This item was submitted to Loughborough University as a PhD thesis by the author and is made available in the Institutional Repository (https://dspace.lboro.ac.uk/) under the following Creative Commons Licence conditions.

For the full text of this licence, please go to: http://creativecommons.org/licenses/by-nc-nd/2.5/
Pilkington Library

Author/Filing Title .................................................................

Vol. No. .......... Class Mark ........................................

Please note that fines are charged on ALL overdue items.


0402155130
A SYSTEMS ENGINEERING AND CONCURRENT ENGINEERING FRAMEWORK
FOR THE
INTEGRATED DEVELOPMENT OF COMPLEX PRODUCTS

by

GEILSON LOUREIRO

A Doctoral Thesis
Submitted in partial fulfilment of the
requirements for the award of
Doctor of Philosophy
Loughborough University

Loughborough University
Department of Manufacturing Engineering
1999

© Geilson Loureiro 1999
Acknowledgements

I, the author, wish to thank:

God for the inspiration, energy and spirit of contribution.

CNPq, the Brazilian Council for the Development of Science and Technology, and INPE, the Brazilian Institute for Space Research, my employer, for the sponsorship.

Dr. Clovis Solano Pereira, head of the Integration and Testing Laboratory (LIT) of INPE, for supporting my case to remain in Loughborough until finishing the writing-up of this thesis.

Dr. Paul G. Leaney for the contacts, resources and patient supervision of the work performed over the last four and a half years.

Dr. Ian C. Wright for supporting my application to Loughborough University.

The Department of Manufacturing Engineering staff, especially Dave Walters, Mary Treasure and Sheralyn Thorne, for their always prompt attention and service.

Mike Hodgson, Bob Jaques, David Pickman, Barrey Sunley and all those at the Powertrain Control Systems Engineering Division of Ford (now of Visteon) at Dunton, England, for their collaboration.

Lee Barnett for lending his manufacturing engineering expertise to this work and, Neil Adcock for providing information, for the development of the seat height adjust mechanism example.

3SL and their staff for Cradle, the main software used in this research.

Dr. Carolyn S. Griner, Deputy Director of the NASA’s Marshall Space Flight Centre, for her words of encouragement to this work, after my paper presentation at the 49th Congress of the International Astronautical Federation, in Melbourne, Australia, October, 1998.

The Loughborough Brazilian community for the many lessons learned on how to live with solidarity.

The Sutton Bonington Baptist Church for adopting my family as part of theirs.

My friends, Mr. Silas Peres da Costa and Mrs. Sely Maria de Sousa Costa, for their love, support, wise advice and example of faith as they received me as part of their family for the period March-July, 1999.

My parents, Mr. Getulio Pimentel Loureiro and Mrs. Nadir Vicente Loureiro, for everything I am.

My wife Maristela and my born-in-England daughter Isabel for making this journey with me with love and patience.
Abstract

The aims of this research are to investigate the development of complex products (e.g. cars, satellites, aeroplanes), to identify areas for improvement and to make a contribution to those areas. The premise is that the shortcomings of a component design focused concurrent engineering (CE) and of a product focused systems engineering (SE) can be addressed by seeking a 'total view framework' that encompasses both, CE and SE. A broader scope of product development becomes necessary for coping with the complexity of the current very dynamic manufacturing business environment. Initial work was to undertake a comprehensive literature review drawing out the perceived needs in general as well as the particular needs of the space and automotive industries. This was supplemented with periods spent in industry with a major automotive manufacturer.

It was found that complex product manufacturing industry faces a very dynamic and highly competitive global marketplace. In such a dynamic environment, the ongoing success of a development organisation is translated by its capacity to continuously shorten development cycle time, reduce cost, manage risks and, at the same time, improve product performance. The achievement of these objectives is highly dependent on the organisation's ability to cope with changes and with the complexity that may result from them. This involves identifying the elements that are likely to change and the interactions among them very early in the product development life cycle, in its conceptual stage. These elements are not only part of the product itself but are also part of the product life cycle processes and their performing organisations. Traditional development approaches provide only a partial picture of these elements and their interactions. For example, the traditional automotive approach fails to capture the interactions among the product elements. It uses a component approach that treats the product as a set of isolated components rather than as an integrated system. CE tools treat life cycle process elements in isolation of each other. Traditional satellite SE approach develops the product as a system but does not consider its life cycle processes as part of the system. In order to cope with product inherent complexity and with the complexity that may arise from changes, it is necessary to adopt an integrated development approach for complex product development.

In response to the needs, this thesis defines integrated development as an approach to develop concurrently and in an integrated manner a product, its life cycle processes and their performing organisations as elements of a balanced solution. The approach is stakeholder-driven instead of only customer-driven and is centered on the concepts of requirements and attributes as the interface between the stakeholders in the problem domain and the product, process and organisation in the solution domain. The backdrop to this was the development of two case studies:

- the gasoline PCS (Powertrain Control System) case study illustrated the automotive component approach and showed the consequent complexity growth in a product that evolved over the years since 1978;
- the diesel PCS case study illustrated the application of a structured approach to the development of a new automotive subsystem and laid down the basis of the framework developed in this thesis.

The proposed framework for the integrated development of complex products is called 'total view framework' because it provides the total set of product, process and organisation elements and their interactions from the outset in the development process. It uses SE and CE in an integrated manner, as part of the same framework. The framework extends the application of the SE process to life cycle processes and their performing organisations and applies CE at all levels of the hierarchical product breakdown structured.

The 'total view framework' is supported by a method, called 'concurrent structured analysis method', that consists of the three analysis processes: requirements, functional and physical. These processes mirror the bulk of the SE process and are applied concurrently to product, process and organisations. The outputs of the method are requirements, functional attributes, physical attributes and the interactions among them. These outputs are then analysed using a clustering algorithm and a complexity metric based on cohesion and coupling shows the clustering effects.

The 'concurrent structured analysis method' is demonstrated through its application to three automotive subsystems: the diesel and gasoline PCSs and a seat height adjust mechanism (SHM). The demonstration of the framework uses a commercial systems engineering environment software package, called Cradle, for modelling product, life cycle processes and organisation and capturing the interactions among their requirements and attributes. The demonstration also uses Chaco-2.0, a graph partitioning algorithm, and Excel to analyse and present these interactions, respectively.

Conclusions are that the framework and method are supportive of the trends in the space and automotive industries by encompassing CE and SE. Also the framework addresses the needs for a broader scope of complex product development, for the integration of product, process and organisation and for complexity management. The method and use of commercial tools demonstrate the feasibility of the framework and launch the challenge to make it more pragmatic. Opportunities for further work include the extension of the framework to include product families, integration of the systems engineering environment with simulation and design tools and a more systematic attributes engineering process. The value of the framework, concepts and method challenges for their adaptation to the real industrial environment and to the current level of technology available in a move towards pragmatism.
# Table of contents

Title page  
Declaration  
Acknowledgements  
Abstract  
Table of contents  
List of figures  
Acronyms  
Glossary  

## Part I Needs

### 1 Introduction
1.1 Background  
1.2 Aims and objectives  
1.3 Research approach  
1.4 Structure  

### 2 Literature review
2.1 Introduction  
2.2 Complexity, complex products and approaches to manage complexity  
2.2.1 Concepts  
2.2.2 Complex products  
2.2.3 Conditions to manage complexity  
2.2.4 Complexity metrics  
2.2.5 Interactive management  
2.2.6 The ‘total’ approaches  
2.2.6.1 Total design  
2.2.6.2 Total quality control  
2.2.6.3 Total quality development  
2.2.6.4 Total systems approach  
2.2.7 Business process reengineering (BPR)  
2.2.8 Holistic enterprise evolution  
2.2.9 Holonic manufacturing systems  
2.2.10 The fractal factory  
2.2.11 Information modelling  
2.2.12 Evolvability, enduring architectures and product families  
2.2.13 Modularity and integration  
2.2.14 Mechatronic, Complex Electro-Mechanical and VLSI development approaches  
2.2.15 Partitioning and hierarchy  

### 2.3 Integrated development
2.3.1 IPD  
2.3.2 IPPD  
2.3.3 IPPED  
2.3.4 Additional ‘P’s in integrated development  

### 2.4 Concurrent engineering
2.4.1 Process initiatives  
2.4.2 Computer-based support


2.4.3 Data interchange methods 39
2.4.4 Formal techniques 40
  2.4.4.1 Quality function deployment [and the Kano model] 41
  2.4.4.2 Taguchi methods 44
  2.4.4.3 Axiomatic design 44
  2.4.4.4 Theory of inventive problem solving 45
  2.4.4.5 Design for assembly 46
  2.4.4.6 Group technology 47
  2.4.4.7 Value engineering 47
  2.4.4.8 FMEA technique 48
  2.4.4.9 Hazard Analysis 48
2.4.5 Techniques integration 49
2.5 Systems engineering (SE) 50
  2.5.1 System 51
  2.5.2 Systems thinking 52
  2.5.3 Systems engineering definitions 53
  2.5.4 Systems engineering models 55
  2.5.5 Systems engineering standards and processes 56
  2.5.6 Systems engineering methods 61
  2.5.7 Systems engineering tools 62
  2.5.8 SEDRES 65
  2.5.9 New technologies 66
3 Trends in the space and automotive industries 67
  3.1 Introduction 67
  3.2 Space industry 67
    3.2.1 Characteristics 68
    3.2.2 Trends 73
    3.2.3 SE concepts by NASA 74
    3.2.4 NASA’s Faster, cheaper, better 78
  3.3 Automotive industry 80
    3.3.1 Characteristics 80
    3.3.2 Trends 85
    3.3.3 Automotive electronics 86
    3.3.4 Systems engineering 88
    3.3.5 FPDS 89
    3.3.6 C3P 96
  3.4 Comparison 98
3.5 Conclusions 100
4 The gasoline and diesel PCS case studies 101
  4.1 Background 101
  4.2 The gasoline PCS 102
    4.2.1 Stakeholders 105
    4.2.2 Requirements and attributes 107
    4.2.3 Product functions 112
    4.2.4 Product components 116
    4.2.5 The development organisation evolution 122
    4.2.6 The development process evolution 127
    4.2.7 Lessons learned 131
7.4.2 Process modelling
7.4.3 Organisation modelling
7.5 Analysis
7.5.1 Requirements analysis
7.5.2 Functional analysis
7.5.3 Physical analysis

Part IV Implementation

8 Overview of examples and use of tools
8.1 Examples overview
8.2 Tools
  8.2.1 Cradle
    8.2.1.1 Why Cradle?
    8.2.1.2 Cradle use
  8.2.2 Chaco-2.0
  8.2.3 Excel

9 The PCS examples
9.1 Introduction
9.2 The car and powertrain examples
9.3 The diesel PCS example
  9.3.1 Requirements analysis
  9.3.2 Functional analysis
  9.3.3 Physical analysis
  9.3.4 Requirements, attributes and relationships
9.4 The gasoline PCS example
  9.4.1 Background
  9.4.2 The complexity analysis approach and procedure
  9.4.3 Analysis of the results
  9.4.4 Concluding remarks
  9.4.5 Other applications within the total view framework

10 The SHM example
10.1 Introduction
10.2 Requirements analysis
10.3 Functional analysis
10.4 Physical analysis
10.5 Requirements, attributes and relationships
10.6 Lessons learned

Part V - Evaluation

11 Evaluation and discussion
11.1 Introduction
11.2 The ‘total view framework’
  11.2.1 Needs
  11.2.2 Scope
  11.2.3 Integration
  11.2.4 Complexity management
11.3 The ‘concurrent structured analysis method’
  11.3.1 The analysis processes
  11.3.2 Modelling
11.4 The implementation
11.4.1 The diesel PCS example 337
11.4.2 The gasoline PCS example 338
11.4.3 The SHM example 340
11.5 Evaluation against the trends in the complex product industry
11.5.1 Space industry 342
11.5.2 Automotive industry 344
11.5.3 Both industries 346
11.6 Evaluation against the needs highlighted in the case studies 347
11.7 Relationship and comparison with general complexity management approaches 349
11.8 Comparison with integrated development 354
11.9 Comparison with concurrent engineering 355
11.10 Comparison with systems engineering 359
11.11 Critical analysis 361

12 Conclusions and further work 363
12.1 Conclusions 363
12.1.1 The needs for a total view framework 363
12.1.2 The conception of the framework and its main elements 364
12.1.3 The design of a method within the framework 366
12.1.4 Examples of method implementation 368
12.1.5 Evaluation of framework, method and implementation 370
12.2 Results from critical analysis 373
12.2.1 Contribution 373
12.2.2 Issues not addressed 374
12.2.3 Topics for improvement 374
12.3 Opportunities for further work 375

References 377

Appendices
A Structured and object oriented analysis methods A-1 to 14
B Cradle overview B-1 to 9
C System modelling notations C-1 to 20
D Details of the diesel PCS project in Cradle D-1 to 36
E Details of the seat height adjust example E-1 to 34
F Results along the complexity analysis procedure applied to the gasoline PCS F-1 to 39
G Other publications by the author G-1 to 32
List of figures

1 Introduction
   1-1: The structure of the thesis 4

2 Literature review
   2-1: Two problem situations 8
   2-2: A complexity metric 12
   2-3: Block diagram of interpretive structural modelling process 14
   2-4: Total design activity model 16
   2-5: Elements of the PDS 17
   2-6: Redesigned organisation according to the three Cosmos dimensions 20
   2-7: ZOPH's formal language 23
   2-8: Modular VLSI versus integrated CEM 28
   2-9: Integrated product development 30
   2-10: IPD lessons learned 32
   2-11: Functional organisation structure showing IPPD/IPTs 33
   2-12: Manufacturing business supported by IPPED tools 35
   2-13: Typical uses of common CE methods in the product development process 37
   2-14: Scope of CIM 40
   2-15: The translation process of QFD 41
   2-16: General structure of QFD matrices 42
   2-17: The Kano model 42
   2-18: Hierarchy of $Q^2$FD matrices 43
   2-19: Example of failure mode effects analysis 48
   2-20: Classification of risks as a function of gravity and frequency of hazards 49
   2-21: Scope of systems according to modern standards 52
   2-22: Three definitions of systems engineering 54
   2-23: The waterfall model 55
   2-24: The spiral model 56
   2-25: The generic 'vee' model 56
   2-26: The systems versus software boundary 57
   2-27: Overview of SE standards evolution 57
   2-28: The processes for engineering a system 58
   2-29: The systems engineering process 59
   2-30: Description of the systems engineering processes 60
   2-31: Evolution of SE methods 63
   2-32: Data exchange vision 65

3 Trends in the space and automotive industries
   3-1: Overview of programmatic aspects of the space sector and the major technology requirements/drivers 69
   3-2: FireSat life cycle cost estimate 70
   3-3: Development phases of a major DoD space program 71
   3-4: Conventional space program phases 71
   3-5: Conventional space system development approach 73
   3-6: Trends in the space industry 75
   3-7: The spiral model: doctrine of successive refinement 77
   3-8: 'Faster, cheaper, better' space program 78
   3-9: Relationship between product innovation and process innovation 81
   3-10: A typical automobile life cycle 81
   3-11: Changes during product development cycle 83
   3-12: Typical Japanese and U.S supplier systems 84
   3-13: Some trends in the automotive industry 85
   3-14: Rising percentage of cost of electronic components in automobiles (intermediate size Nissan passenger car) 86
   3-15: Increasing use of on-board electronic systems 87
   3-16: Automotive usage of microcontroller technology 87
   3-17: Ford's business process model 90
   3-18: FPDS scaleability 90
   3-19: FPDS and systems engineering 91
   3-20: WCP's component approach versus FPDS' systems approach 92
3-21: FPDS work breakdown structure
3-22: Vehicle attributes
3-23: Attributes cascade
3-24: WCP versus FPDS product/process design paradigms
3-25: WCP emphasis on prototyping and testing versus FPDS emphasis on up-front analysis
3-26: Launch process characteristics under WCP and FPDS
3-27: Early supplier involvement
3-28: FPDS programs team structure
3-29: The product information management approach in C3P
3-30: Automotive versus space industries
3-31: Evolution of product development approaches in the automotive and space industries

4 The gasoline and diesel PCS case studies
4-1: The PCS in the vehicle
4-2: History of the EEC systems for gasoline
4-3: PCS stakeholders evolution
4-4: Federal American and Californian environment agencies organisations
4-5: Federal and California emissions standards compared to emissions from a typical car in 1960
4-6: Overview of emission control regulations
4-7: U.S. Federally imposed fuel economy standards
4-8: Ford's definition and evaluation of driveability
4-9: Voice of the customer for driveability
4-10: Attributes that determine possible PCS configurations
4-11: The evolution of PCS functions and modes of operation
4-12: Mapping between requirements and EEC-I functions
4-13: Mapping between requirements and EEC-II functions and modes of operation
4-14: Mapping between requirements verification conditions and EEC-III modes of operation
4-15: Sensors evolution, from EEC-I to EEC-III
4-16: Actuators evolution from EEC-I to EEC-III
4-17: EEC-IV's sensors and actuators and their variety
4-18(a): EEC-I sensors-actuators mapping
4-18(b): EEC-II sensors-actuators mapping
4-18(c): EEC-IV sensors-functions mapping
4-19: Reuse, ease of calibration and functionality addressed by PCS software
4-20: PTEC's Idle Speed Control subsystem interface
4-21: The future - a proposed architecture for the gasoline PCS
4-22: PCSE in the Ford Automotive Operations matrix organisation
4-23: Worldwide PCSE functional organisation
4-24: PCSE organisation changes in 1995
4-25: Gasoline PCS feature groups
4-26: The Visteon organisation
4-27: Simplified gasoline PCS development process
4-28: Feature/application change process
4-29: History of electronic engine control for diesel systems at Ford
4-30: Overview of the Puma/Lynx PCS development process
4-31: Methods and tools used for the Puma/Lynx PCS development
4-32: DOORS
4-33: EGR feature decomposition structure
4-34: Puma/Lynx PCS Clearcase directory structure
4-35: Clearcase version tree for feature ‘zz’
4-36: TeamWork model derivation tree for a feature
4-37: An example of a top-level Application Model – a CPU architecture
4-38: Comparative analysis diesel x gasoline PCSs
4-39: Product focused gasoline PCS development approach
4-40: Product, process and organisation as part of the system solution in the diesel PCS development approach
4-41: The gasoline and diesel PCS approaches to manage complexity
5 The total view framework  
5-1: Generic model for the integrated development of complex products  
5-2: The total view framework  
5-3: SE scope against the total view framework  
5-4: CE scope against the total view framework  
5-5: The constituents of a system  
5-6: Building block structure as according to the EIA-632 and IEEE-1220 standards  
5-7: Building block of highest complexity  
5-8: Building block of minimal complexity  
5-9: Building block of a satellite system  
5-10: Building block of an autopilot system  
5-11: Building block of a passenger car  
5-12: The analysis dimension and the systems engineering process  
5-13: Example hierarchy of building blocks  

6 The concepts of requirements, attributes and relationships  
6-1: Requirements, attributes and relationships in the generic complex product development model  
6-2: The evolution of requirements over the requirements analysis process  
6-3: Correspondence between what attributes describe and types of attributes  
6-4: Relationships of interest: traceability and impact links  
6-5: Generic deployment matrix format  
6-6: Requirements and attributes deployment along system layers  

7 The concurrent structured analysis method  
7-1: The scope of the method  
7-2: Method overview – method applied at a given system layer  
7-3: Proposed modelling approaches and notations  
7-4: Overview of product modelling  
7-5: Overview of process modelling  
7-6: Organisation requirements model  
7-7: Organisation functional modelling  
7-8: The organisation physical modelling  
7-9: Inter-relationships among analysis processes  
7-10: Requirements analysis process overview  
7-11: Mission/objective of a car  
7-12: Life cycle processes corresponding to non-operating functions of a car  
7-13: Scenarios of life cycle process 'maintain car system'  
7-14: Scenarios of 'use car system'  
7-15: Organisation performing non-operating functions  
7-16: Product - DFD representing product, stakeholders and concerns  
7-17: Processes - BD representing life cycle processes, stakeholders and their concerns  
7-18: Organisation - UCD representing organisation, stakeholders and their concerns  
7-19: Examples of stakeholder requirements organisation  
7-20: Functional analysis process overview  
7-21: Product - DFD showing a context diagram of a car  
7-22: Process - BD showing a functional analysis a car life cycle processes  
7-23: Organisation - UCD showing modelling an organisation’s interfaces with its environment  
7-24: Product - car functional decomposition  
7-25: Links between functional attributes and functional model elements  
7-26: Overview of the physical analysis process  
7-27: Product - DFD representing a car and its functional relationships with the environment  
7-28: Product - an example of an architecture flow diagram of a car  
7-29: Functional allocation to subsystems  
7-30: Product - FBD representing a car and its physical interfaces with its environment  
7-31: Product - FBD representing an architecture interconnect diagram of a car  
7-32: Process - Physical model of FPDS  
7-33: Links between physical model elements and physical attributes
8 Examples overview and tools used

8-1: The diesel PCS example 228
8-2: The gasoline PCS example 229
8-3: The SHM example 230
8-4: Tools used for implementation 231
8-5: Cradle setup for the total view framework 233
8-6: Events as sources of requirements 237
8-7: Example of requirements structuring using Cradle 237
8-8: Analysis of stakeholder requirements using Cradle 238
8-9: An example of technical requirement in Cradle 239
8-10: Example of traceability from technical requirement to stakeholders 239
8-11: Traceability over the requirements analysis processes 240
8-12: Distinction between cross-reference types and groups 240
8-13: Transitivity within cross-references groups 241
8-14: List of Cradle’s modelling notations used 241
8-15: Adaptations required on Cradle notations 242
8-16: Attributes management in Cradle 244
8-17: Cradle link types linking items of information 245

9 The PCS examples

9-1: Example of a partial matrix ‘stakeholder requirements versus technical requirements’ for a car 250
9-2: Example of a partial matrix ‘technical requirements versus functional attributes’ for a car 251
9-3: Example of a partial matrix ‘functional attributes versus physical attributes’ for a car 253
9-4: Car system partitioning and examples of functional attributes deployment 255
9-5: Example of a partial matrix ‘stakeholder requirements versus technical requirements’ for the powertrain 256
9-6: Example of a partial matrix ‘technical requirements versus functional attributes’ for the powertrain 257
9-7: Example of a partial matrix ‘functional attributes versus physical attributes’ for the powertrain 258
9-8: Product_The diesel PCS requirements model 259
9-9: Process_The diesel PCS life cycle processes requirements model 260
9-10: Organisation_The PCSE organisation requirements model 260
9-11: Product_The diesel PCS product functional model (top level 1) 261
9-12: Product_PCS functional decomposition (level 3) 262
9-13: Process_PCS life cycle processes (top level of functional decomposition) 262
9-14: Process_the process 'improve feature' at the fourth level of process functional decomposition 263
9-15: Organisation_ the PCSE organisation functional model 263
9-16: Product_PCS components and their links 264
9-17: Product_PCS physical decomposition into software subsystems or features 265
9-18: Product_functional analysis of the EGR control feature 266
9-19: Product_structure of the EGR software modules 266
9-20: Process_diesel PCS evolution process physical analysis 268
9-21: Organisation_PCIE functional units 269
9-22: Organisation_resources in the PCSE organisation for diesel PCS development 269
9-23: Product_TRACEability links between requirements and functions of the diesel PCS product 270
9-24: Product_TRACEability links between PCS functions and PCS components 270
9-25: Product_TRACEability links between PCS functions and EEC module components 270
9-26: Traceability links between PCS functions and microcontroller components 270
9-27: Traceability links between PCS functions and software subsystems (features) 270
9-28: Product, process and organisation_Impact links between functional and physical attributes of the EGR_control feature 271
9-29: The feature sections 273
9-30: Distribution of functionality among strategy document sections 274
9-31: Number of partitions of the software 274
9-32: Ford’s partitioning into 5 sets and Chaco’s sequencing 278
9-33: Balanced spectral bisection partitioning into 4 sets 279
9-34: Complexity indices calculated from Ford's partitioning compared with algorithmic partitioning
9-35: Correspondence between the number of software units in Ford's and in the algorithmic partitioning
9-36: Relationships between section 12.1, 12.3 and 12.4 vertices which swap partitions
9-37: Theoretical N2 chart for Ford's partitioning
9-38: N2 chart after Ford's partitioning and sequencing using Chaco-2.0
9-39: Theoretical N2 chart for algorithmic partitioning
9-40: N2 chart after partitioning and sequencing using Chaco-2.0
9-41: Non-expected relationships between Ford's partitions in Figure 9-32
9-42: Non-expected relationships between algorithmic partitions in Figure 9-33
9-43: Hypothetical total system development organisation

10 The SHM example
10-1: SHM, its components and assembly process
10-2: Product – SHM product requirements model
10-3: Process – Identification of SHM's life cycle processes
10-4: Process – The SHM assembly requirements model
10-5: Process – The SHM manufacturing requirements model
10-6: Organisation – Adcock Ltd. organisation requirements model
10-7: Functions of the SHM individual components
10-8: Product – Functional context diagram of the SHM
10-9: Product – Functional decomposition of the SHM
10-10: Process – the SHM life cycle processes top level functional model
10-11: Process – the SHM production process functional model
10-12: Process – The manufacturing part process functional model
10-13: Process – the Assemble_SHM process functional model
10-14: Organisation – the H.R. Adcock Ltd organisation functional model
10-15: Organisation – the 'Produce_SHM' function decomposition
10-16: Organisation – the 'Manufacture_parts' function scenarios
10-17: Organisation – other 'Manufacture_parts' function scenarios
10-18: Organisation – Sequence Diagram for modelling the Manufacturing_tube function
10-19: Organisation – Collaboration Diagram for modelling the Manufacture_tube function
10-20: Product – the SHM physical context diagram
10-21: Product – SHM's architecture interconnect diagram
10-22: Product – SHM's architecture flow diagram
10-23: Process – the SHM manufacturing and assembly process
10-24: Organisation – the H_R_Adcock limited organisation Use Case Model
10-25: Organisation – Package Diagram showing organisation departments
10-26: Organisation – Package Diagram showing different views of the manufacturing department
10-27: Organisation – a Class Diagram representing a resources model
10-28: Organisation – a Statechart Diagram describing the behaviour of a resource 'Tube_Lathe'
10-29: Organisation – an Activity Diagram describing operations performed by a resource 'Tube_Lathe'
10-30: Organisation – a Component Diagram describing a hierarchy of manufacturing processes
10-31: Organisation – a Collaboration Diagram describing flow between manufacturing resources
10-32: Organisation – A Deployment Diagram showing connections between resources
10-33: Requirements versus stakeholders matrix
10-34: Example of derivation of product functional attributes
10-35: Example of derivation of process functional attributes
10-36: Example of derivation of organisation functional attributes
10-37: Requirements versus functional attributes matrix for the SHM
10-38: The functional attributes self-interaction matrix for the SHM
10-39: The functional attributes versus physical attributes matrix for the SHM
10-40: The physical attribute self-interaction matrix for the SHM
11 Evaluation and discussion

11-1: Complexity factors and framework dimensions

Appendices

A Structured and object oriented analysis methods
A-1: IDEF0 ICOM Box
A-2: SADT/IDEFO connectivity arrows
A-3: Top-level function or activity of a manufacturing process
A-4: Expansion of the top level function or activity ‘CODEC Ltd. Gearbox Manufacturing’
A-5: Diagrammatic representation of the HPM
A-6: Requirements-to-architecture iterative loop for HPM
A-7: The vocabulary of the UML
A-8: Classes and interfaces
A-9: Use cases and collaborations
A-10: Active classes, components and nodes
A-11: Interactions
A-12: State machines
A-13: Packages
A-14: Notes
A-15: Dependency relationships
A-16: Association relationships
A-17: Generalisation relationships
A-18: Extensibility mechanisms
A-19: Models of the unified process and their dependencies
A-20: The system life cycle as approached by the Unified Process
A-21: Workflows distribution over the Unified Process phases

B Cradle overview
B-1: Cradle modules
B-2: Cradle Project Database Structure
B-3: Cross-references parameters popup
B-4: Recursion in transitive System Note cross-references
B-5: Re-entrancy in transitive System Note cross-references
B-3: Modelling notations supported by Cradle

C System modelling notations
C-1: The N² chart
C-2: Example of DFD showing its elements
C-3: Example of ERD showing its elements
C-4: Example of STD showing its elements
C-5: Example of DSD showing its elements
C-6: Example of BD showing its elements
C-7: Example of FBD showing its elements
C-8: Example of STC showing its elements
C-9: Example of UCD showing its elements
C-10: Example of PD showing its elements
C-11: Example of SQD showing its elements
C-12: Example of COD showing its elements
C-13: Example of CD showing its elements
C-14: Example of SCD showing its elements
C-15: Example of ACD showing its elements
C-16: Example of CPD showing its elements
C-17: Example of DPD showing its elements

D Details of the diesel PCS project in Cradle
D-1: DFD 01.1.5.1 – Puma Lynx Control System context diagram
D-2: DFD 01.1.5.1.1 – PCS functional decomposition
D-3: DFD 01.1.5.1.1.1 Detect_Crankage State
D-4: DFD 01.1.5.1.1.1.1 DT_Determine_Engine_State
D-5: DFD 01.1.5.1.1.5 – Engine_Cranking
D-6: DFD 01.1.5.1.1.7 Engine_Running
E Details of the seat height adjuster example

E-1: Johnson Controls requirements

E-2: Johnson Controls’ SHM DFMEA

E-3: SHM DFMEA regarding effort, driveability, freeplay, feel, noise, weight and cost aspects

E-4 (a): Customer requirements captured by Ford Motor Company

E-4 (b): Customer requirements captured by Ford Motor Company

E-4 (c): Customer requirements captured by Ford Motor Company

E-5: SHM’s process sheet

E-6: DFD 0 - The SHM functional context diagram

E-7: DFD 16 – The ‘Provide_Seat_Height_Adjust_Funclion’

E-8: DFD 16.41 – The ‘Handle_Interface’ function

E-9: DFD 16.42 – The ‘Holdin Interface_To_Frame’ function

E-10: DFD 16.43 - The ‘Handle_at_a_constant_position_from_user’ function

E-11: DFD 16.44 – The ‘Provide_frame_displacement’ function

E-12: DFD 16.44.41 – The ‘Transmit_handle_rotation’ function

E-13: DFD 16.44.42 – The ‘Transform_rotation_from_handle_into_translation’ function

E-14: DFD 16.45 - The ‘Moving_interface_to_frame’ function

E-15: DFD 16.47 – The ‘Provide_a_constant_handling_effort’ function

E-16: DFD 16.47.41 – The ‘Provide_pre_load’ function

E-17: DFD 16.47.42 – The ‘Balance_friction’ function

E-18: STD 16.46 – The ‘Provide_limits_for_displacement’ function

E-19: BD – 16: SHM life cycle processes

E-20: BD – 16.2: SHM production

E-21: BD – 16.2.1: Manufacture N parts

E-22: BD – 16.2.1.1: Manufacture part 1

E-23: BD – 16.2.3: Assemble SHM

E-24: BD – 16.2.4: Assemble M sub-assemblies

E-25: BD – 16.2.4.1: Assemble subassembly 1


E-27: BD 16.22.21 – Manufacture_spindle

E-28: BD 16.22.21.21 – Produce_blank_D069_P01

E-29: BD 16.22.21.22 – M5_thread_D069_P02

E-30: BD 16.22.23 – Manufacture_hobbin

E-31: BD 16.22.23.21 – Produce_blank_D093_P01

E-32: BD 16.22.24 – Manufacture_tube

E-33: BD 16.22.24.21 – Cut_to_length_D096-P01

E-34: BD 16.22.25 – Manufacture_ball_race_assembly_D103_P01
E-35: BD 16.11.26 - Manufacture_Parts
E-36: BD 16.11.26.21 - Produce_blank_D104_P01
E-37: BD 16.11.27 - Assemble_assembly_1
E-38: BD 16.11.29 - Assemble_assembly_2_D062_P01
E-39: BD 16.11.211 - Assemble_assembly_3_D062_P02
E-40: BD 16.11.213 - Assemble_assembly_4_D062_P04
E-41: BD 16.11.215 - Assemble_assembly_5_D062_P05
E-42: BD 16.11.216 - Assemble_assembly_6
E-43: Top level product functional attributes extracted from the DFD 12 - Seat_Height_Adjust_Mechanism_Context
E-44: Second level product functional attributes extracted from DFD 12.16 - Provide_seat_height_adjustment
E-45: Third level product functional attributes extracted from DFD 12.16.44 - Provide_frame_displacement
E-46: Third level product functional attributes extracted from DFD 12.16.47 - Provide_a_constant_handling_effort
E-47: Third level functional attributes extracted from STD 12.16.46 - Provide_limits_for_displacement
E-48: SHM_production attributes extracted from the SHM life cycle process model BD 16
E-49: SHM_production attributes extracted from BD 16.2
E-50: Process functional attributes extracted from the BD 16.2.1 - Manufacture_N_parts
E-51: Process functional attributes extracted from BD 16.2.1.1 Manufacture_part_1
E-52: Process functional attributes extracted BD 16.2.4 - Assemble_M_subassemblies
E-53: Process functional attributes extracted from BD 16.2.4.1 - Assemble_subassembly_1
E-54: Organisation functional attributes extracted from the top level UCD 16 - H_R_Adcock_Ltd_Business_processes
E-55: Organisation functional attributes extracted from the UCD 16.2.1 - Manufacture_parts
E-56: Organisation functional attributes extracted from the COD 16.2.1.5 - Manufacture_tube
E-57: Organisation functional attributes extracted from the SQD 16.2.1.5 - Manufacture_tube

F Results along the complexity analysis procedure applied to the gasoline PCS
F-1: Top-level sections of the ISC_KBADO strategy document
F-2: Composition of the inlet air control executive feature-IACEX-V3.0_IACEX_section 12.1
F-3: Composition of the ‘generic idle speed control’ strategy module IACEX OVERVIEW_4-sub-section 12.1.1
F-4: Composition of the ‘mode select’ strategy module-IACEX MODE_SELECT_2-sub-section 12.1.2
F-5: Composition of the ‘iac (idle speed control) SCCD output-IACEX_DTY_CONV_2-Sub-section 12.1.4
F-6: Composition of the IACDN – NO_VERSION_LEVEL feature – section 12.2
F-7: Composition of the ‘desired RPM calculation’ strategy module IACDN_DSDRPM_BASE_1-sub-section 12.2.2
F-8: Composition of the ‘DSDRPM (loads) calculation’ strategy module IACDN_DSDRPM_LOADS_1-sub-section 12.2.4
F-9: Composition of the ‘battery charging RPM calculation’ strategy module IACDN_DSDRPM_BAT_V_1-sub-section 12.2.6
F-10: Composition of the ‘IACFB- no_version_level’ feature – section 12.3
F-11: Composition of the ‘IPSIBR calculation’ strategy module IACFB_IPSIBR_2-sub-section 12.3.1
F-12: Composition of the ‘KAM update (ISCKAM_update)’ strategy module IACFB_ISCKAM_1-sub-section 12.3.3
F-13: Composition of the ‘IACFF – no_version_level’ feature – section 12.4
F-14: Composition of the ‘dashpot calculations’ strategy module IACFF_DASPOT_6-sub-section 12.4.9
F-15: Composition of the IACHW (idle air control – heated windshield) feature – section
F-16: Top-level software modules and IAC_EXEC (sub-section 12.1.1.1) structure
F-17: IAC_MODE_SELECT (sub-section 12.1.2.1) structure
F-18: DSDRPM_CALC structure (sub-section 12.2.1.1)
F-19: IPSIBR_CALC (sub-section 12.3.1.1) structure
F-20: ISCKAM_UPDATE (sub-section 12.3.3.2) structure
F-21: DASHPOT_EXECUTION_PROCESS (sub-section 12.4.9.1) structure
F-22: DSDRPM_OUTPUT (sub-section 12.2.4.2) structure
F-23: ISCDTY_CALC (sub-section 12.1.3.1) structure
F-24: ISCDTY_CALC (sub-section 12.1.3.1) structure
F-25: The top-level DFD diagram showing the idle speed control software and its interfaces
F-26: DFD 40.1 - IAC_EXEC decomposition
F-27: DFD 40.1.1 - EXEC_ALL NORMAL_ISC_PROCESSES
F-28: DFD 40.1.1.2 - INTS_UPDATE (subsection 12.1.1.3)
F-29: DFD 40.1.1.3 - N_RATCH_UPDATE (subsection 12.1.1.4)
F-30: DFD 40.1.1.5 - IAC_MODE_SELECT (subsection 12.1.2.1)
F-31: DFD 40.1.1.6 - ISCDTY_CALC (subsection 12.1.3.1)
F-32: DFD 40.1.1.7 - IDLE_DRIVE_MODE (subsection 12.1.8.1)
F-33: DFD 40.1.1.9 - IPSIBR_CALC (subsection 12.3.1.1)
F-34: DFD 40.1.1.9.4 - ISCPST_CONTROL_CALC (subsection 12.3.1.4)
F-35: DFD 40.1.1.10 DASHPOT_EXECUTION_PROCESS (subsection 12.4.9.1)
F-36: DFD 40.1.1.10.1 - BASE_DASHPOT_CALCULATION (subsection 12.4.9.4)
F-37: DFD 40.1.1.11 - DESMAF_PRE_CALC (subsection 12.4.2.1)
F-38: DFD 40.1.1.12 - DSDRPM_CALC (subsection 12.2.2.1)
F-39: DFD 40.1.1.12.1 - DSDRPM_OUTPUT (subsection 12.2.4.2)
F-40: DFD 40.1.1.13 ISCKAM_UPDATE (subsection 12.3.3.2)
F-41: DFD 40.1.2 IN CRANK_MODE ENGINE_MOVING
F-42: DFD 40.1.2.2.1 DASHPOT_CALC (subsection 12.2.2.1)
F-43: DFD 40.1.2.2.1 DSDRPM_OUTPUT (subsection 12.2.4.2)
F-44: DFD 41 ISC_FMEM (subsection 12.1.7.1)
F-45: DFD 42 ISC_PERLOAD_ISC (subsection 12.1.6.1)
F-46: DFD 43 ISC_TIMERS (subsection 12.1.5.1)
F-47: DFD 44 IAC_OUTPUT (subsection 12.1.4.1)
F-48: DFD 45 - ISCKAM_QUALIFICATION (subsection 12.3.3.1)
F-49: DFD 46 - HEAT_WIN_LVL (subsection 12.5.3.1)
F-50: List of vertices in the graph represented by the lowest level DFD processes
F-51: List of edges and associated information
F-52: List of vertices, edge weights and vertex neighbours
F-53: Chaco-2.0 input files
F-54: Sequencing output files from Chaco-2.0
F-55: Example of partitioning output files from Chaco-2.0
F-56: Calculation of the complexity index
F-57: Ford's partitioning into 5 sets and Chaco's sequencing
F-58: Imbalanced, multi-level KL partitioning into 4 sets
F-59: Balanced spectral bisection partitioning into 4 sets
F-60: Ford's partitioning into 32 sets and Chaco's sequencing
F-61: Imbalanced multi-level KL partitioning into 32 sets
F-62: Balanced spectral bisection partitioning into 32 sets
Glossary

Attributes: are the properties and characteristics which are possessed by a product, a process or an organisation and which are intended to meet stakeholders' requirements.

Building block: consists of (1) the system, (2) one or more end products, (3) their life cycle processes and (4) the organisations that perform those processes.

Concurrent engineering: a systematic approach for the integrated, concurrent design of products and their related processes, including manufacturing and support. This approach is intended to cause the developers from the outset, to consider all elements of the product life cycle from concept through disposal, including quality, cost, schedule, and user requirements.

Concurrent structured analysis method: is a method, within the total view framework, that applies the requirements, functional and physical analysis processes, concurrently, to the product, its life cycle processes and their performing organisations at all layers of the product breakdown structure. The method is implemented through the adaptation of structured and object oriented analysis techniques to model product, process and organisation.

Complex product: has the following characteristics:
- affects many different types of stakeholders;
- there is strong coupling between its independently specified objectives;
- serves many different functions;
- has multiple modes of operations;
- is multidisciplinary, i.e., requires engineers from many disciplines during its development process;
- are composed by many subsystems;
- has a large number of interfaces;
- is composed of highly integrated (tightly coupled) units;
- examples are automobiles, aeroplanes, satellites.

Development agency: is the organisation that develops a 'building block' of the building block project hierarchy. The development of complex products requires the work of many development agencies.

Enabling product: a product whose function is to enable an end-product life cycle processes, e.g., getting an end product in service, keeping it in service or ending its service.

End product: performs the operations functions of the system; is delivered to an end-user

Functional analysis: a process that derives a functional architecture, from which functional attributes are identified. As part of the systems engineering process, functional analysis translates [technical] requirements into a functional architecture that provides an arrangement of functions, their decompositions and interactions.

Functional attributes: are the properties that describe product, process or organisation functionality, not related to any specific implementation of that functionality; can be obtained from functional models.

Integrated development: is an approach for the integrated and concurrent development of a product, its life cycle processes and their performing organisations; can be implemented by applying the systems engineering process to develop, concurrently, the product, process and organisation elements of the system solution at all layers of the product breakdown structure; requires multidisciplinary teams to develop a successful system solution.

Layer: a level of abstraction corresponding to the layers of the product breakdown structure.
Life cycle process: a set of activities that characterises a stage of the product evolution. The product evolution initiates by the perception of stakeholder requirements and ends with the product disposal. Examples of life cycle processes are: development, manufacturing, test, distribution, deployment, operations, support, training, disposal.

Organisation: is a set of resources that perform one or more life cycle processes. Resources include machines, personnel, building. The resources must be organised to maximise the productivity. Therefore, an organisation is likely to perform the same life cycle process for many products, many products in the same product family or even many different product families.

Physical analysis: a set of activities that derives a physical architecture from where physical attributes are derived. It models the physical architecture resulting from a synthesis process. The synthesis, is part of the systems engineering process and translates the functional architecture into a physical architecture that provides an arrangement of elements, their decompositions, interfaces, physical constraints, and designs.

Physical attributes: are the physical characteristics that describe a particular implementation of the system solution composed by product, process and organisation elements; are the characteristics of product components (e.g. geometry), process tasks (e.g. timing), organisation resources (e.g. number of employees); can be obtained from physical models.

Process: is a logico-temporal sequence of activities to transform inputs into outputs.

Product: is the main result of the development process; is engineered; is a constituent part of a system; performs the operations functions of the system; end product; consists of one or more of the following: hardware, software, facilities, data, materials, personnel, services, and techniques.

Requirements analysis: a process that identifies stakeholders and their requirements and translates those requirements in technical requirements, if necessary.

Requirements: are the total set of considerations that govern what is to be accomplished, how well it is to be accomplished and under what conditions it is to be accomplished.

Stakeholder requirements: express what stakeholders require of the system solution composed by product, process and organisation elements; expressed as needs, wants, expectations, desires, priorities, objectives, or capabilities; often stated in non-technical terms; generally, not verifiable using technical verification techniques.

Stakeholders: are the people whose satisfaction or dissatisfaction is affected by the attributes of a product, its life cycle processes or their performing organisations, and, therefore, have requirements to be accomplished by a development agency.

System: is a balanced, structured and integrated composite of product, process and organisation elements that meet stakeholder requirements over the entire product life cycle. The contribution of the system solution is greater than the sum of the contributions of its elements in isolation.

Systems engineering: is an interdisciplinary collaborative approach to derive, evolve and verify a life cycle balanced system solution that satisfies stakeholder expectations.

Technical requirements: are derived from the stakeholder requirements; are stated in technical terms so that they are objectively verifiable.

Total view framework: is a modelling framework that integrates the product, its life cycle processes and their performing organisations throughout the requirements, functional and physical analysis processes, at all layers of the product hierarchy, deriving attributes as emergent properties of a whole integrated system.
Part I
Needs
Chapter 1
Introduction

This chapter presents:
• the background and domain of the thesis;
• aims and objectives addressed by this thesis;
• the research approach adopted;
• the structure of the thesis.

1.1 Background

This thesis concerns the management of complexity while developing or evolving a complex product. It aims to justify, develop and demonstrate a framework that contains some necessary ingredients for coping with complexity in the very dynamic environment of the manufacturing business of the 1990s.

The manufacturing business of the 1990s is characterised by [Stevens et al., 1998]:
• increased complexity of products;
• globalisation of the marketplace, with the erosion of national boundaries as trade barriers;
• reduction of product development cycles in the best industrial companies;
• software as the dominant force for change in almost all new products;
• worldwide deployment of technology in ever shorter timescales;
• systems constructed from bought-in technology and components;
• re-use of components, information and knowledge across projects;
• partnerships for product development leading to world-wide teams;
• the transition from a paper-based control to control through electronically managed information;
• an understanding that intellectual capital often is the major part of the assets of modern organisations.

In order to remain competitive in such an environment, a manufacturing company must be able to continuously adapt its products, processes and organisation [Prasad, 1996a]. Coping with change, without being overwhelmed by the complexity that comes from it, requires a total view including all changing elements and how they affect each other [Kidd, 1994]. The earlier this total view is provided during a product development cycle, the better.

Traditional approaches to develop complex products do not provide such a total view. The traditional component approach used for automotive development considers the vehicle as the result of the sum of its components. It is based on the belief that the optimisation of the whole would be an automatic result of the optimisation of the parts [Ziebart, 1991]. The interactions among parts are neglected.

Concurrent engineering recognises the impact of product design on product life cycle processes. It also recommends to anticipate life cycle process requirements to the early stages of product development. However, its tools and techniques treat a product life cycle processes as isolated entities (e.g. design for manufacturing, design for service, design for the environment) as if a given product characteristic could not affect more than one life cycle process simultaneously [Huang, 1996]. Also, most of the examples found in the literature regarding concurrent engineering refer to the application of tools and techniques to
components of a larger system. In this regard, concurrent engineering is used as an enhancement for the component approach mentioned above.

Systems engineering is the traditional approach used for the development of complex products as it has its origins in the space and aerospace industry. Although systems engineering includes many principles, processes, methods and tools useful for complexity management, its scope does not contain all the elements of a total view. Systems engineering develops a product and identifies the requirements for its life cycle processes ([IEEE, 1995], [EIA, 1997]).

Integrated development is based on the assumption that one individual can, at best, do no more than contribute to a solution when developing a complex product. Therefore teams are a fundamental part of integrated development. However, it must be said that teams are not integrated development. Focusing on team alone can distract management from the real goals of integrated development [Armstrong, 1996] in its role to manage complexity.

The existence of a gap between what is necessary for obtaining a total view and what is provided by modern approaches for the development of complex products is the motivation for the development of the work documented in this thesis. However, this thesis does not exclude those approaches. On the contrary, it aims to use them in an integrated manner to provide a total view framework.

1.2 Aims and objectives

The aims of this thesis are:

- **Framework - total view:** To justify, propose and develop a holistic framework for the integrated development of complex products (e.g. modern automobiles, aeroplanes, spacecraft) based on a requirements-driven systems engineering process and providing full-life cycle support according to the concurrent engineering concept.

- **Method - concurrent structured analysis:** To introduce, justify and develop a method for managing complexity throughout the integrated development process within the total view framework

- **Implementation - automotive applications:** To implement and evaluate the framework and the method against examples from the automotive industry.

In order to accomplish those aims, more specific objectives are:

**Needs:** To identify the needs of complex products development by:

- Reviewing the concepts of complexity and complex products;
- Reviewing generic approaches to manage complexity;
- Reviewing approaches to manage complexity in the product development arena;
- Investigating the trends in approaches to product development in the space and automotive industries;
- Identifying and analysing a complex product that evolved in a reactive manner;
- Identifying and analysing a complex product developed through a structured approach from the outset;
- Comparing and contrasting the results of the two approaches - reactive and structured.

**Concept:** To conceive the framework by:

- Developing a generic model of integrated development;
• Analysing the concepts of systems engineering and concurrent engineering from standards and relevant literature;
• Justifying and developing a framework that encompasses the concepts of systems engineering and concurrent engineering;
• Defining a strategy for integration and complexity management.

Design: To design a method supporting the framework by:
• Reviewing structured techniques;
• Reviewing the systems engineering process proposed in standards and relevant literature;
• Adapting structured techniques to be applicable to the elements of the framework;
• Adapting the systems engineering process to encompass the scope of integrated development.

Implementation: To implement a demonstration of the framework and method against an automotive case study by:
• Selecting a systems engineering environment software package and other tools used;
• Implementing the framework and method within the systems engineering environment capability using automotive complex subsystems as examples of application;

Evaluation: To evaluate the framework, method and implementation against the identified needs and to compare it with existing approaches to cope with complexity in the product development arena.

1.3 Research approach

The conceptual basis was developed through a literature review on complexity, general approaches to manage complexity, integrated development, concurrent engineering and systems engineering.

A review on product development at space and automotive industries was carried out. The space industry was chosen because of its tradition in dealing with complex product issues. The automotive industry was chosen because of the increasing complexity and multidisciplinarity of modern automobiles and because of the additional complication of serving the global marketing. This industrial review was based on literature survey, industry documents and observation of industrial practice. The author worked for 7 years at the Brazilian Institute for Space Research (INPE) having dealt with many stages of satellite life cycle: manufacturing, assembly, integration and testing. The author also spent four months in total in a study period at Ford Motor Company, Dunton, UK.

Two automotive case studies were developed. The case studies were developed at Ford-AVT-PCSE in Laindon-Basildon, England. The objects of study were the gasoline PCS and the diesel PCS and their related life cycle processes and product development organisation. The gasoline PCS has been evolving all over the period 1978-1995 in a very reactive manner. The diesel PCS is a new product development, starting in 1995. The case studies were developed in two stages. In the first stage (07/95-09/95), the issues facing PCS development were investigated. In the second stage (04/97), many aspects of the products were analysed: stakeholders, requirements, functions, architecture, life cycle processes and product development organisation. The information gathered for the development of the case studies was captured by:
4

- interviewing 31 Ford engineers. The interviews were guided and recorded. The resulting audio-tapes were transcribed;
- analysing Ford's internal documents;
- browsing the Ford intranet.

For the implementation and demonstration of the framework and method, Cradle™, a systems engineering environment (SEE) software package, was selected and used. The capabilities of many commercial requirements management and systems modelling software packages were investigated. Cradle has been used by a major aerospace company (BAe) and it provides capabilities of an integrated SEE

For the implementation and demonstration of the method, Microsoft Excel 97™ spreadsheet tool and Chaco-2.0, a software that implements graph partitioning algorithms were also used. Chaco-2.0 software was available on the Internet and a licence was obtained from their developers at SANDIA laboratories in the U.S.A.

1.4 Structure

Figure 1-1 illustrates the structure of the thesis. The phrases under each chapter in Figure 1-1 may not be identical to the actual chapter name in the thesis. The phrases in Figure 1-1 serve just as guidance for the subject in the chapters.

![Figure 1-1: The structure of the thesis](image)

The thesis comprises 12 chapters organised into 5 parts, as illustrated in Figure 1-1, and 7 Appendices. The parts contain the chapters that address each set of objectives listed in Section 1.2.

Chapter 1 is this introduction.

Chapter 2 is a literature review on complexity, general approaches for managing complexity, integrated development, concurrent engineering and systems engineering.

Chapter 3 is a review on the trends in terms of product development approaches in the automotive and space industries.

Chapter 4 documents the analysis of the case studies. The case studies represent an insight of the needs in the automotive industry for the development of complex products.

Chapter 5 justifies, proposes and describes the total view framework.
Chapter 6 deepens in the fundamental concepts of the framework, the concepts of requirements, attributes and relationships.

Chapter 7 justifies, proposes and describes the concurrent structured analysis method supporting the total view framework.

Chapter 8 introduces and justifies the examples of application to be presented in Chapters 8 and 9. Those examples demonstrate the framework and method using automotive subsystems and examples. Also, this chapter describes the use of commercial tools used when developing the examples.

Chapter 9 presents the gasoline and diesel PCS examples of implementation.

Chapter 10 presents the seat height adjust mechanism (SHM) example of implementation.

Chapter 11 draws together the topics discussed along the thesis and evaluates the framework and method against the needs that motivated the development of this work. This chapter also compares the framework and method with other approaches to manage complexity reviewed in Chapter 2.

Chapter 12 concludes the thesis and identifies opportunities for further work.

Appendix A describes systems engineering methodologies mentioned along the thesis.

Appendix B provides an overview of Cradle, the systems engineering environment software package used for the demonstration of the framework and method.

Appendix C describes the modelling notations mentioned along the thesis and those provided by Cradle.

Appendix D contains the information and diagrams used for the development of the diesel PCS example.

Appendix E contains the information and models used for the development of the SHM example.

Appendix F contains the detailed information and models regarding the gasoline PCS example.

Appendix G contains a selection of papers, by the author, that have been accepted for academic journals and international conferences.
Chapter 2
Literature review

This chapter aims to provide a review on:

- complexity, complex products and approaches to manage complexity;
- integrated development concepts and approaches;
- concurrent engineering concepts and methods;
- systems engineering concepts, standards, methods and tools;

2.1 Introduction

The need for continuous changes in product, process and organisation in order to maintain manufacturing business competitiveness, if not appropriately managed, may cause complexity to escalate unnecessarily and with it cost, lead-time, and effort increase [Prasad, 1996a]. Arthur (1993), for example, correlates changes and complexity in a general law: "complexity tends to increase as functions and modifications are added to a system to break through limitations, handle exceptional circumstances or adapt to a world itself more complex [...] The secret of evolution is the continual emergence of complexity [as] growing complexity is often followed by renewed simplicity in a slow back and forth dance, with complication usually gaining a net edge over time". Thomas and Mog (1998) too stress that increased complexity is not necessarily bad. Indeed, increased complexity is strongly correlated to increased technical performance. The objective is to manage complexity to one's benefit rather to realise its manifestations to one's detriment. Schuster (1996) explains how nature employs optimisation during times of scarcity, creative innovation in times of abundance, and modular design to control uncertainty over time. While nature invariably uses increasing levels of complexity in successive stages of evolution, the manner in which complexity is employed is neither random nor linear; rather, nature employs complexity prudently. Warfield (1994) affirms that complexity will either be managed, or it will overwhelm the individual and the society.

This chapter provides a review of approaches to manage complexity. The approaches presented here are not restricted to the product development realm. For example, Warfield (1976) developed a method of structuring societal systems. In the product development arena, an important step towards complexity management was the recognition by Andreasen & Hein (1987) that "the ideal company can be thought of as one in which a single person is in charge, and where knowledge of the market, of design and production and of economic mechanisms are collected together in one person, who is also able to make decisions and willing to run a risk." However, the knowledge immediately available to an individual is very limited even if he has a number of books on his shelves [Hubka, 1982]. Therefore, as the market gets highly segmented, the product becomes multidisciplinary, the production requires multiple resources and the company has to grow, it is necessary "to introduce integrated procedures, aims, attitudes and methods into product development" in order to remain successful. Systems engineering and concurrent engineering are approaches that support integrated development.

This chapter is structured as following: Section 2.2 reviews complexity, complex products and general approaches to manage complexity. Section 2.3 reviews the concepts and evolution of integrated development. Section 2.4 focuses on an approach supporting integrated development: concurrent engineering. Concurrent engineering concepts and methods are then reviewed. Section 2.5 concentrates
on systems engineering. Systems engineering concepts, standards, methods and modern tools are reviewed.

2.2 Complexity, complex products and approaches to manage complexity

Section 2.2.1 reviews some concepts on complexity. Section 2.2.2 reviews the general understanding on the definition of complex products. Section 2.2.3 reviews some conditions to manage complexity. Section 2.2.4 reviews complexity metrics. Sections 2.2.5 to 2.2.16 reviews some approaches to manage complexity focusing on those applicable to the development of complex products.

2.2.1 Concepts

This section presents six definitions of complexity provided by people from different backgrounds: biology, social sciences, aerospace engineering, systems engineering, engineering design and physics. All of them serve some purpose in the product development arena as described below.

Lloyd & Pagels (1988) offer the following definition for complexity of physical systems: “complexity is the measure of how hard it is to put something together.” Although this definition is provided within the context of biological systems, the congruity to complex engineering systems is apparent.

Warfield (1976), in developing a methodology to deal comprehensively with large societal problems, found that one common ingredient of such problems is the complexity they entail. Other common aspects are consequences of complexity:

- complexity implies that one individual can, at best, do no more than contribute to a solution;
- complexity implies that a number of elements involved in a problem is large, and there are many linkages among the elements stemming from a multiplicity of important contextual relations;
- complexity feeds on itself. As the complexity of a problem induces many minds to be applied to it, a new complexity arises — that of sharing among these many minds the products of each other’s thoughts and actions. This in turn leads to another complexity — that of interrelating and distilling group efforts into a format where they assume significant utility;
- widespread knowledge of proposed change needs to be understood and supported by a constituency with power to effect change.

Warfield generalised his complexity management approach to include systems other than societal [Warfield, 1994] and formalised a concept of complexity and its escalation. According to him, complexity is the combination of two components: situational complexity and cognitive complexity. Cognitive complexity is the dilemma presented to the human mind when it engages with conceptualisations that are beyond its unaided powers. Situational complexity represents those aspects of phenomena that are intercepted by the mind which induce cognitive complexity. When the mind begins to focus upon a problem or issue, cognitive complexity does not come into play. But when the mind does become oriented toward a complex situation, it is very likely it will escalate.

Escalation may occur because of linkages that take place, in which it may be called “linkage escalation”. Figure 2.1 illustrates two problem situations. In part (a) the problem solver surrounds the problem and
solves it. In part (b), the individual problem solver is enmeshed in a certain perception of the problem. In perceiving an inability to surround the problem, a perfectly reasonable step is taken: to seek out others who will be part of a problem-solving team. Complexity escalation then occurs by the following factors:

- varying perceptions among members of a problem-solving team;
- difficulty in managing group problem-solving efforts;
- the presence of organisational or cultural constraints that suppress useful contributions to problem-solving;
- difficulty of communicating solutions to implementers;
- change in the problem situation with time;

![Figure 2-1: Two problem situations (Source: [Warfield, 1994])](image)

Warfield’s concept of complexity and complexity escalation is very much applicable to the product development activity which is in essence a problem-solving activity.

With an aerospace engineering background, Hitchins (1996) proposes that complexity derives from three factors:

- **variety:** refers to the number of types of elements in a set. For example: in ecology, stability may result from interactions between a profusion of plant and animal types; in an engineering organisation, a richness in skill varieties is important to undertake complex tasks effectively.

- **connectedness:** refers to the number of links or relationships among elements. Different connection schemes exist, some involving more links than others. On some schemes, subsystems function as "post-offices", focal points for collection and dissemination. In others, proliferation of point-to-point connections/relationships exists.

- **disorder:** refers to the level of "tangling" of connections. The more tangled, the more complex.

These factors can be applied to analyse either problem or solution complexity. For example, the number of requirements, how they inter-relate to each other and how they are structured may be seen as correspondence to the variety, connectedness and disorder factors, respectively. Similarly, the number of components in a product, the number of interfaces and the layout of components and interfaces also correspond to variety, connectedness and disorder.

From a systems engineering perspective, Thomas & Mog (1998) propose that system complexity is a joint function of:

- the system architecture: it determines where component interactions are present in the system.
• the technology readiness of components: it determines how much interaction is required to develop the system given the interactions present in the system architecture.

• the team organisational dispersion: it determines the efficiency with which the interactions required to develop the system will be accomplished.

In the engineering design arena, Suh (1990) relates complexity with information content as a measure of knowledge required to satisfy a given functional requirement at a given level of the functional requirement hierarchy. The knowledge required to achieve a task depends on the probability of success. If the task is so configured that it can always be satisfied without any prior knowledge or additional knowledge, the probability of success is unity while the requisite information is zero. In actual design/manufacturing problems it is always tried to transmit the sufficient amount of knowledge so that the probability of achieving the task is as high as possible, although usually less than unity.

Gell-Mann (1995), the physics Nobel Prize-winning author of 'The Quark and the Jaguar', made some remarks on the concept of complexity. He distinguishes computational complexity from algorithmic information content as different ways of expressing the term complexity. Computational complexity is related to time (or number of steps) needed to carry out a certain computation. Algorithmic information content refers to the length of the most concise description of an entity. He proposes the term effective complexity referring to the length of a concise description of a set of the entity's regularities. Thus something almost entirely random, with practically no regularities, would have effective complexity near zero. So would something completely regular, such as a bit string consisting entirely of zeroes. Effective complexity can be high only in a region intermediate between total order and complete disorder. This definition is congruent with the one given by the Chaos theory [Mandelbrot, 1982] which is being used to solve non-linear dynamics problems, e.g., how to return a satellite back to the planned orbit [Macau, 1998]

2.2.2 Complex products

Many authors have been recently mentioning the issue of 'increasing product complexity'.

Focusing on the product itself, Crisp II & Franke (1998) emphasise the following characteristics of complex products: large number of interfaces, multiple modes of operations and multidisciplinary technology implementations requiring high degree of automation. Continuous capability and reliable, fault-tolerant operation are expected as well as a high degree of safety and security. Long service life with periodic upgrades will be required, and these products will need to be integrated with legacy, evolving, and new products.

Prasad (1996a) expands the term 'complex product' to include the complexity of the manufacturing process and of the product development organisation:

• serves many different functions, such as transportation, consumer needs, control and measurements, generation or conversion of energy, etc.

• is composed of highly integrated (tightly coupled) units. The units are interdependent. Each unit may contribute to several required behaviours, and a single required sub-system behaviour may contribute to several required sub-unit behaviours.
• there is strong coupling between independently specified objectives of the product (such as weight, cost, quality, performance, throughput, etc.). In these cases, design decisions are not governed by a single consideration. One-to-one translation from an independent set of organisation goals to the goals of its operating units is often difficult, if not impractical.

• the complexity of the manufacturing process used and the manner in which one designs and develops a product. Even with a simple design, the success in realising a product requires a very large, focused, labour-intensive workforce.

• engineers from various disciplines are called upon during complex decision making.

According to the above characteristics automobiles, satellites and aeroplanes are examples of complex products. Hubka & Eder (1988) provide a scale for product complexity: the simplest level 1 includes a part or component (e.g. shaft, lamination), level 2 contains a group, mechanism or subassembly (e.g. feed box), level 3 contains a machine, apparatus or device (e.g. lathe), level 4 contains a plant. However, they recognise that the stated degrees of complexity are relative within the hierarchy of the product structure.

Thomas & Mog (1998) include multi-disciplinarity and some technologies as distinguishing characteristics of complex products:

• the need for large-scale "hard" real-time embedded computer systems that interact with and adapt quickly to rapidly changing environments and conditions;

• the use of many heterogeneous resources and distributed, highly parallel multiprocessor architectures;

• hardware systems which tend to possess non-trivial amounts of software, and may be said to be information-rich.

2.2.3 Conditions to manage complexity

Warfield (1994) states that the necessary conditions for managing complexity are:

• controlling complexity escalation factors listed in Section 2.1;

• avoiding the situation represented in Figure 2-1(b) in which the individual is required to provide opinions and decisions for which he is not cognitively prepared;

• eliminating other factors (rather than the escalation factors) which detract from problem solving or design activity and provision of those factors that enhance problem-solving or design activity;

• making available suitable conditions or objects or entities or other means or mechanisms for enhancing the capacity of the individual to be effective in a problem-solving situation.

Warfield also mentions some generic elements that may be part of an approach to complexity management: designed processes, designed environments, role specialisation, leadership, quality control.

2.2.4 Complexity metrics

Zuse (1991) provides a comprehensive description of software complexity metrics from which lines of code and McCabe are the most famous. Lines of code measures the size of software through: number of lines of code, number of statements, number of tokens in the program, number of node of an equivalent graph. McCabe used the definition of the cyclomatic number for strongly connected flowgraphs as a software complexity measure. In the graph equivalent to the software: E is the number of edges, N is the
number of nodes, \( p \) is the number of subprograms in the software. Software complexity is calculated by: \( E = N + 2p \). Lines of code and McCabe provide indications of the effort necessary to maintain and test the software.

Besides computational complexity measured as above, another common complexity metric is the informational entropy measure which comes from the same basis as the algorithmic information content mentioned in Section 2.2.1, i.e., the work of Shannon (1948). The theory is, roughly speaking, the greater the complexity the greater the information content. Shannon defines information content (I) for the occurrence of an event as the logarithm of the inverse of the probability (p) of the occurrence of that event: \( I = \log_2(1/p) \). According to this logarithmic definition, the information content is zero when the probability is equal to unity. The base of the logarithm is taken to be 2 so that the information content has the unit of \textit{bits}. Where there is a number of discrete events occurring in an ensemble with probabilities \( p_1, p_2, \ldots, p_i \), the definition adopted for the information content can be extended to compute the average information content of the discrete events as [Brillouin, 1962]: \( I = -\sum p_i \log_2 p_i \). The right hand side of the equation is proportional to the Gibbs entropy. Min and Chang (1991) used this metric to evaluate operational difficulty of complex systems, such as nuclear power plants. These are difficult to operate, maintain, and repair, and hard to understand. This comes from the system uncertainty, which means the difficulty in telling the exact status of the system. To remove such system uncertainty, information for the system is required. The amount of this required information can be used as a measure of the system complexity. Jung and Cho (1996) expand the work of Min and Chang using operational entropy (including desired states) and inoperative entropy (including failed states) to calculate structural complexity of power plants. Suh (1990) used the information content metric to evaluate the quality of a design: “the best design is a functionally uncoupled design that has the minimum information content”.

Dhama (1995) provides metrics for cohesion and coupling in software. It is broadly accepted that cohesion and coupling have great impact in software complexity. Cohesion in a software module refers to that software property that binds together the various statements and other smaller modules comprising the module. Cohesion is an intra-module property that reflects the design considerations for integrating the various components of the module into one unit. Complexity decreases with increasing cohesion. Coupling is a measure of the interdependence between two software modules. It is an inter-module property. Because it is desirable that the changes made in a module affect another module as little as possible, the complexity of a module decreases as module coupling decreases.

Cohesion and coupling are also associated with system complexity. Prasad (1996a) and Yourdon (1989) recognise that when trying to reduce system complexity one should seek to maximise cohesion and minimise coupling when partitioning a system. Group elements are cohesive when they cluster together, sharing common processes, interfaces, communication links, physical features, etc. Functionally-bound elements are candidates for physical grouping in order to reduce the overall system interface complexity. Coupling is the degree of interdependence between elements in different clusters [Hitchins, 1992].

Hitchins (1992) proposes a method of evaluating the quality of clustering. Elements to be grouped or integrated are recorded on the main diagonal of a matrix and interface-related parameters occupy the other cells in the matrix, as illustrated in Figure 2-2. Interface-related parameters (X and Y) represent some characteristic of the interface, such as strength of association, priority, etc. By convention, output
related parameters are on the row containing the element from which the output comes, while input related parameters are on the column containing the element to which the input goes. To score a matrix, one simply determines the distance of each interface from its associated elements in terms of the number of squares, and multiplies the number in the interface by some function of distance. Typical functions, \( f \), of distance would be direct proportionality, square or cube. If two interfacing elements are separated on the matrix by a number of intervening elements, then the interface joining them will score high due to the distance functions, \( dx \) and \( dy \) above. If they were adjacent, their interface score would be low. It is therefore possible to arrange the elements on the matrix so that the matrix has an overall lowest score. In this configuration, all interfaces connecting elements will be grouped such that their scores are low. Competition between elements, each trying to occupy the same position relative to a third element will be resolved by the strength of their respective interface. A better clustering of two is the one with lower score.

Warfield & Staley (1996) propose a measure of complexity called the Situation Complex Index defined as the product of the Miller Index, the Spreadthink Index and the DeMorgan Index. The Miller Index \( (N/7) \) is a measure of size of the set of elements that must be dealt with simultaneously in the design situation. The index divisor is the "magical number" of Miller. If the design situation involved only seven elements, which corresponds to the number of things that can be dealt with in short-memory, the Miller Index would be equal to 1. If the Miller Index is 2, this indicates that there are twice as many elements as could normally be expected to be considered by the unaided human mind at one time. The Spreadthink Index \( (V/5) \) is a measure of the diversity of opinion in the group with respect to the relative saliency of the elements involved in the design situation. Using a voting procedure, the aggregate group opinion on relative importance of elements is measured by \( V \). If \( V = 5 \) this indicates that the group is in perfect agreement on the most important elements of the situation. The DeMorgan Index \( (K/10) \) is a measure of the structural complexity of the relationship being described among the elements of the design situation. For the case where the Miller Index and the Spreadthink Index are both 1, the simplest possible structure that relates all 5 elements would be a linear structure. For such a structure, and a transitive relationship, the number of dyadic relationships \( (K) \) would be 10, and the DeMorgan Index would be 1. The Situation Complexity Index is the product of the above three indices or: \( SCI = (N/7)(V/5)(K/10) = (1/350)NVK \). Staley (1996) reports on the application of the above SCI in a number of different design situations at the Ford Research Laboratory, Dearborn, USA.

Rossi and Brinkkemper (1996) propose complexity metrics for systems development methods and techniques based on the number of classes of objects, properties, relationships and role they contain.

Thomas & Mog (1997) developed COMPRÉ (Complex Organisational Metric for Programmatic Risk Environments). COMPRÉ utilises a covariance matrix to model the interactions over time between component developments within the scope of a complex system development. Consider a system which may be partitioned into components. For each component, define the attributes of technology readiness, development schedule, relative investment (percent of project cost), and organisational data. A
technology covariance matrix then is developed on the basis of the presence, absence, and degree of architectural and organisational interactions. The metric has been used to evaluate the complexity evolution of products developed by NASA.

Goldwasser, Latombe and Motwani (1996) developed complexity measures for assembly sequences. These are based on the following parameters: least number of directions, fewest re-orientations, least number of non-linear steps, minimum depth of an assembly sequence, least number of removed parts.

2.2.5 Interactive management

Interactive Management (IM) is a system of management invented explicitly to apply to the management of complexity. It is intended to be applied intermittently in organisations to enable those organisations to cope with issues or situations whose scope is beyond that of the normal type of problem that organisations can readily solve. [Warfield & Cardenas, 1994]

The development of IM is based on the recognition that for coping with complex situations there is a need for a group of people, knowledgeable of the situation, to tackle together the main aspects of concern, to develop a deep understanding of the situation under analysis and to elaborate the basis for effective action; all these founded in a spirit of collaboration, commitment and within the framework of a serious and organised effort.

IM is based on the “Science of Generic Design” [Warfield, 1994]. The concept of IM was developed at the University of Virginia in 1980. Since that time the concept has been enlarged somewhat, the practice of IM has spread to many places, and many applications of IM have been carried out. Before IM was conceived as a system, numerous other applications of predecessor component parts were carried out, starting in 1974 and continuing until 1980.

The two principal predecessors to IM were nominally referred to as Unified Program Planning (UPP) and Interpretive Structural Modelling (ISM) [Warfield, 1976]. The former was developed at Batelle Memorial Institute in 1971, and the latter was also developed there in 1972-73. Most of the applications in the period 1974-1980 were of ISM.

IM involves three closely-linked phases: the Planning Phase, the Workshop Phase, and the Followup Phase. The Planning Phase identifies the people, information, and facility requirements for the other two phases. The Workshop Phase involves bringing together a selected group of participants who have knowledge about the issue or situation. This group works together in a specially-designed situation room, under the guidance of a skilled IM facilitator, and with the help of a trained staff. The Followup Phase may involve iteration, implementation, or both.

The outcomes of IM are:

- Definition for the situation that is the focus for the work in the form of sets of component problems, sets of component problem types, problematique (a pattern that shows how the problems are related to each other), a problem field (a pattern that shows how problems subdivide into problem types), a set of objectives, a set of organisations, intent structure (to show how the objectives are related), organisations mapped onto either problematiques or problem fields, problem types mapped onto problematiques, intensity of relationships.
- Alternative Designs aimed at correcting the undesired conditions in the defined situation, or otherwise to determine the new possibilities that might be open in that particular situation. The tools used in the process of generating alternatives are the Nominal Group Technique (NGT) [Warfield & Cardenas, 1994] and ISM.

- Choice of a design as the basis for implementing the desired corrective actions. The methodology used to make an initial design choice is the Trade-off Analysis Methodology (TAM). This methodology leads to bar charts which compare all pairs of available design alternatives. The comparisons show total scores for each alternative, derived from the scores on each of the criteria used to make the comparisons.

The main tangible products of IM are annotated graphics, also called: patterns, structural graphics, relationship maps, maps, or interpretive structural models. The products listed below are application structural types and are described in [Warfield & Cardenas, 1994]: DELTA Chart, Problematique, Enhancement Structure, Intent Structure, Curriculum Structure, Priority Structure, Field Representation (Quad), Triply Structured Quad, Tapestry of Quads, Profile Representation, Resolution Structure, Comparison Bar Charts, Unified Program Planning Linked Matrices.

The processes used for the obtention of those products are listed below and described in [Warfield & Cardenas, 1994]. They consist of process for: generating ideas, clarifying (interpreting) ideas, amending ideas, structuring ideas, interpreting structures of ideas, amending structures of ideas. They are: Ideawriting, Nominal Group Technique (NGT), DELPHI, Interpretive Structural Modelling (ISM), Field Development (Options, Problems and Attributes Fields), Profile Development (Options Profile, Attributes Profile), Trade-off Analysis.

More than a hundred applications of IM have been reported. In the product development arena, IM has been used, for example, at Ford Motor Company Research Laboratory, Dearborn, USA for the following applications: Product Information Management at Ford, Rapid Response Manufacturing Joint Application Development, Systemwide Planning for Analytical Powertrain.

Interpretive Structural Modelling

The use of ISM for managing complexity is based on the assumption by Warfield (1976) that structure is a necessary ingredient of an approach for managing complexity.

A process of interpretive structural modelling is shown in very general form in Figure 2-3. Beginning with a mental model (individually or collectively held), a process of embedding is carried out, whereby ingredients of the mental model are supplied to a computer. The computer role in the embedding process is to construct internally a matrix model of the structure of the information supplied by the developer.

![Figure 2-3: Block diagram of interpretive structural modelling process (Source:[Warfield, 1976])](image-url)
The matrix model is then partitioned by the computer, and the computer extracts from this matrix model a second model, called a multilevel digraph. The latter is a basic structural model. Then, by a process of substitution, involving the replacement of computer symbolism with ordinary language, an interpretive structural model is created. This model is then documented and inspected by its human developers. In the process, the structure is tested against the prevailing mental model, which will normally be different from what was held at the beginning due to the learning that takes place in the embedding process.

Sage (1977) used ISM as part of a methodology for large-scale systems development. Loureiro (1994) used ISM to support a computational implementation of QFD (Quality Function Deployment).

2.2.6 The 'total' approaches

This section aims to review some so called 'total' approaches that have an impact on the management of complexity. Some other approaches, although called total, have other purposes such as:

- **total life cycle management**: integrates environmental, occupational health and safety, with recycling as part of the overall management decision making or government policy-making process. Life cycle costing may be used as a means of translating health and environmental impacts. ([Fox & Singh, 1997], [Nasr & Varel, 1997b])

- **total productive maintenance**: is the productive maintenance from a company-wide point of view. Productive maintenance predicts, discovers, and eliminates failures through periodical inspection, repair, and replacement. This is done in order to reduce the rate of wear-out and eventual failure and to prolong the life of an equipment, i.e., the capacity for extended productive use of the equipment under the necessary technological functioning and servicing. For automated manufacturing system, in which the role of human operative work is reduced and failure or breakdown of any portion of the system causes a major loss in productive efficiency, the function of productive maintenance takes on particular importance in the production control system. [Hitomi, 1996]

2.2.6.1 Total design

Pugh (1991) developed a product development process, called total design, to help to understand the complexity and the efficient application of the subjects contained within engineering a product. Pugh did not focus specifically on complex products but as he proposed a generic approach to engineering products, it is reasonable to suppose it includes complex products. Pugh addressed the complexity of the product development process itself as he recognises that almost any product – whether it be tank or turbine, building or burner, bridge or bulldozer – would have required inputs from people of many disciplines, both engineering and non-engineering, in a mix that was almost unique to the product under consideration. To conceive such products would therefore have required co-ordinated inputs from different specialisms. Thus, a typical product may be made up of the many technological and non-technological components that impinge on the product design, and hence the product. Some examples are man-machine interfaces (ergonomics), shape, form, texture and colour (aesthetics). Unless these are in balance, the product may fail in the market place. In industrial terms, the necessary integration comes about only as a result of all of the partial design inputs. Industry is therefore concerned with total design.
Pugh defines **total design** as the systematic activity necessary, from the identification of the market/user need, to the selling of the successful product to satisfy that need — an activity that encompasses product, process, people and organisation. [Pugh, 1991] focuses on the product component of total design. The achievement of the integration of technological or non-technological subject material in an effective and efficient manner is greatly enhanced by having a visible operational structure. Visibility is a crucial factor in bringing about integration. Visibility helps everyone find out what people are doing and why.

Figure 2-4 illustrates the **total design activity model**. Total design may be construed as having a central core of activities, all of which are imperative for any design, irrespective of domain. Briefly, this core, the design core, consists of market (user need), product design specification (PDS), conceptual design, detail design, manufacture and sales. Figure 2-4 also shows that iterations may occur between two subsequent stages of the design core. Iterations occur because of changed circumstances. Although some iteration is inevitable, operating within the design core rigorously and systematically will minimise unnecessary iteration.

Distinguishing aspects of total design in relation to what Pugh calls partial design are: the development of a product to address market/user need, the capture of requirements other than product performance requirements in the PDS (see Figure 2-5), the use of the PDS to envelope and control the subsequent phases of the design core, the use of systematic methods for conceptual design and analysis of alternative solutions to increase visibility, the development of a comprehensive component design specification in the same moulds design of the technology, and the use of life cycle requirements such as manufacturing to the design phases, feedback to market/user need from the selling part of the design core.

### 2.2.6.2 Total quality control

Total quality control is an effective system for integrating the quality-development, quality maintenance, and quality improvement efforts of the various groups in an organisation so as to enable marketing, engineering, production, and service at the most economical levels which allow for full customer satisfaction. [Feigenbaum, 1991]
The underlying principle of the total quality view, and its basic difference from all other concepts, is that to provide genuine effectiveness, control must start with identification of customer quality requirements and end only when the product has been placed in the hands of a customer who remains satisfied. Total quality control guides the co-ordinated actions of people, machines, and information to achieve this goal.

Total quality control's organisationwide impact involves the managerial and technical implementation of customer-oriented quality activities as a prime responsibility of general management and of the main-line operations of marketing, engineering, production, industrial relations, finance, and service as well as of the quality-control function itself.

The institution of quality control significantly broadens and deepens the work and the very concept of quality control in a modern company. It permits what might be called total quality management to cover the full scope of the product and service "life cycle" from product conception through production and customer service.

Quality control evolved with the increase of complexity of product, processes and organisations. Major changes in the approach to quality control work have occurred approximately every 20 years since the end of the nineteenth century. In the end of the nineteenth century, there was the operator quality control, one or a small group of workers responsible for the entire product manufacturing quality. In the early 1900s, with the factory task grouping concept, there was the foreman quality control. As the manufacturing system became more complex during World War I, involving large number of workers, the inspection quality control took place. Inspection organisation grew in size in the 1920s and 1930s. The advent of tremendous mass-production requirements of World War II necessitated statistical quality control. Between the 1960s and 1980s, total quality control developed. Only when firms began to develop a specific decision-making and operating framework for product quality which was effective
enough to take suitable action on the quality-control findings did the firms obtain genuine results in better quality and lower costs. This total quality framework made it possible to review designs regularly rather than occasionally, to analyse in-process results and take control action at the manufacturing or the supplier source, and, finally, to stop production when necessary. Moreover it provided the structure into which the early statistical quality-control tools could later be joined by many additional techniques of metrology, reliability, quality information equipment, quality motivation, and the numerous other techniques now associated with the field of modern quality control and with the overall quality function of a business. As total quality control has come to have a major impact upon management and engineering practices, it has provided the foundation for the evolution in the decade of the 1980s and beyond of total quality control organisationwide, total quality management, and quality as a major new business driver. The latter requires two basic general management steps:

- the total-customer-satisfaction-oriented concept of quality, together with reasonable costs of quality, must be established as one of the primary business and product planning and implementation goals and performance measurements of marketing, engineering, production, industrial relations, and service functions of the company;
- assuring this customer-satisfaction quality and cost result must be established as a primary business goal of the quality program of the company and of the quality-control function itself — not some narrower technical goal restricted to a limited technical or production-oriented quality result.

2.2.6.3 Total quality development

Clausing (1994) developed the total quality development approach. Total quality development goes beyond partial design. Partial design is the discipline specific design focusing on good functionality under limited operating conditions. However partial design cannot cope with the complexity of the product development activity. The emphasis of total quality development is on increasing the scope of the activity to better define the many partial design tasks. Then the team members, with their ever-present skill at partial design, can complete all of the necessary work.

Clausing (1994) considered that a product is developed through the phases of concept, design, and production preparation. This development of a specific new product is supported by new technologies from the technology stream. The specific new product is in the context of the total corporate strategy, which determines when the development of a product will start and when, after the product has been produced, its life cycle will be terminated through withdrawal. Total quality development has three major elements:

- basic improvements in clarity and unity: basic concurrent engineering (see Section 2.4)
- enhanced quality function deployment: QFD (see Section 2.4.4.1) with conceptual design techniques developed by Pugh (1991) (see Section 2.2.6.1).
- quality engineering using robust design: Taguchi’s techniques (see Section 2.4.4.2).

2.2.6.4 Total systems approach

[Hatley & Rushton, 1997] propose a total systems approach to automotive development. The approach consists of expanding the Hatley-Pirbhai methodology (HPM) [Hatley & Pirbhai, 1988] for real-time system specification to include life cycle process considerations. Like in HPM, the method starts
considering only operational requirements (e.g. provide vehicle motion, provide information, communication and entertainment) and the interactions with the environment during product operation. An operational functional model of the product is developed. The architecture development, however, starts considering not only the interacting elements during product operation, but also during other life cycle processes: testing, calibration, service. For example, the architecture model captures the flows and connection of the product with service equipment. The method then iterates in order to enhance the initial list of requirements to include also the life cycle process requirements.

2.2.7 Business process reengineering (BPR)

Hammer and Champy (1993) claim that "reengineering is the fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical, contemporary measures of performance, such as cost, quality, service and speed." 'Dramatic' and 'major' are words that describe reductions or improvements that are ten-fold, not merely 10%. 'Fundamental' means that instead of asking how the company might carry out its work in a better way, one asks fundamental questions about the company. Hammer and Champy describe business process reengineering as 'starting all over, starting from scratch'. It is a conscious decision that underlies the idea of making dramatic improvements and not merely insignificant, marginal changes in the existing organisation.

Reengineering mean taking steps to redesign and simplify business systems and processes, search the best industry practices to develop a more competitive work force, and to explore new process methods. The key to any (manufacturing company) competitive posture lies in its ability to reengineer a business for agility – both physically and logically. Factories, systems, and organisations must differentiate and remove non-value added functions from the chain of design-tasks, manufacturing-tasks and work-tasks, and foster open lines of communications. Reengineering helps to define strategies for bringing manufacturers, suppliers, and customers closer together. [Prasad, 1996]

The risks involved with performing a reengineering project are significant. There are estimates that 50 to 70 percent of companies that try it fail. BPR risks fall roughly into two categories: risks associated with the change process, and risks associated with the technology used. It has been estimated that 80% of the failures are caused by 'soft' factors, such as motivation, management commitment, leadership, the need for expert guidance, and so on. [Jacobson et al., 1995]

Jacobson is convinced that BPR success rate can be dramatically improved if a formal reengineering process is used. He proposes the following as ingredients of a formal reengineering process:

- a description that specifies every activity and deliverable involved. This process description must be adaptable to the reengineering project.
- deliverables, in the form of business models, that focus on the company’s architecture and dynamics. The business models should be presented in an engaging language so that everyone involved – executives, process owners, process managers, process operators, resource owners and customers – can understand them, not just the reengineering team.
- a process for the development of an information system that is truly integral to the reengineered company. A truly information system is one that is developed in parallel with new business process and both influences the design of those processes and is influenced by them.
2.2.8 Holistic enterprise evolution

[Yeh, 1997] affirms that the failures in BPR projects are due to the fact that many organisations attack the problem of change by focusing on a single viewpoint, not realising the multi-dimensional implications of change. Yeh proposes that a systemic approach is needed to produce the most significant business performance improvements. He, then introduces a holistic framework for managing change – a three dimensional model called Cosmos™. The activity structure dimension is concerned with how work is structured in an organisation. It deals with the steps and tasks that must be taken in the flow of work. The infrastructure dimension is concerned with how resources are allocated. It deals with the assets of an enterprise. The co-ordination dimension is concerned with how information is created, shared, and distributed. This dimension deals with the “soft” side of the enterprise – the culture, informal communication among people, and roles and relationships among people.

An important aspect of this model is that the three dimensions are interdependent. Each dimension acts both as an enabler and an inhibitor to the other dimensions. Thus, the key to the model as a guide for managing change is the “co-evolution” of all three dimensions. In doing so, the model provides a “legal space” for navigating from where you are to where you want to be. While there are multiple paths to go from one point to another, Cosmos’ systemic guidelines help managers avoid unnecessary pitfalls.

Cosmos framework deals explicitly with the three essential trade-offs that must be managed in any organisation. When dealing with the work structure, one must trade-off between stability (work must be predictable) and flexibility. In the co-ordination structure one must concern with the trade-off between aligning all the people in an organisation (interconnectedness) and providing each operating unit with sufficient autonomy (modularity) so that they can make rapid decisions in accordance with the changing market situation. Finally, in infrastructure, one must deal with the short term return without losing sight of an organisation’s long term vision. The framework offers an opportunity for managers to see a three-dimensional picture rather than a linear cause-and-effect picture. The Cosmos dimensions mutually enable and constrain each other. That is, an organisation undergoing change can move or change only so far along one dimension before changes in the other two dimensions must be considered.

Figure 2-6 illustrates the characteristics of a redesigned organisation according to the three Cosmos dimensions.

<table>
<thead>
<tr>
<th>Activity attributes</th>
<th>Co-ordination attributes</th>
<th>Infrastructure attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td>Redesigned</td>
<td>Current</td>
</tr>
<tr>
<td>Sequential</td>
<td>Concurrent</td>
<td>Hierarchical organisation</td>
</tr>
<tr>
<td>Ad hoc</td>
<td></td>
<td>Functional organisation</td>
</tr>
<tr>
<td>Dominated by</td>
<td></td>
<td>Manager as supervisor</td>
</tr>
<tr>
<td>non-value-added</td>
<td></td>
<td>Misalignment</td>
</tr>
<tr>
<td>components</td>
<td></td>
<td>Lack of vision</td>
</tr>
<tr>
<td>High percentage of</td>
<td></td>
<td>Control</td>
</tr>
<tr>
<td>rework</td>
<td></td>
<td>Internally focused</td>
</tr>
<tr>
<td>Many handoffs</td>
<td></td>
<td>Exclusive</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2-6: Redesigned organisation according to the three Cosmos dimensions (Source: [Yeh, 1997])
2.2.9 Holonic manufacturing systems

Manufacturing practices of the future will have to address the needs of the customers demanding low cost products. These needs are likely to change quickly. Thus the manufacturing industries of the future will have to be organised differently and will have to be more flexible in responding customer needs. It will use technology differently and in some cases specific technologies will have to be developed specifically for that purpose. Thus rigid, static and hierarchical manufacturing systems will give way to systems that are more adaptable to rapid change. This should allow the producer to respond to specified customer demands with the result that production lots will be small but the cost should also correspond to new specifications. These units will have to be designed so that it may have inherent intelligence and so that they easily communicate with other units. Thus it is envisaged that these subsystems will have to be autonomous, distributed and co-operative. Each of these units may be referred to as a holon [Valckenaers & van Brussel, 1996]. Some fundamental definitions in holonic manufacturing systems are:

- **holon**: an autonomous and co-operative building block of a manufacturing system for transforming, transporting, storing and/or validating information and physical objects. The holon consists of an information processing part and often a physical processing part. A holon can be part of another holon.
- **autonomy**: the capability of an entity to create and control the execution of its own plans and/or strategies.
- **co-operation**: a process whereby a set of entities develops mutually acceptable plans and executes these plans.
- **holarchy**: a system of holons that can co-operate to achieve a goal or objective. The holarchy defines the basic rules for co-operation of the holons and thereby limits their autonomy.
- **holonic manufacturing system (HMS)**: a holarchy that integrates the entire range of manufacturing activities from order booking through design, production, and marketing to realise the agile manufacturing enterprise.
- **holonic attributes**: attributes of an entity that make it a holon. The minimum set is autonomy and co-operativeness.

The link between holon and complexity management is Simon’s conclusion that complex systems will evolve from simple systems much more rapidly if there are stable intermediate forms than if there are not; the resulting complex system in the former case will be hierarchic [Simon, 1981]. Koestler (1967) demonstrates that those stable intermediate forms are holons.

2.2.10 The fractal factory

Warnecke (1993) proposes that the increasing complexity observed within the management, control and operation of manufacturing has led to the need for an alternative view of how the factory of the future may be organised. The use of CIM is one approach, but is found lacking in dealing with the numerous elements bearing multiple, often non-linear relationships with one another within the manufacturing system. Other new approaches such as complexity reduction, concentration on core areas, GT and cells, lean and agile manufacturing are highlighted. Warneck thus introduces the term fractal factory to
establish a common denominator to represent the complex behaviour of modern systems and to determine the commonality of the numerous approaches so that they may be integrated into a holistic approach.

Fractal mathematics is a way of measuring these highly complex natural structures of dynamic change and self organisation [Mandelbrot, 1982]. Two prime characteristics of fractal objects are: self-organisation, where fractals within a factory have the freedom to use the methods appropriate to the particular task, with different fractals free to use different methods. Self-similarity where the structure (fractal factory) is made up from many similar smaller structures (mini fractal factories). Taking the properties of fractal geometry and applying them to manufacturing technology a definition of fractal is that of an independently acting entity whose goals and performance can be precisely described, and exhibit the same characteristics of self-similarity and self-organisation. Thus the fractal factory is an attempt to provide a model for corporate re-engineering that can provide a deterministic modelling of the complex global markets and corporate structures of today’s industry, based upon reflection of nature and the theories of chaos research.

2.2.11 Information modelling

Information modelling assumes that the real world is made of entities (e.g. persons, things, facts, events, ...) linked by relationships (e.g. this car belongs to that person), and described by attributes, where attributes are defined as data and are subject to constraints [Vernadat, 1996], [Martin, 1990]. Therefore the basic elements of information modelling are the same as those of interactive management, although the latter builds one structure of entities per each different type of relationship. Information modelling is applied to product modelling [Krause et al., 1993], process modelling [Prasad, 1996], enterprise modelling [Vernadat, 1996].

An example of the application of information modelling to obtain a comprehensive model of product development systems is provided by [Negele et al., 1997]. Negele et al. developed ZOPH, which stands for “Zielsystem” (goal system), “Objektsystem” (product system), “Prozesssystem” (process system), and “Handlungs(träger)system” (agent system). The model structures all the information relevant to the development system looked at by using these four different system types. Influences across the system boundary also can be taken into consideration by modelling elements of the environment (e.g. suppliers, users, market, legal regulations, environmental conditions) and linking them with the elements of the ZOPH subsystems. ZOPH system types are described below:

“Z” – The goal system describes the outcome aimed at and the conditions for its realisation. This can include user/customer needs and expectations, requirements, specifications, plans and schedules, (individual) objectives of all the different parties involved, legal regulations, standards, constraints, ecological and social restrictions, etc. For example, the goal system can be structured in goals concerning product, process, organisation, and project.

“O” – The product system contains the product to be developed with its components, functions, properties, its (technical) structure, and all the (supporting) elements that are prerequisite for, object or result of the activities of the development system.
"P" - In the process system processes, activities and events are described that have to be carried out in order to attain the goals specified in the product system. Also, the connections (e.g., temporal, logical, causal) and flows (e.g., information, material) between processes, activities and events are modelled.

"H" - The agent system describes the company's organisation with its structures and resources carrying out all the tasks as agents. This includes all the individuals and organisational units (e.g. departments, teams, employees) as well as important resources like installations, machinery, equipment or tools, methods, software, specific know-how, etc.

A formal language has been created specifically for ZOPH. Basis of the formal language is the system definition (see Figure 2-7) structured into four elements:
1) A system consists of elements
2) The elements have attributes (properties and functions)
3) Elements are interacting by means of relations
4) An element can be a system

ZOPH can, in principle, be applied to any system due to its abstract and generic nature. To ease its application to an actual system, predefined, adjustable element or relation types can be added, newly created elements or relations can be derived from. A relational database system has been chosen for computer implementation that also enables simulation of modelled systems. This systemic method has been utilised successfully in several different projects already, e.g., determination and simulation of connections between the functional structure of an automobile and its characteristic features [Walther, 1994]; dynamic model for the simulation of system loads in air traffic scenarios [Kreichgauer, 1995].

ZOPH's capabilities include:
- the model is able to integrate all the different information types;
- the different information types can be connected and interrelated. Links and interfaces can be specified and traced;
- multiple, related views of the modelled development system are possible;
- all kinds of saved information can be changed and updated during the project;
- customisation and tailoring to specific user needs are supported;
- "what if" scenarios are supported;
- the model works in a team environment.
2.2.12 Evolvability, enduring architectures and product families

The need to suit individual customer requirements ever shortening product development lead time and reducing costs has led developers to consider evolvability from the beginning of the product development process. Product evolvability is a trait of a product that allows the product to be easily modified due to changes in the environment. Presently, evolvability of a product is assessed using design criteria or by postulating changes and evaluating the product ability to change. Both of these approaches are weak measures of a product evolvability. [Percivall, 1994]

Product architecture is an important concept related to evolvability. The product architecture is the composition of the product from a number of component products. The term 'product architecture' may include not only the physical architecture of a product (physical components and their interfaces) but also its functional architecture (component functions and functional interfaces) [Erens & Verhulst, 1997]. Although it is generally agreed that the product architecture represents a framework where detailed design is performed, it is not generally agreed what aspects of behaviour and structure should be captured in such a framework, how it should be represented, and how it relates to the specifics of product design [Steiner, 1998]. For example, the INCOSE System Architecture Working Group (SAGW 94) defines the term “system architecture” as “the aggregation of decomposed system functions into interacting system elements whose requirements include those associated with the aggregated system functions and their interfaces requirements/definition”. This is clarified when it is also stated that “when used as a noun the system design is the same as the system architecture”. The Hatley-Pirbhai school of systems design uses the term “architecture” to represent structure or physical design of the system, i.e. the complete collection of and relationship between system components. [Hatley & Pirbhai, 1988]

In order to evolve a product, increasingly companies use the product architecture as a starting point [Erens & Verhulst, 1997]. There are several reasons for this:

- **stability.** Interfaces between components are set so as to reduce the effect of changes in one component for related components. For innovative products, a more generic product architecture can be used. For mature products, a detailed product architecture can be used.
- **communication.** A product architecture provides a framework for associating development documentation.
- **learning organisation.** Companies are often organised around a product architecture. A breakthrough product therefore not only requires a new product architecture, but also a change of the development organisation.
- **commonality and variety.** A pre-defined product architecture creates the possibility of replacing components with components that have identical interfaces in order to cater for the diverse customer requirements. Some components, however, will not have variants and are therefore common for the product family.
- **reuse and upgrading.** A product architecture can be used to create a new version of the product by replacing some of the components with an upgraded version. A number of existing components are reused to reduce time, effort and risk of developing a completely new product. However, incremental development requires a good insight into the life cycle of the different components and the product architecture in which these components must fit.
• **competitive control.** As companies control the interfaces and often the critical components, they are better positioned to develop products that maximise the possibilities of the architecture. Furthermore, they can discipline competing vendors that offer components that are compatible with the architecture. Architectural controllers try to increase the competition among these competing vendors, which results in a price reduction of the overall product and an increased market share for the architectural controller.

Product architectures are essential to separate the stable and variable parts of design. The stable aspects create a framework within which a variety of products can be developed. Following the same rationale, Steiner (1998) proposes the concept of **enduring architecture.** The “enduring architecture” represents a set of constraints placed on the design for the purpose of:

1) ensuring consistency of key characteristics as a product design matures or evolves, and 2) exemplifying a strategy for continuing customer satisfaction and communication.

An enduring architecture, in this case, can contain both structure and behaviour, but at an abstract level. It does not contain the entire system structure or behaviour, but only aspects of the structure or behaviour necessary to accommodate 1) and 2) above. The concept of “enduring architecture” encompasses two other concepts: “growth architecture” and “evolutionary architecture”. A “growth architecture” addresses management of the changes in a design while it matures within a single generation of product design but considering life cycle process aspects. An “evolutionary architecture” addresses management of changes in design while it evolves (from one generation to another). The evolutionary architecture needs to specify the set of enduring characteristics to be designed into a family of products.

Erens & Verhulst (1997) define a product family as a set of products with identical internal interfaces, i.e., interfaces between the product’s components, for all variants in each domain. Interfaces must be standardised in each of the functional, technology and physical domains to allow the full exchange of components. According to this definition, either growth or evolutionary architecture can be used as a product family architecture as the definition does not specify whether the products are contemporary or not. Ramamorthy (1997) defines a product family concept as the idea the new versions of a product can be derived from original model and the members of the family will perform generally the same basic functions in an improved and enhanced fashion. The product family concepts allow designers to deal with continuity of the product and accommodate changes in technology and application. The architecture of a product family defined as such is the evolutionary architecture.

### 2.2.13 **Modularity and integration**

A modular design is considered to be a design in which a restricted number of functions is allocated to a module, or a restricted number of modules is allocated to a physical assembly [Erens & Verhulst, 1997]. Marshall (1998) identifies the following properties of modules:

- modules are co-operative subsystems that form a product, manufacturing system, business etc.
- modules have their main functional interactions within rather than between modules.
- modules have one or more well defined functions that can be tested in isolation from the system and are a composite of the components that form the module.
• modules are independent and self-contained and may be combined with similar units to achieve a different overall outcome.

In terms of functional allocation to technology, the opposite of a modular design is an integrated design, where several functions are allocated to one technology. Function sharing increases the level of integration.

Modularity corresponds to flexibility and changeability. It is an effective mechanism to upgrade and reuse existing functions, modules and assemblies. Reuse multiplies the effectiveness of human problem solving by ensuring that the extensive work or special knowledge used to solve specific development problems will be transferred to as many as similar problems as possible. This reduces initial costs, especially in product development. [Erens & Verhulst, 1997]

In contrast, integration corresponds to stability and optimisation. In an optimised design, a large variety of functions is realised with a limited number of components. The initial costs of such a multi-functional design are usually higher than the costs of a modular design, but the operational costs (in manufacturing and service) are relatively low because of this limited number of optimised components. However integration requires a stable environment and can therefore best be applied in mature products. [Erens & Verhulst, 1997]

Important considerations with respect to modularity and integration should be made in the early phases of the design process, where the product architectures in the different domains (e.g. functional, technology, physical domains) are determined. However, the increase of modularity and integration reduces the design margin, i.e., the possibility to postpone the allocation of less critical requirements. Ulrich & Tung (1991) state that most architectures evolve from being modular to being integrated. In later stages of the product life-cycle the interactions between different aspects are better understood.

Product families need architectures in which both modularity and integration are considered. Modularity is necessary to offer a large product variety, and integration is needed to improve the cost-performance ratio.[Erens & Verhulst, 1997]

2.2.14 Mechatronic, Complex Electro-Mechanical and VLSI development approaches

Examples of modularity and integration applications can be found in the development of mechatronic, complex electro-mechanical (CEM) and very large scale integration (VLSI) products.

Mechatronics is used for the integration of microprocessor control systems, electrical systems and mechanical systems. A mechatronic system is not just a marriage of electrical and mechanical systems and is more than just a control system; it is a complete integration of all of them [Bolton, 1999]. Mechatronics provides a practical example of the difficulties faced by the engineering of complex multidisciplinary products. The combination of just three disciplines imposes problems due to [Buur, 1989]:

• the substance of design problems is different for all three fields;
• there is no common language between mechanical, electronics and software engineers;
there is no cross-functional support either in education or in methods and tools to the development process.

[Loureiro, 1994] summarises the elements of the concept of Mechatronics as following:

- Mechatronics is a new approach to engineering. New in the sense it is not possible to establish boundaries among the engineering areas that compose the product.
- Mechatronics uses technologies from mechanical, electronics and software engineering.
- Mechatronics synergistically integrates those technologies in the product from its concept stage.
- Mechatronics is multidisciplinary.
- Mechatronics uses project teams covering the stages of a product life cycle: marketing, product development, product engineering, process engineering, purchasing, manufacturing, sales and technical support in order to develop the product and the manufacturing process.
- Mechatronics provide energy savings, less weight and lower cost in obtaining a product that contains integrated mechanical functionality and algorithmic control.
- Mechatronics takes into consideration not only product functional performance, but also its manufacturability, testability, ease of integration, serviceability, customer satisfaction and cost.
- Mechatronics makes possible overall product optimisation.

In terms of examples of mechatronic products, Bradley et al. (1991) distinguish between existing products offering enhanced capability and completely new product areas which would not have existed without a mechatronic design approach having been adopted from the outset. Examples in the first category are: automotive engine and transmissions, cameras, power tools. Examples in the second category include: modular robotics, autopilots for small boats, video and compact disc players.

Whitney et al. (1994) use the term complex electro-mechanical (CEM) products to refer to products like: automatic cameras, miniature disk drives, missile seeker heads, automobile drive trains, consumer products like VCRs, CD players, CD-ROMs, and camcorders, commercial and military aircraft and aircraft systems. Whitney and collaborators do not provide a definition for CEM but by the examples given one can infer it encompasses mechatronic products. Whitney et al. (1994) describe the findings of a research program focusing on problems of design and manufacture of CEM products comparing them with digital VLSI. Figure 2-8 summarises those findings by comparing and contrasting the modular design approach, characteristic of VLSI design and, the integrated approach, characteristic of CEM design.

Other conclusions regarding CEM products are [Whitney et al., 1994]:

- products that push the state of the art are without exception designed as integrated systems.
- there is a greater imbalance between the growth potential of the electronic elements and the mechanical elements in CEM products. Growth means increase sophistication, functionality and reliability together with decrease in cost and time to develop.
- CEM design is becoming driven more and more by system-level decisions.
- CEM products depend on systematic design and combination of high precision components for much of their functionality.
Design can be approached on a top-down basis and consists of separate component development and system design phases.

Direct link between function and element connectivity and, between connectivity and final component geometry. Each element’s function can be described with high accuracy even when connected to arbitrary combinations of other elements. Elements typically carry out one single well-defined logic function.

Designed in building-block fashion, by hooking together pre-defined and pre-verified elements into systems of enormous complexity. The elements' behaviour is unchanged regardless of what they are hooked to.

CAD tools support system-level design.

<table>
<thead>
<tr>
<th>Digital VLSI – Modular design</th>
<th>CEM – Integrated design</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Design can be approached on a top-down basis and consists of separate component development and system design phases.</strong></td>
<td><strong>Design cannot be separated into such distinct phases. Instead component and systems are designed together.</strong></td>
</tr>
<tr>
<td><strong>Direct link between function and element connectivity and, between connectivity and final component geometry. Each element’s function can be described with high accuracy even when connected to arbitrary combinations of other elements. Elements typically carry out one single well-defined logic function.</strong></td>
<td><strong>No direct path or automatable procedure linking function to geometry for either the elements or their connections. Each item’s physical embodiment must be thought out in the human mind by combining knowledge of materials, fundamental principles of physics, and 3D geometric reasoning. Elements typically carry out multiple functions and include logic, geometric compatibility, and energy transactions in multiple media simultaneously.</strong></td>
</tr>
<tr>
<td><strong>Designed in building-block fashion, by hooking together pre-defined and pre-verified elements into systems of enormous complexity. The elements' behaviour is unchanged regardless of what they are hooked to.</strong></td>
<td><strong>Cannot use building-block approach. Mechanical components typically back-load each other (that is, their ratio of input to output impedance is not high) and thus cannot be assembled without their individual behaviour changing drastically. CEM components often carry and transform significant power, which inevitably creates side-effects at comparable power. These side-effects cannot be designed out, but must be predicted and accommodated, an effort that often dominates the design process.</strong></td>
</tr>
<tr>
<td><strong>CAD tools support system-level design.</strong></td>
<td><strong>CAD tools support system-level design of electronic products and not of mechanical products. For the latter, they support component level design.</strong></td>
</tr>
</tbody>
</table>

Figure 2-8: Modular VLSI versus Integrated CEM.

- companies that want to prosper in the CEM product arena therefore must have some degree of control over both the system-level design and the specification and manufacture of the important components.
- certain individual precision components have become so hard to make that only one or two companies in the world are able to make them at present with the desired performance at the required cost.
- competent, low cost manufacture and assembly of CEM products has also become so difficult that, again, only a few companies in the world can do it quickly and at low cost.
- US system-level integrators must depend on suppliers for critical components, manufacturing equipment, and manufacturing capability. A large number of these suppliers are Japanese.

### 2.2.15 Partitioning and hierarchy

Partitioning is one of the oldest approaches to manage complexity. Alexander (1964) proposes to partition the design process into minimally coupled groups. Simon (1981) suggests that complexity can be handled by dividing the original problem as a set of “nearly decomposable groups”. Warfield (1976) uses the successive partitioning of a set of ideas that compose the description of a problem or of its solution to structure the problem or the solution. The groups resulting from the partitioning process are characterised by the fact that the strongest interactions or relationships occur within groups and weaker interactions occur across groups [Simon, 1981]. These groups could then be organised in a hierarchy. In terms of complex product development, [Prasad, 1996a] lists many possibilities for decomposing a product into a hierarchy. The product may span into sets from subsystems to components, to parts, to materials/attributes/features/parameters, and then to common representation and standards. Each of the sets may have different views also organised in a hierarchy such as: physical, functional, geometry and activity views. [Prasad, 1996a]
According to the description above, partitioning is fundamental for the structuring of a product. Ulrich & Eppinger (1995) propose some criteria for clustering product architecture elements into chunks: geometric integration and precision, function sharing, capabilities of vendors, localisation of change, accommodating variety, enabling standardisation and portability of interfaces. Pimmler & Eppinger (1994), for example, use physical, energy, material and information interactions as criteria for architecture clustering and team formation. Lanner & Malmqvist (1996) expands the clustering criteria set to include also modularity and economical criteria. Although, these methods use computerised tools for clustering (or partitioning), these tools are not adequate for the partitioning of large sets with thousands of elements, which may be often the case with current levels of product complexity.

With the increased capability of modern computer technology partitioning algorithms have been developed able to partition sets of tens of thousands elements. These algorithms are graph partitioning algorithms especially developed for configuring a parallel computing machine. The processors in such a machine are grouped with the objective of maximising communication within a group and minimising it across groups. Examples of such algorithms are [Hendrickson & Leland, 1995], [Bhargava et al., 1993]: simple, inertial, spectral, Kernighan-Lin (KL), multilevel KL, bold neural networks, genetic algorithms, mean field annealing and simulated annealing. Many software tools available on the Internet implement some of those algorithms. Examples are: Chaco, Metis, Jostle.

Although these algorithms have been developed for parallel computing, they are being applied to the field of product development. Michelena & Papalambros (1995), for example, present the utilisation of graph partitioning algorithms for the optimisation of a powertrain system design. The equations and inequations describing an optimisation problem are translated into a graph format with the objective and constraint functions represented by vertices and independent variables represented by edges. The optimal decomposition of a design problem calls for: minimising the interconnection between sub-problems and balancing the size of the sub-problems. Michelena & Papalambros (1995) exemplify the partitioning of an optimisation problem with 100 objective functions into sub-problems with 20 objective functions each.

Some partitioning algorithms, however, are being developed by people related to the product development arena. Kusiak, Larson & Wang (1994) introduce the triangularisation algorithm for re-scheduling design and manufacturing processes. Derek Hitchins and collaborators developed ©CADRAT (Computer Assisted Design, Relationship Analysis Tool) for re-structuring N2 charts.

### 2.3 Integrated development

Andreasen & Hein (1987) stated that the ideal company would be composed by one person who had knowledge of market, design, production and economics. However, Hubka (1982) recognises that the knowledge immediately available to an individual is very limited even if he has a number of books on his shelves. With increasing product complexity, as the one-person-company becomes unable to master all aspects involved in the product development problem, a perfectly reasonable step is taken: to seek out others who will be part of a problem-solving team [Warfield, 1994]. Teamwork sets the basis for integrated development.
The term 'integrated development' has, since 1987, been accompanied by other words such as product, process, enterprise, that defines the scope of integrated development. During those years, it has been observed that such a scope has broadened.

2.3.1 IPD

Andreasen & Hein (1987) proposed the term Integrated Product Development (IPD) as an idealised model for product development. According to the model, integration takes place in terms of creation of market, product and production and between project and management, including the need for continual product planning. Product development should be integrated with other development activities, and contribute to renewal and adaptation within the company. Figure 2-9 illustrates the IPD model proposed by Andreasen & Hein (1987).

![Figure 2-9: Integrated product development (Source: [Andreasen & Hein, 1987])](image)

The principles supporting the model in Figure 2-9 are [Andreasen & Hein, 1987]:

- Every product development project should take the need as its starting point, even if it is only for product or production improvement. This is necessary to cope with other newly recognised market, product and production conditions.

- At all times the results which are related to market, product and production must be keeping pace with one another in order to keep track of the regularity of the relationships between the different areas.

- The ideal of simultaneous progress in the three areas cannot be attained, but should be striven towards. Three events (keypoints) in a development project should, at any rate, be given special attention:
  - the start of development, i.e. the situation where the target is more or less clear, and where the feasibility of the market, product and production have been roughly established, and the project is set going.
  - investment, i.e. the situation where decisions on investment in production equipment are made.
- the market launch, i.e. the situation where the production is a reality and the products are to be released onto the market.

- All projects are different. The following variables in product development projects are dealt with: the area of renewal (market/product/production); the degree of renewal; the frequency of renewal; in-house/outside development; the relationship between development costs and sales costs; production to order/mass production; the extent to which the project is predetermined.

According to Andreasen & Hein (1987), integration involves:

- Simultaneous realisation of market, product and production. The aim of product development is the creation of good business for the company. Good business is the result of using the market, the product and its production to maximum advantage. If this task is shared among departments concerned - marketing/sales, development/design, and production - there is a considerable risk that the company will fail to maximise its potential and will drift off course.

- Consideration of three time frames: running production and sales in the short term, creating new products in a longer term and, planning and controlling product development in accordance with a long-term strategy.

- Common objective of three levels of activity: setting a strategy, leading product development projects, performing practical tasks. Serious difficulties may arise in creating agreement between aims, means and results on these three levels.

- Controlled interplay between product development projects. If there is no controlled interplay between projects, with establishment of priorities, then the small ones will gain greater priority for themselves and disturb the larger ones, so that the total result will be confusion. Projects therefore must be integrated.

- Controlled interplay between development activities. Many elements of a dynamic company must be in a constant state of development, so that the company can adjust itself to the demands made on it from outside, and to the objectives set by the management. Integration involves at a high level, organisational, market, product and production development. At a lower level, it involves quality control, financial control, stock control, sales, packaging, advertising and competitor analysis as areas of development. In addition there are support areas such as: material workshop, experimental workshop, logistics, electronic data processing, inspection, patenting and so on.

Cusick (1997) provides a collection of IPD lessons learned from a research performed in the following companies: Electric Boat Corporation, Federal Aviation Administration, Hughes Aircraft Company, Hewlett-Packard Corporation, Lockheed Martin Corporation, Saturn Corporation, Software Engineering Institute, Texas Instruments Incorporated. Some of the top level characteristics of companies who have successfully implemented IPD are:

- focus on people and personal commitment;
- organisational structure that is consistent with company goals;
- emphasis on planning;
- focus on measurement and processes;
- careful monitoring of the decision making processes;
- leadership dedicated to IPD.

Figure 2-10 summarises the lessons learned by the companies above.
### ASPECT | LESSON
--- | ---
Team makeup and structure | - Cross functional team following product development from beginning to end  
- Team size of about 10 people  
- Full-time team members preferred but not practical  
- No member should be on more than 2 teams  
- Team performance measure, e.g., decision making speed  
- Disband dysfunctional team if necessary  
- Spend time thinking about the new organisational structure and ensuring that teams are aligned directly with the work that needed to be done  
- Do not focus excessively on the formal structure of the product teams forgetting the objectives supposed to be accomplished
Training | - Commitment to a lot of training was critical.  
- Training used to communicate desired values and develop a new culture  
- Areas of training are: leadership, management, interpersonal skills.
Decision making | - Need for guidelines to determine if a decision is an individual, a team or a management one.  
- Consensus needs to be defined  
- Consensus decision making is critical for success but not all decisions can be made by consensus  
- Decision must be consistent with company goals or product objectives
Organisation structure | - Keep some form of functional organisation  
- Collocation and an open work environment are critical for success
Communication and data sharing | - Keep the information trail short through a team structure
Stakeholder involvement | - Having the customer on the team is critical for success  
- Maintain strong, long-term relationship with suppliers
Leadership | - Management commitment is the number one factor in IPD success
Rewards and recognition | - Company wide bonus system where people negotiated their reward criteria.  
- Reward traceable to organisational goals
Policies, procedures and tools | - Tools need to be compatible with new organisation structure  
- Standardised automation tools enhance effective communication and data sharing  
- Focus on planning  
- Maintain tight control over financial targets  
- Understand why deviations from the plan occurred  
- Understand how work effect other people through integrated scheduling  
- Well defined operation process
Resource allocation | - Effective IPD requires a front loaded funding profile  
- A product team works reasonably well when resources are controlled by another higher level team on the same project  
- A product team does not work well when resources are controlled by a functional organisation separate from the product teams.

**Figure 2-10: IPD lessons learned**

### 2.3.2 IPPD

With the purpose of establishing the basis for a comprehensive, structured, integrated and disciplined approach to major acquisition programs, the USA Department of Defense (DoD) initiated the concept of Integrated Product and Process Development (IPPD) in the mid-1990s [Air Force Research Laboratory, 1999]. IPPD can be defined as "a management technique that simultaneously integrates all essential acquisition activities through the use of multidisciplinary teams to optimise the design, manufacturing and supportability processes. IPPD facilitates meeting cost and performance objectives from product concept through production, and including field support" [DoD 5000.2-R, 1996].

IPPD aims to cause the integration of the various features of design and the organisations involved in the design process. Inherent within the IPPD concept is the establishment of Integrated Product Teams (IPTs), with the objective of addressing certain designated and well-defined issues. An IPT, constituting a selected team of individuals from the appropriate disciplines, may be established to investigate a specific segment of design, a solution for some outstanding problem, design activities that have a large impact on a high priority technical performance measure, and so on. The objective is to create a team of qualified individuals that can effectively work together to solve some problem in response to a given
requirement. Further, there may be a number of different teams established to address issues at different levels, subsystem level, and/or component level. Figure 2-11 places IPPD and IPTs in a functional organisation structure. IPPD is maintained throughout the various phases of the program activity in order to foster the necessary communications across the more traditional functional lines of authority. An IPT may be established to concentrate on those activities that significantly impact selected performance factors, cost of ownership, and configuration management. There may be another IPT assigned to "track" the integrated data environment issue. The objective is to provide the necessary emphasis in critical areas, and to reap the benefits of a team approach in achieving the best solution possible. [Blanchard, 1998]

**Figure 2-11:** Functional organisation structure showing IPPD/IPTs. (Source: [Blanchard, 1998])

IPTs are often established by a Program Manager, or by some designated high-level authority in the organisation. The representative team members must be well-qualified in their respective areas of expertise, must be empowered to make accurate and precise decisions when necessary, must be proactive relative to team participation, success-oriented, and be resolved to addressing the problem assigned. The Program Manager must clearly define the objectives for the team, expectations in terms of results, and the team members must maintain a continuous "up-the-line" communications channel. The longevity of the IPT will depend on the nature of the problem and the effectiveness of the team in progressing toward meeting its objective. Care must be taken to avoid the establishment of too many teams, as the communication processes and interfaces become too complex when there are many teams in place. Additionally, there often are conflicts when it comes to issues of importance and accomplishing its objectives, it should be disbanded accordingly. An established team that has "outlived its usefulness" can
be counterproductive [Blanchard, 1998]. Warfield (1976) launched the concept of TOTO (Task-Oriented Transient Organisation) which can be applied to IPTs formation guidance.

Armstrong (1996) warns that although teams accomplish many requirements for effective IPPD implementation, the use of teams per se does not guarantee IPPD success. Armstrong lists some IPPD requirements that must be fulfilled in a team environment in order to make IPTs effective. They are: collocation, consensus decisions, early involvement of downstream views, anticipation of budget control issues, cover the necessary breadth of functions, consideration of people issues, empowerment or delegation of authority to the team. Also, Popick & Sheard (1996) list ten lessons learned from implementing IPTs, based on their experience at Loral Federal Systems:

1) Strong upper management commitment to drive the implementation of IPTs in the face of opposition is required;
2) Three to six months after adopting IPTs, there is a high level of frustration and a desire to revert to the traditional approaches. Strong leadership is required to continue to employ IPTs;
3) Take the time to clearly define the IPT purpose, end products, customers, process and product measures, resources, and team incentives. Encourage the IPTs to act within their empowerment domains;
4) Carefully define the consensus decision making procedure, when it is to be used, and use it to make some important decisions at all levels of the organisation;
5) Make sure the leadership (including all levels of management) and the IPTs define, record, and commit to the new roles and responsibilities. Periodically the leadership and the IPTs should review and revise the roles and responsibilities;
6) Effective and efficient team communication depends upon the IPT membership recognising which work is better done as a team, as a sub-team, and as individuals;
7) Establish a formal mechanism for communication between IPTs, and identify IPT dependencies early;
8) Make sure that the IPTs are supported with training that defines a core set of engineering discipline skills, interpersonal skills, IPT methods skills, and project management skills.
9) Engineers and managers need to recognise and adopt a different approach to engineering work product development to realise the benefits of integrated development.
10) The IPT approach requires integration into the overall system of management, with a focus on establishing the IPT empowerment and determining how performance appraisals and rewards will be administered.

[Leibrandt & Gunther, 1998] lists the main aspects of IPPD focusing on three major areas:

- IPPD's focus on product and process life cycle;
- early involvement of stakeholders in the development process;
- concurrent development of product and processes.

Those three major areas can be expanded into ten tenets of IPPD [Leibrandt & Gunther, 1998]:

- customer focus;
- concurrent development of products and processes;
- early and continuous life cycle planning;
- maximise flexibility for optimisation and use of contractor approaches;
• encourage robust design and improved process capability;
• event-driven scheduling;
• multidisciplinary teamwork;
• empowerment;
• seamless management tools;
• proactive identification and management of risk.

Some cases of IPPD utilisation are [Leibrandt & Gunther, 1998], [Usher, Roy & Parsaei, 1998]: the DoD's Joint Strike Fighter program, the Navy's assault ship – the LPD17.

Details on methods and tools for IPPD can be found in the Integrated Development Handbook [DoD, 1998].

2.3.3 IPPED

Integrated Product, Process and Enterprise Design (IPPED) was introduced by [Wang et al., 1997] as a means of shortening product development cycle. IPPED is a paradigm that tears down the barriers to effective communications and drastically increases a company's responsiveness to market opportunities. IPPED is characterised as follows:

• the development of a product (or service) is a process which goes from the recognition of a need to satisfaction of that need.
• as it is a process, it can be managed and improved.
• there is a continuum of implementation of this process.
• the best embodiments are IPPED.

The concept of IPPED is relatively simple. As one designs a product, in addition to functionality and performance, other life cycle attributes, e.g. manufacturability, assemblebility, ease of use, maintainability and recyclability, are also considered concurrently. In order to achieve concurrency, given the fact that most products are designed by a team – not by one individual – and that the team members may not be employed by the same business enterprise, networking and data sharing are critical. IPPED consists of a set of tools and models; some are new, and some are new uses of old tools. Figure 2-12 illustrates a scenario in which a manufacturing business is viewed as four integral states of implementation –

Figure 2-12: Manufacturing business supported by IPPED tools (Source: [Wang et al., 1997])
product planning, product design, process engineering and production and service – supported by a suite of IPPED tools. Not all IPPED enterprises are in the same form or operated in the same style. Depending on the need and the sector in which an IPPED business unit is based, the appearance of it may be drastically different from that of another IPPED business unit. Although, as different as they may appear, all IPPED enterprises share the same concept. They all involve the use of simulation, modelling tools and computerised virtual workstations in conjunction with a design environment which allows a diverse group of researchers, manufacture, and suppliers to work within a comprehensive network of shared knowledge. As a result, the time to market is substantially reduced and customer satisfaction vastly improved.

IPPED shifts the focus from teams as in the IPD and IPPD to computer aided tools and networking.

2.3.4 Additional ‘P’s in integrated development

Some authors argue about the need to include other aspects in integrated development. [Armstrong, 1996] proposes the inclusion of people making IPPPD. [Dube, 1998] proposes the consideration of profit aspects from the outset making IPPPD. [Dube, 1998] describes the experience of Raytheon Systems Company’s Naval and Maritime Systems (NAMS) as part of a partnership, the LPD 17 Alliance composed of 4 companies, to develop a new amphibious ship, the LPD 17. The LPD 17 Alliance was obliged by contract with the US Navy to use IPPD. NAMS is leading the Integrated Ship Electronics Team (ISET), the top-level LPD IPT responsible for integrating the ship’s electronic systems. ISET enhanced its implementation of IPPD by special emphasis on two additional ‘P’s – people and profit.

ISET ranks its “people” as the first ‘P’ in IPPPD recognising the fact that an essential element of IPPD on a large and complex program is to provide a team that has maximum ability to quickly and effectively shift its focus to fit an ever-changing situation. As program phases evolve, requirements shift, work share grows, funding changes, or program goals evolve, a primary characteristic of an effective IPT is that, as an entire team, it be able to quickly adapt to such changes in its situation and stay focused on its program task. Specifically, for ISET this meant forming Cross Product Teams (CPTs) within the program to provide specialised, discipline-oriented support for the IPTs, as opposed to using functional groups that serve the entire enterprise. More generally, this requires a team of attentive and committed people, working in an environment of trust and openness, and consistently seeking consensus and trade-offs. Obtaining such team environment requires attention to the ways people are selected, the team structure is architectured, the teams are aligned and the behavioural expectations are defined.

ISET includes profit as the fourth ‘P’ in IPPPD because NAMS as a business organisation depends on the productive output of its IPTs. Profit is the ultimate goal of its teams. This profit requirement requires that IPTs and CPTs be more than engineering teams. It requires that they take responsibility for all aspects of their task: programmatic, technical and contractual. In addition to technical/engineering responsibilities, each IPT and CPT is also responsible for managing such programmatic responsibilities as cost, schedule and communication.
2.4 Concurrent engineering

The term 'concurrent engineering' was coined in 1986 by the Institute for Defence Analysis (IDA) Report R-338 [IDA, 1986]: 'Concurrent engineering is a systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support. This approach is intended to cause the developers, from the outset, to consider all elements of the product life cycle from concept through disposal, including quality, cost, schedule, and user requirements.'

Figure 2-13 illustrates the phases of a product development process using a CE approach and some techniques used to support each phase. The essence of CE is the integration of product design (represented in Figure 2-13 by 'concept development' and 'detailed design') and process planning (represented in Figure 2-13 by 'manufacturing system concept development' and 'process design'). These tasks are performed, not by individual specialised groups (as in the traditional approach to product development), but by one multidisciplinary team of experts [Bedworth et al., 1991] As a consequence of CE, the following objectives are achieved [Syan & Mennon, 1994]:

- decreased product development lead-time;
- improved profitability;
- greater competitiveness;
- greater control of design and manufacturing costs;
- close integration between departments;
- enhanced reputation of the company and its products;
- improved product quality;
- promotion of team spirit.

Figure 2-13: Typical uses of common CE methods in the product development process (adapted from [Syan & Mennon, 1994])

There are four main broad classes of support for CE [Syan & Mennon, 1994]: process initiatives, computer-based support, data interchange methods and formal techniques. This section reviews briefly the first free aspects and focuses on formal techniques, some of which are presented in Figure 2-13 supporting the phases of the product development process.

2.4.1 Process initiatives

There are mainly two aspects of this type of support for CE. One is the team formation and operation, including its management and support. The other is the organisation of structural and cultural change to
accommodate and to enable the team approach to work effectively. As these aspects were covered in Section 2.3.12, focus is given here to the main process-related characteristics of the CE approach.

According to [Simms, 1993], the Department of Defence (USA) set features that characterise CE, and these became known as the "10+1 commandments of CE":

1. **Create multifunction teams.** CE is structured around multifunctional teams that bring specialised knowledge bases together. Called product development teams (PDTs), these groups are chartered with developing and integrating all the products a program requires. The PDT's charter encompasses all aspects of a product's life cycle, including requirements, design, analysis, cost, quality, manufacture, operations, and disposal. The teams, which replace the traditional functional departments, are often organised by hardware end-item.

2. **Improve communications with customer and user.** PDT members are often collocated, which facilitates communication and helps to break down the organisational barriers that can exist in functionally organised projects. Improved communication with customers enables better understanding of their requirements. CE enhances this understanding because it necessitates customer involvement from the start of the development cycle.

3. **Design manufacturing and support processes concurrently.** The resulting products represent an improved balance between performance and affordability. Traditionally, engineering drawings and specifications. With CE, many companies use an integrated data package approach wherein all the products associated with a particular task are released concurrently. They include those that define not just the hardware, but the associated processes as well. Often none of these products is officially released until all of the elements have been prepared and integrated.

4. **Involve subcontractors and suppliers early.** Many of today's products are not built by the prime contractor. As subcontractors and suppliers best know their process capabilities and must ultimately produce the appropriate hardware, they must be allowed to influence the product's design as it evolves.

5. **Incorporate lessons learned.** On-line databases with keyword search capability permit teams to capitalise on both positive and negative past experiences.

6. **Create a digital product model.** It enables electronic data sharing by everyone involved in product definition. Such models involve more than just CAD, encompassing design definition, schedules, build instructions, and operation manuals. These common databases eliminate duplicate effort and give everyone access to current data.

7. **Integrate CAE tools with digital product model.** CAE activities such as finite element modelling have become very specialised and often are not integrated with other models being created. CE stresses use of a common database that, for example, permits analysis models to be created from and tied to the design models.

8. **Simulate product performance.** It helps designers to understand, before hardware is built and tested, how a product will perform. Simulations are used early and the results incorporated during the development cycle. Digital mock-ups simulate the hardware assembly and, for example, check interference minimising the need for physical mock-ups.

9. **Simulate manufacturing processes.** Statistical process control (SPC) is one tool that many companies are using to understand process variables, control processes, and reduce inspection requirements.
10. Improve process continuously. Many companies have found that CE allows them to improve the internal plans and processes significantly. Metrics assist in monitoring team progress and provide feedback for continuous improvement. For example, gathering data performing to out-of-specification history by part can provide an excellent metric for driving the corrective action process. In addition, PDTs use selected metrics to influence product development. Metrics for design simplicity, such as number of parts or processes, can be effective for reducing costs.

10+1. Integrate technical reviews. It helps to co-ordinate team's activity. The CE team approach results in more meetings, so it is essential that the various reviews be focused and productive. Informal reviews are held at the project level, within the team, and between related teams. Those reviews start after the initial "thinking/creation" period and before the design, process, procedure, and analysis are finalised. They continue as the task matures and help keep team members informed and co-ordinated.

2.4.2 Computer-based support

Computer based support include [Bedworth et al., 1991], [Syam & Menon, 1994], [Prasad, 1996a]:

- computer-aided design and manufacturing (CAD/CAM) tools: have the capabilities of three-dimensional shape modelling and the ability to derive physical properties (e.g. weight, centre of gravity, etc.) and to produce manufacturing data such as numerical control data and programme files. Solid modelling provides opportunities to integrate design, process planning, production planning and production through the ability to carry out interactive verification of manufacturing, assembly, and rapid prototyping machines. These can take a model form a package and produce a plastic prototype in minutes for verification purposes. These facilities enable the whole modelling, draughting and manufacturing process to be speeded up greatly.

- computer-aided engineering (CAE) tools for electronic, mechanical and other disciplines.

- engineering data management (EDM) tools: design projects of even small scale can require up to 1000 drawings and there may be up to ten times that of related information. This information is made up of product specifications, simulations, test and analysis results, costs and schedules, manufacturing, tooling, assembly and maintenance, and so on.

- modelling and simulation tools, production planning and costing tools.

- computer integrated manufacturing (CIM): is a management philosophy in which the functions of design and manufacturing are rationalised and co-ordinated using computer, communication and information technologies. Figure 2-14 shows scope of CIM in the major elements of a manufacturing business.

2.4.3 Data interchange methods

As the provision of computer tools increased together with the variety of suppliers world-wide and due to the fact that products are rarely designed, manufactured and maintained entirely by a single company, product data should be defined and exchanged unambiguously and consistently during the entire course of product realisation. Neutral file formats is an increasingly adopted solution for compatibility among computer tools. The more common standards for neutral file formats are: IGES (initial graphics exchange specification); SET (standard d'exchange et de transfert); VDA (verbad der automobilindustrie
STEP is an emerging international standard [ISO-10303-1, 1994] that uses a high level, feature-based and object-oriented approach to define products. It is able to provide a complete, unambiguous, computer interpretable definition of the physical and functional characteristics of each unit of a product throughout its life cycle. The new standard enables the following:

- communication among heterogeneous computer systems;
- integration of manufacturing functions/processes, such as design and analysis, design and manufacturing;
- automatic, paperless updates of system documentation.

As an example of the use of STEP, consider virtual workstations developed and integrated in such a way that they are compatible with the STEP application protocols. A STEP-based application protocol (AP) suite is developed. These application protocols serve as integration tools between CAD and other applications. In each application protocol, the product data structure required for the specific application is defined. The product data from CAD are processed based on the AP data structure through EXPRESS models [ISO-10303-11, 1994], and object databases are formulated. The objects in the database are ready to use for any engineering application.

A STEP server in the virtual server cluster manages product data exchange and record integrity. It maintains product configuration and record change rationale that provide a design audit trail for product and process design.

2.4.4 Formal techniques

Concurrent engineering (CE) is traditionally seen in terms of considering manufacturing requirements during the early design stages of a product. However, a broader view of CE anticipates to the early design stages all the requirements imposed to a product development during its entire life cycle, not only the manufacturing ones. Concurrent engineering delivers better, cheaper and faster products via the continuous consideration of all constraints. In a perfect CE world all constraints are considered at every decision point – be they manufacturability, reliability, testability, performance, etc. In traditional product
development, each constraint has been dealt separately as a matter of policy. The success of CE, in
general or in any single CE tool, is achieved by increasing the number of constraints to be considered at
each decision point.’ [Parsaei & Sullivan, 1993]. The ability to capture those constraints or
requirements is decisive to the success of the product design.

This section provides a brief review of the CE techniques used throughout the manufacturing industry to
capture, communicate and analyse those requirements. These methods and their aims are:

- **Quality Function Deployment (QFD):** communicates customer requirements throughout the
development, manufacturing and production stages of the product life cycle;
- **Taguchi methods:** brings robustness concerns through experimental development of products;
- **Axiomatic design:** summarises a set of good practice rules into design axioms and use these axioms as
  a reference to guide and evaluate the decisions taken during the design process.
- **Theory of inventive problem solving (TRIZ):** proposes a systematic way for evolving engineering
  systems
- **Design for Assembly (DFA):** anticipates assemblability requirements to the early stages of product
development;
- **Group Technology (GT):** most commonly used for production planning, but its cluster algorithms can
  also be used to relate product requirements to customer requirements and manufacturing requirements
  to product requirements and to optimise the allocation of resources.
- **Value Engineering (VE):** makes a balance of the value of product functions to the customer and the
cost to implement them. It can anticipate cost constraints to the early stages of product development.
- **Failure Mode Effects Analysis (FMEA):** analyses the effects of potential failure modes at all levels of
  the product breakdown structure.
- **Hazard Analysis:** analyses the gravity, frequency and risks of hazards related to the product life cycle.

### 2.4.4.1 Quality function deployment [and the Kano model]

QFD is a structured planning tool of concurrent engineering [Syan & Mennon, 1994]. It maps the
customer requirements into specific engineering design features (and eventually into manufacturing
process) through one or more matrices of ‘expectations and fulfilment options’. QFD is used as a
systematic approach to both identify and prioritise customer requirements, and to translate these
requirements into product and process specifications [Eureka & Ryan, 1992]. (see Figure 2-15)

![Figure 2-15: The translation process of QFD (adapted from [Greenall, 1995])](image-url)
The structure of all translation matrices is similar. The first translation matrix which is often called 'house of quality' or 'voice of customer' has the most general structure (see Figure 2-16).

QFD was initially developed and applied by ship industries in Japan [Akao, 1990]. Today, its first matrix (the house of quality) is largely used within the automotive industry (e.g. Toyota [Hauser & Clausing, 1988], Ford, General Motors [Schilke, 1994]). [Akao, 1990] provides also examples of the use of QFD in service enterprises. Some applications use QFD together with the Kano model [Greenall, 1995], where requirements can be classified as customer excitement requirements, performance requirements and basic requirements and customer satisfaction can be evaluated over time (Figure 2-17).
An example of excitement feature is: car with navigation systems. Examples of performance features are: shape, road holding at speed, ABS, airbags, etc. In the past, switch on a car in the winter at the first attempt was considered good, today it is taken for granted.

Recently, many variants of QFD have appeared such as EQFD [Clausing, 1994] and Q^2FD [Maier, 1996].

**EQFD – Enhanced QFD**

Clausing (1994) proposes some enhancements to the conventional QFD concept based on Pugh’s total design techniques (see Sections 2.2.6.1 and 2.2.6.3). In the conventional QFD concept, total product system expectations are directly deployed into piece-part characteristics. This procedure is straightforward for simple products that are conceptually static. For conceptually dynamic products, Clausing proposes the use of Pugh’s concept selection process [Pugh, 1991] before piece-parts are considered. Also, for complex products, Clausing proposes the total systems expectations to be deployed down to the subsystem level and then to the piece-part level.

**Q^2FD – Quantitative QFD**

Q^2FD extends conventional QFD to directly include quantitative performance models in the cascading. In Q^2FD, a “satisfaction model” calculates a numerical value for each performance objective from engineering parameter values. For complex products, the product can be decomposed in successive subsystem levels. The engineering parameters at one level will be the performance objectives of the next level down, as illustrated in Figure 2-18. [Maier, 1996] proposes a computer spreadsheet implementation of Q^2FD. Setting target objective values, parameter values are modified until calculated objective values match to target. For complex products, for example, this approach would potentially show the effects in the objective values of a modification in the component parameters. Therefore Q^2FD corresponds to a dynamic QFD. Whereas QFD provides a ‘picture’ of a cascading of interactions from requirements to production operations, Q^2FD shows the modifications in the requirements when a component and potentially its production operation changes.
2.4.4.2 Taguchi methods

According to [Taguchi et al., 1989], product design has the greatest impact on product quality. It is essential to consider all aspects of the design (including factors built into the product) that affect deviation of functional characteristics of the product from target (nominal) values. It is also necessary to consider methods to reduce the undesirable and uncontrollable factors (such as noise) that cause functional deviations.

Taguchi defines 'quality of a product' as the (minimum) loss imparted by the product to the society from the time the product is shipped. This loss function is expressed as a parabola, where the minimum is the target performance which deviates from ideal with deviations in the value of the quality characteristic.

According to Taguchi, three steps are applied to a design of a product. They are:

- **System design**: denotes the development of a basic prototype design that performs the desired and required functions of the product with minimum deviation from target performance values. It includes the selection of materials, parts, components, and the assembly system.

- **Parameter design**: determines the optimal combination of levels of parameters and all components of the prototype, that minimises or diminishes the effects of various noises while keeping performance as close as possible to its nominal (target) value. The key is to identify the parameters which have the greatest effect on product performance and design the product to desensitise the function to variations in the parameter, statistically separating the effects created by each of the individual factors by plotting the factors orthogonally [Bedworth et al., 1991]. The controllable and noise factors are placed in different arrays so that the experiment can calculate the ratio (Signal/Noise) of how the controllable factors affect the result when compared to the noise factor. This helps to produce a more robust product. Other types of analysis that can be performed are measures of interaction between two factors [Ross, 1988a].

- **Tolerance design**: determines the most economical tolerances, those that minimise product cost for given tolerable deviation from target values. It determines the tolerance of each individual parameter (factor) by trading off quality loss and cost. In effect, it is necessary to define allowable ranges of deviation in the parameter value. Obviously, the narrower the range of deviation, the more costly the product becomes, as a result of increased manufacturing cost. On the other hand, the wider the range of deviation, the larger the deviation in the product function from a specific target value.

Examples of manufacturing enterprises where Taguchi methods have been used are [Taguchi, 1989] and [Ross, 1988a]: AT&T, Ford, Xerox and Toyota.

2.4.4.3 Axiomatic design

Axiomatic design is the application of design axioms and corollaries during the design process. According to Suh (1990), there are two fundamental axioms that govern good design:

Axiom 1 - The independence axiom: Maintain the independence of 'functional requirements'.

Axiom 2 - The information axiom: Minimise the information content of the design.
Axiom 1 states that during the design process, the mapping between 'functional requirements' in the 'functional domain' and 'design parameters' in the 'physical domain' must be such that a perturbation in a particular 'design parameter' must affect only its referent 'functional requirement'. Axiom 2 states that, among all the designs that satisfy Axiom 1, the one with minimum information content is the best design.

These axioms have corollaries that can be demonstrated from the axioms. These corollaries may be applicable not only to product design but also to process design, such as manufacturing and assembly, looking at the relationships between product design parameter and a process characteristic. Examples of such corollaries are [Bedworth et al., 1991]:

- minimise the number of functional requirements and constraints.
- satisfy the functional requirements from most important first to least important last.
- decouple the separate parts of a solution if functional requirements are coupled or interdependent.
- integrate functional requirements, in a single part if they can be independently satisfied in the proposed solution.
- everything being equal, conserve materials.
- there may be several optimal solutions.
- part count is not a measure of productivity.
- cost is not proportional to surface area.
- minimise the number and complexity of part surfaces.
- if weaknesses can be avoided, separate parts.
- use standardised or interchangeable parts

The ideal application of the axiom 1 over an entire product decomposition tree (system, subsystems, components, parts, features, materials), including manufacturing and assembly processes, would lead to a tree-like diagram with functional requirements at the top deriving successive levels of product design parameters, until manufacturing and assembly design parameters at the bottom of the tree. The modification of any one parameter at the lower levels of the tree would affect only one functional requirement at the top. The modification of any functional requirement at the top would lead to a cascade of changes identifying the necessary changes at the bottom of the tree at the process levels. The application of axiom 2 would lead to the minimum complexity (if complexity is measured by information content) to perform those modifications.

2.4.4.4 Theory of inventive problem solving

The theory of inventive problem solving (TRIZ is its Russian abbreviation) has been developed in the former Soviet Union by Genrich Altshuller, starting in the 1950s. The main postulate of TRIZ is: 'evolution of engineering systems is not a random process, but it is governed by certain laws'. Altshuller analysed about 400,000 invention descriptions from different fields of engineering. The most effective solutions were selected and examined to reveal the objective laws (trends) of evolution of engineering systems. The evaluation of the solutions' effectiveness was based on the concept of 'engineering contradiction': a problem becomes a creative one when an attempt to improve system's parameters by conventional means leads to deterioration of other parameters, i.e., generates an 'engineering contradiction'. From the standpoint of TRIZ, solving a problem means overcoming this contradiction, or satisfying all conflicting requirements. Altshuller formulated eight laws of evolution of engineering
systems [Altshuller, 1984] summarised as follows:

1) Engineering systems follow a life cycle of birth, growth, maturity, decline.
2) Engineering systems evolve towards increasing ideality.
3) In the course of an engineering system evolution, uneven development of subsystems result in contradictions.
4) Engineering systems evolve from a rigid structure into a flexible or adaptive one.
5) Engineering systems evolve towards increasing use of controllable fields.
6) Engineering systems evolve towards increasing complexity, followed by simplicity through integration.
7) Engineering systems evolve through matching and mismatching of parts.
8) Engineering systems evolve towards increasing dispersion of their components, from macro to microlevel.

Fey et al. (1994) describe an analytical tool that provides specific mechanisms for developing engineering systems according to TRIZ's laws. The tool is called ARIZ (Algorithm for Inventive Problem Solving). Solving a problem by application of ARIZ starts with the formulation of a mini-problem that defines the function to be fulfilled giving that everything else in the system remains unchanged. The engineering contradiction is then formulated and a 'conflict zone' is identified. The problem is treated by formulation of the Ideal Final Result (IFR). Usually, to realise IFR, the critical component of the system in the conflict zone must possess contradictory physical properties. This situation is called a physical contradiction. The next part of ARIZ offers three basic groups of methods for overcoming the physical contradiction: 1) separation of contradictory properties in time, or in space; 2) system transformations (combination of systems, combination of system and anti-system, separation of contradictory properties between system and its sub-systems); 3) phase transformation of substances, or physical contradiction is provided by maximal utilisation of material resources in the system and is supported by a database of physical, chemical and geometrical effects. If the problem has not been solved, it should be reformulated and the process re-started. ARIZ is being implemented by software tools developed by Ideation International Inc.

Facilitating the identification of the engineering contradiction helps to understand the right problem to be solved and as a consequence to encapsulate its complexity. Once the physical contradiction is identified, no efforts will be wasted in terms of an evolutionary approach. The inventive approach will be then pursued.

TRIZ has been used by many manufacturing companies such as: Allied Signal Aerospace Sector, Chrysler Corp., Emerson Electric, Ford Motor Co., General Motors Corp., Rockwell International, UNISYS and Xerox Corporation.

2.4.4.5 Design for assembly
The two parts of DFA include, first, a catalogue of generic part shapes and types by group technology methods according to ease of feeding by parts and ease of assembly by manual or automatic means. Estimates are given for assembly times. Assembly times usually are directly proportional to assembly costs. The second part of DFA is a source of rules, advice, or prompting questions concerning good
DFA practice. It is assumed that if these criteria are met, the product design will have achieved some savings in assembly costs.

The technique is manifested in a suite of software modules as described below. The user begins by knowing the design of each of the assembled components and an exploded view of the assembly. The first package assists the designer in determining if the product is a candidate for automatic assembly. The software queries the user the values of various parameters that are important in automated assembly. The results are presented as a list of costs for using several assembly systems together with estimated costs of manual or automated assembly versus production volume.

Once it is determined which of the assembly methods will be suitable, the user enters a procedure to optimise the assembly itself. Part codes are given based on answers to questions about geometry, function, and foreseen problems, such as part tangling or nesting during feeding. The codes give the designer an idea of assembly costs and possible location of bottlenecks in assembly or large costs. Finally, the designer is asked questions to determine the theoretical minimum number of parts in the assembly based on relative part motion, different required materials, and repeated assembly-disassembly of certain parts to allow assembly of other parts. The software will tell the user if parts should be eliminated or perhaps combined to be multifunctional in assembly. (see [Boothroyd, Dewhurst & Knight, 1994]).

Examples of manufacturing industries which use the DFA method are [Bedworth et al., 1991]: Nippondenso (Toyota) and Lucas.

2.4.4.6 Group technology
The principle of GT is that the characteristics of a part can be codified into a multidigit code. This code can then be used to classify into families for ease of design retrieval and variant process planning. During the design process, the engineer can examine all parts currently made by a company to determine if a part with the desired characteristics already exists. If it does then a new design is not necessary. It may be that a similar part exists, in which case a simple design change may bring it into specifications for the current application. GT can also help in the manufacture of parts by being used to extract the process plan of a similar part and modify it to be used on the new part. Whether this technique should be used depends on the quality of the old part’s process plan. One would not want to replicate a poor plan for a new part [Bedworth et al., 1991].

[Bedworth et al., 1991] provides examples of the use of GT in aerospace and automotive industries.

2.4.4.7 Value engineering
Value engineering is an evaluative technique that assigns a value to a product. The value is defined as the ratio of the product by raising the functional worth or, more commonly, decreasing the cost of each function. The goal is to eliminate unnecessary features and functions by optimising the value ratio. This value can be divided into two components: a use, or functional, value and a esteem value. The use value reflects how the product satisfies the user’s needs, and the esteem value is a measure of the desirability of the product, a marketing and advertising concern. The two values are investigated
analytically by a team of experts based on a preliminary design. A creative attempt is made to maximise them. There are really no guidelines except experience to direct the design optimisation.

The primary component of value engineering is the functional statement. Each component of the product as well as the overall product itself is given a brief functional statement and then that function is given a numeric value based on a discussion by design team members. A cost for that function is also determined based on manufacturing and other costs.

The most difficult aspect of value engineering is estimating the values of the functions and costs. Value engineering attempts to use the values in a systematic way. (see [Bedworth et al., 1991] and [Cross, 1989]).

[Cross, 1989] provides various examples of the use of value engineering for the development of: turbine gears, generators, air valve, pindle, piston, electric motor, tubular heater, especially for the aerospace and automotive industries.

### 2.4.4.8 FMEA Technique

Failure-mode effects analysis (FMEA) provides the design team with an organised approach to evaluate the causes and effects of various modes of failure in a product. The goal is to increase overall reliability by anticipating failures and designing them out of the product.

The first step in FMEA is to list all possible types of failure. All of these modes of failure are then ranked according to their effect. Failures that are catastrophic are ranked very high, and those that do not affect function are ranked low. Each failure is addressed one by one from most to least important. Design changes are then made to reduce the chance of failure. Design simplification begins with functional analysis: determining what the product should do rather than what it does now.

Figure 2-19 presents an example of FMEA chart. Failure modes are identified, ranked and associated to specific requirements from initial drawings of the product. Prioritisation of the failure modes is obtained by taking the product of the frequency (F), severity (S), and detection (D) and then ranking the risk (R) factors in order of highest to lowest. (see also [Juran & Gryna, 1988])

**Figure 2-19**: Example of failure mode effects analysis (Source: [Bieda, 1993])

### 2.4.4.9 Hazard Analysis

Hazard analysis [MISRA, 1994] is used to identify, evaluate and record system hazards to assist the design of a system to meet required safety levels. The objective is to determine the maximum tolerable risk and achieve a level of risk that is as low as reasonably practicable and below the maximum tolerable risk. The procedure broadly splits into 2 sections. The first section is a preliminary analysis of the project before any significant design work is started. This phase identifies potential hazards to focus design
effort. The second phase is done once the basic design work has been completed to verify that sufficient action has been taken to cover the identified hazards and determine any new hazards generated by the design.

Once the hazards are captured, they are assessed in terms of probability of occurrence and controllability. Risk classes are then attributed to each hazard according to Figure 2-20. Depending on the risk, adequate (design) actions are assigned to each hazard.

Risk Classes: Class I - Intolerable risk, Class II - Undesirable risk, tolerable only if risk reduction is impracticable or if its cost is grossly disproportionate to the improvement gained, Class III - Tolerable risk, if the cost of risk reduction would exceed the improvement gained, Class IV - Negligible risk

<table>
<thead>
<tr>
<th>Probability</th>
<th>Uncontrollable</th>
<th>Difficult to control</th>
<th>Debilitating</th>
<th>Distracting</th>
<th>Nuisance only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequent</td>
<td>9-10</td>
<td>I</td>
<td>5-6</td>
<td>3-4</td>
<td>1-2</td>
</tr>
<tr>
<td>Probable</td>
<td>7-8</td>
<td>I</td>
<td>II</td>
<td>III</td>
<td>IV</td>
</tr>
<tr>
<td>Occasional</td>
<td>5-6</td>
<td>II</td>
<td>II</td>
<td>III</td>
<td>IV</td>
</tr>
<tr>
<td>Remote</td>
<td>3-4</td>
<td>III</td>
<td>IV</td>
<td>IV</td>
<td>IV</td>
</tr>
<tr>
<td>Incredible</td>
<td>1-2</td>
<td>IV</td>
<td>IV</td>
<td>IV</td>
<td>IV</td>
</tr>
</tbody>
</table>

*Figure 2-20: Classification of risks as a function of gravity and frequency of hazards (Source: [MISRA, 1994])*

### 2.4.5 Techniques integration

Although CE started focusing on the integration between design and manufacturing, the utilisation of CE, during the 1990s, increased in depth and scope [Huang, 1996]. For example, under ‘design for manufacturing’, there is design for assembly, for dimensional control, for inspectability. Also other life cycle processes are increasingly being included in the CE concept with methods such as: design for effective material storage and distribution, design for reliability, design for electromagnetic compatibility, design for serviceability, design for recycling. Approaches for optimisation of the entire product life cycle is presented by [Yoshimura, 1996]. Also, [Watson, Radcliffe & Dale, 1996] proposes a meta-methodology that uses a weighted matrix approach to determine interactions between competing ‘design for ...’ (DFX) techniques.

An example of a commercial computer tool that integrates CE techniques is Team SET. TeamSET [CSC, 1994], developed by Lucas Engineering & Systems and traded by CSC, is a commercial computer tool which aims to integrate, through a common database, the following methods:

- DFA: reduces parts count and assembly cost
- MA (Manufacturing Analysis): selects the most cost-effective material and process
- FMEA: identifies and prioritises corrective action
- DTC (Design to Target Cost): monitors product cost against targets
- QFD (not integrated yet): focuses design effort on real customer priorities
- Con-Con (Controlled Convergence, not integrated yet): selects iteratively concepts against criteria.

Design Function Deployment [Jebb et al., 1993] developed at City University (London), is a comprehensive design system, which incorporates the features of a prescriptive design model and associated design methods (Finite Element Analysis, FMEA, DFA, Taguchi, Simulation, etc.) for the integration of manufacturing, use and other downstream issues into the design and thus enabling a
'concurrent engineering' approach to product or process development. It uses the QFD-4 matrix-model as a core model and proposes the use of various engineering design tools to provide information to those matrices in order to develop customer driven design, product and manufacturing process. The prescriptive design model used in Design Function Deployment is: obtain customer requirements and translate them into design requirements (specifications and constraints) in a solution neutral form; propose different architectures or conceptual solutions; establish viable layouts within each architecture; establish viable materials, manufacturing process and geometries for each layout; establish production plans for each layout.

2.5 Systems engineering (SE)

The discipline of 'systems engineering' originated in the late 1940s and early 1950s from a blending of theoretical foundations of systems science with the World War II production experience [Brill, 1998]. SE has always been devoted to the development of complex products, starting with the development of ballistic missiles in the late 1950s. SE had its peak in the mid-1960s when the US-DoD adopted and mandated the use of SE on the development of all DoD projects. The SE concepts were reduced to a set of regulations (set forth in the MIL-STD-499 and associated standards) and educational materials which were taught to DoD personnel and every DoD Prime Contractor in the USA. With the great aerospace crash of 1968-1969 with funding being diverted from weapons development to weapons production, interest in SE decreased. Since there was no Journal of Systems Engineering or any professional organisation of systems engineers, SE knowledge became dispersed. Although the aerospace industry revived with increasing funding in the 1970s, the practice of SE deteriorated at a time when systems were becoming increasingly complex due to the introduction of computers into systems. As a result, the failure rate of delivered systems increased. At that time SE was very much viewed as a document-driven approach [Parth, 1998].

From the late 1970s and during the 1980s, as software complexity became an issue, many powerful new notations and methodologies were developed. Also, the concept of concurrent engineering was introduced in the mid-1980s. During the same period computer aided tools were developed. The first CASE (Computer Aided Software Engineering) tools implementing the notations and methodologies were developed. Also, VHSIC Hardware Description Language (VHDL) for chip design, System Description Language (SDL) for telecommunications, CAD (Computer Aided Design) and CAE (Computer Aided Engineering) for mechanical and electronic parts were developed supporting other engineering disciplines. Systems engineers themselves also developed tools, usually small proprietary systems to address specific SE needs: traceability from requirements to specifications, modelling, and simulations [Malcolm & Alford, 1993].

In the late 1980s and 1990s, a revival of SE was due to two main factors: the founding of INCOSE (International Council on Systems Engineering, formerly NCOSE) and the availability of automated tools for SE [Honour, 1998]. The introduction of the personal computer in the early 1980s led to the development and use of a large number of PC-based tools to support SE; the late 1980s saw the introduction of the first commercial integrated SE tools. Some automate specific tasks such as requirements traceability or simulation. Others combine requirements traceability with document
creation, provide modelling with simulation, or address a whole spectrum of SE tasks. Because systems engineers must communicate with engineers from all other disciplines, SE automation has increased the demand for an integrated tool set for concurrent engineering. Full SE with full life cycle support is a challenge currently being addressed. This would facilitate the trend of SE moving from a document-driven to a trade-off-driven discipline [Parth, 1998].

2.5.1 System

According to the [United States Army Field Manual - Systems Engineering, FM 770-78] system is "a composite of equipment, skills and techniques capable of performing and/or supporting an operational role. A complete system includes related facilities, equipment, material, services, software, technical data and personnel required for its operation and support to the degree that it can be considered a self-sufficient unit in its intended operational and/or support environment. The system is employed operationally and supported logistically".

This definition is also used by [Reilly, 1993] and is used here because it includes life-cycle aspects such as users (skills and techniques) in addition to the physical system and its logistics support. The military definition is oriented around operational roles. The use of the term "self-sufficient" means that systems have well-defined functional boundaries crossed by inputs and outputs. Constraints placed on the system limit its operation and define the boundary within it is intended to operate. The boundary may contain a simple function or it may contain many functions. Everything that remains outside the boundaries of the system is considered to be the environment.

"Systems" have the following properties (compiled from [Blanchard & Fabrycky, 1990] and [Reilly, 1993]):

• every system performs a purposeful action which is called its function. The function of a system must be explicitly defined and understood so that system elements may provide the desired output for each given set of inputs. Once defined, the objective or purpose makes it possible to establish a measure of effectiveness indicating how well the system performs.

• systems consist of an interdependent group of items (subsystems) that perform together as a functional unit. The functional boundary is what determines the system definition in any given application of the concept.

• systems are composed of elements, attributes, and relationships. These are described as follows:
  • elements are the operating parts of a system consisting of input, process, and output. Each system element may assume a variety of values to describe a system state as set by control action and one or more restrictions.
    • the properties and behaviour of each element of the set has an effect on the properties and behaviour of the set as a whole.
    • the properties and behaviour of each element of the set depends upon the properties and behaviour of at least one other element in the set.
    • each possible subset of elements has the two properties listed above;
    • the elements cannot be divided into independent subsets.
attributes are the properties or discernible manifestations of the elements of a system. These attributes characterise parameters of a system.

relationships are the links between elements and attributes.

• a system is more than the sum of its elements parts because the set of elements comprising a system always has some characteristic or behaviour pattern that cannot be exhibited by any of its subsets.

• the elements of a system may themselves be systems, and every system may be part of a larger system in a hierarchy. The definition of a system is not complete without consideration for its position in the hierarchy.

Examples of systems are those which alter material, energy, or information. They are composed of structural elements (static parts), operating elements (perform the processing), and flow elements (material, energy, or information being altered). Structural, operating and flow elements have various attributes that affect their influence on the system.

Modern SE standards (IEEE-1220-1994, EIA-632) expand the scope of a system to include not only an operational end-product element but also its life cycle process elements as illustrated in Figure 2-21. However, only the end product and its subsystems are object of the development effort and only requirements are set for the life cycle processes.

2.5.2 Systems thinking

[Blanchard & Fabrycky, 1990] states that there are two dominant ideas for understanding the world: reductionism and mechanism. Reductionism consists of the belief that everything can be reduced, decomposed, or disassembled to simple indivisible parts. Reductionism gives rise to an analytical way of thinking. Analysis consists of: 1) taking apart what is to be explained, disassemble it, if possible down to indivisible parts of which it is composed; 2) explaining the behaviour of these parts; and, finally, 3) aggregating these partial explanations into an explanation of the whole. The limitation of reductionism is that it does not consider that the parts may interact with each other in such a way it is not possible to divide the whole into completely independent parts. The behaviour of the whole may not be resulting only from the behaviour of its parts but also from the interactions among parts. Mechanism consists of the belief that all phenomena are explainable by using only one ultimately simple relation, cause and effect. A cause is the necessary and sufficient condition to explain an effect. The limitation of the mechanistic view is that it ignores other elements in the environment that may affect the cause-effect relationship.

[Blanchard & Fabrycky, 1990] proposes that to cope with the increasing complexity and scope and increased pace of technological growth and change in the engineering work it is necessary to shift from
the doctrines of reductionism and mechanism and the analytical mode of thought to the doctrines of expansionism, teleology, and a new synthetic mode of thought. These characterise systems thinking. Expansionism is a doctrine which maintains that all objects and events, and all experiences of them, are parts of larger wholes. It provides another way of viewing things, a way that is different from, but compatible with, reductionism. In synthetic thinking, something to be explained is viewed as part of a larger system and is explained in terms of its roles in that larger system. This way of thinking is based on the observation that, when each part of a system performs as well as possible, the system as a whole may not perform as well as possible. To be teleologically oriented refers to the preoccupation of seeking a goal or being purposeful.

The above characteristics of systems thinking match with [Paul, 1997] in his research about the 'system' meaning. According to him, the essence of system relies on three main aspects: wholeness, interdependence and purpose.

2.5.3 Systems engineering definitions

[MIL-STD-499A, 1974] defines 'systems engineering' as: "the application of scientific and engineering efforts to:

• transform an operational need into a description of system performance through the use of an iterative process of definition, synthesis, analysis, design, test, and evaluation;
• integrate related technical parameters and ensuring compatibility of all physical, functional, and program interfaces in a manner that optimises the total system definition and design;
• integrate reliability, maintainability, safety, serviceability, human engineering and other such factors into the total engineering effort to meet cost, schedule, survivability, and technical performance objectives".

[Reilly, 1993] defines 'systems engineering' as "the systematic application of proven standards, procedures, and tools to the technical organisation, control, and establishment of:

• System requirements;
• System design;
• System management;
• System fabrication;
• System integration;
• System testing; and
• System logistics support".

Reilly adds the word 'systematic' to the first definition. 'Systematic' means, according to Reilly, 'definite generic sequence of activities that must be consistently followed'. Reilly also translates the 'application of engineering and scientific efforts' of the military definition into 'application of proven standards, procedures, and tools'. 'Systems requirements' in Reilly's definition is the 'transformation of an operational need into a description of system performance' in the military definition. System design is included in both definitions. Like in the military definition, Reilly also integrates life cycle aspects (fabrication, integration, testing and logistics support) into the engineering effort. However only Reilly mentions the 'system management' function (which includes configuration management and systems engineering management planning).

The [IEEE-Std 1220-1994, 1995] provides a modern definition of 'systems engineering': "an interdisciplinary collaborative approach to derive, evolve, and verify a life cycle balanced system solution that satisfies customer expectations and meets public acceptability".
In this case ‘systems engineering’ is considered to be ‘an approach’ rather than ‘application of efforts’ or ‘application of proven standards, procedures and tools’. The approach is interdisciplinary and collaborative (not explicitly mentioned in the two other above definitions) which is necessary to handle complexity. Like the two definitions above, also this definition ‘derive and verify a life cycle balanced system solution’, but the term ‘system solution’ gives a more generic character to this definition. The solution may be a product, a process or an organisation, for example. Only this definition includes the term ‘evolve’, which fits the needs of automotive systems development, for example. The overall objective of the system is established as: ‘to satisfy customer expectations and meet public acceptability’ and is broader than in the two other above definitions, where systems engineering is limited to meet ‘operational needs’ or ‘systems requirements’.

The [IEEE-Std 1220-1994, 1995] also defines two other terms to support its definition of systems engineering:

- **Life cycle**: the system or product evolution initiated by a user need or by a perceived customer need through the disposal of consumer products and by-products.
- **Customer**: a person or organisation ordering, purchasing, receiving, or affected by a product or process. Customers include developers, manufacturers, testers, distributors, operators, supporters, trainers, disposers, and the general public.

Figure 2-22 compares these definitions and summarises what was mentioned above.

<table>
<thead>
<tr>
<th>SYSTEMS ENGINEERING</th>
<th>ACCORDING TO:</th>
<th>ULTIMATE GOAL:</th>
<th>WORKS THROUGH:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IS:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[MIL-STD-499A, 1978]</td>
<td>application of engineering and scientific efforts</td>
<td>to transform operational need into a description of system performance; to optimise the total system definition and design; to meet cost, schedule, survivability, and technical performance objectives;</td>
<td>iterative process of definition, synthesis, analysis, design, test and evaluation; integration of related technical parameters and assurance of compatibility of interfaces; integration of life cycle aspects into the total engineering effort.</td>
</tr>
<tr>
<td>[Reilly, 1993]</td>
<td>systematic application of proven standards, procedures and tools</td>
<td>to meet systems requirements</td>
<td>technical organisation, control, and establishment of: System requirements; System design; System management; System fabrication; System integration; Systems testing; and System logistics support.</td>
</tr>
</tbody>
</table>

*Figure 2-22: Three definitions of systems engineering*

According to Figure 2-22, from the first to the third definition, the ultimate goal of systems engineering becomes broader going from system products performance (basis of old definition of quality) to customer satisfaction (basis of TQM’s definition of quality), i.e. from the product to the market. The three definitions present an increasing level of details, from the third to the first.
2.5.4 Systems engineering models

From the early 1980s when systems started to become more software intensive, a number of models were developed in the software engineering arena and used to describe a system life cycle. The "waterfall model" is probably the oldest and most widely used. This model, shown in Figure 2-23, is based on a top-down approach for software development and includes the steps of initiation, requirements analysis, design, test, and so on. Often, in its implementation, the steps are viewed as being relatively independent from one another and must be executed in strict sequence. Additionally, the required interfaces with the other elements of the system (e.g., hardware, the human, facilities) are not usually considered [Blanchard, 1988]. Therefore, the waterfall model reflects the departmentalised, sequential and document-driven approach used in the beginning of the SE history.

![Figure 2-23: The waterfall model (Source: [Boehm, 1981])](image)

In the mid-1980s, a generic "spiral model" was developed for software-intensive systems. The model continually examines objectives, strategies, design alternatives, and validation methods. System development results through several iterations of this model. Figure 2-24 illustrates a modified version of the original generic approach, evolving from a prototyping model. Note that rapid prototyping is used in each cycle, and that the model emphasises risk analysis. This approach is particularly useful in high-risk development because design sometimes evolves as detailed requirements emerge [Blanchard, 1998]. NASA uses this model to describe its system development life cycle. The spiral model also reflects the evolutionary nature of systems development.

In the early 1990s, the "vee model" was introduced and reflects a top-down and bottom-up approach to system development [Forsberg & Mooz, 1991]. In Figure 2-25, the left side of the 'vee' represents the...
evolution of user requirements into preliminary and detail design, and the right side represents the integration and verification of system components through subsystem and system testing. Ford Motor Company uses the 'vee' to describe a vehicle development cycle. Ford calls the left side of the 'vee' as 'designing to requirements' and the right side as 'verifying against requirements'.

Figure 2-26 represents an extension of the 'vee' model concept. Of particular note is the interface between the system and the software subsystem. Although the software may be predominant within the structure of a system, it is not a system per se. It does not fulfil a purpose of the system by itself. Figure 2-26 emphasises the fact that there are system engineering activities that lead into the software development process.

The models presented here are only a few among the numerous models developed over the last decades. These are the most used ones.

2.5.5 Systems engineering standards and processes

The history of SE standards reflects the level of interest in SE. Figure 2-27 shows the profusion of standards and capability maturity models (CMM) that were developed in the 1990s, during the SE revival. Also, the increased complexity of commercial products made SE to become of more general interest instead of the traditional military driven interest. SE standards started to be developed by professional organisations (e.g. EIA, IEEE, INCOSE) and international standardisation bodies (e.g. ISO). Another aspect reflected in Figure 2-27 is the influence of software engineering on the development of SE. For
example, the capability maturity models (CMM) developed for SE originated from the software engineering CMM. Also the ISO/IEC 15288 on systems life cycle processes will be a compilation of the

**Figure 2-26: The systems versus software boundary** (Source: [Downward, 1991])

**Legend:**  
- derived from  
- influenced by  
* not released

Source organisations and names of the standards:
- USAF 375-5: US Air Force systems engineering handbook
- MSL-STD-499: US DoD Engineering management
- MSL-STD-499A: US DoD Engineering management
- MSL-STD-499B: US DoD Systems engineering
- USA, FM770-78: US Army Field Manual, Systems Engineering
- EIA/JS 632: Electronic Industries Association - Interim Standard - Processes for engineering a system
- IEEE-P-1220: Institute of Electrical and Electronics Engineers

**Figure 2-27: Overview of SE standards evolution** (based on [Sheard & Lake, 1998], [Brill, 1998])

ISO/IEC 12207 (on software life cycle processes) together with the currently most accepted standards, EIA-632 and IEEE-P-1220/1994.
Capability models provide a way to evaluate an enterprise's or project's SE capability. They provide a multi-point scale, in each of these cases containing six levels, showing a road map along which an organisation's process improvement effort can progress. The development of SE capability maturity models is being led by INCOSE in order to address the assessment of SE capability. These efforts are moving towards the unification of existing CMMs into the EIA/IS 731 standard, to a unified software and systems engineering CMM and to an IPD CMM, recognising the broader scope of SE is assuming. Details on CMMs can be found in [Paulk et al., 1993], [Bate et al., 1995] and [INCOSE-SECAM, 1996].

U.S military standards originally supported contracts, to aid the government in delivery of quality products or to ensure utilisation of consistent processes by contractors. In general, commercial standards are not imposed on contracts. In the case of commercial SE standards, use by an enterprise is ordinarily voluntary, as is imposing a standard on a supplier as a basis for competition or performance. The most influential standards to current SE activities are the MIL-STD-499B (never released), the EIA-632 and the IEEE-P-1220. The EIA/IS 632 standard released in 1994 is a commercialised version of the MIL­STD-499B developed by the Electronic Industries Alliance (EIA) working group composed of representatives from the Aircraft Industry Association, Department of Defense, National Security Industries. That was done with the understanding that considerably more industry input would go into a replacement version, to be called EIA 632. In parallel, the IEEE also released in February 1995 a commercial "trial-use" standard IEEE 1220.

The EIA-632 standard describes the processes to engineering a system. These processes are depicted in Figure 2-28. The acquirer-supplier agreement process establishes the technical provisions for an agreement between acquirer and supplier. The planning process plans the technical effort necessary to ensure the integrated development of end products, including definition of the control mechanisms for measuring and assessing technical performance.}

![Figure 2-28: The processes for engineering a system](Source: [EIA, 1997])
performance and progress. The system design progress collects and validates the stakeholder requirements. It converts these, first, to system technical requirements and then, to detailed technical requirements. It formulates and establishes a design that effectively satisfies these requirements. The control process captures work products from the technical activities. Monitor technical progress relative to planned technical activities. Initiate corrective actions, as appropriate. The system verification & validation process verifies the design definitions (both product and process) to ensure the specified requirements have been met. It verifies that development of enabling products for associated processes is progressing satisfactorily. It validates the products against the stakeholder needs and expectations to ensure stakeholder satisfaction.

The IEEE-1220-1994 defines the interdisciplinary tasks that are required throughout a system's life cycle to transform customer needs, requirements, and constraints into a system solution [IEEE, 1995]. Figure 2-29 illustrates the SE process (SEP) as described by the standard. Figure 2-30 describes each of those processes in greater detail.

Figure 2-29: The systems engineering process (Source: [IEEE, 1995])
### SEP’s Element

<table>
<thead>
<tr>
<th>Requirements Analysis</th>
<th>Aim (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>To establish:</td>
<td></td>
</tr>
<tr>
<td>• what the system must be capable of accomplishing;</td>
<td></td>
</tr>
<tr>
<td>• how well system products must perform in quantitative, measurable terms;</td>
<td></td>
</tr>
<tr>
<td>• the environments in which system products must operate; and</td>
<td></td>
</tr>
<tr>
<td>• constraints that will affect design solutions.</td>
<td></td>
</tr>
</tbody>
</table>

### Requirements Validation

<table>
<thead>
<tr>
<th>Requirements Validation</th>
<th>Aim (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• To guide the remaining activities of the SEP and definition of the problem that must be solved.</td>
<td></td>
</tr>
<tr>
<td>• To describe:</td>
<td></td>
</tr>
<tr>
<td>• how the system products will serve their users (operational view);</td>
<td></td>
</tr>
<tr>
<td>• what the system products must do to produce the desired behaviour described in the operational view and of the methodology used to develop the view and decision rationale (functional view);</td>
<td></td>
</tr>
<tr>
<td>• the physical considerations of the system products development and establishment of requirements for technology and for physical interfaces among operators and equipment (physical view).</td>
<td></td>
</tr>
</tbody>
</table>

### Baseline Functional Analysis

<table>
<thead>
<tr>
<th>Functional Analysis</th>
<th>Aim (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>To translate the validated requirements baseline into a functional architecture without consideration for a design solution;</td>
<td></td>
</tr>
<tr>
<td>(Groups of subfunctions generated during functional analysis set the criteria that will guide the definition of the product and subsystem solutions during synthesis)</td>
<td></td>
</tr>
</tbody>
</table>

### Baseline Functional Architecture

<table>
<thead>
<tr>
<th>Functional Architecture</th>
<th>Aim (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>To describe the functional arrangements and sequencing of subfunctions resulting from decomposing (breaking down) the set of system functions to their subfunctions.</td>
<td></td>
</tr>
</tbody>
</table>

### Baseline Functional Verification

<table>
<thead>
<tr>
<th>Functional Verification</th>
<th>Aim (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• To assess the completeness of the functional architecture in satisfying the validated requirements baseline;</td>
<td></td>
</tr>
<tr>
<td>• To produce a verified functional architecture for input to synthesis</td>
<td></td>
</tr>
</tbody>
</table>

### Physical Architecture

<table>
<thead>
<tr>
<th>Physical Architecture</th>
<th>Aim (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• To define system product solutions;</td>
<td></td>
</tr>
<tr>
<td>• To identify subsystems to satisfy the requirements of the verified functional architecture;</td>
<td></td>
</tr>
<tr>
<td>• To evaluate the requirements baseline to ensure that it represents identified customer expectations and project, enterprise, and external constraints;</td>
<td></td>
</tr>
<tr>
<td>• To assess the requirements baseline to determine whether the full spectrum of possible system operations and system life cycle support concepts has been adequately addressed.</td>
<td></td>
</tr>
</tbody>
</table>

### Physical Verification

<table>
<thead>
<tr>
<th>Physical Verification</th>
<th>Aim (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• To assess that the requirements of the lowest level of the physical architecture, including derived requirements, are traceable to the verified functional architecture;</td>
<td></td>
</tr>
<tr>
<td>• To assure that the physical architecture requirements satisfy the validated requirements baseline.</td>
<td></td>
</tr>
</tbody>
</table>

### Systems Analysis

<table>
<thead>
<tr>
<th>Systems Analysis</th>
<th>Aim (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• To resolve conflicts identified during requirements analysis;</td>
<td></td>
</tr>
<tr>
<td>• To decompose functional requirements;</td>
<td></td>
</tr>
<tr>
<td>• To allocate performance requirements during functional analysis;</td>
<td></td>
</tr>
<tr>
<td>• To evaluate the effectiveness of alternative physical element solutions and select the best physical solution during synthesis;</td>
<td></td>
</tr>
<tr>
<td>• To assess system effectiveness;</td>
<td></td>
</tr>
<tr>
<td>• To manage risk factors throughout the systems engineering effort.</td>
<td></td>
</tr>
</tbody>
</table>

### Control

<table>
<thead>
<tr>
<th>Control</th>
<th>Aim (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>To provide:</td>
<td></td>
</tr>
<tr>
<td>• a complete and up-to-date picture of SEP activities and results that are used in accomplishing other subprocesses activities;</td>
<td></td>
</tr>
<tr>
<td>• planning for and inputs to future applications of the systems engineering process;</td>
<td></td>
</tr>
<tr>
<td>• information for production, test, and customer support;</td>
<td></td>
</tr>
<tr>
<td>• information for decision makers at technical and project reviews.</td>
<td></td>
</tr>
</tbody>
</table>

---

**Figure 2-30: Description of the systems engineering processes** (Source: [IEEE, 1995])

Comparing the IEEE-1220 with the EIA-632 provides indication that the EIA-632 standard has a broader process scope than the IEEE-1220 because it considers acquisition processes as part of the engineering process. The bulk of the IEEE-1220 standard is a detailed description of the EIA-632's system design process. Whereas the EIA-632 provides guidelines for performing such a process, the IEEE-1220 adopts a more prescriptive approach. In summary, the IEEE-1220 goes to a greater level of detail than the EIA-632. This, on the other hand, provides a broader process scope for the engineering activity.

The SEP described in the IEEE-1220 reflects very much a generic problem-solving approach from the engineering design community which includes the following steps: planning and clarifying the task, conceptual design, embodiment design and detail design [Pahl & Beitz, 1996].
[Caple, 1997] proposes a systems engineering meta-model, called G.U.S.E.M., the (Caple) Generic Unified Systems Engineering Model. Caple defines the SEP as 'not a single process, but a product & process pair structure, which is passed from project phase to project phase and operated on by everybody's favourite project phase process [P.E.R.T.'d] in order to: minimise risk and achieve product completion'. Caple argues that the IEEE-1220 and the EIA-632 standards do not define a practical or useable model corresponding to his definition of the SEP. Caple, then, proposes G.U.S.E.M., a model composed of three domains: product, process and time domains. In the product domain, the knowledge of everything from design requirement specifications to trial plans and actual hardware may be found. It is implemented through a product database containing all input and output products of all processes carried out to develop, manufacture and test the product. The process/activity domain contains all the knowledge on 'how to do it', i.e., how to perform the development, manufacturing and testing processes. It is implemented through analysis methods such as Soft Systems Methodology [Checkland & Scholes, 1990] and Systemigrams [Boardman, 1994]. The time domain is where the project is run, managed, resourced and executed. It is implemented through the use of P.E.R.T.-type tools with a time vector. The P.E.R.T. contains the results of the process analysis.

2.5.6 Systems engineering methods

SE methods had their origin in the software engineering community in the late 1970s when software complexity had reached a level where it was very expensive to test, correct and maintain the software code. Software grew in complexity mainly because the approach to meet user requirements was dealt with at the physical or code level. When requirements changed and the software needed modification, as the code grew in size, difficulties arose in the following areas: traceability to requirements, visibility of software functionality, communication among software designers, documentation (redundant, wordy, physical, tedious to read and unbearable to write). It was necessary to adopt a more structured approach to software development with more emphasis on the analysis phase than was given before. Analysis is the phase in software development when the user requirements are identified and analysed in order to generate a list of functions in the software product that meet those requirements. DeMarco (1978), one of the pioneers of structured analysis, addressing the problems mentioned above, proposed a structured analysis method with the following characteristics:

- high maintainability of the products of analysis;
- effective method of partitioning to deal with size problems;
- extensive use of graphics;
- differentiation between logical and physical considerations;
- existence of a logical model so that the user can gain familiarity with system characteristics before implementation;
- requirements partitioning and partitioning documentation before specification (tool proposed DFD);
- interfaces traceability and evaluation before becoming unduly physical (tool proposed: Data Dictionary);
- use of tools (other than narrative text) for description of logic and policy (tools proposed: structured english, decision tables and decision trees).
These characteristics are the basis for the modern systems structured analysis and design methods developed since the late 1970s. Modern software engineering tools tend to provide a collection of notations covering the whole software development life cycle: requirements, analysis, design and implementation. Traceability is also provided among the elements in each phase of the life cycle. As the software development phases correspond to a typical SE process (see Figure 2-29) these tools are used for general systems development.

Addressing the reuse issue, in order to increase productivity in software development, object oriented methodologies were developed in the 1980s. Object orientation is based on the encapsulation concept initially defined by Parnas in the early 1970s. This approach considers all resources, data and modules as objects. Each object encapsulates its data and data manipulation procedures (functionality). Using this concept, the designer can create his own abstractions to solve the problem. The object-oriented development is typically bottom-up. A trade of software components would thus be possible, facilitating reuse. Another concept introduced by object orientation is ‘inheritance’. Object instances are grouped in classes in such a way that attributes and functionality of the class is inherited by its object instances. The core of object-oriented methods (e.g. Booch, OMT) is an object model consisting of an entity-relationship diagram showing classes and their relationships. Therefore, for modelling very large systems the object model can potentially have many classes and loose visibility. For this reason, object model has been used essentially as a software design methodology instead of a system architecture development tool. Also, object oriented methods did not use to include a requirements phase in the development process. The development process started from an analysis phase. More recently, methods using the UML (Unified Modelling Language) notation, started to include a requirements phase for software development by the use of Use Case Models and a hierarchy of models.

The trend of migrating from implementation-only methods to methods supporting the earlier stages of the software/system life cycle can be seen in Figure 2-31. The methods shown in Figure 2-31 and others are reviewed in [Calvez, 1993]. Figure 2-31 shows that more recent methods seek to cover the entirety of the software/system development life cycle. Some of these methods (SADT, Yourdon, HPM and UML based method) are reviewed in Appendix A.

2.5.7 Systems engineering tools

Systems engineering tools are computer tools that support the systems engineering process. Computer aided software engineering (CASE) tools implement the methods mentioned in Section 2.5.6. Modern tools tend to cover the earlier stages or the entirety of the software development life cycle and are being used for general systems development. Types of tools available are:

- requirement management workbenches: support the input, tracking, and all other management of project requirements, including structuring users’ original requirements into system requirements. A project requirement database developed by the tools can be regarded as a system model itself. It is however a static model. Other static and dynamic models of a system/project can be produced by the use of system modelling tools [Williams & Allan, 1995].
system modelling environments: build up an executable model to study the feasibility of design and to highlight dynamic design errors. Typically, functions are defined and assigned to system components in the system modelling to meet the requirements specified in the requirements database. Allocation of system resources can also be carried out by the execution of the model. The system dynamic model can either be produced on the basis of the static model of a system or created directly by using general purpose system thinking packages such as Stella [Williams & Allan, 1995]. The system modelling environments may be classified into behavioural and object oriented models.

systems engineering environments: provide full life cycle support from requirements capture through requirements analysis, functional analysis and synthesis phases including control functions such as configuration management, traceability, document generation, performance analysis [3SL, 1998].

Examples of commercial requirements management workbenches are:

- RTM (Requirements Traceability Management), was initially developed in 1992 for internal use within Marconi Systems Technology [Marconi, 1992]. It is a requirements traceability and management workbench centred around a database management system, and have tools to document, parse, organise, edit, interlink, change, and manage requirements. RTM does not support hierarchical requirements definition. Its distinguishing characteristic is the ease of creating traceability matrices between user requirements and system requirements. [Rader & Haggerty, 1995]

- DOORS (Dynamic Object Oriented Requirements System) [QSS, 1994b] is a software tool and a networked, multi-project, multi-user requirements management system. The prime functions of DOORS are: creation of hierarchical requirements sets; traceability between a structured requirements set and external descriptive documents; traceability between structured requirements sets; linkage between a structured requirements set and an external structured tool (e.g. a CASE tool); sorting, selection and processing of requirements.

- CORE (Controlled Requirements Expression) [BAe, 1990] is a graphical method developed by British Aerospace (Warton) and System Designers in the late 1970's. It aims to: provide techniques
for requirements capture, analysis and specification; capture operational (customer) requirements, functional requirements and constraints and implementation requirements and constraints; analyse the captured requirements to ensure completeness. CORE modularises the problem into hierarchical structure through the use of an abstract concept called viewpoints. The viewpoints are used to view the system in a number of different ways enabling the analyst to capture information. The procedure consists of: select functional and non-functional viewpoints; structure functional viewpoints into a hierarchy; specify non-functional requirements with structured text; identify scenarios for each functional viewpoint; analyse each functional viewpoint in isolation from each other in terms of starting and stopping conditions; identify sequence or concurrency among functional viewpoints, obtaining threads; combine all threads into an operational diagram; assess the impact of non-functional requirements on functional requirements; compile an specification (viewpoint structure diagram, combined operational diagram, combined threads, node notes).

Examples of system modelling environments are:

- **SLATE** [TD Technologies, 1995], System Level Automation Tool for Engineers [Fararoy et al., 1995], was initially developed for internal use within Texas Instruments, and was commercialised in 1994. It is a dynamic, multi-user, information system for organising and managing the entire product life cycle -- from initial definition through design, development, manufacturing, deployment, maintenance, and eventual disposal. It is a UNIX-based client/server application. The basic building block in SLATE is known as Abstraction Blocks (ABs), which can be any object (a sensor, a train, or a track circuit). The user begins building hierarchies using ABs. The system allows the user to look up these ABs hierarchically and functionally and view them from many different perspectives, such as organisation, cost, reliability. Each of these different views can be linked to show interrelationships between the different views so changes or impacts in one view can be seen in other views. SLATE also provides a flexible graphical report generation capability that allows the user to get the information in the database.

- **RDD-I00** [Ascent Logic, 1995] traces its heritage to SREM (Software Requirements Engineering Methodology) and DCDS (Distributed Computing Design System) [Alford, 1985]. It was initially developed in 1987 and is a workstation based, interactive tool. RDD-100 is used by multi-disciplinary teams to analyse, specify, track, verify, document and manage large complex enterprises and development projects. The different software products comprising the RDD-100 Product Family are designed to help you with all phases of system engineering. These products include: System Designer, System Analyst, Requirements Manager, Requirements Editor, System Browser, Dynamic Verification Facility (DFV) Option, Interleaf Interface Option, Cadre TeamWork Interface Bridge Option. RDD 100 uses IDEF0, DFD, STD, FFBD, N2, SSL and structure diagrams.

- **TeamWork** [Cadre, 1994] is a CASE tool developed by Cadre. It implements an enhanced Yourdon notation including DFD, STD, Entity Relationship Diagrams, Process Activation Tables, Decision Tables, Process Specifications, Control Specifications and Data Dictionary. It has got a linkage tool in order to reuse existing models in other projects.

- **Statemate** [i-Logix, 1991] The statemate model is a graphic model based on co-operating sequential processes, extended abstract state machines, predicate logic and data flow. The model contains templates, Statecharts and Activity Diagrams. Templates are completed for the following objects:
state, condition, event, action, activity (function), signals/variables, modules, and channels (which connects modules). A Statechart is a visual extension to conventional STD. An Activity Diagram shows data and control flow and is similar to a DFD. The Statechart is a major contribution of Statemate. A number of STD can be shown on the same chart, displaying parallelism, selection, and decomposition. Statemate can model cooperating sequential processes or support structured analysis methods.


- **X Math** [Integrated Systems, Inc., 1994] provide capability for control systems design, modelling and simulation through: solving control problems using a number of different approaches; analysis and synthesis of robust control systems; system identification, model reduction and signal analysis; interactive design of continuous-time single input single output linear time-invariant controllers. Consists of four modules: control design module, robust control module, interactive system identification module, interactive control design module.

An example of a systems engineering environment is **Cradle** [3SL, 1998] described in Appendix B. Appendix C contains a brief description of some notations implemented by the above tools: DFD (Data Flow Diagram), STD (State Transition Diagrams), ER or ERA model (Entity Relationship or Entity-Relationship-Attribute model), FFBD (Functional Flow Block Diagrams), Structure diagrams, N2 chart and UML notation. Appendix A contains a description of the IDEF0 notation under the description of SADT.

### 2.5.8 SEDRES

SEDRES stands for Systems Engineering Data Representation and Exchange Standardisation [Johnson, 1998]. It is a three-year European Commission-funded ESPRIT project, running 1996-1998, which was initiated by Aerospatiale, Alenia, British Aerospace, DASA and Saab. Also contributing are the Universities of Loughborough, Linkoping, and the Australian Centre for Test & Evaluation.

Integrated product development frequently involves multi-company and multi-national teams, using heterogeneous design tool sets. SE must be able to operate in this environment. SEDRES is producing a neutral data exchange standard based on STEP (ISO-10303), that will embrace product definition aspects crucial to successful SE: product requirements, systems architectures, product functionality, allocation, traceability and configuration management information. The standard will enable SE tools to exchange such information and should be applicable to many industries.

Figure 2-32 illustrates the four areas in which such exchange capability gives benefits to the design process:

![Figure 2-32: Data exchange vision](Source: [Johnson, 1998])

1. Facilitating the process of moving design data from one design tool, to another, as an emerging design moves through tools used in the design process;

2. Subsequently opening up the possibility to store the design data in a tool-independent way, rather than in the tool proprietary formats that are currently used;
3. Enabling the possibility of performing checks and analyses on the emerging designs that previously were impossible and/or very labour intensive, with design data across two or more design tools;

4. Extending the ways that the design can be ‘viewed’, again using design data from multiple design tools.

The focus of the envisaged standard is primarily on design data aspects including product functionality, behaviour and requirements, extending to product environment (context) definition and physical interaction and control, for instance, between sub-systems. The standard will extend into project management data to address issues such as configuration management and change control.

2.5.9 New technologies

New technologies which can be potentially used for systems engineering include the following:

- **Fuzzy logic** ([Adeli & Hung, 1995] and [Shimizu & Jindo, 1995]): makes possible to introduce probabilistic evaluation of the presence of a requirement in the set of requirements which define customer satisfaction whenever there is an uncertainty.

- **Multiple regression analysis** [Shimizu & Jindo, 1995]: provides the relationships between multiple explanatory variables (e.g., systems requirements) and target variables (e.g., user requirements).

- **Fuzzy regression analysis** [Shimizu & Jindo, 1995]: useful when there is ambiguity in the system structure that expresses the relationships between inputs and outputs.

- **Genetic algorithms** [Santillan & Wright, 1995]: provides a way of component selection using requirements fulfilment as the selection criteria.

- **Knowledge based systems** [Koh, 1989]: expresses requirements and its relationships as events, procedures, rules and logic.

- **Neural networks** [Ishihara et al., 1995]: allows creating a stable self-organising system in a very dynamic environment.

- **Kansei engineering** [Nagamachi, 1995]: aims to grasp consumer’s feeling (Kansei) about the product in terms of ergonomic and psychological estimation; identify design characteristics of the product from the consumer’s Kansei; building Kansei engineering as an ergonomic technology; adjust product design to the current societal change or people's preference trend. It uses semantic differentials [Snider & Osgood, 1969] as a main technique to grasp consumer’s Kansei. 600 to 800 customer’s feeling words are collected from selling shops and industry magazines, and 100 out of them are selected. Surveys or experiments are carried out to look at the relations between Kansei words and design elements. Artificial intelligence, neural network, genetic algorithms and fuzzy logic are utilised to build a systematic framework. Kansei engineering databases are updated every three or four years. Examples of use are provided by Mazda.

Having comprehensively reviewed approaches to manage complexity in the development of complex products, the thesis continues in Chapter 3 analysing the trends in product development in two major industries of complex products – the space and the automotive industries. These first four chapters of the thesis provide the needs that justify subsequent development.
Chapter 3
Trends in the space and automotive industries

This chapter aims:

- to review the characteristics and trends in two industries of complex products: the space and the automotive;
- to compare and contrast the approaches used by those industries;
- to draw some conclusions.

3.1 Introduction

The space industry is characterised by the space product complexity and by the adoption of systems engineering as an approach to develop such complex space products. Complexity is also increasing in the automotive industry not only because of its product complexity but also because of the need to meet increasingly stringent regulatory, customer and corporate requirements. These requirements may affect not only the automobile but also its manufacturing process and the company organisation. It is proposed in this chapter that the approaches to product development in both industries are converging to a common approach with a broader scope. An illustration of this, is the fact that the concept of system engineering used at Ford Motor Company has its origins at Boeing. Systems engineering is how Boeing is able to sell their first plane (they have no full engineering prototypes on programs newer than the 777) and is how they have confidence in the functional performance capability the first time that first plane takes off.

This chapter, together with Chapters 1, 2 and 4, aims to identify the needs in the complex product development arena.

The information presented in this chapter is a result of literature review and actual experience in the space and automotive industry. The author's background is related to satellite development as he worked during the period 1988-1994 at INPE (the Brazilian Institute for Space Research), having dealt with many stages of a satellite life cycle, such as: manufacturing, assembly, integration and testing. The author also spent four months (July-September, 1995 and April, 1997) in a major automotive company developing the case studies documented in Chapter 4.

Section 3.2 reviews the characteristics and trends in the space industry. Section 3.3 reviews the characteristics and trends in the automotive industry. Section 3.4 compares both industries. Section 3.5 draws some conclusions.

3.2 Space industry

The space industry started in the 1950s with artificial satellites [INCOSE, 1996] and had great boost with the Apollo program to take man to the moon during the 1960s [Parth, 1998]. From there on, the
utilisation of space products have increased dramatically including applications such as: weather, environment and ecology, navigation and localisation, risk management, mobile telecommunications, multi-media, broadcasting, cartography, hydrology, agriculture, urban development, coastal zone, space transportation, space exploration, earth observation, infrastructure for man in space [Atzei et al., 1998].

In addition to the space product functional requirements, a space product must meet the operational conditions in space, with varying constraints of temperature, vibration, humidity, and shock with reliabilities in excess of 99% [Parth, 1998]. As a result of such stringent set of requirements, the space product was born as a complex product already: many functions, many subsystems, much integration effort required, many disciplines involved [Ruth, 1994]. For example, satellite development and operation includes disciplines such as onboard data processing, orbitography, thermal control, attitude control, structure, electrical power supply, vibration, electromagnetic compatibility and interference, telecommunications.[Gory & Keller, 1993]

Once the space product cost too much (10s to 100s million dollars) and is virtually non-repairable [Ruth, 1994], a risk avoidance philosophy [Atzei & Novara, 1997] became characteristic of space mission development. This risk avoidance philosophy required a systematic formal approach in order to guarantee full traceability from requirements and plans to parts and execution. Such approach is systems engineering [INCOSE, 1996a].

In the following, Section 3.2.1 describes some characteristics of the space industry and Section 3.2.2 describes the trends in the space industry. Systems engineering concepts adopted by NASA (the American Space Agency) are described in Section 3.2.3. NASA was chosen for being ranked top as having system engineering capability maturity (SE-CMM) according to the SE-CMM model developed in the USA by the Enterprise Process Improvement Collaboration (EPIC) consortium (Hughes, Lockheed, Texas Instruments, SEI and the Software Productivity Consortium) [Bate et al., 1995]. The modern 'faster, better, cheaper' philosophy being used by NASA is presented in Section 3.2.4.

3.2.1 Characteristics

The space industry has been traditionally characterised by the following aspects:

- **government as main stakeholder.** Examples of such government organisations are: DoD, NASA, DERA, MoD. After World War II and the war in Korea, the U.S.A space program began a huge technology push in parallel with that created by the Vietnam conflict. Unprecedented amounts of government money were spent on large, complex, risky projects. Private industry could never have done the level of development that was demanded by the Apollo program, the space shuttle program or the international space station program. However, with the advent of mobile telecommunications, there is an increasing interest of global private companies in the space industry [Parth, 1998].

- **emphasis on performance requirements and risk avoidance.** Space products must operate under widely varying constraints of temperature, vibration, humidity, and shock to reliabilities typically in excess of 99%+. Since the government puts a higher emphasis on a perfectly workable system than it does on making a profit, there is much greater emphasis on examining development issues as thoroughly and as completely as possible regardless of cost. While schedule impacts are sometimes a
concern, the times the government gets the most worried is when a space launch window is threatened (due to the high dollar cost and significant schedule impact). When technological risk is high, the dollar cost is high, and/or human life depends on the proper operation of the system, little expense or schedule is spared to do a thorough engineering job. There could be no compromise to knowing that they had thought of the best achievable design solutions to very difficult requirements [Atzei & Novara, 1997] [Parth, 1998].

- **wide range of applications.** The space industry covers a very broad scope of applications including weather forecast, mobile communications, launching, space exploration as illustrated in Figure 3-1 [INCOSE, 1996a] [Atzei et al., 1998].

- **many disciplines involved.** Figure 3-1 lists the many technologies under research and development in order to cover the broad range of applications of space industry products. The multidisciplinarity, however, is not a recent trend. It has always been a characteristic of space product development. For example, satellite development and operation include disciplines such as onboard data processing, orbitography, thermal control, attitude control, structure, electrical power supply, vibration, electromagnetic compatibility and interference, telecommunications. [Atzei et al., 1998] [Gory & Keller, 1993]

<table>
<thead>
<tr>
<th>SPACE SERVICE/DOMAIN</th>
<th>PROGRAMMATICS</th>
<th>DRIVERS FOR TECHNOLOGY RESEARCH &amp; DEVELOPMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PUBLIC SERVICES</strong></td>
<td>Guaranteed continuity of services</td>
<td>Wind lidars</td>
</tr>
<tr>
<td>Weather</td>
<td></td>
<td>Cloud radar</td>
</tr>
<tr>
<td>Environment and ecology</td>
<td>Public funding of infrastructure</td>
<td>Advanced ground computation, simulation and data fusion</td>
</tr>
<tr>
<td>Navigation, localisation</td>
<td></td>
<td>High accuracy timing</td>
</tr>
<tr>
<td>Major risk management</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Telemedicine</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>COMMERCIAL SERVICES</strong></td>
<td>Short development (&lt;2 years)</td>
<td>Autonomy and lifetime</td>
</tr>
<tr>
<td>Mobile telecon</td>
<td>Continuity of service</td>
<td>On-board processing &amp; switching</td>
</tr>
<tr>
<td>Multi-media</td>
<td>Global private financing</td>
<td>Resources flexibility (beams, power, frequency)</td>
</tr>
<tr>
<td>Broadcasting</td>
<td>High production rate</td>
<td>Use of higher frequencies</td>
</tr>
<tr>
<td>Cartography</td>
<td>Large constellations</td>
<td>Satellite constellations management</td>
</tr>
<tr>
<td>Hydrology</td>
<td></td>
<td>Electric propulsion</td>
</tr>
<tr>
<td>Agriculture</td>
<td></td>
<td>Inter-satellite link</td>
</tr>
<tr>
<td>Urban development</td>
<td></td>
<td>Low debris production</td>
</tr>
<tr>
<td>Coastal zone</td>
<td></td>
<td>Integrated EO-GIS and navigation</td>
</tr>
<tr>
<td>Space transportation:</td>
<td></td>
<td>High resolution SAR</td>
</tr>
<tr>
<td>* improved expandable launchers</td>
<td></td>
<td>Stratospheric platforms</td>
</tr>
<tr>
<td>* future re-usable systems</td>
<td>Long development</td>
<td>Improved cryo-propulsion</td>
</tr>
<tr>
<td>* small launchers</td>
<td>Operational flexibility</td>
<td>Reusability</td>
</tr>
<tr>
<td></td>
<td>Drastic cost reduction</td>
<td>Re-entry environment</td>
</tr>
</tbody>
</table>

*Figure 3-1: Overview of programmatic aspects of the space sector and the major technology requirements/drivers (Source: [Atzei et al., 1998]) continues...*
Figure 3-1 continued

<table>
<thead>
<tr>
<th>SPACE SERVICE/DOMAIN</th>
<th>PROGRAMMATICS</th>
<th>DRIVERS FOR TECHNOLOGY RESEARCH &amp; DEVELOPMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCIENCE/EXPLORATION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td>• Often-one-of-a-kind</td>
<td>• Interferometry</td>
</tr>
<tr>
<td>• Solar system</td>
<td>• 10 years development cycle</td>
<td>• Robotics</td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td>• typical for large missions</td>
<td>• Large, precise/stable optical systems</td>
</tr>
<tr>
<td>• Earth observation</td>
<td>• Public funding</td>
<td>• Ultrasonic cryogenic detectors</td>
</tr>
<tr>
<td>• Microgravity</td>
<td>• International cooperation</td>
<td>• Multi-hyperspectral imagers</td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td>• Radar-altimeter</td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td>• Autonomy/on-board processing</td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td>• Nuclear power</td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td>• Electrical propulsion</td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Astrophysics &amp; astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Solar system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Moon/Mars exploration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Earth observation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Microgravity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Exobiology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Interferometry</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| • Astrophysics & a...
• long space product life cycle. Figure 3-3 illustrates the development phases of a major DoD program. Every space program progresses through the top-level phases. Subphases may or may not be part of a given program. The time required to complete the process varies with the scope of the program. Major programs take as much as 15 years from concept exploration to initial launch, whereas some programs may require as little as 12 to 18 months. [Larson & Wertz, 1992]

<table>
<thead>
<tr>
<th>Phase</th>
<th>Concept exploration</th>
<th>Detailed development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Needs analysis</td>
<td>Concept development</td>
<td>Demonstration and validation</td>
</tr>
<tr>
<td>Production and deployment</td>
<td>Operations and support</td>
<td></td>
</tr>
<tr>
<td>Typical DoD milestones</td>
<td>ADM</td>
<td>SRR</td>
</tr>
<tr>
<td>Typical products</td>
<td>Statement of needs; Studies</td>
<td>Brassboards and studies</td>
</tr>
<tr>
<td>Time required in major program</td>
<td>Continuous</td>
<td>1-2 years</td>
</tr>
</tbody>
</table>

Figure 3-3: Development phases of a major DoD space program (Source: [Larson & Wertz, 1992])

• product development process has high overhead and has strict reporting requirements. Not only is there extensive analysis work required, but it has to be strictly documented and reported. The product development process writes a lot of documentation (e.g. Mission Needs Statement, Systems Engineering Management Plan) and gets involved in the numerous reviews that are part of large, risky product development. The purposes of the documentation requirements in the space industry are to make it as easy as possible for the government to understand and follow what is going on in the project, to audit it, and to prove that they have full control over it. One other reason is the fact that full traceability has to be pursued, due to the fact that any failure in the spacecraft has to be analysed through the documentation available, since the product is virtually not repairable. [Parth, 1998]

• a conventional space program is structured in a high number of sequential phases, and is monitored through a sequence of formal reviews particularly during the Phases B and C/D (see Figure 3-4). The development process of a space system is inherently sequential (mission, space segment, and finally operations). It can be thought of in-terms of a single-parameter optimisation of performance. [Atzei & Novara, 1997]

Figure 3-4: Conventional space program phases (Source: [Atzei & Novara, 1997])

• the systems engineering process carried out on phases B and C/D is traditionally described by the ‘Waterfall Model’ (see Figure 2-23, in Chapter 2). This model is based on a top-down approach for system development and includes steps of initiation, requirements analysis, design, test,
and so on. Often, in its implementation, the steps are viewed as being relatively independent from one another and must be executed in strict sequence. Additionally, the required interfaces with the other elements of the system (e.g., human, facilities, processes) are not usually considered.[Blanchard, 1998]

- **compliance with performance requirements is usual design driver.** The current paradigm for space activities says that low-cost missions are simple (i.e., few requirements) missions, while requirements are generally considered the primary determinant of mission costs. The need for performance drives the need for new technology and for complex systems development and operations. For example, large instrument apertures, high data rates, 3-axis stabilised pointing with sub-arcsecond tolerance drive the design of spacecraft and ground systems more than do simpler instruments, lower control needs, and moderate data rates. Similarly, requirements for real-time monitoring and instrument adjustment, for measurement concurrent with other missions, and for quick reaction to targets of opportunity drive all missions costs. [Atzei & Novara, 1997]

- **impact of requirements on life cycle cost not fully appreciated.** Most of the conventional wisdom of investing more at the beginning of the life cycle in order to reduce costs in production, operations, maintenance and disposal is generally difficult to justify in the space industry using conventional metrics. As the life cycle of large, military and commercial space systems is measured in decades, usually well beyond the ability of individuals to remain in a function to see the consequences of their decisions. This is especially true for the government where mobility is encouraged and for politicians who have 2, 4 and 6 year terms. This, in turn, limits opportunities for feedback to the original product development decisions, which is key for process improvements, in the decision making, risk management, and technical manufacturing. [Atzei & Novara, 1997] [Ruth, 1994]

- **one-of-a-kind to low volume production (generally 1-25 units).** There are a number of satellites that are one-of-a-kind. However, unlike many other high cost one-of-a-kind items, the space industry is only beginning to think about the possibility of a follow-on production contract and the savings that would accrue by including manufacturing issues in the early concept phase. [Ruth, 1994]

- **high reliance on testing.** A typical spacecraft is developed through 3 prototyping phases: engineering, qualification and flight models. During the engineering model the functional concept of the spacecraft is proven through traditional electronics, mechanical engineering functional tests. The qualification model is an identical copy of the flight model but it is developed to assure that the spacecraft complies with the vibration, thermal, structural, electromagnetic interference and compatibility. Tests are performed in large test facilities with controlled humidity and temperature environment. [Atzei & Novara, 1997]

- **optimisation carried out at subsystem level only.** The traditional breakdown of space systems into subsystems was first applied by NASA at the beginning of the 1960s and corresponded to the state of technology at that time. Each subsystem is made of a set of onboard/ground hardware/software constituent equipments and participates in the optimisation of a basic onboard or ground function
(onboard data processing, orbitography, thermal control, attitude control, structure, electrical power supply etc.). [Gory & Keller, 1993], [Atzei & Novara, 1997]

• time shift between system and subsystem level configuration and requirements. First system requirements and a system outline are compiled, then subsystem requirements and subsystem trade-off analysis and design are performed (see Figure 3-5). [Atzei & Novara, 1997]

• 'cost reduction exercises' fairly common outcome. Cost requirements are not considered from the outset. After system design review and reconfiguration, cost estimates are performed and the development process reiterates, re-evaluating the initial system requirements. (see Figure 3-5) [Atzei & Novara, 1997]

3.2.2 Trends

Major change drivers in space systems development are [Parker & Braddock, 1994]:

• shrinking government budgets for science and technology and as a consequence decreasing public investments in space;
• increasing commercial utilisation of space services such as mobile telecommunications and space transportation;
• computing and information technology push.

Given the change drivers above, the Systems Studies Division of the European Space Agency's Technology Centre (ESA/ESTEC) for science-oriented missions states that success requires [Atzei & Novara, 1997]:

• a reduction of a factor 2 to 3 in life cycle costs;
• reduction from 6 to 3 years in development schedule (preliminary and detailed definition, production/ground qualification & testing);
• a factor 2 to 3 increase in launch rate.

The French Space Agency (CNES) states that, in the space industry, the length of time between the date on which a design began and the date the system was no longer produced decreased from 12 to 6 years [Gory & Keller, 1993].

As publicly funded support to science declines, current space systems engineering faces some challenges, including the following [Atzei & Novara, 1997][INCOSE, 1996a]:

• satellite constellations where the satellites may be implicitly related by performing co-ordinated mission or more explicitly by communicating with each other. The benefits of such system architecture would be: the workload in the mission centre would be reduced; the constellation would be more robust than a single, large satellite; the constellation would be more flexible; the constellation could be more reactive.
• the integration of commercial-off-the-shelf (COTS) subsystems to reduce cost and time to develop space segment and ground support systems.
• the development of well-integrated, robust, low cost low Earth orbit systems with appropriate allocation of functions among onboard, other spacecraft, and ground support systems.
• the increase of autonomy in onboard and ground support system operations since operations is a major component of mission life cycle costs.
• the implementation of micro-technologies to make satellites smaller and therefore financially and commercially more attractive.
• various launch options need to be considered, if LCC reduction has to be achieved: batch launch on larger launcher, one-off launch of replacements on small launcher, insurance costs of satellite and launch, Re-usable Launch Vehicle (RLV)

Given the traditional aspects in the space industry described in Section 3.2.1, Figure 3-6 lists the trends in that industry in order to adapt to the change drivers listed above.

### 3.2.3 SE concepts by NASA

NASA defines a system as a set of interrelated components which interact with one another in an organised fashion towards a common purpose. The components of the system may be quite diverse, consisting of personnel, procedures, software, equipment, and/or facilities. Every system exists in the context of a supersystem, which has a broader scope. Managers in the supersystem set system policies, establish objectives, determine system constraints, and define what costs are relevant.

The NASA management instruction for the acquisition of major systems (NMI 7100.14B) defines a program as “an organised set of activities directed toward a common purpose, objective, or goal undertaken or proposed by an agency in order to carry out responsibilities which have been assigned to it.” The similarity to the above definition of system is not accidental.
<table>
<thead>
<tr>
<th>TRADITIONALLY</th>
<th>TREND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Government as main customer</td>
<td>Increasing partnership with commercial organisations, other space agencies and academia</td>
</tr>
</tbody>
</table>
| Emphasis on performance requirements and risk avoidance | • Trade-off among performance, risk, cost and schedule  
• Risk management                                      |
| Complex space products                           | • Smaller and more functionally dedicated space systems;  
• Increased autonomy of space segment in order to reduce operations cost;  
• Functional partitioning among on board subsystems, other spacecrafts and ground segment being considered;  
• Reduction in the number of equipment items          |
| New product emphasis                              | • Re-use of proven concepts and technologies;  
• Increasing use of commercial off the shelf solutions.  
• Synergies with military applications                |
| High overhead and reporting requirements          | • Documentation automation;  
• Reuse of documentation (e.g. plans) from previous programs |
| High number of sequential phases and formal reviews | • System design/ integration/ verification done in a more concurrent and integrated basis;  
• Trend to analyse problems and identifying potential solutions prior to review meetings  
• Off-line technology development                     |
| Uses ‘waterfall model’ for systems engineering process | ‘Spiral model’ for systems engineering process |
| Life cycle requirements                           | Production, operations and disposal requirements being considered from the outset |
| One-of-a-kind or low volume production            | • Standardisation and modularity;  
• Economies of scale through the production in series of standard spacecraft components, including structures.  
• Centralised batch procurement across several projects. |
| High reliance on testing                          | • Rapid prototyping methods for specification and design;  
• Use of commercial, aeronautical, military off-the-shelf (COTS) equipment and components  
• More reliance on analysis instead of test  
• Ship & shoot concepts                               |
| Optimisation carried out at subsystem level only  | Risk management, life cycle cost minimisation and shorter time to launch requires overall system analysis and optimisation |
| Time shift between system and subsystem levels configuration and requirements | System requirements drive preliminary subsystem models, subsystem cost and engineering models are developed and then integrated to compose system model. System optimisation and review takes place form this integrated system model. |
| Cost reduction exercises                          | Life cycle cost considered from the outset |
| Classical matrix organisation                     | • Teams of specialists become multi-functional teams  
• Multi functional teams may include people from different organisations including government, prime and sub-contractors and academia  
• Integrated product teams                            |

Figure 3-6: Trends in the space industry (Source: based on [Atzei & Novara, 1997])

In the NASA context, a project encompasses the design, acquisition, and operation of a major system, and is generally managed by a NASA field centre. A program, on the other hand, is what NASA Headquarters manages, and encompasses not only the engineering of the system, but all other activities required to achieve the desired end. The term mission is often used for the system's purpose.

Most NASA systems are sufficiently complex so that their components are subsystems, which must function in a co-ordinated way for the system to accomplish its goals. Each subsystem is a system in its own right. Spacecraft systems often have such subsystems as propulsion, attitude control, telecommunications, and power.
NASA considers systems engineering as a robust approach to the design and creation of systems to accomplish desired needs. In simple terms the approach consists of identification and quantification of system goals, creation of alternative system design concepts, performance of design trades, selection and implementation of the best design, and post-implementation assessment of how well the system meets (or met) the goals. The approach is often applied recursively, with several increases in the resolution of the system baselines (requirements definition, design details, verification procedures and standards, cost and performance estimates, and so on). System engineering is performed in concert with system management. Formally, NASA’s System Engineering Handbook [MSFC-HDBK-1912A, 1994], while not truly defining SE, gives a description of it as:

... a continuous, iterative process with a built-in feedback mechanism that is used throughout a project or program’s life cycle to arrive at the best system architecture and design possible.

The objective of systems engineering is to provide a system that accomplishes its purpose in the most cost-effective way possible, considering performance, schedule and risk. In order to handle the often complex end-to-end and life cycle issues, systems engineering must also rely on the speciality engineering disciplines, in addition to the traditional design disciplines. These specialist engineering areas, which typically include reliability, maintainability, logistics, test, production, transportation, human factors, and safety engineering, are highly technical fields in themselves. Although specialist engineers contribute their functional expertise and analytic methods throughout the system engineering process, the system engineer’s role is to see that these functions are coherently integrated into the project at right times.

The systems engineering process at NASA reflects a doctrine of successive refinement. The model adopted by NASA is the spiral model ([Chamberlain & Shisko, 1997], [Griner & Schneider, 1998]), illustrated in Figure 3-7. The spiral model reflects the fact that the systems engineering process operates at successively greater resolution as the system design is developed. The steps in the NASA’s systems engineering process are: recognise need/opportunity, identify and quantify goals, create alternative design concepts, select design, increase the resolution of the design, implement the selected design decisions, perform the mission. These steps are detailed in [Chamberlain & Shisko, 1997]. These steps are applied again and again during the system life cycle as the system is developed. As the system is realised, the issues addressed evolve, and the particulars of the activity change. Most of the major system decisions (goals, architecture, life cycle cost, etc.) are made during the early phases of the project, so the turns of the spiral (that is, the successive refinements) do not correspond precisely to the phases of the system life cycle. Much of the system architecture can be “seen” even at the outset, so the turns of the spiral do not correspond exactly to the architectural hierarchy, either. Rather, they correspond to the successively greater resolution by which the system is defined.
Recognise need/opportunity

Identify and quantify goals

Increase resolution

Implement decisions

Select design

Do trade studies

Create concepts

Mission

Figure 3-7: The spiral model: doctrine of successive refinement (Source: [Chamberlain & Shishko, 1992])

Like the waterfall model, the spiral model is also derived from a model for software life cycle. Although all activities in the waterfall model can be depicted in the spiral model, the spiral model reflects the iterative nature of software and systems engineering processes. Figure 2-24, in Chapter 2, illustrates the spiral model for software life cycle. Note that rapid prototyping is used in each cycle, and that the model emphasises risk analysis. This approach is particularly useful in high-risk developments because design sometimes evolves as detailed requirements emerge [Sage, 1992]. The spiral model is risk-driven whereas the waterfall model is document-driven. [Ruth, 1994]

NASA management instructions (NMI 7100.14B) define the phases of a major system acquisition as follows:

Phase A – Preliminary analysis
Phase B – Definition
Phase C/D – Design/Build/Integration/Verification

This list is rather truncated because the acquisition activities do not include the pre-proposal part of the process, and tend to emphasise the remaining early phases to the exclusion of the later portions of the life cycle. A more complete view of the life cycle phases is shown in Figure 3-8.
3.2.4 NASA’s Faster, cheaper, better

The paradigm of ‘faster, cheaper, better’ space missions, as heralded by NASA with the Discovery and New Millennium Programmes, relies on a much closer integration of system design, development and verification, and draws heavily on a robust and comprehensive programme of technology development, which must run in parallel and off-line with respect to flight programmes. (see Figure 3-8).

![Diagram of mission phases]

**Figure 3-8 ‘Faster, cheaper, better’ space program** (Source: Adapted from [Atzei & Novara, 1997])

The “faster, cheaper, better” paradigm was used for the expansion and adaptation of the capabilities of the Mission Control Center (MCC) of the NASA Johnson Space Centre. At an overall control centre level, significant changes have occurred in four major areas [Parker & Braddock, 1994]:

- **fundamental technology.** Technology used in the MCC is now firmly founded in a distributed COTS hardware and software platform supporting Open Systems standards;

- **generalisation of capabilities.** The MCC consists primarily of a generic control centre set of capabilities architected as a distributed network of Unix workstations and servers, augmented with vehicle-specific telemetry, command, and end-user applications;

- **incremental delivery to operations.** New control centre capabilities are being delivered into operations at least every six months, resulting in extremely focused development activities, immediate user feedback, and isolation from major program phases.

- **change in customer/contractor relationship.** NASA and its contractors operate as a team with common goals and a common vision. NASA JSC has formed product delivery teams with NASA, development contractor, and operations contractor members. Six-month deliveries have major mandatory components, as well as general enhancements that get prioritised into the work plan based on schedule and funding availability. Success is clearly defined for the product team: delivery on schedule within budget of all mandatory capabilities and as much of the enhancement capabilities as possible. When technical or resource problems arise, they are solved in a team environment. NASA is clearly the team leader in all of these areas, however it is recognised that contractors are critical to the success of the team. This team environment has fostered a continuous process improvement approach to doing business, as well. Always looking for faster, better, cheaper solutions allows teams to question current methods and improve upon their execution. The systems engineering process at NASA has evolved from traditional ‘waterfall’ approach to concurrent engineering.
Results of this application of the "faster, cheaper, better" approach are [Parker & Braddock, 1994]:

- **faster.** Shuttle MCC equipment replacement will complete 5 years ahead of the previous schedule. Training simulations using the new Shuttle MCC equipment and software begin 12/94, with full on-orbit shuttle support by 6/95, and ascent/entry support by 11/95.

- **better.** The new Control Centre uses COTS hardware and software, providing a distributed, open systems-based complex for space vehicle command and control. The new Control Centre builds on a generic base to provide rapid reconfiguration to support different missions. User interfaces in the new Control Centre are graphical compared to the textual displays of the old MCC.

- **cheaper.** The new MCC provides over US$150 million in development savings alone. The number of equipment racks has been reduced from 1400 in FY1993 to 700 by FY1999, resulting in significant maintenance reductions. In addition, maintenance of this equipment is over 99% vendor-provided, further driving down the cost of ownership. On the software side, the number of lines of code was reduced from 4.6 million lines to support today's Shuttle program to 3 million lines to support Shuttle, Space Station, and other vehicles. By reducing maintenance costs and increasing automation, the current Shuttle MOD staffing profile was reduced by 180 while simultaneously growing overall operations capabilities to support both Shuttle and Station operations.

The Mars Pathfinder mission is one of the first NASA projects to be developed under the "faster, cheaper, better" paradigm. Key was the early and comprehensive use of computer modelling and simulations during the study period, which allowed efficient concurrent engineering. The team also focused on integrating previously developed products. Some items such as a heat shield for Mars atmospheric entry are not commercially available (as yet). To take advantage of prior experience developed twenty years earlier on the Viking missions, heat shield design and fabrication experts were recalled from retirement. The Mars Pathfinder team successfully reduced the development cost by about one-half, compared to similar projects, and shortened the development schedule from five years to three years. There was a well-developed project management process and risk mitigation activity for the entire project development span, even though it was a departure from the approach used on previous large JPL projects. They implemented new things, combined in new ways, with the following highlights:

- flatter management structure, which shortened decision times;
- collocation of the entire team;
- greater authority given to subsystem managers;
- necessary documentation was created, but in a non-traditional, less formal way;
- preference for analysis over test during development,
- the focus on "cheaper" caused careful management of cost and schedule reserves.

In addition the team had a well-defined quantitative risk analysis process. For instance, they modelled the entry, descent, and landing sequences, and used Monte Carlo analysis to provide data for risk management decisions [Forsberg & Mooz, 1998]
3.3 Automotive industry

The automotive industry started at the end of the nineteenth century in the U.S.A. Big automotive companies were developed in the U.S.A, Europe and Japan over the twentieth century. Their main competitive strategy has been volume production seeking market share and volume sales in order to keep production costs per unit down and maximise their profit margins.

During the energy crisis in the mid-1970s, the automotive industry had to answer to fuel economy requirements. The increase in environmental awareness from the end of the 1970s introduced strict emissions regulations. The introduction of devices to control fuel economy and emissions worsened vehicle driveability (idled rough, often hesitated or stumbled during acceleration if the engine was not fully warmed up, and had little power). As a consequence, during the 1980s, the customer came into the picture. Electronics and software had to be introduced in order to balance fuel economy, emissions and driveability. Thus, the automobile has evolved from a purely mechanical or electromechanical product to include also electronics and software.

In the highly competitive global market of the 1990s, automotive companies, in order to survive, must seek not only product quality, but also short time to market and lower cost. The automobile must meet more functionality in order to keep or increase customer satisfaction. More functionality requires more components and their integration. The product became more complex with a range of disciplines involved in its development.

Sections 3.3.1 and 3.3.2 describe the characteristics and trends, respectively, of product development in the automotive industry. As a main driver to increase product complexity, trends in vehicle electronics are described in Section 3.3.3. The needs and trends for the adoption of systems engineering by the automotive industry are described in Section 3.3.4. Section 3.3.5 describes Ford Product Development System (FPDS) highlighting its approach to systems engineering and concurrent engineering against the traditional product development approach adopted by Ford in the past. On the concurrent engineering side, Section 3.3.6 describes the technology Ford is using to integrate CAD, CAM and CAE.

3.3.1 Characteristics

Stakeholders: The main stakeholders in the automotive industry are the customers, the competition, the corporation and the government. The customers are split into three major markets (U.S.A, Japan and Europe), low-end and emerging markets (Latin America, Eastern Europe, India and China) and niche markets. The competition includes mostly American, European and Japanese companies who compete in a global market. The corporation includes stockholders and upper management of the company that develops, manufactures, tests and distributes the product. The government includes each country’s environment protection agencies, road and traffic regulators, safety regulators, energy departments.

Requirements: The main drivers of automotive product development are return-on-investment, customer satisfaction, regulations (including emissions, fuel economy, road and traffic and safety regulations), and competition benchmarking. As high volume production (hundreds of thousands vehicles) being the main competitive strategy, market share and volume sales are the goal priority.
Higher productivity, shorter development lead-time and better product quality are the management monitoring parameters [Clark & Fujimoto, 1991]. Other important drivers are cost and weight.

**Product:**

The automobile has evolved from a mostly mechanical device to an electromechanical system that includes electronics and computer software. [Percivall, 1992]

An automobile is complex in terms of its internal product structure (i.e. number of distinct components and production steps, number of interfaces, and technological difficulty of and severity of the trade-offs among different components), and of its product-user interface (i.e., number and specificity of performance criteria, importance of measurable versus subtle and equivocal dimensions, and holistic versus narrow criteria). [Clark & Fujimoto, 1992]

[Pugh, 1991] describes the automobile as based on a static concept. The model ‘T’ Ford of 1907, in setting the vogue for the modern automobile, became the dominant design and established a conceptual plateau for all automobiles. Characteristics of a product based on a static concept are:

- product improvements occur through incremental design at the subsystem and component levels, and not at the overall system level.
- major innovations and emphasis swing from the product to the production processes. (see Figure 3-9)

**Life cycle processes.** A typical automobile life cycle is represented in Figure 3-10.

![Figure 3-9: Relationship between product innovation and process innovation (Source: Utterback & Abernathy, 1975)](source)

![Figure 3-10: A typical automobile life cycle (Sources: adapted from Nasr & Varel, 1997a), Clark & Fujimoto, 1991, Ford’s internal documents)](source)
Product development process:

Typical lead time: 43 months in Japanese companies, 62 months in American companies and 58 months in European companies. [Clark and Fujimoto, 1991]

Major stages: concept and project co-ordination, advanced vehicle design, styling, layout, component design, prototype building and testing, process engineering. [Clark & Fujimoto, 1991]

No formal specifications from customers: customer does not provide a detailed specification and does not reward producer until the system is available for his use. Even if the customer’s wants and needs are understood explicitly, it remains a challenge for the product to be developed from concept to showroom-ready before customer’s tastes change. [Percivall, 1992]

Use of prototypes: Vehicle development has a historical reliance upon the use of mule and pre-prototype vehicles. (Mule vehicles are production vehicles modified with experimental parts.) The heavy reliance is traditional because of the possibility of producing a vehicle for development testing, the significant amount of carried-over parts from year to year, and, the tremendous proving ground resources of the automotive companies. [Percivall, 1992]

Component focus: In the beginning of the 90s, the prevailing design approach in the automotive industry, especially for vehicle electronics development, was still to optimise a component [Gormley & MacIsaac, 1989]. The evolution of the overall product was believed as resulting from the evolution of its components. This component approach was founded on the belief that any design could be broken into independent and self-supporting components [Ziebart, 1991]. Each function required one component. The optimisation and the evolution of the whole was believed as resulting from the optimisation and evolution of every component. However, rather fast, the need for more functionality asked for more components and more interconnections among them, and a non-systematic evolutionary development process took place. Better components were substitute for old ones and new components were simply added to the current design version. Examples of the consequences of this approach show that it has not led to optimisation of the whole product [Ziebart, 1991]:

- more ECUs, sensors and actuators need more space. However, the rooms at the systems’ disposal get less and less because they are required for the convenience of the passengers;
- the overall system reliability may decrease because of the increasing number of parts;
- the increasing complexity of the system structure may deteriorate the serviceability and the handling of the electronic systems;
- in many cases, the driver has been overtaxed by the number of differently reacting electronic systems;
- interfaces often look like amateur solutions because they must be implemented at that time when the system definition was already finished.

Use of concurrent engineering techniques and tools. Volume producers may have production volumes of hundreds of thousands. Large production volumes require early emphasis on life cycle processes such as manufacturing, assembly, service and disposal. The automotive industry is well developed in the use of concurrent engineering techniques and tools and integration of CAD, CAE, CAM. [INCOSE, 1996]

Engineering changes. On U.S. programs the peak in changes occurs just before start of production. On Japanese programs the peak occurs in the design stage [Womack, 1990]. Figure 3-11 illustrates this fact.
A product begins as a concept, part of a strategy for attracting and satisfying customers. To turn a concept into a product, designers and product planners must make choices about product content. For an automobile firm, the choices pertain to trim levels, engine-body combinations, degree of innovation in the product and process, and the role of suppliers and carryover parts from predecessor models, among other things. These choices establish both how a firm will realise its product concept in the marketplace and who will undertake the necessary design and engineering work. Decisions about innovation and variety affect product complexity; degree of supplier involvement and use of off-the-shelf parts affect the volume of engineering work to be done in-house, what is called the scope of the project. Together, these choices determine the complexity of the project, which in turn influences productivity, lead-time and total product quality. Greater project complexity, particularly greater project scope, though it increases lead time and engineering hours, also increases overall product quality [Clark & Fujimoto, 1991].

Variety. Traditionally the theoretical total number of differently specified vehicle models (e.g. Fiesta) is of the order of 10000s considering all possible combinations of different parts performing the same functions.[Lorenz, 1998]

Reliance on carry-over parts. Automotive industry emphasises piece cost and investment. This emphasis creates a heavy reliance on carry-over parts. This places more constraints on the design and an emphasis on interface definition. [Clark & Fujimoto, 1991] shows that U.S. designs called for more than twice as many off-the-shelf parts as Japanese designs (38% versus 18%). European designs lie between these extremes.

Role of suppliers: The relationships of the automobile producers with their suppliers are a distinction of the Japanese system. The reliance on supplier engineering resources allows reduction in the automaker's engineering workload. Two-thirds of Japanese parts are black-box procurements. The black box approach allows the automaker to specify performance, the supplier designs and develops the component
to the specification. US companies reliance on suppliers is one-fifth of the Japanese [Percivall, 1992]. Figure 3-12 highlights the differences between the Japanese and American supplier systems.

**Legend:**
- Car maker
- Supplier with engineering capability
- Supplier without engineering capability

![Figure 3-12: Typical Japanese and U.S. supplier systems](Source: [Clark & Fujimoto, 1991])

**Product development organisation:**

**Functionally specialised organisations.** All major carmakers group engineering and planning expertise in separate subunits under functional managers (e.g., chief engineers). Though sometimes given different names, all product development organisations have standard departments organised around either car parts or development activities. Examples include body engineering, chassis engineering, powertrain engineering, testing, manufacturing engineering, engineering administration, planning, styling, and advanced engineering [Clark & Fujimoto, 1991].

**Cross-functional co-ordination.** The straightforward functional organisations of the 1960s virtually all incorporated formal mechanisms for cross-functional co-ordination by the late 1980s. This cross-functional co-ordination is carried out by multi-functional task forces and small teams organised around components or particular problems and full-time project co-ordinators called “product managers”. Although similar in structure, in practice, the role of teamwork and product managers are different in Western and Japanese companies. In the U.S., for example, a project team may be composed only of liaison people from each functional department and not containing any of the people who actually do the design or planning. In Europe, a product manager may be considered just as a co-ordinator or conflict manager. Japanese companies on the other hand, empower a program manager and assign a team of engineers, transferred from functional areas, for the life of the project [Clark & Fujimoto, 1991].

**Peak of people involvement and timing of trade-off decisions.** In the Japanese model, the number of people involved is highest at the very outset to confront difficult trade-offs early. In the U.S approach, the peak number of people on the project occurs at the time of launch as hundreds or thousands of extra
bodies are brought in to resolve problems that should have been cleared up in the beginning. [Womack, 1990]

### 3.3.2 Trends

Three driving forces have transformed the world automobile industry [Clark & Fujimoto, 1991]:

- **the emergence of international competition.** In the late 1950s, the auto industry counted four or five major world players. Today, more than twenty companies are capable of playing on a global scale.

- **the creation of fragmented markets populated by demanding, sophisticated customers.** The top-selling model in the U.S. in 1994 accounted for only about 0.4 million units compared to the 1.5 million units it sold in the late 1950s. All the major automobile markets have been characterised by an increase in number of models and a decline in volume per model.

- **the diversity resulting from technological change.** In 1970, 80 percent of all U.S production utilised one basic power-train technology — a water-cooled, carburetted, V8 engine that was longitudinally mounted and connected to a three-speed automatic transmission and rear-wheel drive. There were only five such power-train packages in production in the entire U.S. industry in 1970. By the mid 1980s, there were 35, representing a sevenfold increase in technical diversity. If electronics, new materials and new components were also considered, the growth would be even greater.

Figure 3-13 lists some trends in the automotive industry for the characteristics described in Section 3.3.1.

<table>
<thead>
<tr>
<th>CHARACTERISTICS</th>
<th>TREND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stakeholders</td>
<td>Increasing variety of stakeholders in the form of new niche markets or new environment agencies, for example. Competitors merging into bigger corporations seeking market share, volume sales and market coverage</td>
</tr>
<tr>
<td>Requirements</td>
<td>Increasingly stringent regulatory requirements Customer gets more sophisticated, taking for granted existing features in the vehicle Increased emphasis on customer/vehicle interface (e.g. Kansei engineering by Mazda/Ford [Nagamachi, 1995], “Fahrvergnugen” by Volkswagen, Humanware concepts by Mitsubishi [Rowand, 1989])</td>
</tr>
<tr>
<td>Product</td>
<td>Increasing vehicle complexity Increasing functionality Increasing use of electronics and computer control Increasing utilisation of safety critical systems</td>
</tr>
<tr>
<td>Life cycle processes</td>
<td>Use of total life cycle cost approaches Life cycle processes getting more integrated with product development through the use of DFX tools</td>
</tr>
</tbody>
</table>

*Figure 3-13: Some trends in the automotive industry continues...*
<table>
<thead>
<tr>
<th>CHARACTERISTICS</th>
<th>TREND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product development</td>
<td>More formal requirements capture and analysis</td>
</tr>
<tr>
<td></td>
<td>Anticipation of life cycle process requirements</td>
</tr>
<tr>
<td></td>
<td>Shorter average development cycle</td>
</tr>
<tr>
<td></td>
<td>Greater automation and integration of CAD, CAM, CAE, PDM, rapid</td>
</tr>
<tr>
<td></td>
<td>prototyping tools</td>
</tr>
<tr>
<td></td>
<td>Move towards systems engineering</td>
</tr>
<tr>
<td></td>
<td>More focus on the total vehicle and key systems (e.g. powertrain) a</td>
</tr>
<tr>
<td></td>
<td>means of meeting customers expectations. Delegation of subsystems</td>
</tr>
<tr>
<td></td>
<td>development (e.g. powertrain control systems, seat, etc.) to tier</td>
</tr>
<tr>
<td></td>
<td>suppliers.</td>
</tr>
<tr>
<td></td>
<td>Move towards Japanese supplier system</td>
</tr>
<tr>
<td></td>
<td>More engineering analysis upfront as a means to reduce lead time due</td>
</tr>
<tr>
<td></td>
<td>to prototyping and testing</td>
</tr>
<tr>
<td></td>
<td>More empowerment of project teams</td>
</tr>
<tr>
<td></td>
<td>Reduction in the total number of possible vehicle model specifications,</td>
</tr>
<tr>
<td></td>
<td>concentrating on the aspects that are most important to gain</td>
</tr>
<tr>
<td></td>
<td>customers with different tastes.</td>
</tr>
<tr>
<td></td>
<td>Move towards integrated development</td>
</tr>
<tr>
<td></td>
<td>Emphasis on reuse but with a more formalised process.</td>
</tr>
<tr>
<td></td>
<td>Recognition of different part reuse levels.</td>
</tr>
<tr>
<td>Product development</td>
<td>Delegation of subsystems development (e.g. powertrain control systems,</td>
</tr>
<tr>
<td>organisation</td>
<td>seat, etc.) to tier suppliers.</td>
</tr>
<tr>
<td></td>
<td>Move towards Japanese supplier system</td>
</tr>
<tr>
<td></td>
<td>More empowerment of project teams towards the Japanese approach</td>
</tr>
</tbody>
</table>

3.3.3 Automotive electronics

Since the 1970s electronics has improved incredibly the automotive performance and the operating features of the car. Consequently, the electronic content has grown considerably. In other terms, the value of electric and electronics in top range cars can easily exceed 20% of the total costs [Ziebart, 1991]. Figure 3-14 shows the rising percentage of cost of the electronic components used in an intermediate-size Nissan passenger car, from 1979 to 1991.

Figure 3-13: Some trends in the automotive industry

Figure 3-14: Rising percentage of cost of electronic components in automobiles (intermediate size Nissan passenger car)
(Source: [Miura, 1991])
Also, the number of on-board electronic systems in Japanese automobiles was increasing at a rate of around 10% per year (as can be seen in Figure 3-15) [Miura, 1991].

Figure 3-16 shows the increase in memory, functions, and inputs and outputs (I/O) since model year 1980 and an estimate for model year 2000. The most dramatic increase has occurred in the memory (RAM). Program memory (ROM) is also increasing. [Jurgen, 1995]

Figure 3-15: Increasing use of on-board electronic systems (Source: [Miura, 1991])

Figure 3-16: Automotive usage of microcontroller technology (Source: [Jurgen, 1995])

Electronics in automobiles was initially used to meet emissions regulations, but its use now covers also active safety control and on-board information and communication. Major applications of automotive electronics are ([Miura, 1991] and [Ziebart, 1991]):
• speed control;  
• electronic ignition;  
• engine control;  
• fuel injection;  
• cruise control;  
• automatic transmission;  
• automatic air-conditioning;  
• airbags;  
• drowsiness warning system;  
• rear-end collision warning system for trucks;  
• anti-skid control;  
• electronic suspension control;  
• four-wheel steering;  
• brake control (ABS);  
• traction control;  
• air-fuel ratio control;  
• mobile communication;  
• navigation systems;  
• advanced man-machine interfaces.

[Davies, 1996] researching about the driving revolution of the 21st century foresees a move towards automated driving. He proposes that by 2020 there will be automatic systems for most functions in a car including electronic tracking and convoy travel.

3.3.4 Systems engineering

The driving forces for the use of a systems engineering approach in the automotive industry, and particularly for automotive electronics development, include market and technical issues. They are mainly:

• increased global competition, with the ability to make products of high quality, low cost and strong consumer's appeal based on the application of many methodologies and technologies within a systems engineering approach [Parnaby, 1995];

• need for integration of new technologies and subsystems. Future automotive systems will include numerous new technologies (fuzzy logic, neural networks, expert systems, ergonomics) and subsystems (information and communication subsystems) to enable enhanced operation and to provide additional functionality for the driver and passengers. These technologies need an integrated framework only provided by systems engineering [Ziebart, 1991];

• increasing functionality and complexity. Functions need to be integrated to meet the overall system requirements (e.g. human factors, reliability and serviceability). The systems engineering framework is key in the development of complex systems which involve specialised disciplines (such as human factors, reliability and serviceability) and many interdependent design issues, because its rigorous structure ensures technical integrity. Monitoring the requirements ensures that the design requirements are optimised with respect to system requirements and constraints. At the same time, the systems engineering process encourages innovation and flexibility by postulating and evaluating many design alternatives relative to the requirements [Schilke, 1994];

• need for integration of techniques to capture and analyse requirements into a vehicle engineering process so that customer/user requirements can be balanced with many other issues and constraints that influence design decisions which must be made. The principles and processes of systems engineering offer a thorough framework for engineering for the customer and keep the focus on the life-cycle objectives to be satisfied [Schilke, 1994];

• need for safety critical electronic systems development, which requires a structured development [Parnaby, 1995];
• need for integration of concurrent engineering objectives (shortened lead-time, better quality (e.g.,
  better assemblability, better reliability, better robustness, better customer satisfaction), and lower cost)
  which provided benefits in isolated aspects of product design [Blanchard & Fabricky, 1990];
• need for viewing product and process (including the overall organisational process) as an integrated
  whole [Parnaby, 1995].

In 1988 a paper was published indicating some interest in systems engineering at General Motors
[Schilke, 1988]. More dramatically, General Motors announced the establishment of a Systems
Engineering Center in August of 1988 [Higgins, 1988]. A General Motors spokesperson said that a
thorough systems engineering approach would greatly reduce the cost of developing a new car.

The Society for Automotive Engineering has held forums on systems engineering [SAE, 1992]. The
forums on systems engineering were chaired by the Director of GM Systems Engineering, and included
presentations from GM, Ford, and other companies involved in automotive systems engineering.

Similar endorsements have recently been made in Europe. Fiat and Renault have created systems
engineering organisations headed by senior executives, and more subsystem level testing labs are
becoming apparent [Rivard 1992].

In addition to the observations by the Harvard and MIT studies, Japan has grown many of the total quality
control methods. Many of the concepts of systems engineering are similar to total quality control
[McMunigal, 1990]. One analysis indicates that total quality management, as it is developing in the
United States, is total quality control minus systems engineering aspects [Rohde, 1991]. The use of total
quality control and other observations indicates that the principles of systems engineering have been
adopted by Toyota, Nissan, Mitsubishi, and Honda [Rivard 1991].

3.3.5 FPDS

(The information presented in this section comes from the analysis of Ford internal documents such as
manuals for training on FPDS.)

Since 1993, Ford has been developing processes to describe the way it works to develop its products. The
evolution of these processes has been as following [Loureiro, 1996a]:
• CTC (Concept To Customer): this took place in the end of 1993 and consists of an improved
  product development process;
• WCT (World Class Timing): it followed the CTC stage and consists of a specific plan for
  establishing more efficient ways of working together in the quest to develop products that provide the
  highest level of satisfaction to the customers;
• WCP (World Class Process): followed WCT from the 1st. January 1995. It consists of an improved
  product development process, which is an expansion of CTC, and is based on the Fast Cycle Time
  principle (the on-going ability to identify, satisfy and be paid for meeting customer needs faster than
  anyone else). It is a timing/gateway process.
• FPDS (Ford Product Development System): process design happened from August 1995 to March
  1996. Release 1.0 happened on the 6th of May 1996 in support of the Ford 2000 program according
to which Ford aims to be the leading automotive company in the world. It is described in the following.

Ford 2000 identified 5 core business processes within Ford. They are: Management Systems, FPDS, Ford Production System (FPS), Order to Delivery and After Sales Service. These processes and their relationships are represented in Figure 3-17.

*Figure 3-17: Ford's business process model*

FPDS is the development process system by which Ford plan, design, develop and launch new vehicles. FPDS was designed to achieve the following objectives:

- 90% customer satisfaction at 3 months in service and 85% at 6 years in service;
- 25-45% reduction in time-to-market across a range of program milestones and magnitudes;
- 25-30% reduction in warranty due to the FPDS system approach to total program management;
- 30-40% reduction in resources (hours/program) due to time reduction;
- 25-30% improvement in investment efficiency due to reusability.

Meeting these objectives will increase Ford’s ability to launch more new vehicle programs, improve cash flow and enhance employee pride.

In comparison with WCP, FPDS promotes a reduction from 68 to 50 months to develop a vehicle. FPDS introduces the concept of scaleability by creating different scales of vehicle programs depending on its scope. Figure 3-18 shows that there are 6 different scales of program development, from a completely different vehicle (S6P6) up to minor modifications (S1P1). Depending on the scale, a vehicle program duration, from strategic intent (SI) to Job 1 may vary from 42 to 18 months.

*Figure 3-18: FPDS scaleability*
In order to achieve FPDS objectives, it is necessary changes in people attitude, changes in processes and changes in technology. Changes in people attitude include:

- **systems thinking**: decisions need to be made to optimise performance for, in decreasing order of priority: 1. FAO (Ford Automotive Operations), 2. Vehicle Program, 3. Individual function or activity. This approach contrasts with the traditional way of prioritising local optimisation of vehicle programs and individual work groups.

- **product development factory concept** focusing on optimising total FAO throughput leading to higher levels of financial returns.

- **reusability** of design concepts, experience, knowledge, parts, tools, manufacturing processes, facilities in order to achieve better levels of quality, cost and speed.

- **external customer focus** in contrast to placing excessive importance on internal issues and inhibitors.

- **Job1 like commitment**;

- **teamwork**. FPDS’s concept of empowerment includes clear and aligned goals and objectives, boundary conditions communicated and understood, accountability to deliver.

In terms of process changes, the major change was the adoption of a systems engineering process. FPDS follows systems engineering methodologies originally from the aerospace industry (e.g. Boeing). Figure 3-19 shows that FPDS’ systems engineering process is based on the ‘V’ model. Systems engineering cascades from customer-driven vehicle targets to systems level targets to sub-systems target to component targets. The prove-out starts with the component and progressively adds variables as testing/evaluation/analysis proceeds through sub-system and system levels to the vehicle.

![Figure 3-19: FPDS and systems engineering](image)

Figure 3-20 illustrates how the traditional automotive development approach, the component approach, described by WCP differs from the FPDS’ systems engineering approach. In WCP teams typically created a program level target and cascaded immediately to the part level. The complexity of this analysis made it difficult to optimise the Quality, Cost, Weight, Function, Timing (QCWFT) of the vehicle to desired customer attribute desires. The best parts do not make the best vehicle for the customer.
In FPDS, there are four basic types of tasks, as illustrated in Figure 3-21: define product and process, design product and process, verify/build/produce product and process, manage program. The key concepts supporting those tasks are listed in Figure 3-21 and some of them are described below.

**Figure 3-21: FPDS work breakdown structure**

Defined Pre-Strategic Intent Process was developed to provide the tasks and deliverables enabling a program team to transform inputs from Ford's annual process into a compatible proposal for vehicle program. Annual process inputs includes business goals and product assumptions, business plans, technology deployment plans, manufacturing strategy, bookshelf data, benchmarking data. The process focuses on the development of compatible vehicle target ranges for all QCWFT (Quality, Cost, Weight, Function, Timing). Target ranges include voice of the customer (e.g. target customers, volumes and markets, etc.), business aspects (e.g. investment, development cost, return on sales), product aspects
(scaleability, platform, vehicle weight and functional attributes, corporate wants/regulatory requirements), manufacturing aspects (e.g. reusability of parts, tools, facilities and processes, plant sourcing), purchasing aspects (early sourcing workplan), timing aspects (preliminary Job 1 date). No comparable such process definition existed prior to FPDS.

**Targets drive design.**
Customer, corporate and regulatory inputs are translated into 15 vehicle level attributes (see Figure 3-22) which are used to define the Vehicle Design Specification (VDS). The targets cascade and balance process is the left side of the systems engineering 'V' (see Figure 3-23). The approach balances functional, manufacturing, corporate and regulatory targets.

**Reusability.** Typically Toyota carries over 40% of parts versus comparable Ford programs at 5-25% carryover parts. Program reusability targets are set at <SI> (Strategic Intent milestone) for parts, processes, tools, facilities. Fit to Pre-<SI> and targets processes drive cross-

**Engineering for reliability.** In summary, to all vehicle systems, process flow charts and process FMEAs are developed. Some systems will be selected for robustness focus in order to optimise function in the presence of noise.

**Predictive and analytical engineering.** Consists of the use of CAE and C3P (see Section 3.3.6) tools for target setting, target cascade and validation, design experimentation of alternatives, design verification that the design meets the target, systems and attribute integration.

**Integrated package, function and appearance.** The advantage of this process in relation to WCP
process is the fact that as requirements were captured upfront, the limits of feasibility for appearance model are known. This contributes to reduce in 20% the engineering resources used under WCP model.

**Total manufacturing involvement.** Characteristics are: concurrent product/process design, maintain interface with plant personnel, ensure lean manufacturing and robust manufacturing design methodologies, manufacturing engineering co-leads system development teams, utilise flexibility and reusability practices. The change of paradigm from WCP is shown in Figure 3-24.

<table>
<thead>
<tr>
<th>WCP</th>
<th>FPDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Design and make feasible”</td>
<td>“Select the process and design to requirements”</td>
</tr>
<tr>
<td>Sequential engineering and manufacturing processes</td>
<td>Parallel and proactive, concurrent product/process design</td>
</tr>
<tr>
<td>Assumes minimal reusability</td>
<td>Optimises reusability</td>
</tr>
<tr>
<td>Quality capability must be developed</td>
<td>Quality capability designed in up-front</td>
</tr>
<tr>
<td>Drives design changes</td>
<td>Avoinds late design changes</td>
</tr>
</tbody>
</table>

*Figure 3-24: WCP versus FPDS product/process design paradigms*

Vehicle attribute focus. Figure 3-25 illustrates shows the shift in the prototype used for attribute development from WCP to FPDS – emphasising and increased reliance on analytical and lab/rig versus vehicle prototypes. In FPDS, the lower prototypes are applied when the upper ones are leverage.

![Diagram showing WCP and FPDS product/process design paradigms](image)

*Figure 3-25: WCP emphasis on prototyping and testing versus FPDS emphasis on up-front analysis*

Launch with quality and speed. Launch consists of the manufacturing validation and production preparation. Figure 3-26 illustrates the main differences between launch under WCP and FPDS.

<table>
<thead>
<tr>
<th>PROCESS</th>
<th>WCP</th>
<th>FPDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-functional team interaction in CP build</td>
<td>Minimal/active participation</td>
<td>Commitment/active participation</td>
</tr>
<tr>
<td>Prototype build process</td>
<td>Not very production-representative</td>
<td>More production-representative</td>
</tr>
<tr>
<td>Upfront skilled trades involvement</td>
<td>Limited</td>
<td>Involved; good input</td>
</tr>
<tr>
<td>Use of full service tooling suppliers</td>
<td>Partial</td>
<td>Increased</td>
</tr>
<tr>
<td>Tool suppliers</td>
<td>Prototype not always production</td>
<td>Same of prototype and production</td>
</tr>
</tbody>
</table>

*Figure 3-26: Launch process characteristics under WCP and FPDS*

Sourcing. Sourcing is the process of awarding business to suppliers. In FPDS this occurs in 3 phases as illustrated in Figure 3-27. Selected suppliers become part of the program team on a just-in-time basis and
work jointly to deliver program objectives. Suppliers include in-house and outside sources of systems, commodities, and end-item components. Relationships with supplier are oriented towards a long-term basis.

**Product development discipline.** Defines project management/deliverables/metrics, program team staffing and structure, design reviews, technology deployment guidelines, bundled changes. Program team structure is outlined in Figure 3-28.

![Diagram](image)

**Figure 3-27: Early supplier involvement**

**Program Steering Team (PST)**
- Chief Program Engineer

**Program Module Team (PMT)**
- Number varies by Program due to magnitude
- Chair: PMT leaders (Product and Manufacturing co-chair)

**Program Activity Teams (PAT)**
- Cross PMT Vehicle Attributes
- Chair: manager as required

**Program Task Forces (PTF)**
- Specific issues
- Chair: as required

PST - provides program direction for product, investment, quality and process. Admisters program timing, targets process and maintain vigilance of status with respect to team deliverables and milestones.
PMT - manages the design and release and manufacturability of the specified vehicle system, subsystem and component to meet the functional, quality, timing, weight and cost targets.
PAT - manages issues/events that affect multiple PMT's. For the FPDS targets process, a PAT vehicle integration is set up to manage trade-offs between attributes.
PTF - created to address a specific issue that arises within the PMT activities.

**Figure 3-28: FPDS programs team structure**
3.3.6 C3P

(The information presented in this section is extracted from Ford's training material on C3P)

C3P stands for CAD/CAM/CAE/PIM. It is a strategy that links and integrates processes and software tools for computer-aided design (CAD), computer-aided manufacturing (CAM), and computer-aided engineering (CAE) through global product information management (PIM). The business reasons for moving to C3P are: improved availability and quality of product information, reduced time to market, reduced costs. C3P offers two key technological advantages: improved access to information through integrated tools and a single data source. Initial rollout of C3P occurred in 1996 and its full capability will be available by the end of 1999.

Currently, the variety of software tools used by Ford Motor Company operations and suppliers makes the access and exchange of information difficult and inefficient. Instead of multiple vendors and technologies, a single vendor - SDRC (Structural Dynamics Research Corporation) - provides the core of Ford's next generation C3P software. This approach provides a foundation for process optimisation and the deployment of a comprehensive information management system.

SDRC's product line includes:

- I-DEAS Master Series - mechanical design automation software with integrated CAD, CAM and CAE applications.
- Metaphase - a market-leading product information management tool developed in a joint venture with Control Data Systems.

I-DEAS™ is the core CAD tool and provides a design methodology that readily anticipates analysis, assembly conditions, and other critical parameters. The result is a single math model for all components and assemblies. This computerised solid model becomes the electronic master - the only authoritative - and eliminates the need for paper drawings and other methods of access and distribution. Metaphase provides an environment where both graphical and non-graphical product information can be managed together.

The long-term strategy for C3P integrates all product information under a global PIM system, as illustrated in Figure 3-29. PIM is the organisation, access, and management of information directly related to the product(s) of an enterprise. It includes the processes and methods we use to generate, distribute, and manage robust product-related information. In simple terms, PIM is the process of getting the right information to the right people at the right time so they can do work or make decisions. PIM tools include electronic software tools and paper-based information. The electronic software tools are: world wide web, metaphase, engineering and business information systems. PIM also includes paper-based information, and identifies where the information is located and who can provide access to it. C3P PIM is not a single information environment, but rather several customised types of PIM structures that are integrated through processes and tools. There are two primary PIM structures currently under development:

- Vehicle Program PIM, which focuses on developing product information for a vehicle platform
- Non-program specific PIM, which focuses on core and advanced information relative to new technologies, processes, methodologies, and benchmarks for use in Vehicle Development
Both of the above PIM structures are made up of workgroups. A workgroup includes individuals who perform a similar activity and need to share information. The group follows a common process, and each member plays a specific role.

![Product Information Management](image)

**Figure 3-29: The product information management approach in C3P**

PIM libraries link Vehicle Program PIM and Non-Program Specific PIM by providing a central collection of information that needs to be shared with others, they organise it into electronic books. This information is promoted to a central library so it can be accessed by other groups.

C3P acts as a key enabler of FPDS by integrating advanced CAD/CAM/CAE technology with product development and manufacturing processes and associated data. C3P will lead to: earlier definition of component design objectives, an ability to test each part against these objectives through simulated tests, a reoriented design process that permits analysis while the design is in progress, increased linkage among geographically dispersed locations, improved effectiveness of teams leading to higher quality and reduced timing.

Much important functionality is embodied in Ford-specific computer applications. Tools currently being integrated with I-DEAS are: mechanisms analysis (kinematic, static, dynamic) (e.g. for suspension analysis, engine roll, wiper mechanism), human modelling (e.g. for manufacturing assembly capability, vehicle ergonomics/packaging), assembly analysis (e.g. for assembly optimisation), vehicle visualisation.

Issues being addressed by C3P software include: design in context of assembly, open architecture facilities, electrical systems harness design enhancements, manufacturing community’s needs, associative relationships between CAD, CAM and CAE data.

C3P is based on solid modelling instead of wireframe and surfaces. A primary difference between solids and wireframe and surfaces is that solid models have physical properties, such as: mass, weight, volume, centre of gravity and moments of inertia. As a result, a solid model can provide instant visual feedback about size and shape. In addition, solid-based assembly environment software enables you to perform operations such as interference detection, weight calculations, and dynamic installation checks more easily and more accurately. The result of such improved capabilities is a reduction of physical prototypes through the use of electronic, or virtual, prototypes and an increase in overall design quality. Also, it is easier to build solids than wireframes or surfaces. For example, if manufacturing only requires surface
information for NC machining, it may seem excessive to build a solid. In fact, it is easier to build a solid and then extract the surface information than it is to build separate surfaces and integrate them. Finally, solid models are the basis of FEA analysis and NC tool path generation. CAE models are based directly on solid CAD models and linked back to them. Solid models can be used for automatic mesh generation for CAE, thus significantly reducing lead-time for analysis. A solid model can also be used for rapid prototyping; for example, stereolithography.

Ford and its supplier(s) have several different CAD systems such as PDGS, CADDS, Pro-E, etc. It is difficult to exchange data between systems. C3P will reduce losses due to translation and will, therefore, help ensure quality and speed. Consolidate from PDGS 28 data collectors to a single C3P file management system. It is necessary to:

- avoid translation losses/issues/time with common integrated CAD/CAM/CAE/PIM toolkit which utilise exchange of native data;
- use solid modelling full electronic part definition – major CAD technology breakthrough;
- facilitate concurrent cross-functional work with improved product information sharing and easier access
- assembly modelling of CAD files allows build, function and appearance concerns to be identified before drawing release for Confirmation Prototypes (CP).

Concurrent engineering which has been a goal at Ford for some time, is the practice of sharing information before a process or a segment of a process is complete. FPDS and other initiatives bring increased process definition, and C3P provides the technology required to make concurrent engineering a reality.

### 3.4 Comparison

Figure 3-30 compares the characteristics and trends of the space and automotive industries described in Sections 3.2 and 3.3, respectively.

<table>
<thead>
<tr>
<th>ASPECT</th>
<th>AUTOMOTIVE</th>
<th>TRENDS</th>
<th>SPACE</th>
<th>TRENDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>Local market</td>
<td>Global market</td>
<td>Government</td>
<td>Government/Commercial Organisation</td>
</tr>
<tr>
<td>Requirements capture</td>
<td>Product quality, cost, weight, emissions</td>
<td>Time-to-market, functionality and the traditional</td>
<td>Performance and risk</td>
<td>Performance, risk, life cycle cost, development time</td>
</tr>
<tr>
<td>Requirements capture</td>
<td>Very little</td>
<td>Increased emphasis on customer/vehicle interface</td>
<td>Specification provided by specialised customer</td>
<td>Specification provided by formalised customer</td>
</tr>
<tr>
<td>Product functions</td>
<td>Static concept</td>
<td>Static concept for basic functionality. Increasing number of functions</td>
<td>Large product with many functions</td>
<td>Smaller products with distributed functionality</td>
</tr>
<tr>
<td>Product architecture</td>
<td>Much part variety enough to produce 10s of thousands of vehicle model combinations</td>
<td>Vehicle variety tuned with customer required variety</td>
<td>Redundancy in order to increase reliability</td>
<td>Redundancy and cost trade-off</td>
</tr>
<tr>
<td>Product development approach</td>
<td>Component approach</td>
<td>Systems engineering</td>
<td>Document focused systems engineering</td>
<td>Trade-off focused systems engineering</td>
</tr>
</tbody>
</table>

*Figure 3-30: Automotive versus space industries continues...*
In 1996, the author performed a literature review of the systems engineering and concurrent engineering methods and tools used at aerospace (including space) and automotive companies in the period 1978-1996. The results of that review are reported in [Loureiro, 1996b].

The study revealed that until the mid 1980s the only tools used by automotive companies were CAD/CAM based tools. With the increasing computer capability, CAD/CAM tools began to be integrated to CAPP, CAE and modelling and simulation tools. In the 1990s with the advent of the computer network technology, CIM environments were created. In the 1990s a greater emphasis was given to the use of concurrent engineering methods and tools by the automotive industry. Methods such as QFD, Taguchi methods, FMEA, DFMA, Value engineering, Group Technology, Design for Safety started to become common place in the automotive industry. Therefore, concurrent engineering tools enhanced the component approach traditionally adopted by the automotive industry.

On the other hand, in the aerospace industry, the use of concurrent engineering methods and tools happened in much smaller scale. The emphasis in the aerospace industry, since the late 1970s was the development of methods and tools for requirements capture and analysis (SADT, SART, DCDS, SA/SD) helping in the product specification process. Over the 1980s, these methods were computerised. Only from 1994, these methods became to be used also at the automotive industry.

In the 1990s, in both industries, in the organisational arena, changes were publicised towards: cross-multidisciplinary teams, concurrent engineering with suppliers and project management.

<table>
<thead>
<tr>
<th>ASPECT</th>
<th>AUTOMOTIVE TRADITIONAL</th>
<th>AUTOMOTIVE TREND</th>
<th>SPACE TRADITIONAL</th>
<th>SPACE TREND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product development timing</td>
<td>Sequential</td>
<td>Concurrent</td>
<td>Sequential</td>
<td>Concurrent</td>
</tr>
<tr>
<td>Verification</td>
<td>Prototypes</td>
<td>Analysis and fewer prototypes</td>
<td>Test</td>
<td>Analysis and fewer tests</td>
</tr>
<tr>
<td>Reviews</td>
<td>As required</td>
<td>As required</td>
<td>Planned and formal</td>
<td>Planned, fewer and issue solutions anticipated</td>
</tr>
<tr>
<td>Product development organisation</td>
<td>Matrix emphasising functional</td>
<td>Matrix emphasising teamwork</td>
<td>Matrix emphasising functional</td>
<td>Co-located teams</td>
</tr>
<tr>
<td>Relationship with suppliers</td>
<td>Purchase order</td>
<td>Early involvement</td>
<td>Acquisition contract</td>
<td>Partnership</td>
</tr>
<tr>
<td>Life cycle process drivers</td>
<td>Feasibility considered late in the development process</td>
<td>Design for manufacturing, design for assembly, design for service, design for the environment</td>
<td>Feasibility considered early</td>
<td>Life cycle cost</td>
</tr>
<tr>
<td>Manufacturing</td>
<td>Mass production</td>
<td>Mass customisation Lean production</td>
<td>One-of-a-kind production</td>
<td>Seeking economies of scale through commonalities among space programs, military and commercial applications</td>
</tr>
</tbody>
</table>

*Figure 3-30: Automotive versus space industries*
3.5 Conclusions

- The space industry is moving from the traditional systems engineering to a systems engineering approach that encompasses the concurrent engineering objectives of 'faster, cheaper and better'
- The 'faster, cheaper, better' approach of the space industry is a move towards integrated development in the sense it promotes concurrent engineering instead of the traditional sequential engineering approach, teamwork and partnership with suppliers.
- The automotive industry is moving from its traditional component approach towards systems engineering. From the late 1980s the component approach has been enhanced by the use of concurrent engineering methods and tools. In the 1990s, the automotive industry started to recognise the opportunities of a systems engineering approach. Teamwork, empowerment, supplier involvement are also increasing trends in the automotive industry.
- Both industries are enlarging the scope of their product development approach to include product and processes. Systems engineering provides the framework for product development. Concurrent engineering ensures that life cycle processes requirements will be anticipated to the early stages of product development so to enable that processes such as the manufacturing process in the automotive industry be developed simultaneously to the product.
- The pillars of integrated development organisation such as the integrated product teams find similarities with teamwork approaches heralded by modern approaches in both industries.

Figure 3-31 summarises the evolution of product development approaches in the automotive and space industries presented in this chapter.

![Figure 3-31: Evolution of product development approaches in the automotive and space industries](image)

With the objective of identifying the needs in the area of complex products development, Chapter 2 comprehensively reviewed approaches to manage complexity and this chapter investigated the product development trends in the space and automotive industry – two industries of complex products. In Chapter 4, this thesis deepens the investigation of the needs, providing insight of real industrial situations. Two automotive case studies will be presented.
Chapter 4
The gasoline and diesel PCS case studies

This chapter aims:
• to identify the needs to manage complexity throughout the product development and evolution process by analysing the gasoline powertrain control system (PCS) case study;
• to present the approach used by Ford Motor Company to develop the diesel PCS, coping with complexity;
• to draw some conclusions and lessons learned from the case studies regarding the identification of needs for complex product development

4.1 Background

A powertrain control system (hereafter called a PCS) is an automotive subsystem to control operating characteristics of the powertrain under all operating conditions to meet customer requirements including not only legal exhaust emissions but also, performance, fuel economy, driveability, safety, durability, quality, reliability and customer image. Operating characteristics of the powertrain include: air/fuel mixture, ignition timing, idle speed, torque. Operating conditions include: engine state, engine operating temperature, altitude, failure situations.

Figure 4-1 situates the PCS in the vehicle breakdown structure. A PCS consists of sensors, an electronic module (that runs the control software) and actuators to control, e.g., engine, exhaust gas recirculation system and automatic transmission.

This chapter presents two case studies of PCS development at Ford Motor Company: the gasoline and diesel PCS case studies. The gasoline PCS is a product that has been evolving since 1978 in a reactive and non-structured manner. This chapter describes the gasoline PCS development process as being mainly change driven, based on improving past versions of the product, through implementation driven requirements (e.g. 'change this subsystem to another version', 'add that sensor', 'delete that subsystem'). The effects of changes have only been evaluated when the whole vehicle is tested to see whether it meets the emissions regulations. This led to high complexity and low software maintainability of PCSs. On the other hand, motivated primarily by the need to develop a safety critical system and beginning from scratch, in 1994, a diesel PCS began to be developed through a structured approach. The development process was based on the European Space Agency (ESA) PSS-05-0 standard and supported by methods (QFD, HPM, FMEA, Hazard Analysis, SA/SD) and particular commercial tools (DOORS™, TeamWork™, Matrix-X™, Excel™, ClearCase). It was a solution adopted to deal with the complexity of the system.
Section 4.2 aims to highlight the reasons for managing complexity during product development by analysing the gasoline PCS case study. Initially a history of product evolution is presented. Then, stakeholders, requirements, functions and component evolution is examined. The development process evolution is then analysed as well as the organisations modifications related to this evolution. The section ends with a summary of the lessons learned from the gasoline PCS case study analysis.

Section 4.3 aims to present the structured approach used for the diesel PCS development as a means to cope with complexity. As the diesel PCS is a recent development, it is not possible to talk in terms of evolution. However, its brief history is presented and a comparison with the gasoline is performed.

The information gathered and analysed for the elaboration of this chapter is a result of two study periods at the Ford-AVT-CAPE-PCSE\(^1\) (Advanced Vehicle Technology – Core and Advanced Powertrain Engineering – Powertrain Control Systems Engineering), at Dunton, England and of published information about Ford’s PCS. The PCSE department is responsible for the PCS requirements analysis, systems architecture, software development, hardware specification and delivery to vehicle program. Hardware specification includes existing hardware change requests or decisions about purchase orders. The two study periods were:

- from July to September, 1995 with the objective of starting to identify best practice in the use of a systems engineering approach for the requirements capture and analysis and their deployment into, and within, subsystem level. The study was developed through a planned series of interviews with people working at various levels of the system breakdown structure and at various stages of product development life cycle. The study period is documented in a report entitled: ‘History of EEC at Ford: from evolutionary to systems engineering approach’ [Loureiro, 1996a]

- in April, 1997, with the objective of analysing product-process-organisations relationships, more interviews were carried out, information collected and documentation analysed. Types of information gathered referred to requirements, functions, components of a PCS subsystem and its development/evolution/configuration management process. For the gasoline side, focus was given to the Idle Speed Control subsystem and for the diesel side, to the whole PCS application developed for the New Diesel engines with emphasis to the EGR control subsystem. Information gathered during this period was also used for the elaboration of Chapter 9.

### 4.2 The gasoline PCS

The gasoline PCS developed by Ford is also called EEC (Electronic Engine Control) systems for historic reasons (e.g., it started controlling only engine parameters). Between 1978 and 1998, Ford had 5 versions of EEC, from EEC-I to EEC-V. The EEC evolution is summarised in Figure 4-2. EEC-I’s application was primarily spark control as Ford was interested in developing a more robust ignition system. EEC-II was introduced in 1979 and EECIII in 1980. In 1981, central fuel injection replaced the feedback carburettor as part of the EEC III system on some engine applications. Gradually as emission regulations

---

\(^1\) In September 1997, Ford Motor Company founded a company called Visteon. AVT-CAPE-PCSE is now Visteon-PCSE.
tightened, Ford was limited to basic engine design and looked at various other functions that could be performed by using electronics to optimise the operating range of the engine. In late 1982, EEC-IV was first introduced. Between 1978 and 1982, electronic engine management was developed exclusively in the USA and for the American market. In 1983, Ford started developing EEC systems in Europe. Between 1983 and 1988, many functions were added to the original EEC IV functionality, e.g.: EGR, adaptive strategy, cruise control, FMEM, neutral idle, MAF replaces MAP, top vehicle speed. In 1988, the Sierra became the first car produced in Europe by Ford using an electronic engine

1978 - EEC I was introduced and used on the 1978 and 1979 5.0-liter Lincoln Versailles.
1979 - EEC II was introduced
1980 - EEC III was introduced with expanded application
1981 - Central fuel injection replaced the feedback carburettor as part of the EEC III system on some engine applications.
1982 - EEC III was introduced on some light truck applications
late 1982 - EEC IV was first introduced
1984 - EEC IV was the only computerised engine control system Ford was using
1985 - Beginning in 1985 all EEC IV applications use rotary type TPS.
1985 - Beginning in 1985 some light truck EEC IV applications were equipped with an electronic module (IMS) that contained an E-cell.
1985 - EGR Vacuum Regulator (EVR) Solenoid was introduced.
Late 1985 - Adaptive strategy was introduced to EEC IV.
1986 - EEC IV system's ECU had an expanded adaptive strategy capability.
1989 - A slight variation of the high pressure CFI system was introduced.
1986 - The 5-liter engine featured sequential fuel injection.
1988 - On limited applications, the cruise control system was integrated into the EEC IV system.\[1991\] - Beginning in 1987, selected Ford vehicles were equipped with a Programmermed Ride Control (PRC) system.
1987 - Beginning in 1987, a limited number of Ford Motor Company vehicles were produced with a Malfunction Indicator Light (MIL).
1987 - Beginning in 1987, MAP and/or BP sensors have to be repositioned
1988 - MPG Lean Cruise strategy was introduced to EEC IV.
1988 - Beginning in 1988, selected engine applications were equipped with a mass air flow (MAF) sensor instead of a MAP sensor.
1988 - Cruise control system integration to EEC IV systems became more universal.
1989 - Beginning in 1988, on some applications of the 5.0 litre engine, the ECU is programmed to control top vehicle speed.
1988 - Cylinder Balance Test was improved so that a less severe cylinder miss could be identified.
1988 - EEC IV Sierra made in Europe.
1991 - Application of generic features to Zetec engine.
1995 - PTEC
1996 - First vehicle using EEC V, developed in Europe.

Figure 4-2: History of the EEC systems for gasoline

control system, the EEC-IV. From 1988, various advances were incorporated into EEC IV, e.g.: generic spark, EDIS, MAF, SEFI, dynamic fuel, generic air bypass ISC, electric air pump, generic EGR (Delta PFE), electronic transmission, DCL, OBD I, knock control, variable valve timing, adaptive calibrations, OBD 2, on board vapour recovery, direct electronic transmission shift control. Development with EECV started in 1992. From EEC IV to EEC V there is only a change of processor, the functionality is the same and the software can be re-used. The processor for EEC IV had a 32Kbyte-memory (maximum) and a 15MHz clock (maximum). Current gasoline systems have about 216KBytes of assembler code. The processor for EEC V is an Intel customised chip, developed especially for Ford.
EEC-V is part of the general EEC-IV evolution. The EECs evolved to control not only engine but also transmission parameters and are affected also by driveline axle characteristics.

The gasoline PCS evolution has been very much driven by emission regulations, including, for example, the requirements for on-board-diagnostics. However it has not evolved in a structured manner. Various features (or control software subsystems) and strategies (control algorithms) have been combined into it since 1978. This combination, however, has not been done in a structured manner. An existing version of a whole system software was modified for the inclusion of additional functionality in such a way that it was not possible, for example, to identify the correspondence between functionality and pieces of code. The whole system was then tested into the vehicle, and if adjustment could be found it would be used, otherwise further development would be necessary or discontinued. The adjustment criteria was the final emission numbers. However, the contribution of each individual feature to the whole emission figure was rarely analysed.

The number of applications grew very fast as a function of different air measurement system, engine ranges, number of valves, cylinder arrangement, engine orientation, induction type, fuel system, ignition type, transmission type, transmission type category, number of gears, drive lockup type, drive wheels, drive type. As the software was developed in an unstructured manner and the number of lines of code was steadily growing, the effort to modify or maintain the software became paramount.

In order to cope with the increasing software complexity and limit the growth of number of versions, Ford adopted some alternative concepts, such as:

- in 1991, the concept of generic features began to be applied to the Zetec engine. The generic concept consists of a common system framework for all next generation of world-wide EFI (Electronic Fuel Injection) powertrain programs. This framework identifies basic configuration of: system design, operating strategy, software design, electronic module design and sensor and actuator technologies. Flexibility was designed into the concept to accommodate unique user requirements. The use of generic system was intended to provide: world wide technical support focused on a single product range; optimisation of development resources; reduced complexity; improved calibrator understanding of system.

- in 1994, Ford-Europe launched all the vehicles with single strategy path. The PCS software for Scorpio, Transit, 1.3 Fiesta, was from a single original piece of code. Feature people took all the work that had been done, pulled it together and used one piece of code for every one.

- in 1995, Ford introduced the concept of what it calls 'the PTEC strategy', which consists of a basic PCS with the highest complexity, supporting all the features already developed, which is depopulated to meet different user requirements.

Despite all efforts to reduce variety of strategies and software, in 1995, world-wide, there were [LUT, 1995]: 70 EEC IV software families; 30 EEC IV strategy paths; 1100 EEC IV calibrations.

Driven by the need to improve software reuse, from 1995 onwards, Ford has been moving from ladder logic to the C language in order to develop the control software, reengineering the whole software using an object-oriented approach, restructuring the development organisation or adopting a structured approach for the development of new features.
In the following sub-sections, it is examined particular aspects that better characterise the gasoline PCS evolution over the period 1978-1998.

### 4.2.1 Stakeholders

This sub-section aims to demonstrate how the number and variety of PCS stakeholders increased over the period 1978-1998. Figure 4-3 illustrates PCS stakeholders evolution. The stakeholders presented in Figure 4-3 are those who provide operational or life cycle process requirements to the PCS development organisation.

<table>
<thead>
<tr>
<th>PCS stakeholders</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1978-1979</td>
<td>In the late 1960s, environment agencies were established by the American government in order to promote emissions regulations which were initially met by electro-mechanical devices. In the early 1970s, the oil embargo and an energy shortage drove fuel mileage standards in addition to the already established emissions standards. As lower emissions and better mileage were largely in opposition to each other, what was needed was a much more precise way to control engine calibration. In 1968 and in 1975, Volkswagen and Cadillac, respectively, introduced computer-controlled electronic fuel injection systems. In 1976 and 1977, Chrysler and Oldsmobile, respectively, introduced computer-controlled electronic spark control systems. Ford started developing its EEC-I system in the late seventies. Requirements driven by competition and environment agencies were interpreted by the Vehicle Program Office and passed onto the engine calibration community who would, then, require a new functionality or modifications to the existing one. Also, requirements or last minute modifications from the end-of-line engine test and hot test were also an input to the PCS development process. Constraints provided by manufacturing feasibility, electrical engineering and ACD (Automotive Components Division, who manufactures the electronic module) were also taken into consideration. EECI and EECII had no self-diagnostic capacity. Special test equipment and procedures were developed for the dealer community.</td>
</tr>
<tr>
<td>1980-1988</td>
<td>During the 1980s the consumer (referred as Joe Public) got into the picture. In order to meet tightened emissions targets, not only did the consumer's car get poor fuel mileage, it had poor driveability (slid roughly, often hesitated or stumbled during acceleration if the engine was not fully warmed up, and had little power). In order to improve driveability, EEC-III, for example, had a modulator strategy to compensate for conditions requiring extreme calibration commands: cold engine, overheated engine and high altitude. Some early versions of EEC-IV already came with programmed ride control and shift indicator light which can be considered as part of a driver information system. PCS development started also to integrate with the driver information systems development. EEC III has some self-diagnostic capability. EEC-IV had an outstanding self-diagnostic capacity. It included: malfunction indicator light, quick-test/self-test, failure mode effects management. EEC-IV diagnostic routine provided means to verify driveability complaint. At this stage also, the PCS started to be integrated with the whole powertrain, by considering also powertrain parameters to control the engine.</td>
</tr>
</tbody>
</table>

*Figure 4-3: PCS stakeholders evolution continues...*
...Figure 4-3 continued

<table>
<thead>
<tr>
<th>PCS stakeholders</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1989-1998 [Ford Motor Company, 1995g]</td>
<td>From the late 1980s and during the 1990s, the PCS started to control also transmission parameters. An example of such a functionality is the torque converter clutch control for automatic transmissions. Also, as the PCS became a safety critical component, security issues were also taken into consideration for its development. In terms of diagnostics, since 1996, Ford's cars are compliant with OBD II regulations.</td>
</tr>
</tbody>
</table>

*Figure 4-3: PCS stakeholders evolution*

Over the years the stakeholders shown in Figure 4-3 also changed or grew in variety. As environmental world awareness increases, so does the number of environment agencies around the world. The developing countries, also called emerging markets, for example, are starting to enforce emissions regulations. The leading environmental agencies in the world are the Californian, the Federal American, the Japanese and the European. Also, each individual environment agency has its own organisation responsible for elaborating emissions requirements. Figure 4-4 illustrates the Federal American and the Californian organisation.

Customers are also becoming increasingly differentiated as the claim for customised products increases. Opportunities to explore niche markets with their own peculiar requirements are now being considered.

*Figure 4-4: Federal American and Californian environment agencies organisations*
4.2.2 Requirements and attributes

This sub-section analyses how requirements related to some vehicle attributes evolved over the years. The vehicle attributes under consideration are: emissions, fuel economy and driveability. In order to optimise these three parameters simultaneously, a precise way to control engine functions or calibration is required. The PCS has a mission: to reduce emissions, improve fuel economy and improve driveability. The priority varies, however, under some driving conditions. For example, during warm-up, driveability has a higher priority than fuel economy.

Emissions regulations

Figure 4-5 presents the Californian and Federal American regulations evolution in the late 1970s and 1980s. It can be seen how they have tightened if compared with emissions levels from a typical car in 1960.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1978</td>
<td>0.41</td>
<td>1.5</td>
<td>9.0</td>
<td>15.0</td>
<td>1.5</td>
<td>2.0</td>
</tr>
<tr>
<td>1979</td>
<td>0.41</td>
<td>0.41</td>
<td>9.0</td>
<td>15.0</td>
<td>1.5</td>
<td>2.0</td>
</tr>
<tr>
<td>1980</td>
<td>0.39</td>
<td>0.41</td>
<td>9.0</td>
<td>7.0</td>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>1981</td>
<td>0.39</td>
<td>0.41</td>
<td>7.0</td>
<td>3.4</td>
<td>0.7</td>
<td>1.0</td>
</tr>
<tr>
<td>1982-84</td>
<td>0.39</td>
<td>0.41</td>
<td>7.0</td>
<td>3.4</td>
<td>0.4</td>
<td>1.0</td>
</tr>
<tr>
<td>1985-88</td>
<td>0.39</td>
<td>0.41</td>
<td>7.0</td>
<td>3.4</td>
<td>0.7</td>
<td>1.0</td>
</tr>
<tr>
<td>1989-92</td>
<td>0.39</td>
<td>0.41</td>
<td>7.0</td>
<td>3.4</td>
<td>0.4</td>
<td>1.0</td>
</tr>
<tr>
<td>1960</td>
<td>10.6</td>
<td></td>
<td></td>
<td></td>
<td>84</td>
<td>4.1</td>
</tr>
</tbody>
</table>

Figure 4-5: Federal and California emissions standards compared to emissions from a typical car in 1960 (Source: [King, 1997])

Figure 4-6 illustrates the variety of regulatory requirements and emissions test procedures in order to meet the USA 49 states, California, Europe and Japan regulations. Also, from 1992 onwards, the emissions regulations included durability tests at 80000 and 160000 Km. As the emissions targets tightened, the regulations also became more complicated. For example, for the USA-49 states market: 'manufacturers must certify a minimum of 40% of their 1993, 80% of their 1994 and 100% of their 1995 and later passenger cars plus light-duty trucks to the primary or low emission standards, with the remainder certifying to the secondary standards'. One important modification concerning the definition of pollutants is the specification of non-methane hydrocarbon limits (NMHC) for methanol-operated vehicles and as NMOG (Non-Methane Organic Gas) for “clean-fuel” vehicles. Starting in 1994, the Californian Air Resources Board introduced very low-emission vehicle plans:

- TLEV (Transitional Low-Emission Vehicles) for 1994 and later model year vehicles,
- LEV (Low-Emission Vehicles) for 1997 and later model year vehicles;
- ULEV (Ultra-Low Emission Vehicles) will start in 1997, but will only become mandatory for higher shares of overall sales from the year 2000;
- ZEV (Zero-Emission Vehicles) from 1998, an emissions standard of NMOG=0.0 g/mile (NMOG=Non_Methane Organic Gas) must be complied with by 2% of the vehicle fleet. This limit must be complied with by 5% in 2001 and by 10% of the fleet in 2003.
European emissions limits, traditionally less stringent than the American counterparts, from 1996 are being considered as restrictive as those of the USA. Recent European regulations are known as Stage I, Stage II and Stage III emissions limits. Up to the present, a vehicle that was certified in accordance with US test procedure FTP-75 cannot be refused an EEC Type Approval. This is likely to change, since the trend in Europe is towards more stringent standards combined with the new, comprehensive test procedures to reflect European driving conditions better than the U.S standards.

<table>
<thead>
<tr>
<th>Place</th>
<th>Test</th>
<th>Year</th>
<th>HC g/m</th>
<th>Nox g/m</th>
<th>CO g/m</th>
<th>Durability</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA 49 States</td>
<td>FTP 75</td>
<td>1992</td>
<td>0.41</td>
<td>1.0</td>
<td>3.4</td>
<td>50Kmiles</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1994/95/96 40/80/100%</td>
<td>0.25/0.31 (NMHC)</td>
<td>1.0/1.25</td>
<td>3.4/4.2</td>
<td>50Kmiles/100Kmiles</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2001</td>
<td>0.125 (NMHC)</td>
<td></td>
<td></td>
<td>50Kmiles</td>
</tr>
<tr>
<td>USA California</td>
<td>FTP 75</td>
<td>1992</td>
<td>0.39 (NMHC)</td>
<td>1.0</td>
<td>7.0</td>
<td>100Kmiles</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1994/95/96 40/80/100%</td>
<td>0.31 (NMHC)</td>
<td>1.0</td>
<td>4.2</td>
<td>100Kmiles</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1999</td>
<td>TLEV (NMOG)</td>
<td>0.125/0.156</td>
<td>0.4/0.60</td>
<td>3.4/4.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1999</td>
<td>LEV (NMOG)</td>
<td>0.047/0.056</td>
<td>0.125/0.188</td>
<td>2.125/2.625</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2001</td>
<td>ULEV (NMOG)</td>
<td>0.025/0.034</td>
<td>0.125/0.188</td>
<td>1.063/1.313</td>
</tr>
<tr>
<td>Europe</td>
<td>ECE 15 MVEG Proposal + EUDC Proposal Germany</td>
<td>1992</td>
<td>HC+Nox g/km</td>
<td>g/km</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1992</td>
<td>0.97</td>
<td>2.72</td>
<td>80/101 km</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1993</td>
<td>0.97</td>
<td>1.6/1D</td>
<td>80/160/101 km</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1994</td>
<td>0.9/1D</td>
<td>1.0/1D</td>
<td>80/160/101 km</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2001</td>
<td>0.5</td>
<td>0.5</td>
<td>80/160/101 km</td>
<td></td>
</tr>
<tr>
<td>Japan</td>
<td>Japan 10.15 cycle *inertia weight (test)</td>
<td>1992</td>
<td>0.40</td>
<td>0.5/ &lt;1250 kg</td>
<td>2.10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1993</td>
<td>0.40</td>
<td>0.5/ &lt;1250 kg</td>
<td>2.10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2000</td>
<td>0.40</td>
<td>0.4</td>
<td>2.10</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4-6: Overview of emission control regulations**

**Fuel Economy regulations**

Figure 4-7 presents the evolution of American fuel economy standards. In 1986, the mileage per American gallon (MPG) was reduced from 27.5 to 26.0 by the federal American government.

Japanese fuel consumption is measured directly when the emissions test cycle is run, in contrast with U.S practices that calculate fuel consumption from emission figures. The Japanese fuel consumption limits should only be regarded as guidelines. No regulations designed to limit fuel consumption have so far been passed in the European Community.

<table>
<thead>
<tr>
<th>Model Year</th>
<th>MPG</th>
<th>L/100km</th>
<th>Total Improvement over the 1974 Model Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1978</td>
<td>18.0</td>
<td>13.06</td>
<td>50%</td>
</tr>
<tr>
<td>1979</td>
<td>19.0</td>
<td>12.38</td>
<td>58%</td>
</tr>
<tr>
<td>1980</td>
<td>20.0</td>
<td>11.76</td>
<td>67%</td>
</tr>
<tr>
<td>1981</td>
<td>22.0</td>
<td>10.69</td>
<td>83%</td>
</tr>
<tr>
<td>1982</td>
<td>24.0</td>
<td>9.80</td>
<td>100%</td>
</tr>
<tr>
<td>1983</td>
<td>26.0</td>
<td>9.05</td>
<td>116%</td>
</tr>
<tr>
<td>1984</td>
<td>27.0</td>
<td>8.71</td>
<td>125%</td>
</tr>
<tr>
<td>1985</td>
<td>27.5</td>
<td>8.55</td>
<td>129%</td>
</tr>
<tr>
<td>1986-1988</td>
<td>26.0</td>
<td>9.05</td>
<td>116%</td>
</tr>
<tr>
<td>1989</td>
<td>26.5</td>
<td>8.88</td>
<td>120%</td>
</tr>
<tr>
<td>1990-1997</td>
<td>27.5</td>
<td>8.55</td>
<td>129%</td>
</tr>
</tbody>
</table>

**Figure 4-7: U.S. Federally imposed fuel economy standards**
Driveability

[King, 1997] defines driveability as those factors, including ease of starting, idle quality, acceleration without hesitation, and so on, that affect the ease of driving and reliability. Although driveability has been an issue since the 1970s, only recently systematic attempts to evaluate it occurred such as the 1993 CRC Driveability Workshop [CRC, 1994].

Ford's Corporate Engineering Test Procedure 00.00 - R-202 [Ford Motor Company, 1996e] relates driveability to how well the engine is calibrated and the quality of the fuel charging system. It recommends to evaluate driveability during cold and hot operation and at low and high altitudes. Figure 4-8 presents the various aspects Ford considers when evaluating driveability.

The attributes described in Figure 4-8 maps back to the customer's definition of driveability that Ford captured on its QFD Wants Template (00.00QFD-D02-1) [Ford Motor Company, 1995c]. Figure 4-9 selected some of these customers' wants.

<table>
<thead>
<tr>
<th>STATE</th>
<th>ATTRIBUTE</th>
<th>PROCEDURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting</td>
<td>Cranking time</td>
<td>Consider engine cranking time. Does it come up to idle speed smoothly? Specify starting conditions, i.e., A/C on/off, electrical load, etc.</td>
</tr>
<tr>
<td></td>
<td>Stalling</td>
<td>Does engine cut-out following start-up? Specify starting conditions, i.e, A/C on/off, electrical load, etc.</td>
</tr>
<tr>
<td></td>
<td>Stumbling</td>
<td>Stumbling is engine hesitation and RPM drop without stalling. Specify starting conditions, i.e., A/C on/off, electrical load, etc.</td>
</tr>
<tr>
<td>Idle</td>
<td>Idle stability</td>
<td>Consider the smoothness of the engine torque pulses and whether the engine is consistently firing on all cylinders at idle. Evaluate under various loads, i.e., A/C on/off, etc.</td>
</tr>
<tr>
<td></td>
<td>Surging</td>
<td>Does engine RPM fluctuate from a normal RPM to a higher RPM?</td>
</tr>
<tr>
<td></td>
<td>Idle undershoot</td>
<td>Does the RPM drop significantly below the desired idle speed after kicking the throttle or rapid release of the throttle from a higher RPM?</td>
</tr>
<tr>
<td></td>
<td>Return to idle</td>
<td>Assess time and smoothness for the engine to return from 2000 RPM to the desired idle speed after rapid throttle closing.</td>
</tr>
<tr>
<td>Stabilized drive</td>
<td>Hesitations</td>
<td>Does engine pause during stabilised drive? Does the engine consistently fire on all cylinders?</td>
</tr>
<tr>
<td></td>
<td>Drive idle</td>
<td>Does the vehicle shuffle during drive without throttle operation?</td>
</tr>
<tr>
<td></td>
<td>Surging</td>
<td>Does engine RPM surge and cause a decrease in vehicle speed during stabilised drive?</td>
</tr>
<tr>
<td></td>
<td>Cruise stability</td>
<td>Does the engine run smoothly at steady state speeds? Is there a feeling of insufficient power during any drive cycle?</td>
</tr>
<tr>
<td>Low speed driveability</td>
<td>Hesitations</td>
<td>Does engine pause during low speed operations? Does the engine consistently fire on all cylinders?</td>
</tr>
<tr>
<td></td>
<td>Surging</td>
<td>Does engine RPM surge and cause decrease in vehicle speed during low speed operation?</td>
</tr>
<tr>
<td></td>
<td>Sagging</td>
<td>Does the engine drop off momentarily in acceleration rate?</td>
</tr>
<tr>
<td></td>
<td>Deceleration</td>
<td>Look for any change in a smooth deceleration from a longitudinal force standpoint.</td>
</tr>
<tr>
<td>High speed driveability</td>
<td>Hesitations</td>
<td>Does engine pause during high speed operations? Does the engine consistently fire on all cylinders?</td>
</tr>
<tr>
<td></td>
<td>Surging</td>
<td>Does engine RPM surge and cause decrease in vehicle speed during high speed operation?</td>
</tr>
<tr>
<td></td>
<td>Sagging</td>
<td>Does the engine drop off momentarily in acceleration rate?</td>
</tr>
<tr>
<td></td>
<td>Deceleration</td>
<td>Look for any change in a smooth deceleration from a longitudinal force standpoint.</td>
</tr>
</tbody>
</table>

Figure 4-8: Ford's definition and evaluation of driveability.
EXCELLENT POWERTRAIN (ENGINE AND TRANSMISSION)

<table>
<thead>
<tr>
<th>9.7.0</th>
<th>Dependable engine and transmission (for example, always starts quickly, never stalls, no hesitation or surging, and never overheats) [Anchor Statement – Upper/Positive]</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.7.1</td>
<td>Starts quickly every time whether hot, cold or wet</td>
</tr>
<tr>
<td>9.7.2</td>
<td>Never stalls when cold or hot</td>
</tr>
<tr>
<td>9.7.3</td>
<td>Quick to idle and settle</td>
</tr>
<tr>
<td>9.7.4</td>
<td>Consistent start times</td>
</tr>
<tr>
<td>9.7.5</td>
<td>No overheating or coolant loss</td>
</tr>
<tr>
<td>9.7.6</td>
<td>No unpleasant noise from engine</td>
</tr>
<tr>
<td>9.7.7</td>
<td>Battery should not discharge when vehicle is parked with lights on</td>
</tr>
<tr>
<td>9.7.8</td>
<td>No axle leaks or failures</td>
</tr>
<tr>
<td>9.7.9</td>
<td>No major problems for 100,000 miles</td>
</tr>
<tr>
<td>9.7.10</td>
<td>Long transmission and engine life</td>
</tr>
<tr>
<td>9.7.11</td>
<td>Maintain good power and pickup</td>
</tr>
<tr>
<td>9.7.12</td>
<td>Transmission does not break down</td>
</tr>
<tr>
<td>9.7.13</td>
<td>Engine does not break down</td>
</tr>
<tr>
<td>9.7.14</td>
<td>Adjustment/maintenance free transmission operation</td>
</tr>
<tr>
<td>9.7.15</td>
<td>Engine and transmission maintain good sound over time</td>
</tr>
<tr>
<td>9.7.16</td>
<td>Engine and transmission are durable (they will be trouble free and have a long life)</td>
</tr>
<tr>
<td>9.7.17</td>
<td>Engine and transmission are dependable (always starts easily and never stalls)</td>
</tr>
<tr>
<td>9.7.18</td>
<td>An engine that always starts easily</td>
</tr>
<tr>
<td>9.22.0</td>
<td>Smooth engine response</td>
</tr>
<tr>
<td>9.22.1</td>
<td>No surge at cruise</td>
</tr>
<tr>
<td>9.22.2</td>
<td>Smooth response (no hesitation) when accelerator pedal is pushed</td>
</tr>
<tr>
<td>9.22.3</td>
<td>Steady and predictable acceleration</td>
</tr>
<tr>
<td>9.22.4</td>
<td>No surge at a stop in drive gear (automatic transmission)</td>
</tr>
<tr>
<td>9.22.5</td>
<td>Smooth engine and transmission response</td>
</tr>
<tr>
<td>9.22.6</td>
<td>When pressing down on the gas pedal to accelerate, the vehicle responds the way I like</td>
</tr>
</tbody>
</table>

Figure 4-9: Voice of the customer for driveability

Effectiveness of the catalytic converter

One of the major reasons for creating a computerized engine control system is to have the ability to maintain a 14.7 to 1 air/fuel ratio and therefore enhance the effectiveness of the catalytic converter. The catalytic converter is a device for controlling exhaust emissions. Placed in the exhaust system, the catalyst agents cause low temperature oxidation of HC and CO and reduction of NOx, yielding H₂O, CO₂, O₂, and N₂. A stoichiometric air/fuel ratio (approximately 14.7 to 1 at sea level) is essential to make the catalytic converter work effectively. If the fuel mixture is leaner than 14.7 to 1, the reduction of NOx is not effective. If mixture is richer than 14.7 to 1, the oxidation of HC and CO is ineffective. A stoichiometric air/fuel ratio provides just enough air fuel for the most complete combustion resulting in the least possible total amount of leftover combustible materials: oxygen, carbon, and hydrogen. This minimizes the potential for producing CO, HC, and NOx. It also provides good driveability and economy. The most power is obtained from an air/fuel ratio of about 12.5 to 1, although the best economy is obtained at a ratio of about 16 to 1. Air/fuel ratios far from stoichiometric, however, are not compatible with the catalytic converter. The leaner mixtures also increase engine temperature as a result of slower burn rate.

Applications

On the top of the variety of emissions requirements to be met, more stringent fuel economy requirements and compulsory driveability requirements, the PCS has also to be compatible with an enormous amount of possible combinations of different types of its neighbouring subsystems in the powertrain. Figure 4-8 illustrates the many possible configurations the PCS has to fit in.
Considering, for example, only the fuel system (3 types), ignition type (4 types), number of gears (4 types) and drive type (4 types) in Figure 4-10, 192 (3 x 4 x 4 x 4) possible configurations should be addressed by the PCS development. This could lead to increased number of PCS versions or increased complexity of the PCS in order to meet the requirements for all configurations in a small number of reusable versions. As mentioned above, in 1995, there were 70 code modules, 30 control strategies and 1100 different calibrations.

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Subsystem attribute</th>
<th>Possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Air measurement</td>
<td>mass flow, speed density or vane meter</td>
<td></td>
</tr>
<tr>
<td>Engine</td>
<td>Engine range</td>
<td>1.8, 1.9, 2.0, 2.3, 2.5, 3.0, 3.8 or 5.0 L</td>
</tr>
<tr>
<td></td>
<td>Number of valves</td>
<td>2 or 4</td>
</tr>
<tr>
<td></td>
<td>Cylinder arrangement</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>Engine orientation</td>
<td>East West or North South</td>
</tr>
<tr>
<td></td>
<td>Induction type</td>
<td>TC or not applicable</td>
</tr>
<tr>
<td></td>
<td>Fuel system</td>
<td>SEFI, CFI or EFI</td>
</tr>
<tr>
<td></td>
<td>Ignition type</td>
<td>EDIS, TFI, LDIS or DP-DIS</td>
</tr>
<tr>
<td>Transmission</td>
<td>Transmission type</td>
<td>5R70E, 5R70W, 92332G, A4LD, A4LD-E, A4LDPE, ALL E, AOD, AOD-E, AOD-EW, AOD-I, AOD-IW, AODW, AODW-P, ATX, MTX</td>
</tr>
<tr>
<td></td>
<td>Number of gears</td>
<td>DC, 3, 4 or 5</td>
</tr>
<tr>
<td></td>
<td>Drive lockup type</td>
<td>MLUS, ON-OFF, MECH-4&quot; or SPLIT TORQUE</td>
</tr>
<tr>
<td></td>
<td>Drive wheels</td>
<td>2, 4 or not applicable</td>
</tr>
<tr>
<td></td>
<td>Transmission type category</td>
<td>manual or automatic</td>
</tr>
<tr>
<td></td>
<td>Drive type</td>
<td>RWD, R/4WD, A/RWD or FWD</td>
</tr>
</tbody>
</table>

*Figure 4-10: Attributes that determine possible PCS configurations*

**Reuse**

In order to improve investment efficiency, it is a corporate requirement within Ford to seek reuse. This reuse requirement is also a driver for the PCS software development. A reuse index can be used to evaluate strategy reuse by dividing the number of applications delivered by the number of existing strategy paths. The smaller the number of strategy paths, the bigger the reuse. Currently, Ford has around 30 strategy paths, i.e., 30 different algorithms implemented by the whole software code that goes into the electronic control module. The ideal number of strategies is of course 1, but Ford engineers believe that a figure around 10 strategy paths is feasible. Another reuse index of interest is the software subsystems reuse that can be obtained by dividing the number of strategy paths times the number of software subsystems by the total number of existing software subsystems. Software subsystems implement the PCS functionality such as ignition control, EGR flow control. For example, for 5 strategy paths and 200 software subsystems there will be potentially a total of 1000 different software subsystems. However, as there is reuse the actual number of existing software subsystems is 500. That means that each software subsystem is used, in average, by 2 strategy paths. Another reuse metric can be obtained by dividing the total size of all files shared with any other application by the total size of all files in application and it is called application commonality.

**Ease of calibration**

Within an entire vehicle development life cycle of 4 years, the powertrain calibration process may take between 1 and a half and 2 years. The calibration process consists of providing values to constants in the software so that verification parameters such as, emissions figures, fall into pre-specified acceptable levels. In 1995, each PCS had around 21000 of such constants to be input. One output of the PCS
development time because calibrators work iteratively with the PCS software developers trying to modify the software to suit the calibration requirements. Therefore, easing calibration reduces the burden on PCS software developers.

**Code size constraint and execution time**

In the early years of EEC-IV, the code size restriction was 32Kbytes. This figure moved to 88K in the late 1980s, 112K in 1995 and is now going to be constrained at 216Kbytes. However, as memory capacity grows, more functionality is required to programmes and execution timing becomes a problem. The PCS software routine has to run every engine cycle. In general, the bigger the code size, the longer the execution time for the same processor characteristics.

**4.2.3 Product functions**

This section aims to demonstrate how the PCS functions have increased in number and in modes of operations over the period 1978-1995 as an answer to the requirements evolution. Also, it demonstrates the evolution of the mapping between emissions, driveability and fuel economy requirements and, PCS functions.

Figure 4-11 shows the evolution of PCS functions and modes of operation from EEC-I to EEC-V. It can be observed that, for example, for EEC-I there were only the two basic modes of operation: normal and default. For EEC-II, the air/fuel ratio control function allowed the inclusion of open and closed loop operation. Closed loop operation is used for keeping the air/fuel ratio as close as possible of the stoichiometric value. The open loop is used during periods of time when a stoichiometric air/fuel ratio is not appropriate such as during engine warm-up or at wide-open throttle (WOT), when driveability and fuel economy take precedence to emissions requirements. A description of all modes of operation in Figure 4-11 can be found in [King, 1997].

Figure 4-11 also shows how the number of functions increased with time. From 3 in EEC-I, 6 in EEC-II and EEC-III, 15 functions in EEC-IV and 26 groups of functions in EEC-V. If the possible individual functions that can be implemented in EEC-V are counted, they are more than 90 individual functions, according to Ford’s application feature plan in 1997. For example, ignition control in EEC-V does not refer only to ignition timing but also to: spark timing, knock control, ignition diagnostics, etc. Obviously, not all possible functions in EEC-IV and EEC-V are part of every application. For example, the torque converter clutch control in EEC-IV is only for automatic transmission applications.

Summarising the functional evolution shown in Figure 4-11, the following can be said. EEC-I does not control the air/fuel mixtrure, but only spark advance. There is, therefore, no closed loop operational mode. This system was the last, best effort of the mechanical carburetor system to maintain driveability while optimising emissions. New with EEC-II is the air/fuel mixture control. Hence EEC-II is the first Ford system that can operate in closed-loop mode. Carbureted EEC-III vehicles have the same functionality as on EEC-II. The difference is that EEC-III systems can work with electronic fuel injection (EFI) systems, also called central fuel injection (CFI) systems. Also, in EEC-III, there are 3 additional operational modes: base engine strategy, modulator strategy and limited operational strategy. EEC-IV supports much more functionality than its predecessors. Its primary function is to control the air/fuel ratio. This is achieved through either a feedback carburettor, a single-point injection system (which Ford
diagnostics capability. EEC-V controls the fuel mixture delivery and the ignition throughout the range of the engine’s operational capacity. It is OBD-II (On-Board-Diagnostics-II) mandated, which is a regulatory requirement from the CARB (Californian Air Resources Board). It also determines the gradual wear and age of the vehicle over time and any changes in altitude or barometric pressure; it can then make compensatory adjustments in its own programs to correct for whatever needs to be altered because of the changed input information.

Figures 4-12, 4-13 and 4-14 provide the mapping between requirements and, functions and modes of operations. The requirements in those figures are detailed in terms of the conditions under which they are going to be verified. For example, emissions are verified during a cold start, stabilised drive or hot start. Fuel economy is verified by a city test and a highway test.

**Table:**

<table>
<thead>
<tr>
<th>EEC I</th>
<th>EEC II</th>
<th>EEC III</th>
<th>EEC IV</th>
<th>EEC V</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MODES OF OPERATION</strong></td>
<td><strong>NORMAL</strong></td>
<td><strong>DEFUALT MODE</strong></td>
<td><strong>NORMAL</strong></td>
<td><strong>DEFUALT MODE</strong></td>
</tr>
<tr>
<td><strong>Operation</strong></td>
<td><strong>Normal</strong></td>
<td><strong>Operation:</strong></td>
<td><strong>Normal</strong></td>
<td><strong>Operation:</strong></td>
</tr>
<tr>
<td><strong>Default mode</strong></td>
<td><strong>Limited operational strategy</strong></td>
<td><strong>Open loop</strong></td>
<td><strong>Closed loop</strong></td>
<td><strong>Base engine strategy:</strong></td>
</tr>
<tr>
<td><strong>Timing</strong></td>
<td><strong>2</strong></td>
<td><strong>Thenactor air control</strong></td>
<td><strong>3</strong></td>
<td><strong>EGR flow</strong></td>
</tr>
</tbody>
</table>

**Figure 4-11:** The evolution of PCS functions and modes of operation.
Figure 4-12: Mapping between requirements and EEC-I functions

<table>
<thead>
<tr>
<th>EEC I functions</th>
<th>Emissions control</th>
<th>Fuel economy</th>
<th>Driveability</th>
<th>Fuel economy</th>
<th>Driveability</th>
<th>Driveability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ignition timing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Throttle body air control</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EGR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 4-12 illustrates the EEC-I functions map to emissions, driveability and fuel economy requirements. It can be observed in the figure, the focus on emissions and fuel economy requirements. Some of the driveability requirements are not addressed by the EEC-I functions, as indicated, in Figure 4-12, by the empty cells under driveability. Also, many different requirements are addressed by the same functions. For example, city fuel economy is addressed by the ignition timing and the EGR flow control functions. The fact that many requirements are coupled by the same functions is, mainly, not due to the PCS conceptual development. The other powertrain subsystems controlled by the PCS have a major responsibility in this fact. An example of this is the catalyst converter that clearly couples driveability, emissions and fuel economy requirements once: leaner air/fuel ratio is better for fuel economy but increases NOx emissions and richer air/fuel ratio is better for driveability but increases HC and CO emissions.

Figure 4-13 demonstrates how the EEC-II functions and modes of operation map to emissions, driveability and fuel economy requirements. It can be observed the effect of the use of modes of operation on the requirements coupling. Instead of one big group of requirements affected by the same functions as in Figure 4-10, now 3 smaller groups of requirements can be clustered.

Also, Figure 4-13 shows an evolution, from EEC-I to EEC-II, in meeting driveability requirements.

Figure 4-14 demonstrates how the use of EEC-III modes of operation that reflect the requirements testing conditions makes the requirements x functions relationship closer to 1 to 1. This is desirable if one considers the fact that a change in a requirement will imply a change in a smaller number of functions. The testing conditions taken into consideration are cold or hot temperature, low or high altitude, highway or stabilised city test.
### EEC II functions and modes of operation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Air fuel ratio</td>
<td>Low</td>
<td>Stable</td>
<td>Stable</td>
<td>Low</td>
<td>Stable</td>
<td>Stable</td>
<td>Low</td>
<td>Stable</td>
<td>Stable</td>
</tr>
<tr>
<td>Ignition timing</td>
<td>Low</td>
<td>Stable</td>
<td>Stable</td>
<td>Low</td>
<td>Stable</td>
<td>Stable</td>
<td>Low</td>
<td>Stable</td>
<td>Stable</td>
</tr>
<tr>
<td>Throttle air control</td>
<td>Low</td>
<td>Stable</td>
<td>Stable</td>
<td>Low</td>
<td>Stable</td>
<td>Stable</td>
<td>Low</td>
<td>Stable</td>
<td>Stable</td>
</tr>
<tr>
<td>EGR</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>idle Speed Control</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Air fuel ratio</td>
<td>Low</td>
<td>Stable</td>
<td>Stable</td>
<td>Low</td>
<td>Stable</td>
<td>Stable</td>
<td>Low</td>
<td>Stable</td>
<td>Stable</td>
</tr>
<tr>
<td>Ignition timing</td>
<td>Low</td>
<td>Stable</td>
<td>Stable</td>
<td>Low</td>
<td>Stable</td>
<td>Stable</td>
<td>Low</td>
<td>Stable</td>
<td>Stable</td>
</tr>
<tr>
<td>Throttle air control</td>
<td>Low</td>
<td>Stable</td>
<td>Stable</td>
<td>Low</td>
<td>Stable</td>
<td>Stable</td>
<td>Low</td>
<td>Stable</td>
<td>Stable</td>
</tr>
<tr>
<td>EGR</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>idle Speed Control</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 4-13: Mapping between requirements and EEC-II functions and modes of operation**

### EEC III modes of operation

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Emissions, Fuel Economy and Driveability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cranking</td>
<td>Cold, Highway, Low speed, High speed</td>
</tr>
<tr>
<td>Part throttle</td>
<td>Cold</td>
</tr>
<tr>
<td>Wide open throttle</td>
<td>Base</td>
</tr>
<tr>
<td>Cranking</td>
<td>Overheated</td>
</tr>
<tr>
<td>Cranking</td>
<td>High</td>
</tr>
<tr>
<td>Cranking</td>
<td>Base</td>
</tr>
<tr>
<td>Closed throttle</td>
<td>Cold</td>
</tr>
<tr>
<td>Closed throttle</td>
<td>Overheated</td>
</tr>
<tr>
<td>Closed throttle</td>
<td>High</td>
</tr>
<tr>
<td>Closed throttle</td>
<td>Base</td>
</tr>
<tr>
<td>Part throttle</td>
<td>Cold</td>
</tr>
<tr>
<td>Wide open throttle</td>
<td>Overheated</td>
</tr>
<tr>
<td>Part throttle</td>
<td>Overheated</td>
</tr>
<tr>
<td>Wide open throttle</td>
<td>Overheated</td>
</tr>
<tr>
<td>Part throttle</td>
<td>High</td>
</tr>
<tr>
<td>Wide open throttle</td>
<td>High</td>
</tr>
</tbody>
</table>

**Figure 4-14: Mapping between requirements verification conditions and EEC-III modes of operation**
4.2.4 Product components

This section aims to describe the evolution of PCS components including: the electronic control module, sensors, actuators and control software. Also, the mapping between sensors and actuators or control functions is investigated to demonstrate how functions are made dependent on each other (or coupled) by the design solution chosen. Section 4.2.3 has already shown how the requirements were made dependent by the functional concept choice.

The electronic control module

EEC-I’s electronic control assembly (ECA):

- uses an externally mounted calibration unit to fit the unit to specific vehicles, accounting for weight, axle ratio, high altitude use and so forth;
- includes a fuel octane adjustment switch to retard ignition timing either 3 or 6 degrees if spark knock occurs (later systems use an “octane rod” on the distributor to do the same thing)
- provides a 9-volt reference voltage to its sensors
- default mode: if the ECA fails, it stops sending commands to the three actuators it controls. Ignition stays fixed at base timing; the Thernactor dumps air into the atmosphere as long as the engine is running, and the EGR valve stays closed. Obviously, neither driveability nor emissions quality will be at the desired levels, but the vehicle can be driven at reduced power and greater fuel consumption until it is diagnosed and repaired.
- the ECA includes a power relay to protect it from the reversed polarity, a protection retained on later Ford systems. Both the relay and the ECA itself are on the passenger side of the instrument pedal, near the brake pedal.

The EEC-II’s ECA and power relay are similar to EEC-I’s except the ECA includes more internal circuitry to perform additional functions. It also provides a 9-volt reference signal to each of its sensors. The default mode is now called limited operational strategy (LOS). This mode is triggered by an electrical failure in some critical component or circuit, and in it all actuators are left in their deactivated mode. Driveability, fuel economy, and exhaust emissions quality are all compromised in that state.

EEC-III’s ECA became more complicated, and there are some electronic fuel injection (EFI) systems with the EEC-III system. The fuel injection system is the throttle body type, effectively an electric carburettor, that Ford technical literature most often calls central fuel injection (CFI). Once the Ford Motor Company went to the EEC-IV system, electronic fuel injection refers specifically to multi-point injection (MPI). The LOS is like that of the EEC-II system on carbureted vehicles. For CFI-equipped vehicles the LOS keeps the injectors operating, but at a fixed pulse rate to produce a full, rich mixture.

EEC-IV’s ECA has much more capacity than its predecessors. It is the first to include KAM, a keep-alive memory enabling it to store codes related to faults that it has previously observed but that are no longer present. The ECA’s engine calibration assembly contains the calibration constants to fine-tune the ECA’s commands to the specific needs of that vehicle and engine’s weight, axle ratio, and transmission. Ford calibration assembly is an integral part of the ECA, and it cannot be removed or serviced separately. The processor for EEC V is an Intel customised chip, developed especially for Ford. EEC-V is part of the general EEC-IV evolution.
Sensors and actuators

The evolution of sensors in the EEC systems became important after EEC-IV's introduction. Up to EEC-III the important change was the introduction of the EGO sensor, from EEC-I to EEC-II. The EGO allowed the PCS to work in closed loop mode, being able to control air/fuel ratio. The introduction of ACT in EEC-III made the calculation of air/fuel ratio and spark advance more precise. Figure 4-15 illustrates the fact that up to EEC-III, the number of sensors remained almost the same and that there were not many different types of each sensor.

<table>
<thead>
<tr>
<th>Sensors</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEC I</td>
</tr>
<tr>
<td>ECT carried over</td>
</tr>
<tr>
<td>MAP and BP in one housing</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
<tr>
<td>EGO modified for central fuel injection applications</td>
</tr>
</tbody>
</table>
| EGO modified for central fuel in
### Sensors

- ECT, MAP, BP, CP, EGO, EVP, ACT carried over
- TP was linear for early applications and became rotary (1985)
- Profile Ignition Pickup (PIP) for SEFI (1986) or for DIS (1989)
- Cylinder identification (CID) for SEFI and DIS, 2.3, 3.0 and 3.8 liter engines.
- Ignition diagnostic monitor (IDM) different for TFI IV ignition system and for DIS
- Pressure feedback EGR (PFE)
- Vane airflow (VAF) for MPI applications
- Vane air temperature (VAT) for MPI applications
- Mass airflow (MAF) instead of MAP in some applications from 1988
- Idle tracking switch
- Neutral drive switch (NDS) for most automatic transmissions
- Neutral pressure switch (NPS) and transmission hydraulic switches (THS 3-2 and 4-3) for AXOD (automatic overdrive transaxle) transmission.
- Inferrred mileage sensor (IMS) from 1985 for some light trucks
- Knock sensor
- Vehicle speed sensor
- Brake on/off (BOO) switch for A4LD (automatic four-speed light-duty) transmission applications
- Power steering pressure switch (PSPS)
- Air-conditioning clutch (ACC)
- Ignition switch

### Actuators

- Air/fuel mixture control:
  - Feedback carburetors (3 types)
  - Central fuel injection (CFI) (2 types)
  - Multipoint injection (MPI)
- Fuel supply system:
  - Fuel pump (1 type for CFI, 2 types for CFI and MPI)
  - Fuel pressure regulator (for MPI)
  - Fuel Pump Relay
- Ignition system:
  - Thick film integrated (TFI-IV) Ignition system
  - Distributorless Ignition System (DIS)
  - Idle speed control:
    - Throttle kicker (carburetor or CFI)
    - DC motor idle speed control
    - Idle air bypass valve solenoid (MPI)
  - Thermactor air management (2 types):
    - Thermactor II
    - TAB, TAD and combination
  - Canister purge (3 types):
    - Constant purge system (CFI and MPI)
    - In-line canister purge solenoid (carburetor)
    - Canister/purge/heat control system
- EGR control (4 types):
  - EGRC and EGRV solenoids
  - EGR shutoff solenoid
  - EGR shutoff solenoid with back-pressure transducer
  - EGR vacuum regulator (EVR)
  - Torque converter clutch control (for A4LD, AXOD, 4EAT and E40D automatic transmissions)
- Turbocharger boost control solenoid
- A/C and cooling fan control (2 types):
  - A/C and cooling fan electronic module
  - Wide-open throttle A/C cutoff relay
- Inlet air solenoid (IAS)
- Variable voltage choke (carburettor)
- Temperature-compensated accelerator pump (TCP) solenoid (2150A carburettor)
- Engine fuel injector cooling tube (MPI)
- Shift light indicator (manual transmission)
- Vehicle speed control (from 1988)
- Programmed ride control (PRC) (from 1987)

### Figures

**Figure 4-17: EEC-IV's sensors and actuators and their variety**

Figures 4-18 demonstrates how control functions are made dependent on each other by the use of the same sensors, for example. If a sensor fails or becomes inaccurate, all functions related to them will not be performed adequately. Also, a change in the sensor may lead to a change in the control function, although there are mechanisms to avoid this effect. Between the sensor and the control software subsystem may exist input device hardware (e.g. A/D converter), low level input drivers (e.g. software for calculating units, for example, Volts), high level input drivers (e.g. software providing logical values of throttle position). Therefore, a change in a sensor does not mandate a change in the actual control software subsystem if the structure described above is used. Unfortunately, this layered structure is being considered by Ford only from 1995, when the software code was already too complicated to maintain.

Figure 4-18 (a) shows the mapping between EEC-I actuators and sensors. A '1' in the cell of the matrix indicates that the corresponding actuator in the column is controlled by using the information provided by
the sensor in the corresponding row. Figure 4-18(b) shows the mapping between EEC-II actuators and sensors. Figure 4-18(c) shows the mapping between EEC-IV control functions and sensors. In this case, a ‘1’ in the cell of the matrix indicates that the function in the corresponding column is accomplished by using the information provided by the sensor in the corresponding row. It can be observed that a basic set of sensors tend to be common to all actuators and functions. These sensors tend to be those used since the early versions of EEC systems, EEC-I and EEC-II. They are: ECT, TAP or TP, MAP or MAF (used alternatively) BP, CP, EGO and EVP. They also correspond to the main functionality performed by a PCS: air/fuel mixture control or fuel control and ignition control.

Software evolution

Since the start of PCS development, in the late 1970s, up to 1995, software algorithms, also called at Ford, strategies, were developed using ladder logic. Ladder logic is a micro-controller dedicated notation to describe a control algorithm [Collins and Lane, 1995]. If translated into a high-level computer
language, for example, it would be like a sequence of 'if...then...else' clauses or nested decision loops. These algorithms have been developed for each control subsystem, then put together to compose the PCS strategy. Ladder logic is not appropriate for developing large systems because as the algorithm grows in size, it does not provide visibility of the whole system being developed. For a large system, it may be appropriate as a bottom-up approach for system development when the functionality has already been partitioned among its components. It reflects the component approach to product development mentioned in Chapter 3.

With the exception, of course, of the EEC-I system, all of its successors' software has been developed by modifying previous versions of existing software. For example, when gasoline PCS development started in Europe, the first U.S.-developed-PCS strategy was already 10 years old. This continuous modification of existing software in order to meet various applications requirements has led to an increase in the number of strategy variants, also called strategy paths. In 1987, for example there were about 10 strategy paths with each strategy containing the whole PCS functionality. When the PCS started to control not only engine, but also transmission, the number of strategy paths was multiplied by the number of different transmission configurations. In 1989, there was an entire re-writing of the strategies available at the time in an attempt to reduce the number of strategy paths. This brought the number down to about 30 strategy paths. This re-writing however kept the ladder logic as the algorithmic language.

In 1991, it was introduced the concept of generic features. ‘Features’ is the name Ford calls the control software subsystems. These generic features were able to meet the requirements of all the applications launched in 1991. In 1995, the concept of generic features was expanded to PTEC. PTEC was an attempt that Ford made to reduce the number of strategy paths to ‘1’. The PTEC strategy contained all the functionality to meet all known possible applications. The strategy consisted of 700Kbytes of ‘C’ code. The idea was to depopulate the PTEC strategy depending on what features the application required. Also, calibration constants were set up differently depending on the application in order to enable a convenient piece of software and disable another that would not be used for that application.

The strategies are translated into Assembler code appropriate to the microprocessor used. Also the calibration constants are kept in the engine calibration assembly. These calibration constants contain the necessary information to fine tune the software to the specific needs of that vehicle and engine's weight, axle ratio, and transmission. As mentioned in Section 4.2.2, the size of the code is constrained by the memory available. EEC-IV started with about 30Kbytes of program memory. In 1995, the limit was 130Kbytes and in 1997 it was already 216Kbytes. However, in 1997, for example, the appropriate execution time was only achieved for a code size of 112Kbytes. Therefore, the PTEC has not become a reality in terms of software code.

The analysis of the PTEC strategy reveal the attempt to improve reuse, ease calibration and implement required functionality, meeting emissions, driveability and fuel economy requirements, all through the PCS software. Figure 4-19 presents a piece of the Idle Speed Control feature strategy which is part of the PTEC PCS strategy. Figure 4-19 provides example pieces of code for reuse, ease of calibration and functionality.
This is a typical way for easing calibration. The software itself is used to test different calibration constant values. Calibration constants are supposed to be set up by the calibrator and input to the software.

(a) Ease of calibration

(b) Reuse

Figure 4-19: Reuse, ease of calibration and functionality addressed by PCS software

The modifications of software strategy at the subsystem or component level, the attempts to improve reuse through generalising the software capability, the use of the software itself as a means of easing calibration by the identification of the appropriate calibration constants led to a situation in which it was not possible to map a given control function to the actual piece of software that implements it. Or, when it is possible its interface with other control subsystems or with software drivers is very complicated. Figure 4-20 illustrates the complexity of the PTEC’s Idle Speed Control subsystem interface.

Figure 4-20: PTEC’s Idle Speed Control subsystem interface

For new features, the gasoline PCS software is trying to adopt a more structured approach to strategy development. This approach is, however, constrained by the need to document the resulting strategy fitting the old gasoline PCS format, dictated by the U.S. based development groups. The overall proposed physical architecture for the PCS, according to this new structured approach, can be illustrated
in Figure 4-21. Up to this date, however, as far as the old gasoline features are still in use, the software code does not reflect the structure proposed in Figure 4-21.

**POWERTRAIN AND REST OF VEHICLE**

**Figure 4-21: The future - a proposed architecture for the gasoline PCS (Source: Ford Intranet, 1997)**

### 4.2.5 The development organisation evolution

Ford Automotive Operations has a matrix organisation as represented in Figure 4-22. Functional organisations such as product strategy, manufacturing, marketing & sales support the product development organisation. The product development organisation is organised in Vehicle Centres. The Small & Medium Vehicle Centre is located in Europe, whereas the Large and Truck Vehicle Centres are located in the USA. Vehicle Centres are split into Vehicle Lines and these into the various Vehicle Programs. The PCSE organisation is part of the AVT-CAPE (Advanced Vehicle Technology - Core & Advanced Powertrain Engineering). The PCSE is the organisation responsible for developing the PCS in support of the vehicle programs. Each vehicle program will generate PCS applications, although these applications are likely to have commonalities.

Figure 4-23 describes the worldwide PCSE organisation. There are PCSE organisations in Europe and in the USA. The PCSE in Europe is located in Dunton, England. The PCSE organisation is divided into Applications, Systems and Technologies groups.

The Applications group is responsible for capturing the requirements from the program office and documenting them in a PRD format. The PRD is the project requirements document. It contains the sensors, actuators and features that are going to be used on that PCS application. The Application group is also responsible for assuring that the PCS delivered accomplishes all the requirements in the PRD. Applications engineers may also be responsible for project management.
The Systems group is responsible for translating the PRD into the first three chapters of a system/subsystem design specification, the SDS. The SDS contains: Chapter 1 - System overview; Chapter 2 - Subsystem and design specific functional requirements; Chapter 3 - General system and
context requirements; Chapter 4 – Subsystem design; Chapter 5 – Test and integration reports; Chapter 6 – Requirements traceability report; Chapter 7 – Appendices (includes transfer documentation); Chapter 8 – Revision control sheet (change history). The Systems group is responsible for the development of new systems or new software subsystems, for providing subsystem or component requirements to the Technologies group and for validating the delivered subsystem and components against those requirements. The Technologies group is responsible for specifying hardware components and developing software components/subsystems in order to meet the PRD and/or the SDS requirements. The hardware components specification may be an HCR (Hardware Change Request) to be implemented on an existing hardware design by ACD (Automotive Components Division), or a specification for a new sensor or actuator. The development of software components/subsystems may be achieved by modifying existing strategy or software or by developing new ones by using the SDS information provided by the Systems group.

Before 1995, there were no Systems groups, only Applications and Technologies group. In the USA, the Systems group’s responsibility has been, mostly, the elaboration of the SDS. In Europe, the Systems group has never had a clear role so that it has never appeared on the organisation charts as a separate organisation unit. It has so far been linked either to the Applications group or to the Technologies group. For new software subsystems, like 'the smart alternator control', a systems engineer has worked close to the hardware and software engineers, writing subsystem documents. There was no systems engineering work for the entire gasoline PCS up to 1997. The systems work done is for new subsystems.

Before 1995, the software was developed for each vehicle program. The PCS was responsible for providing strategies and hardware components. The software was developed independently for each vehicle program. For example, there was no systematic way of preventing that two different pieces of software were developed for the same strategy. This situation led to an increase in the number of software versions for the same strategy. This justifies the fact that, for example, for 30 strategy paths in 1995, there were 70 software codes.

Before 1995, the Technologies group included groups responsible for sets of strategy paths. Each strategy path contained the various versions of strategies. Each strategy included the whole control algorithm implemented by the whole software codified in the module hardware. The organisation of the Technologies group was driven by the various applications implemented by each strategy. There were 5 groups responsible for the 30 strategy paths. When a change was requested on a strategy, if approved, all strategies had to be changed. With so many different supervisor groups implementing the same changes into various strategies, many duplicated implementations happened. This can be seen as an example of the effects of organisation on the development process.

This application driven organisation, where a new application was obtained by just implementing changes on old ones might have been adequate when the whole PCS software controlled just spark timing or implemented just six functions as in EEC-III. Demand for functionality started to grow, and with functionality, the whole PCS system became too complex to be efficiently maintained. The complex PCS system needed to be partitioned into more manageable subsystems and the development organisation should reflect that partitioning.
In February 1995, an organisational change put the strategy and software groups under the same supervisors. The groups in the new organisational structure would have feature responsibility instead of strategy responsibility. Each feature represented a control software subsystem. Each feature had its corresponding algorithm (strategy for that control subsystem) and software code version. In the old organisation, each strategy contained all features. In the new organisation, the application teams pick up the work from the feature teams and use released features to construct their applications. This reduced the re-work of feature coding tremendously. However, as the PCS strategy and software were developed as a whole, the features have developed very complicated interfaces. As a consequence, the feature teams themselves did not have a clear interface at that time. The previous organisation of the PCSE into strategy groups made the task of partitioning the control software system into features a very difficult, and sometimes impossible, task. The resulting features were tightly coupled to each other. This can be seen as an example of effects of organisation on product.

Figure 4-24 illustrates the organisation changes implemented in February 1995.

From February 1995 onwards, the Technologies group were organised into feature groups and component groups. Figure 4-25 lists the feature groups existing in April 1997. There were 10 feature groups for gasoline: 6 feature design groups, 1 device dependent software group and 3 transmission design. There is only 1 gasoline feature design group in Europe. All others are in the USA. As mentioned before, the features resulting from the gasoline PCS evolution from 1978 to 1995 were tightly coupled so that they have a very complicated interface among themselves. The change of one feature is very likely to affect the other ones. With the organisation split among Europe and USA, the maintenance effort may be increased by the need of communication among the feature development teams. For example, the Inlet Air Control features developed by Feature Design Group F, based in Europe, exchanges much information with the Fuel features developed by Feature Design Group B, based in the USA. When a change is required in any of these features, their developers need to communicate. The need for
communication, however, due to the physical and organisational distance among the groups, may increase development time and cost.

Figure 4-25 also gives an idea of the size of the American PCSE organisation if compared with the European one. The American is of the order of 10 times bigger than the European one. It is necessary to be said that only the gasoline PCS features are listed in Figure 4-25. The diesel PCS features are not included.

<table>
<thead>
<tr>
<th>Feature Development Group</th>
<th># Strategy + Software Engineers</th>
<th>Feature Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature Design Group A</td>
<td>7</td>
<td>Catalyst Monitor, EGO monitor, Electrically heated catalyst, Fuel level input processing, Fuel tank pressure input processing, Inferred catalyst temperature, OBDII diagnostics, Output state control, Purge control, Purge monitor, Self-test</td>
</tr>
<tr>
<td>Feature Design Group B</td>
<td>11</td>
<td>Adaptive fuel, Air charge estimator, Closed loop fuel, Dynamic fuel, Foreground fuel, Fuel pump, Hot injector compensation, Injector synchronisation, Injector timing, Open loop fuel, Transient fuel, Variable cam timing</td>
</tr>
<tr>
<td>Feature Design Group C</td>
<td>18</td>
<td>AC input processing, Air charge temperature, Alternator load, Anti-heel messages, Anti-plug fouling control, Barometric pressure, Battery voltage, Catalyst temperature, Clutch pedal position, Comprehensive component monitor, Data output link, Engine coolant level input, Engine oil temperature, Engine temperature, Engine temperature output, Flex fuel input processing, Fuel rail, IMCC, IMRC, Intake manifold swirl control, Neutral device switch, Power steering and brake on/off, RPM processing, SCP PID, SCP PIDn -- strategy specific, Secondary air, Speed control, Standard corporate protocol, Switchable calibration, Throttle position input processing, Throttle position output, Turbo control, Utilities</td>
</tr>
<tr>
<td>Feature Design Group D</td>
<td>9</td>
<td>Ignition diagnostics, Ignition timing, Knock control, Knock I/O, Misfire detection, Spark input/output processing, Software architecture</td>
</tr>
<tr>
<td>Feature Design Group E</td>
<td>10</td>
<td>Electronic throttle control, Electronic throttle control monitor, Exhaust gas recirculation, KAM qualification, Raster, Powertrain limiting &amp; protection, Torque based driveability, Torque control, Traction control</td>
</tr>
<tr>
<td>Device Dependent Software Group</td>
<td>3</td>
<td>Context FPC, Input drivers, LibCLL, Lib Kern FPC, Low level drivers, Output drivers, PTECCRT0, RTOS FPC, Sector Map FPC</td>
</tr>
<tr>
<td>Feature Design Group F</td>
<td>10</td>
<td>Air conditioning, EGR output driver for EEC, Alternator, Anti-theft, Exhaust gas ignition, Fan control, Inlet air control desired RPM, Inlet air control executive, Inlet air control feed forward, Inlet air control feedback, Inlet air control heated windshield, Integrated EDIS, Manual transmission input processing, Ride control, IPATS, PATS, Cold start fuel, Engine cooling control</td>
</tr>
<tr>
<td>Transmission Design Group A</td>
<td>8</td>
<td>Close loop pressure control, Closed loop adaptive control, Closed loop engagement control, Closed loop non-synch shift, Closed loop swap shift control, Closed loop synchronous downshifts, Closed loop synchronous upshifts, EPC executive, Transmission calculations, Transmission input processing, Transmission output control</td>
</tr>
<tr>
<td>Transmission Design Group C</td>
<td>8</td>
<td>Electronic pressure control, IPSC transmissions, Shift adaptive EPC, Transmission torque modulation, Transmission torque truncation</td>
</tr>
</tbody>
</table>

**Figure 4-25: Gasoline PCS feature groups**

On the 9th September 1997, Ford Automotive Products Operations (Ford APO) underwent a major organisational change and became a separate business enterprise called Visteon. Figure 4-26 shows how Visteon is organised. Visteon's 7 main divisions are: Global Powertrain Systems, Exterior Systems, Glass Systems, Climate Control Systems, Chassis Systems, Interior Systems, Electronic Systems. Visteon also has centred business direction not represented in Figure 4-26 including: Business Strategy, Finance, Public Affairs, Marketing, Human Resources and Logistics. The Global Powertrain Systems Division is fundamentally a PCS development organisation with research in other powertrain functionalities. The PCS Europe department is a transformation of the AVT-CAPE-PCSE (Europe) organisation. It focuses on the direct injection (DI) gasoline and diesel PCS features. It still keeps the Applications-Systems-Technologies type organisation.
4.2.6 The development process evolution

As it can be inferred from the product and organisation evolution described in the previous sections, the gasoline PCS development process is a very much change driven process. In February 1995, it moved from a strategy changing process to a feature changing process. This section shows also that the hardware development is a result of changing existing versions.

Figure 4-27 provides a simplified overview of the gasoline development process. The inputs to the gasoline PCS development are: program letters describing what functionality is required from a given PCS application, identified market opportunities from WCR (Worldwide Customer Requirements) and QFDs, informal contacts with program managers and calibrators. These inputs are formally translated into a PRD (Project Requirements Document) by the Applications group. Although called a requirements document, in practice the PRD is a list of features, sensors and actuators that the vehicle program managers or the calibrators would like to have on the vehicle. The URDs are software change requests that reflect the PRD or the calibrator will. If the development of a new feature is required as a consequence of the URD the Systems group develops a corresponding SDS. For existing features, the URDs are analysed and if the modification is approved, an EMR (Engineering Modification Request) is elaborated by the Strategy engineer and the change is codified in the software by the Software engineer.

If a change in the hardware is also required, it is specified by an HCR (Hardware Change Request) elaborated by a Module Configuration Engineer. The hardware change is implemented by the Automotive Components Division which has a library of bookshelved hardware designs in CAD file.
format. Once the hardware and software changes are implemented, a transfer document is prepared in order to guide the integration and testing of the PCS on the vehicle.

Before February 1995, the implementation of an EMR implied the modification of all strategies affected by that change. As there were strategies under different supervisors, there could be different implementations of EMRs. The resulting strategy release could also be codified differently if the same strategy was going to be used by different vehicle programs.

With the creation of the feature groups, all existing versions of the feature were put under the same supervisor. The trend became the reduction of the number of feature versions as a result of the implementation of the same change by the same way. Also the software version followed the feature version as the corresponding codified software was under the same supervisor as well.

The creation of the feature groups improved feature reuse, reduced the number of feature strategy and software versions and as a consequence reduced feature development time. The remaining difficulty was that the strategy could not be easily partitioned into features and different strategies could be partitioned differently. Depending on the strategy path chosen, the feature could have very different interfaces. The complexity of these interfaces also become an issue when gluing the resulting modified features back into an application strategy.

Figure 4-28 details the software change process at Ford, implemented from February 1995. In the following description, it will be highlighted how PCS software characteristics affected this process.

The requestors of a feature/application change are generally calibration and/or application engineers who knows the PCS functionality and can propose changes in order to correct or improve functionality. The changes come in the format of a URD (User Request Document), describing reasons for the request, symptoms associated with a problem or improvements required and test conditions in which the problem was discovered. The URD also indicates the application strategy chosen and the feature to be modified.
URDs regarding each feature are distributed to the corresponding feature groups in order to be analysed. The URDs accepted will lead to a different feature version described in the feature version plan.

As the features in the application strategies do not have well defined boundaries, a modification in one feature may require a change in another one. Therefore accepted URDs can lead to URDs in other features. The identification of all features affected beforehand is very difficult because of the lack of a model of the application strategy architecture. The source of change is the application strategy code itself, which was until recently written in ladder logic and, therefore, providing very low visibility of the whole application system. Some iterations are expected in order to identify the right set of features to be modified. These iterations may be difficult due to the fact that there are feature groups in the USA and in Europe and these features may interact with each other.

The feature version plan is discussed in the RDM (Release Development Meeting) which plans the release of the application software. At least the supervisors of the feature groups affected by the accepted URDs should be present in such a meeting. This may be difficult due to location distances and to non-identification beforehand of a feature affected.

Accepted URDs will lead to feature strategy modifications. Strategy engineers issue EMRs (Engineering Modification Requests) asking for software modifications and these are implemented. The time and effort to modify strategy and software increase in direct proportion of number of lines of code, cohesion, coupling metrics which are PCS software product characteristics.

Although very little testing is done at feature level, testing effort and time is a function of the PCS software product characteristics measured, for example, by the McCabe Cyclometric Complexity index [McCabe, 1976].

After the feature software being tested and reviewed, it is installed in the application. Very little is currently being done in terms of integration test and integration review. The application software is actually tried and tested by the calibration community when applying the PCS to the vehicle. Mainly emissions figures are evaluated to validate the PCS. If the application software do not meet the emission requirements, for example, or if the calibrator foresees room for improvement, a new set of URDs are emitted and the process iterates. When no more URDs are necessary, the application is released.

The process illustrated in Figure 4-28 was developed with the aid of a computer database tool developed by Ford, the EDTS (Electronic Development Tracking System). The first EDTS version remounts back to January, 1989. The last review happened in February, 1995.

From February 1995, the concepts of versioning and bookshelving were also introduced. Versioning is the identification of a combination of strategy and software modules that will fulfil a specific functionality. A feature build list is used to specify the exact file names and generations. Optimally this identified combination is reused in many applications. Once all combinations are identified, they can be optimised to maximise reuse. Bookshelving is a technique whereby development is carried out in advance of its requirement by any application. A complete set of documentation (requirements, design, implementation, and testing) is finalised and put on a ‘shelf’ in library for future use. The documentation should be extensive enough so that when pulled off the ‘shelf’ the package is ready to be used ‘as is’, without requiring further development.
The powertrain calibration process may have a duration of up to 2 years within a vehicle development life cycle varying from 4 to 6 years. Therefore the PCS software characteristics of size, maintainability, complexity, cohesion, coupling are an important factor for calibration timing, and therefore for vehicle development time. Also some special routines are built in software functionality in order to facilitate the calibration process. For example, in the PTEC strategy, there are routines testing some conditions and
deciding which calibration constant to use. In order to reduce software complexity, for example, a calibration guide tool should be used in order for the calibrator to decide the constant to use. This is a major difference between the gasoline and diesel PCS approaches.

4.2.7 Lessons learned

The gasoline PCS case study provides different aspects of evolution of a complex product and how the complexity growth was (or was not) managed during the evolution process. The following summarises these different aspects:

- the number and types of stakeholders increased. For example: the market became more differentiated, number of interacting subsystems increased, new environment agencies were created;
- new requirements appeared and constraints became more tightened. For example: new driveability requirements, more stringent emissions and fuel economy requirements;
- the need to comply with different powertrain configurations made the number of PCS applications increase dramatically, especially with the different ignition, injection and transmission systems created;
- emissions control being the main driver for PCS development made the PCS a tool for improving the effectiveness of catalyst converters. Catalyst converters made emissions control, driveability and fuel economy parameters very much dependent on each other. Therefore the PCS concept cannot help in making these parameters less coupled;
- although not considered as formal requirements, ease of calibration and reuse have always been drivers of the PCS development;
- the number of functions in the product increased. Examples of new functions are: on-board diagnostics II, cruise control. There is no one to one correspondence between PCS functions and requirements. The same set of functions may affect different requirements. For example: EGR control affects NOx control and driveability.
- the number of modes of operations increased. Initially, modes of operations were related only to protection of the system being controlled and to make possible the vehicle to work without the PCS, now there are different modes of operation depending on throttle position, temperature, altitude, engine state. As the emissions, fuel economy and driveability tests are performed considering various conditions, these conditions are reflected in the modes of operations. This improves the requirements x functions correspondence;
- the number and variety of sensors and actuators increased in order to meet different applications requirements. One sensor could be involved in the implementation of more than one function. By analysing the sensor x actuators correspondence it is not possible to derive clusters of functionality;
- the difficulty to partition functionality is reflected in the PCS software developed. From 1978 to 1995, it was written as a whole entity. It was not possible to find a correspondence between functions and piece of code;
- the choice for writing PCS software as a whole entity was in part due to the memory and execution timing constraints;
- the PCS evolution developed by changing incrementally existing software and hardware versions;
in order to meet different application configurations the number of versions of the PCS software increased dramatically. In 1995, there were 30 different application strategy paths. Any change in one strategy could require change in others requiring a great maintenance effort. In 1995, the software code size was 150KBytes;

as software and strategy groups were under different supervisors, the number of software versions for the same application strategy also increased;

besides the increasing amount of functionality embedded in the software in order to meet emissions, driveability and fuel economy requirements, the software was also supposed to meet reuse and ease of calibration requirements. This culminated with the launch of the PTEC strategy in 1995;

in February 1995, an organisational change put together strategy and software groups under the same supervisors creating the feature groups. The whole strategy would be now treated as groups of features. The partitioning problem, however, made the interfaces between features very complicated and the boundaries between feature groups not clear;

also in February 1995, a new software development process became operational, the application/feature change process. The process aimed to implement the feature-focused approach as a replacement to the old application-focused approach.

The gasoline PCS evolution provides an example of unmanaged complexity growth. Complexity grew through the increasing of number and diversity of stakeholders, number and diversity of requirements, number and diversity of functions, number and diversity of product solutions. The software was treated as a whole entity from 1978 to 1995. Changes were implemented locally without consideration of their effects on the whole. A complex problem was tried to be solved as a whole without using the basic concept of 'divide and conquer' or the basic concept of 'structure'.

Up to 1995, the software was used as the only answer to all the requirements imposed on the PCS development organisation. PTEC is an example of this fact. Ease of calibration and reuse were built in the software as if the software was not complex enough to implement its required functionality. Also memory and execution timing constraints were also compromised by the attempt of easing calibration and improving reuse built in the software.

From 1995 onwards, the PCS started to be considered as a system composed of features, its subsystems. As PCS functionality was incrementally implemented in the strategies and software code without visibility of the whole PCS picture, the system partitioning became a difficult task. Anyway, they started working with features with very complicated interfaces, dividing the complex PCS problem into more manageable ones.

Also in February 1995, Ford started recognising that the PCS product was not the only way of meeting all PCS requirements. Reuse, for example could be better managed with a different development approach within a different development organisation. These could also affect the ease of calibration. Therefore, the PCS development process and the PCS development organisation were recognised as part of the solution to manage the complex problem of evolving the PCS.

The gasoline PCS case study highlights some needs to manage complexity while developing a complex product:
• maintain traceability from stakeholders to their requirements, from these to functions, from these to implementation elements, so that a change in one of them can be quickly responded by the others;
• consider not only the product as the solution of the complex problem but also its life cycle processes and their performing organisations;
• keep visibility of the interactions or relationships among the various elements of the solution implementation, including not only the product elements but also the process and organisation elements.

4.3 The diesel PCS

The history of electronic engine control for diesel at Ford is much shorter (see Figure 4-29). There was some limited electronics on Diesel powertrain, primarily pump control. As emission legislation tightened up, the need for more electronics became apparent. Ford of Europe had the opportunity to actually design the system from scratch, using the knowledge it had from gasoline. Ford of Europe actually designed it as a system.

<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1992-1995</td>
<td>Lucas D-b-w on the Transit</td>
</tr>
<tr>
<td>1994</td>
<td>Beginning of the NEW DIESEL powertrain control system development</td>
</tr>
<tr>
<td>1996</td>
<td>Escort 1.8 IDI EEC V/Mondeo EEC V</td>
</tr>
<tr>
<td>1999</td>
<td>NEW DIESEL powertrain control system</td>
</tr>
</tbody>
</table>

*Figure 4-29: History of electronic engine control for diesel systems at Ford*

The 1996 model year diesel was the first step to apply engine controllers to diesels at Ford. Up till 1995 Ford was using entirely mechanical pumps with some very simple controllers, usually to control EGR and also some simple on/off valves on the pump to provide cold start advance and light load retard. They were generally external supplier’s systems (from Lucas, for example) and were very minor. On the 96 model (Mondeo) Ford added an EEC module for the first time to the diesel. In this case the pump was not fully electronically controlled, it still used mechanical determination of fuel quantity, but there was a closed loop injection timing control for the EEC module. So the conventional gasoline module architecture was used, software was written in assembler with very little structured design for the software. It did use a new kernel, but it really just lifted what gasoline were doing on a day to day basis, and with a very small team, delivered a controller. To give an idea of the complexity, Ford's gasoline systems at that moment were about 128KBytes of code and that first electronic controller for Diesel was 17KBytes, so it was about 1/5 to 1/6 of the size of a current Stage II gasoline control system. Otherwise, it was using an EEC processor, and it benefited from all those economies of scale that Ford already had on that EEC module.

However, since 1992, Ford has used a Lucas drive-by-wire system on the Diesel Transit. The electronic engine control consisted of just pump control, using the Lucas’ System. As the Mondeo 1996 mentioned above, the Escort 1.8 IDI 1996 also uses an EEC-V as a diesel control system. But the first big step up for diesel systems will certainly be the 1999 New Diesel powertrain control system, whose development process is described in the following sections.

The New Diesel PCS is a safety critical subsystem. As such it had to comply with automotive software development standards such as the MISRA guidelines and ESA-PSS-05. It had also to use a commercial-off-the-shelf microcontroller instead of Ford's EECs. In terms of functionality, the New Diesel PCS had the same level of functionality as modern EEC-V gasoline systems such as the PCS used for the Zetec
application. The New Diesel PCS was developed by a co-located team of 20 people including: applications engineers, systems architects, software engineers, module configuration engineers. It was completely developed by Ford of Europe in Dunton, England.

4.3.1 The structured approach

The structured approach used for the development of the New Diesel PCS is summarized in Figure 4-30. It is based on the Software Engineering Standards (PSS-05-0) of the European Space Agency (ESA) [Mazza et al., 1994] and on MISRA guidelines [MISRA, 1992]. Although these are a software engineering standards, they are being applied to a complete system development (hardware + software) in this context.

The structured approach comprises 6 phases: customer requirements phase, system requirements phase, architectural design phase, detailed design phase, transfer phase and operations and maintenance phase. These phases are presented in Figure 4-30 following the 'systems engineering V' with the requirements and design stages represented in the downstream side of the V and the validation stages represented in the upstream side. These phases are described as following [Ford Motor Company, 1995a]:

- **Customer Requirements phase**: shall produce a general description of the expected use and scope of the system, the operations that the customer want to perform with the system, all specific customer requirements with attributes, requirements such as costs, reliability and 'quality' as well as 'functional' requirements, any specific constraints imposed upon the system. All requirements are recorded so that they are traceable throughout the various design phases of the project. The outputs of this phase are: CRD, Management Plans SFMP/SR, SCMP/SR, SVVP/SR, SQAP/SR and Acceptance test plans SVVP/AT.
- **System Requirements phase**: shall produce a document which include intended functions, operating modes, interfaces (context diagram), application, operating environment, target cost, functional requirements, performance requirements, operational requirements (environment and interfaces), safety
requirements, verification requirements, acceptance testing requirements, documentation requirements, security requirements, quality requirements, reliability requirements, maintainability requirements, known design constraints. As the above requirements will be used to evaluate different design alternatives, it is important that the requirements are unambiguous and verifiable. The functional requirements and applicable design constraints will be embodied into an Essential Model using CASE tool TeamWork. The model shall be used to assist in the identification of any omission and/or ambiguities in the functional Customer requirements. Irrespective of whether they can be physically represented within TeamWork, all requirements will be recorded such that they can be traced through to specific design implementation details. The outputs of this phase are: SRD, Preliminary System Block Diagram, Preliminary Hazard Analysis (including an Initial System FMEA), System Test Plans SVVP/ST and Management Plans SCMP/AD, SVVP/AD, SPMP/AD, SQAP/AD

• Architectural Design phase: there are potentially many Architectural Designs that can be developed to meet the System Requirements. To aid consideration and evaluation of potential design, the subsystem will be deconstructed into features. For each of these features and ultimately the entire subsystem, alternative design solutions are generated and then evaluated against the Systems Requirements Document. This evaluation may be a subjective paper study or an objective assessment based on prototype evaluation, modelling and other feasibility studies. The culmination of this phase is the selection of the best alternative or 'Go for one' decision. After the selection process, the chosen design is refined. This design will be embodied in a Processor Implementation Model (PIM) using the CASE tool TeamWork. This will be used to provide design specifications for the system components - software, module and sensor/actuator hardware. Testing plans (integration testing) shall be written for each of these component parts (SVVP/IT). The outputs of this phase are: AD document (Sensor and Actuator Design Requirements, Module Design Requirements, Software Design Requirements, Installation Requirements), control system hazard analysis (including system FMEA), integration test plans SVVP/IT, management plans SCMP/DD, SVVP/DD, SPMP/DD, SQAP/DD.

• Detailed Design phase: the component parts are designed and manufactured. The components of this system fall into three categories: module hardware, software, sensors/actuators/wiring. As with the Architectural Design phase, there are potentially multiple designs for each component that will meet their specification. It is also possible that the detail design work will reveal that it is not possible to meet the specification and this will require an additional pass through the Architectural Design phase. The initial software 'Architectural design' is completed using a Task Implementation model (TIM), and from this the software module design is completed using a Module Implementation Model (MIM). Once MIMs for particular features and an operating Kernel have been established, subsequent passes through the process will require less emphasis on an initial TIM construction, but will require effort in reintegration existing feature MIMs with new parts of the design. Test plans are produced such that the manufactured components can be tested to ensure conformance to the component specifications detailed in the ADD. The outputs of this phase are: components, module and integrated software meeting specification from ADD, detail design document (describing the code and the various components), software user manual, procedures for system prove out and fault analysis, training guides, component FMEAs, Management Plans SCMP/TR, SVVP/TR, SPMP/TR, SQAP/TR.

• Transfer phase: once all the component parts have been manufactured they are integrated and tested according to the integration test plans detailed in the SVVP/ST written in the SR phase. Final
acceptance testing is carried out according to the SVVP/AT before the system is handed over to the customer. The outputs of this phase are: verified and validated system, system transfer document (detailing the system being handed over to the customer), final system FMEA.

- Operations and Maintenance phase: once the system has entered operation the system shall be monitored to ensure that the high level customer requirements are continuing to be met.

Ideally, these phases are executed in a sequential manner and thus the exit from each phase represents a key milestone of the project. Documentation is produced at the exit of each phase and this acts as the input specification for the subsequent phase.

### 4.3.2 Methods and tools

Figure 4-31 illustrates the use of structured analysis methods and tools for the New Diesel PCS development.

<table>
<thead>
<tr>
<th>PHASE</th>
<th>AUTOMATION TOOLS USED</th>
<th>DIAGRAMATIC PHASE REPRESENTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Requirement Phase</td>
<td>DOORS</td>
<td>Functional Requirements Document</td>
</tr>
<tr>
<td>Systems Requirement Phase</td>
<td>TeamWork</td>
<td></td>
</tr>
<tr>
<td>Architectural Design Phase</td>
<td>TeamWork</td>
<td></td>
</tr>
<tr>
<td>Detailed Design Phase</td>
<td>TeamWork</td>
<td></td>
</tr>
<tr>
<td>Construction and build</td>
<td>ClearCase</td>
<td></td>
</tr>
<tr>
<td>Workstation based testing</td>
<td>Test harness generator</td>
<td></td>
</tr>
<tr>
<td>Testbench based testing</td>
<td>Test harness generator</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 4-31: Methods and tools used for the New Diesel PCS development*
The customer requirements document led to a set of functional requirements which were hierarchically organised using DOORS (Dynamic Object Oriented Requirements Specification), a commercial tool developed by QSS. DOORS [QSS, 1995] aims to create structured requirements sets, keeping traceability between structured requirements sets and between a structured requirements set and external descriptive documents (see Figure 4-32). DOORS also provides means of sorting, selection and processing of requirements and linkage between a structured requirements set and an external structured tool (e.g. a CASE tool).

Figure 4-32: DOORS

The functional requirements and applicable design constraints were embodied into an 'essential model' which aimed to describe the PCS among all the other systems with which it interacts, describing all the necessary inputs, outputs and interface requirements. This 'essential model' led to the systems requirements and was developed using the CASE (Computer Aided Software Engineering) tool, TeamWork.

TeamWork is a commercial CASE tool developed by Cadre and implements the Yourdon structured analysis and design method [Yourdon, 1988] enhanced by the use of PAT (Process Activation Tables) and DT (Decision Tables) proposed in the Hatley-Pirbhai methodology (HPM)[Hatley & Pirbhai, 1988]. TeamWork implements some ideas embedded in the Yourdon and HPM methodologies:

- the separation between an essential model and an implementation model.
- the separation between data processing and control;
- a hierarchical representation of the system either in the essential model or in the implementation model;
- the use of Process Specifications (PSPECs) at the bottom level of the hierarchy to further describe the data processing;
- the use of STDs (State Transition Diagrams), DTs and PATs for control specification;
- the use of data dictionary.

The essential model models what the system is required to do, its functions, rather than how these functions would be physically implemented. It starts as a context diagram setting the boundaries between the system and its environment and describing their interactions. The essential model is represented in a hierarchy of DFDs (Data Flow Diagrams).

The implementation model models how the system is going to be implemented. It also uses mainly DFDs for system representation and decomposition. It starts with a context diagram, the architecture context diagram. The diagram models the actual flows of information between the system and the elements in the environment. The PCS is successively decomposed until the level where the control software is going to
be modelled. The sequential unveiling of the PCS reveals its sensors and actuators, the hardware input and output drivers, the software low level input and output drivers and the software high level input and output drivers, similarly to what is shown in Figure 4-21. For hardware, the model is used only to specify hardware components or describe the attributes of hardware components that are certain to be part of the product. The actual control software is modelled then into a PIM (Processor Implementation Model) showing the relationships among the various control features. Each PIM feature is further decomposed into a TIM (Task Implementation Model). The TIM models the tasks necessary to perform the processor function, the feature. For example, ‘monitor engine speed’ every 10 milliseconds is a task within the Idle Speed Control feature. When the TIM models are decomposed to a level when the various tasks can be described by pieces of code, the MIM (Module Implementation Model) level is achieved. The tasks in the MIM model are called modules and are further specified in M-SPECs which contain pieces of code written in C. Structure Charts can be used to represent the structure of software modules in terms of its constituent routines. Each MSPEC written in C will then correspond to a piece of code written in Assembler. The links between MSPECs and code is done by a UNIX tool called Clearcase. As a consequence, if the whole PCS decomposition up to code level could be represented in a structure diagram, it would look like a triangle with the PCS at the top vertex and the code modules at the bottom. Correspondence between the implementation model elements and the functional model elements is also pursued. Figure 4-33 illustrates the EGR feature decomposition and relationships with requirements and essential models.

![Figure 4-33: EGR feature decomposition structure](image)

The definition of the way to implement the features is aided by a behavioural modelling tool called X-Math. X-Math is part of the Matrix-X group of tools. It provides mathematical modelling and simulation of control systems. This is helpful, for example, to calculate PID (Proportional, Integral and Derivative) control parameters as a function of the vehicle configuration. EGR flow control, for example, uses this sort of aid.

At all levels of development, modules, application or PCS, comprehensive testing takes place. Test harness generators are used for automate the testing procedures.
4.3.3 Calibration

The diesel subsystems group within PCSE developed a Calibration Guide tool. The tool simulates the software features and can be used by calibrators in order to find the best set of calibration constants for a given application. Also, as the tool makes extensive use of graphical notation, it provides the calibrator with a better understanding of the software algorithms without the need for him to be a software specialist. The use of the Calibration Guide to ease calibration simplifies the PCS in the sense that it is not anymore a PCS requirement to identify which application is running. When the calibrator runs the Calibrator Guide he is aware of the conditions to which the PCS is going to be applied.

4.3.4 Configuration management

The New Diesel PCS development process has a configuration management procedure in order to store and control the changes of the work-products (hereafter called configuration items or CI) produced by the New Diesel PCS development process.

The New Diesel development is based on modelling. Many CIs are models such as those produced in DOORS, X-Math and TeamWork. Clearcase is the configuration management tool used by New Diesel. Clearcase is a software tool supplied by Atria. It provides the changes control environment for programmers. It provides all CI storage. It is a UNIX-based file management system and does not embody the necessary functionality to manage models. All models must then be reduced to a file before they are managed. The modelling tools provide a mechanism to reduce their work-products to a file. Data is exported from the design tool into a dump-file and manually placed into the configuration management storage. With the exception of models, work-products are created and modified within the Clearcase environment. The environment provides change control at the point of use. It also provides change records submitted as comments by the designer at the time of checking-in or checking-out a CI. Change request management and defect tracking is provided by Cleartrack, which is also a software tool provided by Atria. For the models, TeamWork has a sophisticated mechanism for controlling access to model configuration management functionality based on UNIX permissions on a per model basis. There is a UNIX closed user group known as CM. Only this group and root have permissions to create, delete, baseline, unbaseline a model.

The New Diesel configuration management policy is based on the ‘branch and merge’ method of parallel development [Ford Motor Company, 1996f]. Each feature is developed on its own ‘feature branch’. Any modifications to a CI will require the creation of a temporary development branch from this ‘feature branch’ which is then used to develop the necessary changes. When the changes are complete, any development that has been checked-in to the ‘feature branch’ will have to be merged into the development branch ready for testing and reviewing. After satisfactory quality has been met, the CI is merged back into the ‘feature branch’ thereby completing the development of that change.

Clearcase uses the UNIX filestore as a structural standard. It controls a versioned object base (VOB) that takes the form of the underlying UNIX filestore. Figure 4-34 illustrates the New Diesel directory structure. A1 is a sub-directory that holds application documentation, models (dumped), files and a makefile for the A1 application. A1’s models sub-directory holds three types of dumped models: DOORS, Xmath and TeamWork models. There are 8 feature sub-directories shown. The ‘pd’ sub-
directory is expanded to show the structure for each feature. 'pd' has 2 tasks. Each task sub-directory holds all the information required to build a task and test it. Project documentation is also stored in the same structure in the following subdirectories:

- Deliverables: PRD, SRD, ADD, SDS, Calibration Guide, HCR, ARM, PHA, FMEA and Compliance Matrix;
- Admin_Documents: meeting minutes, project log, organisation chart and roles and responsibilities documents;
- Project_plans: SPMP, SVVP, SQAP, SCMP, Timing Charts, Risk Management documents, Process Deviations, Project Audit Reports etc.
- Process_documents: system development process, work-instructions, training material and standards documents.
- Quality_records: review error logs, review meeting minutes, change requests, metrics etc.

```
[//vol/new_diesel_196]
[docs]          makefile
[models]      application files
[DOORS_dumpfile] xmath_dumpfile twk_dumpfile task1/  task2/  twk_models/  include/
task1.c      task1.cal  meta/  test_file.out results
feature_p_model_dumpfile  feature_s_model_dumpfile

Figure 4-34: New Diesel PCS Clearcase directory structure
```

Within the sub-directories there is a Clearcase version tree of files. Figure 4-35 illustrates the Clearcase version tree for a hypothetical feature 'zz'. Only the 'makefile' development is shown, all other files will have a similar structure.

```
Figure 4-35: Clearcase version tree for feature 'zz'
```
For TeamWork models, Figure 4-36 illustrates the TeamWork model derivation tree for a feature. Models appear with indentation in the TeamWork model index.

<table>
<thead>
<tr>
<th>EI</th>
<th>Essential Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>Application Model</td>
</tr>
<tr>
<td></td>
<td>Feature Processor Models</td>
</tr>
<tr>
<td>zz_p.esp_v1</td>
<td>first variant of the feature</td>
</tr>
<tr>
<td>zz_p.esp_v1.001</td>
<td>version 1 of the feature variant</td>
</tr>
<tr>
<td>zz_p.esp_v1.002</td>
<td>version 2 of the feature variant</td>
</tr>
<tr>
<td>zz_p.esp_v2</td>
<td>new algorithm variant</td>
</tr>
<tr>
<td>zz_p.esp_v2.001</td>
<td>small change</td>
</tr>
<tr>
<td>zz_p.man_v1</td>
<td>new feature variant (e.g. new technology)</td>
</tr>
<tr>
<td>zz_p.man_v1.001</td>
<td>small change</td>
</tr>
<tr>
<td>zz_p.man_v1.002</td>
<td>another small change</td>
</tr>
<tr>
<td></td>
<td>Feature Software Models</td>
</tr>
<tr>
<td>zz_s.esp_v1</td>
<td>first variant of the software</td>
</tr>
<tr>
<td>zz_s.esp_v1.001</td>
<td>version 1 of the feature variant</td>
</tr>
<tr>
<td>zz_s.esp_v1.002</td>
<td>version 2 of the feature variant</td>
</tr>
<tr>
<td>zz_s.esp_v2</td>
<td>new algorithm variant</td>
</tr>
<tr>
<td>zz_s.esp_v2.001</td>
<td>small change</td>
</tr>
<tr>
<td>zz_s.man_v1</td>
<td>new feature variant (e.g. new technology)</td>
</tr>
<tr>
<td>zz_s.man_v1.001</td>
<td>small change</td>
</tr>
<tr>
<td>zz_s.man_v1.002</td>
<td>another small change</td>
</tr>
</tbody>
</table>

Figure 4-36: TeamWork model derivation tree for a feature

4.3.5 Reuse and evolution

Like in the gasoline case, the development of a diesel PCS should only start once a formal Product Letter defining the program has been issued. It is the responsibility of the VC Program Offices to issue the Product Letters. From the Product Letter, the Application Engineer within the PCSE identifies all functional content that the powertrain control system needs to provide and describes this in the PRD (Project Requirements Document). In addition to this, the PRD also contains other requirements that affect the design of the Powertrain Control System, which might include requirements for business, marketing, development tools, service, security, etc. As the requirements change through related Program Module Teams (PMT) or new Product Letters, the PRD is updated to reflect the latest information and the impact of the changes is assessed.

An Application Requirements Matrix is created for the project to ensure that common requirements have common solutions. The matrix supports the System Requirements phase assisting the identification of common functionality across multiple vehicle applications. This matrix should be a simple way of understanding how different requirements affect different applications within the programme. It could be as simple as a single matrix of applications against requirements with a check mark where a requirement applies to a given application. The ARM will be updated based on revisions to the PRDs for each application and is also updated if the requirements should subsequently change.

For software only evolution, which is the case for most of recent New Diesel PCS applications, many of the work-products developed during the first issue of the PCS are reused, e.g.: SPMP, SQAP, SCMP, SVVP, Essential Model, Concept FMEA, Preliminary Hazard Analysis, PCS architecture (sensors, electronic module, actuators), electronic module architecture (microcontroller and hardware drivers), microcontroller architecture (CPU and low level software drivers). The need for change is analysed from the CPU architecture (see Figure 4-37) which includes all features and high level software drivers. This CPU architecture is the top level PIM of the original New Diesel PCS development. As mentioned in Sections 4.3.1 and 4.3.2, the PIM is decomposed into TIMs for each feature and each task is decomposed.
into MIMs. The set of PIM, TIMs and MIMs specific to an application is called Application Model. Each module in the MIMs will correspond to a codified software module.

The development of a new application, starts then from:

- an existing entire Application Model;
- putting together existing features, tasks or modules from different existing applications to compose a Application Model.

The adoption of an Application Model instead of the original textbook proposed PIM, TIM, MIMs is due to the fact that when developing the New Diesel PCS, the PIM developers tended to go too far into the TIM. TIM developers tended to go too far into the MIM, almost to the point that the code resulted directly from the model.

Having requirements from Applications Engineers or from Diesel Engineers when calibrating an engine, a System Architect and a Software Engineer work together to compose and/or modify an Application Model for a particular application. The System Architect and the Software Engineer work together in order to avoid the fact that the system architect propose an algorithm that cannot be implemented. The Application Model is developed up to a point in which the software development is a direct derivation of it. When the software modules are codified the whole application software is mounted through a 'c_make' command. The idea is to implement the changes request very quickly and deliver software prototypes to be tested by the requester. When the requester approves the software delivered, than the requirements that originated the changes are formally documented and linked to the change performed.

The configuration management tools mentioned in Section 4.3.4 helps in this purpose, as change requests can be managed by Cleartrack and change history and versioning by Clearcase. PCSE's diesel engineers are making an effort to keep one only root from which all applications can be developed by keeping track and justifying all modifications made to existing features, tasks and modules in the root Application Model. This root Application Model can, of course, have versions and variants.

From the time software starts to be written, the Calibration Guide (mentioned in Section 4.3.3) begins to be developed from the Application Model. The identified calibration constants and the approved software are codified and transferred to the PCS corresponding memories.

As much reuse is being obtained through this procedure, much of the testing done for the original New Diesel PCS development is being replaced by reviewing in the evolving versions and variants. This, also, helps to shorten the time and effort to develop the PCS.

As features get more complex, the trend is to have 'makefiles' per feature instead of per application as it is now. However, the current way of evolving the diesel PCS fits the current Ford organisation well. Another trend is to have a greater involvement of the calibration engineer requiring software changes when the Application Model starts to be put together and modified. Current Ford's organisation however does not facilitates this involvement as it requires to go through the formal gasoline PCS process of URDs, EMRs, etc... The expected more flexibility of the Visteon's organisation, on the other hand, opens the way for an increased involvement.
Figure 4.37: An example of a top-level Application Model – a CPU architecture
4.3.6 Comparison with the gasoline

Figure 4-38 shows a comparative analysis of a gasoline PCS with the New Diesel PCS.

Figure 4-38 highlights a more efficient diesel PCS development organisation than the gasoline PCS one. With a co-located people of 20 people the diesel organisation was supposed to meet the requirements of 10 applications over the period 1997-2002. It represents a rate people/application of 2 against a gasoline rate of 4. This comparison is done considering PCSs with almost the same complexity in terms of functionality and hardware.

<table>
<thead>
<tr>
<th>CRITERIA</th>
<th>GASOLINE</th>
<th>DIESEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of developers</td>
<td>~400</td>
<td>~20</td>
</tr>
<tr>
<td>Number of scheduled applications to be delivered in the period 1997 - 2002</td>
<td>~100</td>
<td>~10</td>
</tr>
<tr>
<td>Location</td>
<td>9 feature groups in the USA, 1 feature group in Europe</td>
<td>Co-located in Europe</td>
</tr>
<tr>
<td>System complexity - I/O - Features</td>
<td>96 pins used (Zetec), 14 Analogue l/P, 8 DCO</td>
<td>100 pins used (New Diesel), 29 Analogue l/P, 16PWM, CAN, more complex</td>
</tr>
<tr>
<td>Safety critical system</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Evolutionary process</td>
<td>Very complex and time consuming</td>
<td>Simple and quick</td>
</tr>
<tr>
<td>Evolve from</td>
<td>Strategy Code</td>
<td>Architecture design called Application Model</td>
</tr>
<tr>
<td>Calibration process</td>
<td>Calibration constants and tests in the code (21000 variables to calibrate)</td>
<td>Calibration procedures developed separately</td>
</tr>
<tr>
<td>Customers</td>
<td>Program Office, PMT, PAT, calibration community</td>
<td>Mainly Diesel Engineering</td>
</tr>
<tr>
<td>Non-functional requirements considered</td>
<td>Non-functional requirements considered: cost</td>
<td>Non functional requirements considered: cost, reliability and safety (hazard analysis and FMEA, New Diesel is a safety critical project) and manufacturability (change of the processor on New Diesel PCS must consider manufacturing feasibility and end-of-line test study).</td>
</tr>
<tr>
<td>System documentation</td>
<td>PRD, URD, EMR, HCR, SDS (not fully implemented), widespread</td>
<td>Complete system documentation, centralised</td>
</tr>
<tr>
<td>Design quality assurance</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Performance modelling</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

*Figure 4-38: Comparative analysis diesel x gasoline PCS*
engineering process. Also, the gasoline PCS is evolved from a ladder logic or, in the best case, C coded strategy document with very low visibility of the interactions among features. The diesel PCS is developed from an Application Model that clarifies the interactions among all features involved. The gasoline PCS modifies the software at a very physical level without having a clear picture of how the stakeholder requirements are going to be affected. The diesel PCS maintains track from requirements, functions (essential model), features, task, modules, code (implementation models) so that any change can be reversed back to the requirement it affects. As the PCSs get more complex, the gasoline PCS is on the way to collapse and the diesel PCS to repartition its original modules to keep the links from requirements to code as tree-like as possible, with reduced coupling between features, tasks and modules in between.

4.4 Conclusions

The gasoline PCS case study highlighted some needs to manage complexity while developing a complex product:

- maintain traceability from stakeholders to their requirements, from these to functions, from these to implementation elements, so that a change in one of them can be quickly responded by the others;
- consider not only the product as the solution of the complex problem but also its life cycle processes and their performing organisations;
- keep visibility of the interactions or relationships among the various elements of the solution implementation, including not only the product elements but also the process and organisation elements.

The gasoline PCS focused on the PTEC application strategy as a means of meeting functionality, constraints, ease of calibration, reuse and shorter development time (see Figure 4-39). The result was a strategy with 700Kbytes of code that would probably never become an application software. As interfaces among features were increasingly complex after successive modifications, the strategy size will increase making the maintenance effort even greater.

<table>
<thead>
<tr>
<th>Product: PTEC strategy</th>
<th>Functionality</th>
<th>Constraints</th>
<th>Ease of calibration</th>
<th>Reuse</th>
<th>Development time</th>
</tr>
</thead>
</table>

Figure 4-39: Product focused gasoline PCS development approach

The diesel PCS development team realised that the system solution to meet functionality, constraints, ease of calibration, reuse and shorter development time contained not only the PCS software product but also other work products like the Calibration Guide, process elements and organisation elements as illustrated in Figure 4-40.

The result from the diesel PCS approach was a shorter development life cycle, improved reuse, easier calibration without compromising memory size and execution time constraints and product functionality meeting the functional requirements.

As the diesel PCS product used a structured development approach, it can be evolved in such a way that keeps its complexity, in terms of interactions among its component subsystems, manageable. The effort
has been to keep a tree-like structure from requirements, through functions, to software modules following the structured approach represented in Figure 4-41. The structured approach consists of requirements, functional and physical analysis.

The product focused gasoline PCS development led to an ever increasing complexity. The lack of attention to the gasoline PCS software because it costs US$ 10 out of a PCS costing US$400 is, by itself, a demonstration of how product and component focused the gasoline PCS software has been over the period 1978-1998. Although it costs so little, it affects enormously the vehicle calibration time which is planned to last 18 months out of a total vehicle development time of 48 months. Therefore, besides the obvious effect on the vehicle development life cycle process, the PCS may have an important role to play in the organisation competitiveness as it may affect its ability to shorten time-to-market.
The diesel PCS development approach demonstrates the benefits of a structured approach to product development. Although, when developing the product solution, it did not formally considered processes and organisation as a means of meeting some of the requirements, the final overall solution included these elements. The gasoline PCS demonstrated how related product, processes and organisation are. Therefore, the way forward points towards a structured approach including product, process and organisations as the potentially interacting elements of the final system solution. This is the realm of integrated development. A framework for the integrated development of complex products is then proposed in Chapter 5.

Having identified the needs for the development of complex products, the thesis continues by proposing and justifying a framework for the development of complex products. The following chapter, Chapter 5, presents the framework and Chapter 6 clarifies the key concepts supporting it.
Part II
Concept
Chapter 5
The total view framework

This chapter aims:
• to propose integrated development as an approach for the development of complex products;
• to propose a generic model for integrated development;
• to propose a framework for integrated development, that integrates systems engineering and concurrent engineering;
• to describe the elements of the framework.

5.1 Background

The evolution of product development in the automotive industry is moving from the traditional component approach to systems engineering, as shown in Chapter 3. The traditional component approach was enhanced by concurrent engineering and great effort was spent in the development of computer tools to integrate component design and component life cycle processes such as manufacturing, assembly and disassembly. Although the systems engineering concept being adopted by the automotive industry has a scope that goes beyond the product, its focus is still the product. For example, the systems engineering concept at Ford sets a priority order of optimisation goals: first, Ford Motor Company, second, the vehicle program, third, the vehicle subsystem. However, in practice, out of the 15 attributes that orientates product development: 13 refer to product, 1 to life cycle processes and 1 to the Ford organisation.

In the space industry, the systems engineering approach is moving from the traditional sequentially phased, review orientated, document based approach towards concurrent engineering, integrated product teams and trade-off based approach. They are moving from a very much product performance and risk avoidance focused product development towards a balanced solution that considers development lead time, life cycle cost, risk management with product performance still meeting requirements. Examples of this novel approach is the ‘cheaper, faster, better’ NASA’s approach (see Chapter 3, Section 3.2.4) and the DoD’s IPPD concept (see Chapter 2, Section 2.3.2).

The trends in both industries point towards: a broader scope of product development that includes life cycle processes and organisation requirements from the outset; the adoption of a systems engineering approach for product development; the phasing of product development based on concurrent engineering principles, including integrated product teams. All these are elements of integrated development as defined by [DoD, 1996] and as reviewed in Chapter 2.

This chapter proposes and justifies integrated development as an approach for the development of complex products. Section 5.2 proposes and justifies a generic model for integrated development. Section 5.3 proposes and justifies an integrated development framework containing all elements mentioned above. Sections 5.4, 5.5 and 5.6 describe the elements on the so called dimensions of the framework. The framework provides a total view of product, process and organisation through requirements, functional and physical analysis. In doing so the framework highlights the contribution of systems engineering vis a vis concurrent engineering.
5.2 Generic model for integrated development

Integrated development is a product development approach for the integrated and concurrent development of a product, its life cycle processes and their performing organisations. It takes into consideration, from the outset, life cycle process and organisation requirements which, together with the product specific requirements, drive the product development process. Integration of product, life cycle process and organisation takes place by recognising that their attributes affect each other and that a balanced solution that satisfies stakeholder requirements cannot be achieved without consideration of the relationships among those attributes. The focus of the integrated development effort is to develop a product. However, integrated development recognises from the outset that a product has a life cycle and that each life cycle process is performed by an organisation of resources. The life cycle processes and the organisations constrain or are so constrained by the product design that, in order to avoid late and costly changes, their requirements and attributes are considered from the early stages of product development. The scope of integrated development includes, definitely, a product and its life cycle processes. Some life cycle process performing organisations may also be included.

Figure 5-1 is an entity-relationship diagram that summarises the above definition and proposes a generic model for the integrated development of complex products. The model is intended to be recursively applicable to any product element that composes the complex product breakdown structure. The development of a complex product, its subsystems and components involves many development agencies. The development agencies can be different departments in the same company or even different companies. For example, for a car, Ford’s Product Development organisation is the car development agency; Ford’s CAPE (Core and Advanced Powertrain Engineering) is the powertrain development agency; Visteon-PCSE, another company, is the PCS development agency. Any one development agency is responsible for developing a product, that could be the complex product itself or any of its subsystems or components, together with the product life cycle processes (LCP) and (some of) the LCP performing organisations. The development agency is the product development performing organisation.

![Figure 5-1: Generic model for the integrated development of complex products](image-url)
Figure 5-1 illustrates that there are many stakeholders with many requirements. Some requirements may be common among stakeholders. Requirements are input to the development agency. In order to meet the requirements, the development agency develops products, their life cycle processes and, some life cycle process performing organisations. The set of life cycle process performing organisations includes at least the development agency itself. It refers, for example, to the fact that a product development organisation needs to adapt itself to the required changes in product and life cycle process. Each product has many life cycle processes. Each life cycle process is performed by a performing organisation. Each product, each life cycle process and each performing organisation, including the development agency, have many attributes. These attributes affect the stakeholders' satisfaction or dissatisfaction. The same set of attributes may affect different stakeholders. From them, new requirements may be raised and the evolution cycle starts again. For example, given the product is a Ford car, its life cycle processes are: development, manufacturing, distribution, training, use, service and disposal. The organisations within the scope of the integrated development effort are the ones which perform Ford's core business processes: the FPDS (Ford Product Development System), FPS (Ford Production System), Delivery-to-Order and After-Sales Support. Organisations performing training, use and disposal are outside the scope of the Ford car development effort.

The VDI Guideline 2221 'Systematic Approach to the Design of Technical Systems and Products' [Cross, 1989] divides the development process into two domains: a problem domain and a solution domain. This thesis proposes the elements contained in each domain as following. The problem domain contains the elements that define the problem to be addressed by the development agency. These elements are: the stakeholders; their needs expressed by first draft requirements; the current product, processes and organisation attributes as perceived by the stakeholders. The solution domain contains the elements of the solution that addresses the problem in focus. These elements are: the requirements as perceived by the developers; the development agency; the product; its life cycle processes; life cycle process performing organisations; product, processes and organisation attributes as specified by the development agency. The line separating the problem and the solution domains cuts across the requirements and attributes boxes. This is to express the gaps that exist:

- between the real needs of stakeholders and the way these needs are translated into requirements as perceived by the development agency;
- between the attributes as specified by the development organisation and as perceived by the stakeholders.

The generic model is based on the fact that the product life cycle processes and their performing organisations are highly affected by the product design. That is the main reason to consider, not only the product, but also its life cycle processes and their performing organisations as results of the product development process. Examples of the dependency of life cycle processes and their performing organisations on the product design are: the fact that 70% of production costs are determined during the conceptual stages of a product [Bedworth, 1991]

Modern approaches to the integrated development of complex products (e.g. concurrent engineering [Prasad, 1996a], IPD [Prasad, 1996b], IPPD [DoD, 1996]) also explicitly recognise that the result of the effort of a product development organisation is not only a product itself but also its life cycle processes. The generic model is flexible about the inclusion, in the solution domain, of the organisations that
perform life cycle processes other than the development process, e.g., manufacturing, testing, distribution, training, support, disposal. In case they are not part of the solution domain, they would be considered as stakeholders. Their requirements would then be considered in order to successfully develop the product. The product development organisation would then develop a product and its life cycle processes taking into consideration the requirements of these other organisations involved. The attributes of the product and of its life cycle processes must provide an indication of how well the life cycle process organisations' requirements are being met. In the minimum, the generic model of integrated development, proposed in Figure 5-1, integrates product development, life cycle processes development and product development organisation evolution.

Putting the development organisation in the solution domain reflects the need that modern manufacturing organisations must be able to change themselves in order to remain competitive. The constant verification of how attributes are reflecting stakeholder requirements provides an indication of the need to change.

The need for the integration of product, processes and organisation is reflected by the fact that stakeholders may pose requirements to be met not only by the product itself but also by its life cycle processes (e.g. car availability can be met by the distribution process not only by the car reliability) or by the development organisation (e.g. time to market, productivity, return on investment). Also the analysis of the relationships and trade-offs across all the attributes of product, processes and organisation opens up the possibility of a balanced solution for the stakeholders. Product, processes and organisation would then behave as a whole integrated system.

Clark & Fujimoto (1994) used three quantities to evaluate product development performance: product quality, lead time and development productivity. These quantities map onto product, process and organisation, respectively. Product quality is essentially a product attribute. Lead time refers to the time for developing one product to meet market need. Productivity refers to the use of the resources in the product development organisation, e.g. quantity of engineers to produce a given number of cars. By integrating product, processes and organisation development, a balanced value for those three quantities are being pursued.

The more complex a product is, the more necessary it is to adopt an integrated approach [Prasad, 1996b]. For a complex product, no one person or group in isolation can manage a multidisciplinary product design or understand in depth all of its life cycle processes [Andreasen & Hein, 1987]. IPTs [Usher et al., 1998] are set to manage the attributes of the product and of its life cycle processes as they are highly dependent on each other. The attributes of the product development organisation may also be dependent on the product attributes and how these attributes match with the life cycle process requirements. A bad matching would cause the need for late changes increasing cost, lead time and decreasing productivity.

In summary, the model in Figure 5-1 proposes that:

- the integrated development of complex products must be a requirements driven process;
- the integrated development of complex products integrates product, its life cycle processes and their performing organisations (or at least the development organisation);
- the integrated development of complex products searches for a balanced solution compromising product, processes and organisation attributes as the emergent properties of a whole integrated system.
5.3 The total view framework

In order to move from requirements to attributes in the integrated product development process, it is proposed here a total view approach. The total view approach aims to help to manage the complexity associated not only with a product itself complex, but also with the interactions among the product, its life cycle processes and organisation attributes. Figure 5-2 presents a framework for the total view approach, hereafter, called total view framework.

The 'total view framework' is a modelling framework that integrates the product, its life cycle processes and their performing organisations (e.g. product development organisation, production organisation) throughout the requirements, functional and physical analysis processes, at all levels of the product hierarchy, deriving attributes as emergent properties of a whole integrated system.

The term 'modelling framework' is defined by Vernadat (1996) as a collection of modelling principles, methods, or tools relevant for a given domain of application. In the context of this thesis, being a 'modelling framework' means that the elements of the framework are models (product model, process model, organisation model, requirements model, functional model, physical model) at the various levels of the system hierarchy.

The 'total view framework' encompasses systems engineering (SE) and concurrent engineering (CE). Figures 5-3 and 5-4 present how the 'total view framework' is related to SE and CE respectively.

![Figure 5-2: The total view framework](image-url)
The IEEE-Std-1220-1994 [IEEE, 1995] defines systems engineering as 'an interdisciplinary collaborative approach to derive, evolve, and verify a life cycle balanced system solution that satisfies customer expectations and meets public acceptability.' However, the established focus of that standard is on product-oriented systems such as the automobile, aeroplane, or computers. As proposed in Figure 5-1 and reflected in Figure 5-2, the solution domain does not consist only of a product but also of its life cycle processes and [some of] their performing organisations.

Also, the EIA 632-I standard [EIA, 1997] was developed to be applicable only for the development of end products and their subsystems. Only requirements are determined for the product life cycle processes.

Based on those standards, the scope of systems engineering within the 'total view framework' can be defined as in Figure 5-3. The grey solid boxes in Figure 5-3 define the SE scope, whereas the white boxes identify the elements left out of the system solution by the systems engineering approach as described by modern standards.

Concurrent engineering is defined by the Institute of Defense Analysis (IDA) Report R-338 [IDA, 1986] as 'a systematic approach for the integrated, concurrent design of products and their related processes, including manufacture and support. This approach is intended to cause the developers, from the outset, to consider all elements of the product life cycle from concept through disposal, including quality, cost, schedule, and user requirements.' Prasad (1996a) states that concurrent engineering is founded on eight fundamental principles: early decision discovery, early decision making, work structuring, teamwork affinity, knowledge leveraging, common understanding, ownership and constancy of purpose. However,
the most frequent terms found in concurrent engineering literature (e.g. drawings, geometry, raw material, manufacturing, CAD/CAM) [Prasad, 1996a], [Parsaei & Sullivan, 1993], [Syan & Mellon, 1994] provide an indication that CE is being applied for component design. The component aspect of CE manifests itself in two ways: the item being engineered does not have to be system and the life cycle processes are very often considered in isolation from each other. For example, a bonnet of a car can be object of CE and it can be designed for manufacturing. The design aspect of CE occurs because the use of CE tools pre-supposes a given functional architecture. Therefore, when CE starts, requirements and functions are considered as given. Only the physical design will be worked on. Also Gardiner (1996) outlines that the phenomenon of "emergence" or 'emergent properties' has not been a feature of the concurrent engineering literature. Such phenomenon is essential in complex product development and is a consequence of the fact that a complex system is different from the sum of its parts. The composition of parts may show behaviour that is explicable but is not necessarily an obvious outcome of the interaction of the parts [Hitchins, 1992].

On the other hand, CE integrates product, processes and organisation development (e.g. [Kunz et al., 1995]). Prasad (1996a) presents the integration of five model-classes within a structure of a model-based CE system: enterprise model-class, specification model-class, product model-class, process model-class and cognitive model class. Current IT capabilities allow these models to be integrated by PPO (product process organisation) files and databases and to have their key parameters in multiple perspectives through trade-offs and constraints.

Therefore, the CE scope within the 'total view framework' can be defined as in Figure 5-4. Figure 5-4

![Figure 5-4: Concurrent engineering scope against the total view framework](image-url)
reflects the fact that CE integrates product, process and organisation but for component design. In order to be applied for the integrated development of complex products, CE needs to be applied to all levels of the hierarchy on which the complex product development is structured and needs to provide means to analyse product functionality and life cycle process requirements simultaneously from the beginning.

In summary, the 'total view framework' applies the systems engineering process for the concurrent engineering of product, processes and organisation.

5.4 The integration dimension

The integration dimension defines the elements to be integrated within the system to be developed. This thesis proposes an expansion of the system concept proposed by modern systems engineering standards to include also the organisations performing the life cycle processes.

Figure 5-5 compares the constituents of a system as according to the EIA-632-Std with the ones proposed in this thesis. According to the EIA-632-Std [EIA, 1997] a system consists of an operations product and its associated processes. An operations product performs the operations function of a system and is composed of one or more end products. The associated processes are implemented using enabling products that perform the non-operations functions of the system -- develop, produce, test, deploy, and support the end product(s), train the operators and maintenance staff of the end product(s), and disposition of end products no longer viable for use. Each end product and enabling product is composed of one or more of the following: hardware, software, personnel, facilities, data, materials, services, and techniques.

The black boxes in Figure 5-5 represent those elements that are within the scope of the development
effort. The grey boxes represent elements that will be further decomposed into their constituent elements. The white boxes represent those elements for which only requirements are determined and no further development takes place. Turning to the terminology proposed in the generic model of integrated development (Figure 5-1), for the black and grey boxes, requirements and attributes will be determined and, for the white boxes, only requirements will be determined. Therefore, according to the EIA-632-Std, although it is recognised that the system consists of the operations product and associated processes, as the end products are defined, the requirements for the associated processes are identified and collected. Therefore, no concurrent engineering of product and processes is considered.

On the other hand, the 'total view framework' aims to identify the attributes of, not only the product but also its life cycle processes and their performing organisations and the relationships among those attributes. Those attributes are identified by mirroring a systems engineering process through three analysis processes: requirements, functional and physical analysis. The analysis processes are applied to every level of the product breakdown structure. The analysis processes start by capturing and analysing product, process and organisation requirements, concurrently, from the outset. The analysis continues through simultaneous functional and physical analysis of product, process and organisation. Therefore, the system, object of the analysis, is composed not only by the product itself, but also by the life cycle processes and their performing organisations as shown in Figure 5-5.

The EIA-632-Std considers the operations process embedded in the operations product. The 'total view framework' considers the operations process as part of the life cycle processes and includes the operations organisation as part of the performing organisations. The life cycle processes (LCP) are divided into the operations process and other life cycle processes and, in the same manner, the LCP performing organisations are divided into operations organisation and other performing organisations. The operations product is made up of one or more end products. The operations process defines the processes, activities and/or tasks that are performed using the end products. The operations organisation relates the end products to other organisational resources such as: people, technology, etc. Other life cycle processes and their performing organisations are implemented by the use of enabling products. Enabling products perform the non-operations functions of the system – develop, produce, test, distribute, and support the end product(s), train the operators and maintenance staff of the end product(s), and disposal of end products no longer viable for use.

The following paragraphs define the elements of the integration dimension: product, process and organisation.

The product element refers primarily to the end products. End products perform the operations function and are delivered to an end user. The product element may also include enabling products, depending on the scope of the development project. Product models capture the information about the interactions among product functions and product components.

The process elements refer to the life cycle processes of the end product. Generally speaking, a process is a set of interrelated activities, actions or tasks that, together, transform inputs into outputs [EIA, 1997]. The IEEE-Std-1220-1994 [IEEE, 1995] defines the following as eight essential life cycle processes: development, manufacturing, test, distribution, operations, support, training and disposal. Process models capture the information about the interactions among the activities, actions or tasks in the processes.
The organisation element refers to the organisations that implement the life cycle processes. An organisation is a structured set of resources which can play a role in the realisation of a certain class of tasks. These resources can be human (people) or technological (machines, applications, etc). Organisation models capture the information about the interactions among these resources in order to carry out a process. The process elements refer to the life cycle processes of one product. The organisation element refers to the organisation that performs these life cycle processes. Very often the organisation is set up to perform a given life cycle process for many products. For example: a production organisation may be set up to produce 1000 different products a day; a service station is set up to support many cars of different types.

The EIA-632-1 [EIA, 1997] and the IEEE-Std-1220-1994 [IEEE, 1995] defined a building block structure for a system as shown in Figure 5-6. The definition of ‘building block’ is provided in the EIA-632-1 [EIA, 1997]: “A building block represents the conceptual framework of a system for organising the requirements, work, and other information associated with the engineering of a system.” The building block consists of (1) the system, (2) one or more end products that make up the operations product, and (3) the ensemble of associated processes. A complex system is composed of a hierarchy of building blocks. Again, in the total view framework, an expanded building block structure is proposed as shown in Figure 5-7. One more set of elements is included in the proposed building block structure: the life cycle process performing organisations. Another difference from the standards is the fact that the operations process and the operations organisation are considered as part of the process and organisation sets, respectively. This happens because, for modelling purposes, the operations process and the operations organisation are modelled as process and organisation, respectively, and not as part of the product model. The standards consider the operations process embedded in the operations product. In other words, the standards consider the operations process and operations organisation implicitly included in the operations product, whereas the ‘total view framework’ makes them explicitly separate. The shadowed region in Figure 5-8 highlights the operations product, process and organisation which are
supposed to meet the operational requirements. The other processes and organisations are related to meeting the non-operational requirements of the system. However, as mentioned in Section 5.2, the scope of development may not include the development of every LCP performing organisation. Some of

Figure 5-7: Building block of highest complexity

them will be outside the development effort. In such cases they will be considered as stakeholders (see Figure 5-1). The building block structure varies from the one of highest complexity in Figure 5-7 to the one of least complexity in Figure 5-8. Figure 5-8 calls the attention to the fact that at least the development organisation should be considered within the integrated development effort.

Figure 5-8: Building block of minimal complexity

The proposed building block structure consists of (1) the system, (2) one or more end products, (3) their life cycle processes and (4) the organisations that perform those processes. If any of the end products needs further definition, the building block also includes the subsystems associated with it. An end
A product can be self-contained in terms of its use and operations (e.g., an aircraft, an automobile, a communications satellite, a nuclear reactor, a space vehicle that is delivered to an operator), or it can be an article that has no use independent of a larger end product, but that is developed as a unit (e.g., a radio for an aircraft, a powertrain for an automobile, a solar panel for a satellite, a life support package for a space vehicle).

The non-operations life cycle processes and their performing organisations of a building block are implemented using enabling products. When the enabling products are also objects of the development effort, they have their own building block as part of the hierarchy of building blocks that compose the project development. Otherwise, only requirements and attributes would be exchanged between the groups that develop the end products and enabling products.

The building block defines the scope of the project that develops a system in terms of subsystems, life cycle processes and performing organisations.

Figure 5-9, 5-10 and 5-11 provide examples of building block structures for different products.

A space research oriented organisation that is trying to master the technology of satellite development, for example, would be involved in the development of end products (satellite and earth station), all life cycle processes and all associated organisations. (see Figure 5-9).

Figure 5-9: Building block of a satellite system

For a commercial aerospace system, the development organisation would not care about developing the operation organisation. The operation organisation (e.g., an airline company) is a stakeholder to the development organisation. Although the development organisation must consider the disposal and distribution life cycle process during product development, it will not be involved in the actual development of the performing organisation. (see Figure 5-10).

Figure 5-11 shows a building block of a passenger car. The development organisation is involved in the development of the car, its life cycle processes and the organisations performing the production and testing of the vehicle, as well as the development organisation.
5.5 The analysis dimension

The analysis dimension defines the different types of analysis that are undertaken to, concurrently, identify the requirements and attributes of the product, process and organisation elements of the system. They are requirements analysis, functional analysis and physical analysis. They can be applied while evolving a system from requirements to actual physical realisation or when analysing an existing system. The analysis dimension follows the systems engineering process as defined by the IEEE-Std-1220-1994 [IEEE, 1995] as shown in Figure 5-12.

Requirements analysis has the purpose of establishing what the system must accomplish (functional requirements), how well it is to be accomplished (performance requirements), and under what conditions it is to be accomplished. It also governs, as appropriate, logical, and physical characteristics of the system.

Requirements begin as stakeholder requirements, and evolve through technical requirements. Stakeholder requirements express what stakeholders of a system require of the end products, their life cycle processes...
and performing organisations under the scope of the project development. Stakeholder requirements may be expressed as needs, wants, expectations, desires, priorities, objectives, or capabilities. These requirements are often stated in non-technical terms that are not adequate for design purposes. Thus, stakeholder requirements often are not verifiable using technical verification techniques. However, stakeholder requirements provide the criteria (but not necessarily quantitative) by which delivered end products are judged. Technical requirements are derived from stakeholder requirements and provide a set of requirements stated in technical terms so that they are objectively verifiable [EIA, 1997].

The total view framework
The analysis dimension

IEEE-Std-1220-1994
The Systems Engineering Process

In modelling terms and within the context of this thesis, requirements analysis consists of context diagrams that establish the boundaries of the system under development and identify the stakeholders. Each identified interaction between the system and the stakeholders is identified by the stakeholder concerns related to that interaction. Concerns derive measures of effectiveness. Measures of effectiveness express the attributes by which the stakeholders evaluate the system, e.g., for a car, comfort, fuel economy, return on investment, manufacturability.

Functional analysis aims to derive a functional architecture and describe it by identifying the functional attributes associated with its elements. It describes the problem defined by requirements in clearer detail. This is to be accomplished by translating the validated set of technical requirements into a functional architecture. A first set of functions is derived from requirements. The functional architecture describes those functions and their interactions and their decomposition into sub-functions. Functional analysis is performed as much as possible without consideration of the physical implementation of the system. [IEEE, 1995].
In modelling terms and within the context of this thesis, functional analysis consists of a set of models that link requirements to functions and decompose these functions into their sub-functions. The models can be reconfigured in order to maximise cohesion and minimise coupling. From the resulting set of reconfigured sub-functions, functional attributes are derived.

The physical analysis aims to derive a physical architecture and describe it by identifying the physical attributes associated with its elements. It models the physical architecture resulting from the synthesis sub-process, as, for example, defined by the IEEE-Std-1220-1994 standard. Synthesis translates the functional architecture into a physical architecture that provides an arrangement of elements, their decompositions, interfaces (internal and external), physical constraints, and designs [IEEE, 1995]. Similarly to the functional analysis, the physical models are also reconfigured using criteria other than functional.

The building block structure defined in Figures 5-7 and 5-8 is used to guide the structure of requirement, functional and physical models.

### 5.6 The structure dimension

The structure dimension defines how the complex system is to be structured. Figure 5-2 shows a layered pyramidal structure. The pyramid is sectioned to represent the fact that every system has a higher layer system to which it belongs. The section of a pyramid represents the hierarchical structure of the system. It is formed by a set of system building blocks (see Figures 5-7 and 5-8). Each subsystem that needs to be developed forms the basis for a lower layer building block. A hierarchy of such systems is shown in Figure 5-13. Where building block subsystems are shown that do not continue down towards

![Figure 5-13: Example hierarchy of building blocks (adapted from the [EIA, 1997])](image-url)
further development, it indicates that the corresponding end product is either an existing product or can be manufactured or coded at that level.

The layers of the hierarchy correspond to the end product breakdown structure. The life cycle processes and their performing organisations for that end product are in the same building block, and consequently, the same layer of that product. If an enabling product is also under the scope of the developer's project as defined in Figure 5-16, a grey box will be linked to the corresponding life cycle process or organisation and will derive its own building block project in a lower layer.

In the same manner that requirements, functional architecture, functional attributes, physical architecture and physical attributes follow the building block structure, they also follow the hierarchy of building blocks. Putting together all the resulting building block structures results in the structure presented in Figure 5-2, the 'total view framework'.

The realisation of the 'total view framework' described in this chapter requires the development of a systematic process. The process consists, basically, of the analysis of requirements, functions and physical architecture of product, processes and organisations in an integrated manner deriving attributes and identifying their relationships. This is the object of the following chapters.

The 'total view framework' provides the elements for the integrated development of complex products. It addressed mainly the issue of scope, i.e., what to include in the development effort. Nothing was mentioned in this chapter on how to integrate the elements of the framework or how to cope with the complexity that may arise from their interactions. Chapter 5 describes the key concepts supporting the 'total view framework' in order to address those other issues. These key concepts are: requirements, attributes and relationships.
Chapter 6
The concepts of requirements, attributes and relationships

This chapter proposes the concepts of requirements, attributes and relationships as essential for the integration of the product, processes and organisation elements within the 'total view framework' and for coping with the inherent complexity in the integrated development of complex products. This chapter outlines the needs for:

• the early identification of product, processes and organisation requirements;
• the simultaneous identification of product, process and organisation attributes;
• the identification of the relationships among requirements and attributes.

6.1 Background

The diesel PCS case study, presented in Chapter 4, demonstrated the benefits of early requirements capture and analysis. Early requirements capture and analysis avoid late changes in the product development cycle and emergency solutions that could unnecessarily complicate the design. Although, formally, only requirements for product functionality were captured and analysed, other requirements were accomplished just by following structured analysis principles.

These other requirements came from the latent needs of the gasoline PCS development process. They fall into three main categories: reusability, ease of calibration and maintainability (or development time) requirements. Clearly, ease of calibration and maintainability are life cycle process categories of requirements and reusability is an organisation category of requirements. In the gasoline PCS development, the aim was to address all these requirements through product functionality. For example, there are numerous 'if-then-else' loops in the gasoline PCS software trying to anticipate various calibration scenarios for many different applications. Also, the reusability issue was addressed by a super-software-configuration, the PTEC (see Chapter 4, Section 4.2.4). Depending on the application requirements, the PTEC strategy software would be depopulated of the unnecessary features.

The diesel PCS development addressed these three sets of requirements separately from the product functionality requirements. For example, the ease of calibration requirements are not met by software features but by a separate application software specially designed to generate calibration constants which would later be automatically incorporated into the software code. The reusability requirements were addressed by the development of an application model which serves as a starting point for the identification of required modifications. The maintainability issue is addressed by a configuration management process which traces any application and software variant or version to the original requirements.

The gasoline PCS development approach resulted in a very interdependent set of product, life cycle process and organisation attributes. Product complexity and maintainability were compromised when
trying to improve reusability and ease of calibration. The diesel PCS structured development approach demonstrated the benefits of uncoupling these attributes.

Many references highlight the importance of uncoupled development.

[Suh, 1990] states in his Axiom 1 that “in an acceptable design, the design parameters and the functional requirements are related in such a way that specific design parameter can be adjusted to satisfy its corresponding functional requirement without affecting other functional requirements”. Suh then separates design into three groups: uncoupled, coupled and decoupled designs. An uncoupled design is the one that satisfies the above axiom.

Genrikh Altshuller, the developer of the Theory of Inventive Problem Solving, states that “a problem becomes a creative one when an attempt to improve system’s parameters by conventional means leads to deterioration of other parameters”. The approach to solve the problem tries to separate the ‘contradicting’ parameters in time, in space and by system transformations. System transformations include combination of systems, combination of system and anti-system, separation of contradictory properties between system and its sub-systems. [Fey et al., 1994]

Stevens et al. (1998) propose a structured approach for coping with complexity in complex product development in which user requirements, systems requirements and elements of the architecture design can be depicted over tree diagrams and related to each other. They also propose to keep track of the relationships among components, functions and requirements as necessary to cope with complexity.

The identification of the relationships among requirements and attributes provide an indication of the level of coupling in the system solution and, therefore, can be used as a complexity management tool.

This chapter deepens the concepts of requirements, attributes and relationships within the context of the generic model for integrated development proposed in Figure 5-1. Section 6-2 stresses the importance of requirements, attributes and relationships in the generic model. Section 6-3 clarifies the concept of requirements used in this thesis. Section 6-4 introduces the concept of attributes. Section 6-5 stresses important aspects of the concept of relationships and their role for integration and complexity management.

6.2 Approach

Figure 6-1 illustrates the important role of requirements and attributes in the generic model for the integrated development of complex products proposed in Figure 5-1. The model proposes the development process to happen over two domains: a problem domain and a solution domain. Stakeholders in the problem domain have requirements that describe their needs, desires, wants, expectations that compose a problem to be addressed. The problem is addressed by a system in the solution domain. The system is composed not only of elements of an end product but also of its life cycle processes and their performing organisations. The system is a balanced solution for the problem described by the requirements. Balance is achieved through the simultaneous consideration of product, processes and organisation attributes that describe the integrated system solution. Attributes provide a measure to the stakeholders of how well their requirements are being addressed by the system solution.
This measure of effectiveness provided by attributes is then the basis for new stakeholder requirements, re-starting the evolution cycle.

Requirements and attributes are considered in the model illustrated in Figure 6-1 as the interface between the problem and the solution domains. Requirements translate stakeholder needs in order to guide the integrated development. A development agency produces the means of meeting these requirements with a system solution. The integrated development should be driven by requirements. Attributes describe various aspects of the system solution to stakeholders. Such a model requires correspondence between the requirements that originate the need for a system solution and the attributes that describe that system.

In the continuous evolution cycle, as attributes may have some sort of dependency among themselves, to meet a new requirement that affects one attribute may not be as simple as it seems. Many other attributes may be affected. Requirements associated with these other attributes would then be also affected. Therefore, the change of an attribute will affect the whole stakeholder satisfaction or dissatisfaction figures.

The early identification of the relationships among attributes and between requirements and attributes provides a picture closer to reality in terms of the amount of work that needs to be done to evolve an existing system. For a complex product, due to the great number of potential relationships, it is much easier to overlook them and as a consequence to only realise problems later in the product life cycle, when the cost of a change is much higher.

When developing a new product, requirements are captured, a concept is developed, an architecture for the product is designed. In integrated product development, not only the product is the object of the development effort but also its life cycle processes and their performing organisations. During those early stages of the integrated development of a new product, requirements, functional and physical models of the product, life cycle processes and organisation are captured in one way or another (e.g.
prototypes, diagrams, draft designs). Attributes can be associated to each element of the functional and physical models.

This chapter proposes requirements and attributes as unifying concepts so that, through them, product, process and organisation can be described in a common language and integrated within the 'total view framework'. The identification of the relationships among requirements and attributes provide a means of managing complexity when changing an evolving product, when deciding between alternative solutions for a new problem or when developing product, process and organisation in a structured way.

6.3 Requirements

6.3.1 Definitions

The Oxford Advanced Learner's Dictionary of Current English [Hornby, 1978] defines a 'requirement' as 'something required or needed' and defines the verb 'require' as 'need; depend on for success, fulfilment, etc.'

The IEEE-Std-1220-1994 [IEEE, 1995] defines a requirement as 'a statement identifying a capability, physical characteristic, or quality factor that bounds a product or process need for which a solution will be pursued.'

A broader and more appropriate definition is provided by the EIA 632-1 Standard [EIA, 1997]. The definition that is the basis for the one used in this thesis is: 'the term requirements refers to the total set of considerations that govern what is to be accomplished, how well it is to be accomplished, and under what conditions it is to be accomplished. They also govern, as appropriate, logical, and physical characteristics of the system.' In this thesis, this definition refers to requirements governing the end product, its life cycle processes and their performing organisations that are the object of engineering according to the 'total view framework'.

Examples of requirements are:

'The car shall transport people comfortably under all weather conditions.'

'The manufacturing process shall last for no more than 1 hour with current facilities.'

'The development organisation must capture requirements consistently according to IEEE-Std-1220-1994 Standard.'

6.3.2 Objectives

The set of requirements for a system describes the problem for which the system is the solution. A system is a set of elements that compose a balanced solution for the problem described by the requirements. The elements of the system solution are the functional and physical elements of the end product, its life cycle processes and their performing organisations.

Requirements provide guidance to the development activities. When captured, analysed and specified requirements identify what the development teams have to address. This can make the difference between an engineering-focused system and a stakeholder-focused system. The former tries to improve
the efficiency of the solution regardless of what the stakeholders require from it. The latter addresses the
efficacy of the solution, developing it to meet the stakeholders needs. Requirements address the issue of
solving the right problem.

Requirements provide the criteria for trade-offs during the development process. During the
development process, there are two main stages when trade-offs are needed. The first is when to decide
for a better concept or functional architecture for the product, process or organisation. The second is
when to decide for the physical parts that will actually implement the functions, the physical architecture.
Requirements provide the criteria to eliminate or to promote concepts or parts during a decision analysis
process.

Requirements provide communication through the system life cycle and between the layers of the system
breakdown structure. Stakeholders responsible for the implementation of the life cycle processes feed the
development process in order to improve product manufacturability, assemblability, testability,
serviceability, up-gradeability, etc. Requirements travel through a stakeholders-suppliers chain in order
to communicate the original stakeholder needs through the current subsystem technical requirements. A
subsystem supplier can sub-contract another supplier to develop part of the system. Requirements
provide communication along the suppliers chain.

Requirements provide the criteria for acceptance or test of the system, subsystem or part. The result of
the development effort is verified against requirements. In that sense, requirements provide the criteria
for a measure of success or failure of a system.

Requirements provide the criteria for the optimization of a system. Given that requirements have
preferences or importance associated with them, requirements provide a criteria for adjusting the
parameters of a given solution in order to maximize stakeholders satisfaction or minimize stakeholders
dissatisfaction.

The early identification of requirements at the conceptual development stage of the system development
process is broadly recognised as a way of avoiding late changes during the system development life
cycle. Avoiding late changes, the system can be developed in a shorter lead-time, at a lower cost and
with better quality. Within the 'total view framework' the early identification of requirements includes
also requirements for the product life cycle processes and their performing organisations within the scope
of the development effort. The early identification of requirements is a concurrent engineering principle
and, in the 'total view framework', supports the integrated and concurrent development of a product, its
life cycle processes and some or all of their performing organisations.

6.3.3 Scope

The definition of requirements in Section 6.3.1 highlights three types of considerations that requirements
govern:

• functions, what is to be accomplished,
• performance, how well it is to be accomplished, and
• conditions, under what conditions it is to be accomplished.
In general, these considerations are not separate in different sets of requirements and sometimes are expressed even in a same requirement statement. In these situations it is worth extracting, for further requirements analysis, these different types of considerations from the requirement set or statement. When in separate format, these considerations characterise functional requirements, performance requirements and conditions. A particular set of conditions are expressed in interface requirements.

**Functional requirement** is a qualitative statement of what an item (organisation, person, product or process) is to accomplish. This can be the behaviour of an item, an effect produced, or an action or service to be performed. An example of behaviour is 'the engine shall move to cranking state'; an example of an effect produced is 'cause an emergency signal'; an example of an action or service to be performed is 'signal shall close valve'. A functional requirement states the actor that is to perform the function, the function to be performed and, if appropriate, the object acted upon.

**Performance requirement** is a statement of a measure of the accomplishment. Performance requirements state how well (or how much, how far, how long, how often, etc.). It complements the function statement with measurement ranges or values that apply to the function performed and, as appropriate, the object acted on.

**Conditions** are statements of under what conditions a function is to be accomplished with the required level of performance. Conditions include statements of the environment within which the function is performed, the conditions that cause the function to start, and the conditions that cause the function to terminate. These statements may contain measurement ranges or values. Conditions of interaction between items are stated in interface requirements.

**Interface requirements** include both logical and physical interfaces. They include, as necessary, physical measurements, definitions of sequences of energy or information transfer, and all other significant interactions between items. There are interfaces between a system and things external to the system, and between elements within a single system. The latter include, but are not limited to, interfaces between the end products and enabling products, the interfaces between items that make up an end product, interfaces between sub-processes that make up the life cycle processes, interfaces between items that make-up an associated organisation. For example, process interface requirements express the needs for exchanging material, energy or information between activities. Organisation interface requirements are, for example, requirements for communication or information technology.

**Constraints** are not considered a separate type of consideration requirements cover. They are requirements that cannot be traded off. As such, constraints can be expressed in any type of consideration requirements cover regardless being function, performance or conditions. Constraints can result, for example, from treaties, laws, regulations, standards, culture, natural laws and firm customer needs. Design or implementation aspects contained in requirements are to be considered as constraints (e.g. I want a car with a radio)
6.3.4 Evolution

Requirements begin as stakeholder requirements, assumptions and goals and evolve to technical requirements, as illustrated in Figure 6-2. Assumptions and goals are included here because it is very important to distinguish, from the outset, the information coming from the stakeholders from the assumptions made by the developers or the goals of the product development organisation.

![Diagram of requirement evolution]

Figure 6-2: The evolution of requirements over the requirements analysis process

Assumptions express what the developers take for granted in terms of requirements, functionality or even physical characteristics of products, processes and organisation. For example, one may assume that a new vehicle will have four wheels, or that the development process will follow the same activities of a previous project or that a given set of resources will be available. Assumptions must be documented and analysed early in the development process in order to avoid later conflicts throughout the life cycle.

Goals express generic objectives of the product development organisation that the system solution will help to achieve. Goals synthesize the key project issues from a development organisation perspective, covering the most important business requirements, major decisions, and important development issues such as cost or schedule. Goals also include the commitments from management to the project. Examples of goals are: to be a leader company by the year 2000, to reduce development cost by 10%, to improve a given product performance attribute by 50%.

Stakeholder requirements express what stakeholders of a system require of the product, processes and organisation of that system. Stakeholder requirements may be expressed as needs, wants, expectations, desires, priorities, objectives, or capabilities. These requirements are often stated in non-technical terms.
that are not adequate for design purposes. Thus, stakeholder requirements often are not verifiable using technical verification techniques.

Technical requirements are derived from stakeholder requirements. They provide set of requirements stated in technical terms so that they are objectively verifiable.

6.3.5 Characteristics

A well defined, mature set of technical requirements has some desired characteristics which apply to an individual requirement or to the set of requirements. It is desired that an individual technical requirement be:

- **necessary**: when omitted or deleted will cause a deficiency in the resulting system, in terms of meeting stakeholders expectations.
- **concise** (*minimum understandable*): the requirement statement includes only one requirement, stated simply and clearly. It is easy to read and to understand. The requirements statements must not contain any explanations, rationale, definitions or descriptions of system use.
- **implementation free**: the requirement states what is required, not how the requirement should be met. A requirement statement should not reflect a design or implementation, a process or an organisation. However, the treatment of interface requirements is generally an exception.
- **attainable** (*achievable or feasible*): the stated requirement can be achieved by one or more developed system concepts at a definable cost. This implies that at least a high level conceptual design has been completed and cost trade-off studies have been conducted.
- **complete** (*standalone*): the stated requirement is complete and does not need further amplification. The stated requirement will provide sufficient capability.
- **consistent**: the stated requirement does not contradict other requirements. It is not a duplicate of another requirement. The same term is used for the same item in all requirements.
- **unambiguous**: each requirement must have one and only one interpretation. The language used in the statement must not leave a doubt in the reader’s mind as to the intended descriptive or numeric value.
- **verifiable**: the stated requirement is not vague or general but is quantified in a manner that can be verified by one of these alternative methods: inspection, analysis, demonstration or test.

Desired characteristics of a set of requirements are:

- **completeness**: the set of requirements has addressed all categories and covers all allocations from higher levels.
- **consistency**: the set of requirements does not have individual requirements which are contradictory. Requirements are not duplicated. The same term is used for the same item in all requirements.

The above characteristics are further detailed and exemplified by the INCOSE’s Requirements Management Working Group [Kar & Bailey, 1996]. This reference also provides additional desired characteristics of requirements including standard constructs and words to avoid. [Ford Motor Company, 1996a] proposes a checklist for reviewing requirement quality. [BAe, 1990] also propose some desired characteristics of requirements.
6.3.6 Elements

So far, requirements have been treated as a statement or descriptive text. However, other information elements may be associated with a requirement, such as: identifier, categories, compliance level, source (e.g. stakeholders), target (e.g. development agencies), required-by subsystem, required-of subsystem, source document, target specification, ownership, verification method, change history. These and other elements are listed and described in [Kar & Baley, 1996] and [Akao, 1990] and in the manuals of requirements management tools such as DOORS [QSS, 1995], Cradle [3SL, 1998] and RTM [GEC-Marconi, 1997].

6.3.7 Capture and analysis

The requirements capture and analysis processes start from identified needs or opportunities in the market place. These needs are basically represented by an initial set of stakeholder and stakeholder requirements. The objective of the requirements capture and analysis processes is to identify other stakeholders and their requirements, and translate these requirements into technical requirements. Many sources, such as [IEEE, 1995], [Martin, 1997], [Stevens et al., 1998] describe requirements capture and analysis processes for product development. In this thesis, a requirements capture and analysis process is proposed within the context of integrated development, taking into consideration, from the outset, the fact that requirements for the product will be affected by life cycle processes and organisation requirements. The anticipation of life cycle process and organisation requirements to the moment when product functional requirements are also captured is a concurrent engineering principle. Chapter 7 proposes a requirements capture and analysis process for the integrated development of complex products.

6.4 Attributes

6.4.1 Definition and description

The Oxford Advanced Learner's Dictionary of Current English [Hornby, 1978] defines the word attribute as: “quality looked upon as naturally or necessarily belonging to somebody or something.”

Ford Motor Company Ltd. specifies the 'somebody' or 'something' of the above definition as being the product and defines attribute as a “characteristic of a product perceived by its customers”[FDI, 1996]. Attributes are then deployed down to sub-attributes, characteristics and properties to cover other characteristics that are not perceived by the customer.

Bunge (1977) defines attribute as a property, trait or character that is attributed to, or predicated of, some subject or other. Bunge differentiates between attributes and properties. According to him, properties are inherent to some subject, independent on whether they are perceived or not. Attributes are the perceived properties.

With the broader scope of the system definition adopted in this thesis, i.e., a system integrates product, process and organisation elements, and the need to describe functional and physical aspects of the system, it is necessary to expand the above definitions. For the context of this thesis, attributes are defined as:
“the properties and characteristics which are possessed by a product, a process or an organisation and which are intended to meet stakeholders’ requirements”. Examples of product attributes are: fuel economy of a vehicle, dimension of a mechanical component, viscosity of a chemical, uniformity of the voltage of an electric power supply. An example of an organisation attribute is: productivity of a factory. Examples of process attributes are: lead time of the development process, promptness to delivery, ease of maintenance, courtesy of service.

The use of the words ‘properties’ and ‘characteristics’ is justified by the need to describe functional and physical aspects of the system, respectively. Morris (1992) associates the word properties with capabilities, functions or the like. Properties qualify the functional elements of a system or the elements of a functional model of a system. For example, properties of an engine are torque and angular speed. Properties are independent of the way the function is implemented. Physical aspects are described by the characteristics of the system. Characteristics are associated with physical parts, implementation aspects. Characteristics qualify the physical elements of a system. Examples of characteristics of an engine are size, mass and material used. In this thesis, properties and characteristics will be called indistinctly as attributes. Attributes may also refer to the existence of a given property or characteristic. For example: the existence of an air conditioning function, the existence of a radio in a car. Examples of process attributes are: development time may be regard as a functional property whereas the time to perform the actual development tasks may be regard as physical characteristics. Examples of organisation attributes are: productivity, capacity, throughput are functional properties that are dependent upon the physical characteristics of a machine or plant.

When developing a system solution that meets stakeholder requirements according to the ‘total view framework’, functional and physical models of the solution are developed as a result of the functional and physical analysis processes.

Functional models are typically composed of functions, inputs, outputs, states, conditions, actions. These elements describe the concept of the product, process or organisation. At this stage of the development process, there is not a concern with how these functions will be implemented. The functional model focuses on capturing what the system solution must accomplish not how it is to be accomplished. Each element of the functional model can be associated to attributes that complement its description. These attributes are denominated functional attributes. Typical functional attributes go from the ‘existence of a function’ to performance factors. An important aspect of functional attributes is that, as they are implementation-independent, they are associated with every candidate physical implementation. As such they can be used to compare different implementations that perform the same function. Examples of functional attributes are: fuel economy for a car, lead time of a development process, productivity of the manufacturing organisation.

Physical models are typically composed of parts or modules and physical or logical links between them. In the case of process and organisations, physical models refer to the physical arrangement of the resources necessary to perform the process and organisation functions. The physical models describe the elements that will implement the functions established in the functional models. They describe how the system is going to be implemented. Each element of the physical model can be associated to attributes
that complement its description. These attributes are denominated physical attributes. Examples of
graphic attributes are: depth, width, length, mass, density, material or the existence of a physical item.

Figure 6-3 shows the correspondence of the elements in the definition of attributes in Section 6.4.1 to
functional and physical attributes. Properties in the definition of attributes correspond to functional
attributes and characteristics, to physical attributes.

![Diagram of attributes, properties, and characteristics]

Figure 6-3: Correspondence between what attributes describe and types of attributes

### 6.4.2 Objectives

In the same way that requirements provide the means of describing the problem to be solved, attributes
provide the means of describing the solution adopted. Product, process and organisation are described by
their corresponding attributes. The following describes some objectives of attributes:

- attributes complement functional and physical architecture models, e.g., those developed using
  structured analysis methods. As structured analysis methods were developed to aid the software
  engineering process, they lack instruments to describe functional interactions other than data
  exchange or state transition. To describe hardware or resources, attributes can be attached to the
  elements of functional and physical models.

- attributes provide the means to determine the quality of a given system solution in terms of meeting
  the requirements for which it has been developed. As such, attributes provide the basis for
  improvement, once they provide a description of the current quality status of a system.

- attributes provide the means to compare different solutions of the same problem, even if they are
  technologically different. For example, the function 'to go from London to Paris' has an attribute
called 'trip duration', this attribute provides a way to compare the solutions 'by plane' or 'by
  Eurostar'.

- attributes provide a way to model the performance of a system. Attributes are related to each other
  and, through the performance model, it is possible to identify their mutual effects, like in a
  spreadsheet.
6.4.3 Elements

An attribute may have, attached to it, additional elements that help in its description. Basic attribute elements are the ones that every attribute must possess. They are:

- **Factor**: is the dimension that represents the attribute. The attribute is identified by the name of the factor. A factor may be quantitative, e.g., temperature, time. A factor may also be qualitative, e.g., type of machine, identification of operator.

- **Level or Value**: is a status or magnitude that a factor may assume. For example: 'comfortable' and 'not comfortable' are levels for the factor 'comfort', '20 degrees Celsius' is a level of the factor 'temperature'.

Attributes whose values express quantities, such as magnitude, size, extent, amount, may have other elements:

- **Tolerance**: the allowable range of deviation from a pre-specified level of the factor. A bilateral tolerance expresses these limits on both sides of a nominal level (the ideal level). A unilateral tolerance has one limit at the nominal value and expresses the other limit entirely in a deviation in one direction from the nominal.

- **Unit**: quantity used as a standard of measurement. For example: 'dollars' is a unit of cost, 'degrees Celsius' is a unit of temperature.

Other attribute elements are: identifier, technical difficulty, weighting, direction, categories, source, ownership, verification method, change history, current value, desired level. These and additional elements are detailed in [Akao, 1990] and in [Juran & Grina, 1988] and can be inferred from systems engineering standards such as IEEE-1220-Std/1994. [Fowlkes & Creveling, 1995] also contains a classification of attributes and the elements that are specific to some classes of attributes.

6.4.4 General desired characteristics

**Traceability to stakeholder requirements**

Every product, process or organisation attribute must be traceable to at least one stakeholder requirement. If this is not true, there is an attribute that does not correspond to any requirement and in this case one out of the two following situations might have happened:

- There is a development effort that brings no value to the stakeholders. There are unnecessary attributes that have a cost to the development agency and does not bring value to the stakeholders. The system could therefore have the same quality but costing less. Finding out unnecessary attributes is part of the process of value engineering. Value engineering has two main objectives: reduce the cost of a development and increase its value. Eliminate unnecessary functions, parts or attributes is a way of reducing costs.

- The attribute brings value to the stakeholders, but there is no corresponding requirement. There is no requirement that captures the stakeholder need being met by that attribute. There are requirements missing in the requirements set or specification. It also reflects the fact that the development process was not requirements driven. It was implementation driven. If there is no
requirement corresponding to that attribute, the process driven by requirements cannot lead to a system solution with that attribute.

The ‘total view framework’ supports an integrated development process that has to go through three stages of analysis: requirements, functional and physical analysis. The integrated development process is forcefully a requirements driven process because the functional and physical analysis processes start from the requirements engineered throughout the requirements analysis phase. Therefore, the attribute characteristic of being traceable to requirements serves the purposes of assuring that all attributes bring value to stakeholders and feeding back the requirements capture process through the identification of missing requirements.

**Hierarchy**

Functional and physical attributes exist on a hierarchy that follows the hierarchy of functions and physical subsystems of a system. Attributes on the upper layers are expected to be a mathematical function of attributes on the lower layers of a system.

Functions at the \(i\)-th layer cannot be decomposed into the next level of the functional hierarchy without first going over to physical domain and developing a solution that performs the \(i\)-th layer functions with all corresponding physical attributes. That is, it is necessary to travel back and forth between functions and physical implementation in developing the functional and physical attribute hierarchy.

**Independence**

Functional attributes shall be chosen from the functional model of a system in such a way they are as much independent as possible from each other. The independence of a functional attribute means that the attribute does not vary by the variation of other functional attributes. A functional attribute is independent if it is not affected by the variation of other functional attributes. Physical attributes and functional attributes shall be related in such a way that specific physical attribute can be adjusted to satisfy its corresponding functional attribute without affecting other functional attributes. This characteristic of attributes is adapted from the design axioms [Suh, 1990] that govern good design.

**Non-uniqueness**

From the same set of requirements, the developer can arbitrarily derive different functional models of the system and to each one of these it can be assigned different sets of functional attributes. The set of functional attributes is not unique. Also, corresponding to a set of functional attributes, there can be many physical solutions, each one with a different set of physical attributes. So, in the same manner, the set of physical attributes is not unique. When the original set of functional attributes changes, a new physical solution must be found. The new solution may not be derived by simply modifying the previously obtained set of physical attributes.

**Minimum number of attributes**

The developers must have the ability to choose a minimum number of (as much independent as possible) functional attributes at each hierarchical layer of the system functional model.

**Verifiability**

Attributes must be chosen in such a way that their values are verifiable.
6.4.5 Derivation

Attributes are derived from information concerning concept development and implementation design resulting from the functional and physical analysis processes, respectively. This information can be presented in the form of models, CAD drawings, circuit diagrams, textual documents, flow chart diagrams, organisation diagrams, etc, which help to describe the functional and physical architectures. Again, many sources ([IEEE, 1995], [Martin, 1997], [Stevens et al., 1998]) describe processes for obtaining functional and physical architectures for product development. In this thesis, functional and physical analysis processes are proposed within the context of integrated development, taking into consideration, from the outset, the fact that product attributes affect life cycle process attributes, organisation attributes and vice-versa. It is a model-based approach based on structured analysis techniques for product, process and organisation systems. According to the proposed approach, product, process and organisation models can be developed simultaneously during the functional and physical analysis processes. Chapter 7 describes proposed functional and physical analysis processes for the integrated development of complex products.

6.5 Relationships

6.5.1 Definition

A relationship expresses how two elements are connected or associated. In the scope of this thesis, the main elements of interest are stakeholder, requirement, development agency and attribute. Other important elements are the functions and component parts for which attributes were derived.

6.5.2 Objectives

The importance of the identification of relationships among these elements is due to:

- traceability: requirements are tracked to functions in the functional models and these to component parts in the physical models. Traceability links are used to assure that the agreed requirements are addressed during the conceptual and physical designs and to justify every designed part by the existence of a requirement. Attributes or other information items are also linked to functions and physical parts through traceability links.
- communication: the translation process from stakeholders requirements to technical requirements promotes a communication channel between the stakeholders and the development agencies. Each of them can have their own jargon language which can only be understood by either part through this translation process. Akao (1990) expresses this importance through QFD matrices
- integration: product, process and organisation attributes are linked together and their contribution to the whole integrated system is identified.
- partitioning: the identification of links among attributes makes possible to identify clusters of attributes that could be treated separately from the whole set of attributes.
optimisation: Papalambros et al. (1994) demonstrated that an optimisation procedure through sub-optimisation of clusters of attributes can lead to satisfactory results.

• team formation: the linkages among attributes can serve as criteria for team formation if the relationship between attributes and development agents is captured.

• change effect analysis: the linkages between requirements and attributes and among attributes indicate the items affected when a requirement or attribute changes.

• design improvements: Suh (1990) suggests that for design improvements purposes, one should pursue functional requirements independence. Functional requirements independence means that attributes and functional requirements are related in such a way that specific attributes can be adjusted to satisfy its corresponding functional requirement without affecting other functional requirements.

6.5.3 Elements

Like requirements and attributes, relationships may be further detailed through some descriptive elements such as:

• Identification: states the phrase represented by the relationship.

• From element: in an asymmetric relationship, refers to the subject of the phrase that identifies the relationship.

• To element: in an asymmetric relationship, refers to the object of the phrase that identifies the relationship.

• Reason: justifies the creation of the relationship between those two specific elements.

• Rationale: describes the basis upon relationships such as this are created. Examples of rationale are: traceability, decision tree, structure tree, etc.

• History: specifies relationship’s creator, creation date, last modifier, and last modification date, for example.

• Strength: defines the intensity of relationship between two elements. Relationships in QFD matrices [Akao, 1990] are assigned values such as: strong (9), medium (3) or weak (1) relationships. Specially for qualitative correlation identification purposes, other accepted values for strength are: strongly positive (+9), positive (+3), negative (-3) and strongly negative (-9).

6.5.4 Types

In this thesis, three types of relationships are considered: the traceability links, the impact links and the hierarchy links. These relationship types are illustrated in Figure 6-4.

Traceability links relate stakeholder requirements to technical requirements, these to functions in the functional models and these to component parts in the physical models. These links assure that every designed part is there in order to meet an identified stakeholder requirement and that every requirement is addressed in the design. Therefore, traceability links are bi-directional. Traceability links also relate attributes to their corresponding model element. Traceability links also represent the allocation,
partitioning or decomposition of requirements and attributes between layers of the system breakdown structure.

Impact links relate two elements characterised by the fact that when one of these element changes the other element is affected. Impact links are unidirectional, i.e., the direction of the link from element A to element B means that 'changing element A, element B will be affected' and that 'changing element B, there is no guarantee that element A will be affected'. Impact links are transitive: if element A 'impacts' element B and element B 'impacts' element C then element A 'impacts' element C. Figure 6-4 suggests that physical attributes may 'impact' each other and that functional and physical attributes in a given layer are impacted by functional and physical attributes, respectively, in lower layers. At every layer physical attributes 'impact' functional attributes, these 'impact' technical requirements, these 'impact' stakeholder requirements, these 'impact' stakeholders. Technical requirements, functional attributes and physical attributes 'impact' their corresponding development agencies.

Hierarchy links represent the hierarchical structure of requirements, attributes, functions and component parts. Requirements can be structured in a hierarchy to reflect the fact that a requirement statement may express more than one requirement. Also a given requirement can be clarified into lower level requirements. Or a given requirement can be further detailed into lower level requirements. Requirements hierarchy may exist in the same layer of the system breakdown structure. Attributes hierarchy reflects the fact that higher level attributes are a function of lower level attributes. This fact can.
occur in the same layer. Hierarchy links are just a way of identifying an independent set of attributes in a
tree structure. The attributes at the lowest level of the tree are then associated with a corresponding
function or physical part. Hierarchies of functions and of component parts reflect functional or physical
decomposition. Hierarchy links are unidirectional links and may express relationships such as: 'is
children of', 'is parent of', 'translates into', 'is translated by', 'is function of', 'is composed of'.

6.5.5 Representation

Figure 6-5 presents a proposed generic format of matrices that capture the relationships among
requirements and attributes. The 'What' elements in Figure 6-5 are those describing objectives and 'how'
elements are those describing means. A functional attribute, for example, can be a 'what' or a 'how'
element depending on the other element it relates to. It
'implements' a technical requirement and 'is
implemented by' a physical attribute. Sources of 'what' and of 'how' reflect the ownership of requirements and
attributes. Cross-interaction matrices relate the items
on the rows with the items on the columns. Self-interaction matrices show the relationships among 'how'
elements and among 'source of how' elements. The matrices generic format is very similar to a QFD
matrix format apart from the inclusion of the sources of 'what' and 'how'. These are included in the
matrix format proposed in this thesis because the relationships among sources of 'what' or among
sources of 'how' can be used as criteria for team formation. Integrated product development teams play a
key whole in the success of complex products development.

Figure 6-6 illustrates the analysis process that deploys stakeholder requirements to physical attributes at
layer i on the system breakdown structure. A complex product may have many of these layers. Figure 6-
6 also illustrates how elements at different layers are related to each other. Requirements and attributes at
a given layer are allocated, partitioned or decomposed into requirements and attributes of the subsystems
at a lower layer [adapted from Maier, 1996b]. Subsystems in any lower layer may have requirements
and attributes originated not only from the upper layers but also from local stakeholders specifically
related to that particular subsystem. Examples of local stakeholders are a subsystem manufacturer, the
owner of a company that develops a specific subsystem, other contractors, etc.

Some requirements and attributes at a given layer can be directly allocated to a lower layer if they can be
assigned to a single functional or physical model element in the lower layer. For example:

- vehicle exhaust emissions requirements and attributes can be directly allocated to the powertrain
  subsystem;
- vehicle comfort requirements and attributes can be allocated to a function 'provide comfort' in the
  vehicle functional model.
Some requirements and attributes cannot be directly allocated from one layer to a lower layer. Before being allocated, they need first to be partitioned. They can be partitioned among the functional or physical model elements at a lower layer if their corresponding value can be divided into values of the same dimension among the elements at a lower layer. Examples of such requirements and attributes are:

- vehicle weight: which can be divided by assigning weights to each element of the physical model in such a way that the sum of powertrain, chassis, body, interior and electric subsystems weight is equal to the vehicle weight.
- time-to-market: which can be divided by assigning time values to all process model components from pre-programme requirements capture to production ramp-up. Time assignment is not as simple as weight assignment. As process models may involve concurrent activities, time allocation is not as simple as weight allocation, which can be done additively. The modelling of concurrent tasks with limited resources requires a more complex procedure for timing allocations. However, the ‘time-to-market’ attribute is still resulting from attributes of the same dimension allocated to elements of the process functional and physical models.

Some requirements and attributes cannot be allocated to any one element of the system models and they must be decomposed in multiple steps into sub-requirements or sub-attributes of different dimensions. They result from the interactions among different system elements. They are the essence of a system based on the principle that a system is more than the sum of its elementary parts. The set of elements comprising a system always has attributes that cannot be exhibited by any of its subsets. These are the emergent attributes. Examples are:
• fuel economy of a car: it depends on the interactions of attributes from the body (e.g. aerodynamics), chassis (e.g., weight), powertrain (e.g. power, emissions).

• detection range of a radar system: it is met only by the interaction of all portions of the system and is specified by the combination of dissimilar parameters, such as system noise figure, detection bandwidth, threshold signal-to-noise ratio, antenna gain, etc.

The relationships between requirements and attributes at different layers of the system breakdown structure can also be represented by matrices such as the ones shown in Figure 6-5. For example, functional attributes for the car can be considered as the 'what' elements and the related functional attributes of the powertrain, the 'how' elements.

6.5.6 Clustering

The relationships among requirements and attributes are represented in matrices. Attributes are derived from functional and physical models. Functional and physical models are graphs containing the elements of the functional and physical architecture, respectively, and the relationships among those elements. Relationships can be used as criteria for clustering the related elements together. Clustering is a way of maximising cohesion and minimising coupling when partitioning a functional or physical model or grouping requirements and attributes. Grouping requirements and attributes according to their relationships and dealing separately with each group is equivalent to dividing a complex problem into simpler sub-problems. The more uncoupled these sub-problems are, the easier to solve them.

Many authors (e.g. [Prasad, 1996a], [Yourdon, 1989]) recognise that when trying to reduce complexity one should seek to maximise cohesion and minimise coupling when partitioning a system. Group elements are cohesive when they cluster together, sharing common processes, interfaces, communication links, physical features, etc. Functionally bound elements are candidates for physical grouping in order to reduce the overall system interface complexity. Coupling is the degree of the interdependence between elements in different clusters [Hitchins, 1992].

[Hitchins, 1992] proposes a method of evaluating the quality of clustering. The method was presented in Chapter 2, Section 2.2.4. This thesis proposes three possible uses for this clustering quality metric:

1) Functional and physical architecture improvement.

From the models developed for product, process and organisation analysis, processes, equipment, time functions and objects can be extracted and placed on the main diagonal of the matrix described in Figure 2-2 (in Chapter 2, Section 2.2.4). Flows and connections between the elements on the diagonal can also be identified from the diagrams. These flows and connections are translated into an interface strength figure and placed on the appropriate matrix cell. Rows and cells of the matrix are re-arranged until the minimal score is found. Criteria for interface strength can be derived in terms of the type of data, energy or material flowing between symbols.

2) Design evaluation

Suh (1990), Michelena & Papalambros (1995) propose that a better design is the one with less interdependent functional attributes. Functional attribute interdependence occurs when they are affected by
the same physical attribute. If functional attributes at any given level of a system breakdown structure are placed on the main diagonal of the matrix in Figure 2-2 (in Chapter 2, Section 2.2.4) and an interdependence index is placed on the other cells, it is possible to derive a metric of functional interdependence and, therefore, of design quality. The interdependence index can be derived, for example, from the number of physical attributes affecting simultaneously two functional attributes, correlation calculations, etc. Important aspects of this metric are that:

- it can compare solutions of different nature because it is based only on functional attributes, which can be met by many different physical solutions;
- it can include product, processes and organisation all together in the scope of the solution analysis.

3) Team formation

Development agencies are defined in this thesis as the sources of requirements and attributes. The relationships between development agencies are derived from the relationships between the requirements and attributes they are related to. Development agencies can represent whole organisations, teams or individuals. If these elements are placed on the diagonal of matrix in Figure 2-2 (in Chapter 2, Section 2.2.4) and the strength of their relationships are placed on the other cells of the matrix, the matrix can be re-arranged in order to guide team formation. The strength of the relationships between two development agencies can be derived from the number of requirements and attributes related to them both and from the strength of the relationships between these requirements and attributes. Again, an important aspect of this team formation criterion is that the team is likely to contain not only product related people but also process and organisation related people.

Chapter 9, Section 9-4, provides an example of the use of the clustering quality metric here proposed and demonstrates the use of computer tools in seeking the best matrix re-arrangement. Due to the potential large size (order of hundred or thousand) of the matrices involved in this calculation, computer aid becomes fundamental.

Chapter 5 justified, proposed and described the ‘total view framework’. Chapter 6 revisited the fundamental concepts of requirements, attributes and relationships as elements for integration and complexity management. The following chapters propose a way of bringing the ideas proposed in Chapters 5 and 6 to life. Chapter 7 proposes a method within the ‘total view framework’ that derives the elements presented here in a structured way, encompassing systems engineering and concurrent engineering principles. The method is called the ‘concurrent structured analysis’ method.
Part III
Design
Chapter 7
The concurrent structured analysis method

This chapter aims:

• to justify and propose a method of structured analysis, within the 'total view framework', to derive
  the requirements and attributes of product, process and organisation and, then, their relationships;
• to present the modelling approaches used to provide a total view of product, process and
  organisation;
• to present the requirements, functional and physical analysis processes comprising the method

7.1 Background

Structured analysis was highlighted by Tom DeMarco [DeMarco, 1976] through the introduction of Data
Flow Diagrams in the late 1970s. The approach was enhanced by Yourdon [Yourdon, 1982] in the early
1980s, who brought the use of State Transition Diagrams, Entity Relationship Diagrams and Structured
Charts. The approach was then further improved by Hatley and Pirbhai [Hatley and Pirbhai, 1988] in
the late 1980s. Structured analysis was conceived for software development to address, mainly, the
productivity, reliability and maintainability issues. Yourdon (1989), for example, states that 50% of the
errors (and 75% of the cost of error removal) in a system are usually associated with errors of systems
analysis. He also says that it is impossible to maintain a system if there is no accurate, up-to-date
functional model of the system. Hatley and Pirbhai expanded the use of structured analysis for real-time
systems addressing also the issues of size and complexity.

As mentioned in Chapter 4, the diesel PCS development used the Hatley Pirbhai methodology, through
the use of a CASE tool called Teamwork [Cadre, 1995]. The methodology and tool were used for
software development and for the analysis of hardware requirements. Although the development focused
on product operations requirements with little consideration of life cycle process and organisation
requirements, the use of a structured analysis approach addressed many of the unwritten needs of the
gasoline PCS development in terms of productivity, maintainability, reusability and ease of calibration.

This chapter proposes a method that expands the diesel PCS structured analysis approach to include
hardware, life cycle process and organisation analysis, addressing not only operations requirements but
also and simultaneously the non-operations requirements. To fulfil the scope of the 'total view
framework', the method supports the requirements, functional and physical analysis of product, process
and organisation.

The traditional structured analysis methodologies mentioned above aims to derive functional and module
specifications for software. In the method proposed here, the aim is to derive functional and physical
attributes. In this case, attributes are treated as individual items of information as opposed to packages of
information in the specifications. Individual attributes can be related to each other and to the specific
requirements that originate them making feasible the approach to manage complexity described in
Chapter 6.
In this chapter, it is not intended to prescribe another method of structured analysis. It is proposed that, with the existing available methods, it is possible to develop the product, its life cycle processes and their performing organisations in an integrated manner as elements of a whole integrated system.

The requirements, functional and physical analysis processes described here are adapted, mainly, from modern systems engineering standards, IEEE-1220-1994 [IEEE, 1995] and EIA-632 [EIA, 1997]. These standards focus on product development and requirements capture for life cycle processes. They are adapted in order to provide a requirement, functional and physical analysis framework to product, process and organisation. The analysis processes are performed simultaneously for product, process and organisation from the outset. A basic principle guiding the method is the identification of requirements, attributes and relationships early in the integrated development of complex products.

This chapter is structured as following: Section 7.2 situates the method within the systems engineering process; Section 7.3 provides an overview of the method; Section 7.4 describes the modelling approaches proposed for product, process and organisation; Section 7.5 describes the requirements, functional and physical analysis process and how they use the proposed modelling approaches.

### 7.1 Scope

Figure 7-1 defines the scope of the method proposed in this chapter within the systems engineering process, the development process and the product life cycle.

![Figure 7-1: The scope of the method](image)

On the left column of Figure 7-1, typical product life cycle processes are listed. A similar set of life cycle processes is proposed in the IEEE-1220-Std94 systems engineering standard [IEEE, 1995]. The set of
life cycle processes in Figure 7-1 refers to a complex product which is composed of several related elements and their interfaces. Each element may be composed of hardware, software, data, facilities, materials, services, and techniques. The middle column of Figure 7-1 contains the generic product development process as proposed by [Ulrich and Eppinger, 1994]. A generic model and a framework for the integrated development of complex products was proposed in Chapter 5. Systems engineering and concurrent engineering are the approaches proposed in Chapter 5 as components of the total view framework, a framework for the integrated development of complex products. The subprocesses of the systems engineering process as defined by [Martin, 1997] are listed on the right column of Figure 7-1. The total view framework consists of the application of the systems engineering process concurrently to product, processes and organisation. The structured analysis method proposed in this chapter concentrates on the requirements analysis and architecture definition subprocess of the systems engineering process.

7.3 Overview

Figure 7-2 provides an overview of the method applied to a given layer of the system under development. This is an iterative process and the processes used will be essentially the same at each layer in the system hierarchy.

![Figure 7-2: Method overview – method applied at a given system layer](image)

Requirements analysis is triggered by the identification of some initial and obvious stakeholders and their needs expressed by stakeholder requirements. The requirements analysis process then identifies other stakeholders, their concerns and their requirements. As part of the analysis process, in the stakeholder requirements set, functions, performance, conditions, constraints, assumptions and goals are
identified. These are then stated in clear, unambiguous, and measurable technical terms constituting the technical requirements set.

Functional analysis translates the technical requirements into a functional architecture, from which functional attributes are derived. Functional attributes describe each element in the functional architecture. The functional architecture describes the functional arrangements and sequencing of sub-functions resulting from decomposing the set of system functions to their sub-functions. Functional analysis is performed without consideration of a design solution.

Physical analysis translates the functional architecture into a physical architecture from which physical attributes are derived. Physical attributes describe each element on the physical architecture. The physical architecture provides an arrangement of elements, their decomposition, interfaces (internal and external), physical constraints, and designs.

The analysis process proposed in Figure 7-2 intends to provide a structured and iterative definition of the problem and development of the solution. The iterative nature of the analysis process is characterised by the requirements, design and verification feedback loops in Figure 7-2. The requirements loop represents the fact that, for example, as new functions are identified, new derived requirements will need to be defined to quantify the functionality. The design loop ensures that as design decisions are made, specific functions, particularly at the lower levels, will be added or rearranged. The verification loop ensures that the solution domain maps correctly to the problem domain. The feedback to requirements indicates the need to confirm (or verify) that proposed solutions meet the requirements.

Requirements, functional and physical analysis processes are carried out through the simultaneous modelling of product, process and organisation. The modelling techniques used result from an expansion of software development techniques such as structured analysis and design ([Yourdon, 1988], [De Marco, 1978], [Marca, 1988]) and object oriented techniques ([Booch, 1993], [Rumbaugh, 1997], [Jacobson et al., 1994]).

As requirements and attributes are identified, the relationships among them can also be. The matrix at the bottom of Figure 7-2 represents, essentially, relationships among requirements and attributes. Section 6.5, in Chapter 6, provides a detailed description of the concept of relationships and their use in the scope of this thesis.

The modelling techniques used are detailed in Section 7.4. The analysis processes are described in Section 7.5.

### 7.4 Modelling

The modelling activity supports the analysis processes overviewed in Section 7.3. The purposes of modelling are to provide a structured way of obtaining requirements and attributes. The models chosen cover different views of product, process and organisation. Together those views are intended to provide a total view of product, process and organisation.
There is no intention to be prescriptive here. The set of models chosen is not assumed as being the best choice. The best set of models will be certainly those providing the best set of requirements and attributes. However, the investigation of the quality of the resulting requirements and attributes and the provision of guidelines for requirements and attributes capture and analysis are beyond the intended scope of this work. They could be the realm of specific disciplines such as requirements engineering and attributes engineering.

The modelling approach proposed in this section demonstrates that, although large, the scope of the 'total view framework' is feasible with currently available modelling techniques. The set of modelling notations chosen may not be the best choice but they cover the necessary (according to the literature) aspects of product, processes and organisation in order to provide a total view.

Figure 7-3 summarises the proposed modelling approaches and notations for the requirements, functional and physical analysis of product, process and organisation. These approaches and notations are well-established techniques for the structured analysis of software, real time systems or embedded real time systems. They are adapted to model hardware, processes and organisations. The use of the approaches and notations in the context of this thesis is explained in the following sub-sections. A more detailed description of each individual approach and notation can be found in Appendices A and C.

Whereas the objective of the approaches listed below is to derive the functional and physical specifications, the objective of the method proposed here is to provide a structured way of deriving requirements and attributes. Product, process and organisations are broken down into their constituent functions and physical parts so that the individual attributes of functions and parts can be derived and their relationships with the whole system requirements and attributes, identified.

<table>
<thead>
<tr>
<th>System Element</th>
<th>Product</th>
<th>Process</th>
<th>Organisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>Approach: proposed Notation: adapted DFD</td>
<td>Approach: proposed Notation: adapted IDEF0</td>
<td>Approach: proposed Notation: adapted UCD</td>
</tr>
</tbody>
</table>

*Figure 7-3: Proposed modelling approaches and notations*

The approaches here presented were selected in order to provide a total view of product, process and organisation, and, therefore, to make possible that a complete set of requirements and attributes would be derived. In order to provide a total view, the approaches chosen must be able to cover all necessary views to model product, process and organisations. Many authors describe which views are necessary for
modelling product, process and organisations and these views are outlined in the following sections. In summary, a list of these views is:

- for product: functional modelling: activity, data and state; physical modelling: behaviour, structure and layout;
- for process: functional, behavioural, organisational and informational;
- for organisation: function, information, resource and structure.

The objective of the method here proposed is to get a set of requirements and attributes as complete as possible. If it is perceived that repeated sets of attributes are coming from a given model, that model can be made redundant. The method, approaches and notations must serve the purpose of deriving requirements and attributes. Otherwise, it will be modelling for the sake of modelling.

Examples of the use of the modelling approaches and notations proposed in this chapter are presented in Chapters 9 and 10. Those chapters provide also an evaluation of the approach and notations for the sake of capturing requirements and attributes. The evaluation is also discussed in Chapter 11.

### 7.4.1 Product modelling

Product modelling is used in this thesis as a way of dividing the product in smaller functional and physical elements that will be further described by functional or physical attributes, respectively. Product modelling models a single end-product. If the object of product development is a family of products, each product in the family will be modelled individually. Obviously, it is expected these models to be very similar so they can be re-used from product to product in the same family. The focus of product modelling is to identify product performance attributes and the physical characteristics that determine them. The risk of not achieving a required performance is also contained in the set of attributes of interest. These attributes must correspond to the requirements analysed in the requirements models.

The product modelling approach proposed in this thesis is an expansion of the Yourdon approach. Yourdon (1989) proposed the use of structured analysis techniques for software development. The choice of the Yourdon approach can be justified by the fact that:

- Yourdon is a well established modelling approach on which many commercial CASE tools are based;
- Yourdon's modelling notations cover the necessary modelling views for requirements, functional and physical analysis. For functional analysis, there is a consensus for modelling in three complementary views: data, state and activity [Calvez, 1993]. Yourdon uses, respectively, entity relationship diagrams, state transition diagrams (STD) and data flow diagrams (DFD) to cover each of these views. For physical analysis, it is necessary to express structure and behaviour [Calvez, 1993]. [Stevens et al., 1998] states the need to represent layout. The use of a hierarchy of DFDs, STDs and structure charts in the Yourdon approach meet the requirements for structure and behaviour. Layout is covered by CAD drawings, prototypes, etc.

The Yourdon approach was developed for software development. As modern complex products contain also hardware elements, the Yourdon approach needs to be expanded. The work of Hubka and Eder (1988), formulating a generic theory of technical systems, shows that it is necessary to model also:
- energy and material couplings in addition to only information (data) couplings;
- physical connections in addition to only functional logical connections.

Figure 7-4 is an overview of the product modelling approach proposed in this thesis. It is an expanded version of the Yourdon approach, including the necessary hardware modelling capabilities. The numbering in Figure 7-4 represents:

1) a DFD is used to develop a context diagram with the end-product mission/objective in a central bubble surrounded by the stakeholders (rectangles) who interact with the product. The flows linking the central bubble to the stakeholders represent the stakeholders concerns (e.g. functionality, assemblability) towards the product. The direction of these flows is irrelevant.

2) a DFD that defines the functional boundaries of the product. It is a context diagram with the end-product mission/objective in the central bubble and the environment elements in the rectangles. These rectangles are called terminators. Differently from the DFD in 1), the terminators in this diagram may represent any (not only human stakeholders) element in the environment interacting with the product. The functions accomplished by the end-product will be expanded from the initial end-product mission/objective. The functions outside the end-product functional scope are supposed to be accomplished by the terminators in the environment. The arrows connecting the central bubble to the rectangles represent not only data flows as in the original DFD conception, but also material and energy flows. The direction of these arrows is relevant and represents data, material or energy coming in or out of the end-product.

3) for each of the terminators in 2), the actions of the terminators towards the product (stimulus) and the response of the product are represented in a list of events.

4) each of the response in the events list potentially represents a function of the product. Each of these functions can be represented by a bubble in a DFD. Some of these process functions may have to be triggered, enabled or disabled by control functions, represented by dashed bubbles. The flows between these functions and the environment are then identified. Consistency must be checked against the diagram in 2) and corrections may be necessary. Flows between functions are identified and they can be used as criteria for clustering functions into higher level functions (bottom-up approach);

5) control functions can be further expanded into state transition diagrams. Product states are identified and conditions and actions for state changes are explored. Among these actions are those to trigger, enable or disable process functions;

6) the relationships among data, energy, material represented by the flows in Diagram 4 can be described using an Entity Relationship Diagram;
1) DFD adapted for requirements context diagram

Requirements analysis

Functional analysis

2) DFD, functional context diagram

Physical analysis

7) FBD, end-product and its connections with environment

8) DFD, flows among component subsystems

9) FBD, connections among component subsystems

10) DFD representing software tasks

11) Structure charts, software module structure

12) CAD drawing, layout

Figure 7-4: Overview of product modelling

7) this constitutes an adaptation of the Yourdon approach in order to model physical connections. Diagram 7 is a Function Block Diagram (FBD) where, again, the central bubble represents the end-
product under development in context. The surrounding rectangles are called terminators and they represent the elements in the environment that are, in any stage of its life cycle, physically connected to the end-product. The links between the end-product and the terminators represent these physical connections.

8) and 9) the end-product in Diagram 7 can be expanded in order to model its component subsystems and their relationships. These relationships can be functional relationships which show data, energy or material flows among subsystems or physical connections. Diagram 8 is a DFD representing an architecture flow diagram [Hatley and Pirbhai, 1988] with the functional connections among component subsystems. Diagram 9 represents an architecture interconnect diagram [Hatley and Pirbhai, 1988] with an FBD showing the physical connections among component subsystems.

10) and 11) software subsystems can be further expanded, from processors through tasks onto modules. A hierarchy of DFDs can be obtained as exemplified in diagram 9. A particular software can be structured into modules organised on a Structure Chart (STC) as according to the Yourdon approach.

12) illustrates what can be a CAD drawing for layout representation.

Diagrams 2 to 6, 8, 10 and 11 follow the Yourdon approach with the flows representing not only data or information but also material and energy. Diagrams 1, 7, 9, 12 are proposed as expansions to the Yourdon approach in order to implement the requirements and physical analysis processes. Diagram 1 is part of the product requirements analysis. Diagrams 2 to 5 are part of the product functional analysis. Diagrams 6 to 10 are part of the product physical analysis.

7.4.2 Process modelling

Process modelling, in this thesis, models the life cycle processes of the complex product under development. For each product under development, there will be a corresponding set of life cycle processes. Process modelling is intended for the derivation of time-related attributes of the product life cycle processes. The risk of not meeting a required schedule is also part of the set of attributes of interest. Therefore the process modelling approach used concentrates on activities modelling.

The review of process modelling by [Curtis et al., 1992] indicates that processes are most commonly represented in four views:

- **functional** represents what process elements are being performed, and what flows are relevant to these process elements.
- **behavioural** represents when process elements are performed (e.g. sequencing), as well as aspects of how they are performed through feedback loops, iteration, complex decision-making conditions, entry and exit criteria, and so forth.
- **organisational** represents where and by whom (which agents) in the organisation process elements are performed, the physical communication mechanisms used for transfer of entities, and the physical media and locations used for storing entities.
- informational represents the informational entities produced and manipulated by a process; these entities include data, artefacts, products (intermediate and end), and objects; this perspective includes both the structure of informational entities and the relationships among them.

[Schlenoff et al., 1996] identified requirements for modelling processes. Among those requirements, the ones which affect the derivation of time-related attributes include the representation of: complex sequences (alternative, concurrent, parallel and serial activities), conditional activities, iterative loops, resource allocation, states, activity attributes.

Available modelling notations which can address most of the requirements listed above are SADT/IDEFO [Marca and McGowan, 1988] and SREM/DCDC's Behaviour Diagram [Alford, 1990]. In IDEF0 covers the functional and partially, the informational and organisational views. In IDEF0: activities, inputs and outputs fulfill the functional view; control represent the informational entities produced and manipulated by a process in the informational view; and mechanisms represent where and by whom the activity is performed in the organisation view. The behavioural view is covered by the Behaviour Diagram. Organisation modelling in Section 7.4.3 complements the informational and organisational views of process modelling.

Figure 7-5 is an overview of the process modelling approach proposed in this thesis. It uses IDEF0 and Behaviour Diagram notations in the same diagram. The numbering in Figure 7-5 represents:

1) Process requirements model: represented by an adapted IDEF0 notation in which the function of the process to be modelled is linked to the stakeholders who are sources of inputs, control and...
mechanisms and to the destination of outputs. The direction of the links is irrelevant and these links represent the concerns of these stakeholders towards the process under modelling.

2) Process functional model: is represented by a Behaviour Diagram but can also include the IDEF0 notation. It focuses on expanding the life cycle process in Diagram I to represent its functions and scenarios over a time line with emphasis to activity, input, output and control elements (in order of decreasing priority). Mechanisms are considered during functional analysis if their use is part of the constraints or assumptions from the resulting requirements analysis. Process functional analysis may model the ideal, to-be or text-book process. The process functional models must contain every process activity required in the requirements set and must justify or provide a rationale for the activities designed in the process physical analysis.

3) Process physical model: is represented by the combination of the IDEF0 notation with the Behaviour Diagram notation. It aims to describe a life cycle process as it is. The life cycle process in diagram I is expanded into the activities actually carried out. Every activity in the Behaviour Diagram comes with its corresponding inputs, control, outputs and mechanisms. Inputs and outputs can also represent states. The time line linking activities may also express the conditions necessary to move from one activity to the next.

The functional and physical process decomposition that plays an important role in the process functional and physical analysis processes is based on the SADT approach described in [Marca & McGowan, 1988] which can be summarised as following:
- Generate a list of major groups and categories of things (data, material, energy) used and generated by the process or activity to be modelled – ‘data list’;
- Generate a list of activities that use each class or collection of data – ‘activity list’;
- Cluster all the activities into aggregates;
- Draw different levels of IDEF0 diagrams corresponding to the activity clustering;
- Review the process and correct diagrams if necessary.

On each IDEF0 diagram, the time relationship between activities is obtained by using the Behaviour Diagram notation.

7.4.3 Organisation modelling

Organisation modelling concerns the modelling of the business organisation that performs a given life cycle process. For each product under development there will be organisations carrying out its life cycle process. The organisation modelling takes into consideration that the organisation may perform the life cycle processes for other products as well. For example, Ford Motor Company develops, produces, distributes, supports many vehicles simultaneously through different vehicle programs. Also a production organisation may be organised to produce a number of products in order to take the most of the resources available. Organisation modelling is intended for the derivation of productivity-related attributes and also the risk attributes associated with them. Examples of such attributes are: cost, capacity, risk of machine failure, etc.
The European Pre-Standard ENV 40 003 entitled 'Framework for Enterprise Modelling' defines four views for organisation modelling [Vernadat, 1996]:

- **the function view**, which provides a hierarchically structured description of the functions, behaviour (dynamics), and functional structure (statics) of the organisation with relevant inputs and outputs;
- **the information view**, which provides the description of a structured set of organisation objects that were identified in the other views;
- **the resource view**, which provides a description of the resource organisation, i.e. the set of resources required to execute the organisation operations;
- **the structure view** (preferred to the original term 'organisation view' for not mixing up with other uses of the word 'organisation' in this thesis) which provides the description of the structure of the organisation, the responsibilities of the individuals, and the organisation units within the enterprise.

In order to cover all views listed above, the organisation modelling approach used in this thesis is based on the approach proposed by Jacobson et al. (1994) but using the UML (Unified Modelling Language) notation for the object model [Texel and Williams, 1997]. The functional view is covered by Jacobson's Use Case modelling approach. The other views are covered by the UML notations, for example: class diagram for information and resource view, collaboration and deployment diagrams resource view; package diagram for organisation view.

Jacobson proposed two types of models of organisations: external and internal. The external model describes the organisation and the world external to it. It describes the processes in the organisation that satisfy the stakeholders' interests. The interface between each process and its external environment is also part of this external model. For this external model, Jacobson proposed and developed the use case model. The internal model of the organisation describes each business process: how they are built up of different work tasks (internal processes) and the various types of resources that they exploit or produce. The organisation can be structured into sub-businesses (functions). The internal model should then be able to show the sub-businesses to which a given task is assigned; that is, where a resource for carrying out a particular task can be obtained. Jacobson's internal model is object orientated with a specific set of notations. In this thesis, it is proposed the use of the UML notations for object modelling. The UML notations include but are not limited to the notations proposed by Jacobson.

In this thesis, organisation modelling is performed through the requirements, functional and physical analysis processes. There is an organisation model corresponding to each of the analysis processes.

The organisation requirements model consists of a use case diagram containing the name of the organisation performing the process linked with the stakeholders who interact with the organisation. The links between organisation and stakeholders represent the stakeholders concerns towards the organisation and their

![Figure 7-6: Organisation requirements model](image-url)
direction is irrelevant. Figure 7-6 illustrates the organisation requirements model.

Figure 7-7 illustrates the organisation functional modelling process. The numbering in Figure 7-7 represents:

1) The organisation functional model starts with a Use Case Diagram, as conceived by Jacobson. The use case model captures the part of the organisation that is of interest. It will describe the organisation's internal processes and external environment. The internal processes are modelled with use cases, while the environment is modelled by actors. The use case model also shows how the external environment interacts with the organisation, that is, how individual actors communicate with use cases within the organisation. The use case model describes the organisation as it is seen externally, that is, how it is perceived by those who wish to use it. Internal processes that cannot be perceived by the actors are not represented in the use case model. The actor represents a role that someone or something in the environment can play in relation to the organisation.

2) Each use case may be complex or compound and may require decomposition into more elemental, simpler use cases. Each use case contains one or more possible sequences or flows of events between the
system and the environment: a normal sequence, and sequences that relate to alternate cases or error handling situations. Each path through these multiple event flows is called a scenario. For example, a use case 'maintain employee information' can have scenarios such as 'create hourly employee pickup delivery', 'review employee', 'delete employee'. For consistency purposes, the scenarios must be linked with the same actors of their parent use cases. Each use case scenario can be further detailed in a Sequence Diagram and/or a Collaboration Diagram described below.

3) A Sequence Diagram depicts the set of events sequenced in time that compose a use case scenario. The Sequence Diagram identifies the external actors and classes (or instances of classes: objects) involved in the interaction. Objects in the organisation model must represent: the things and products that are used during a flow of events in the organisation, the tasks that must be performed during a flow of events and, the way by which the organisation communicates with the outside world. Examples of objects for modelling a restaurant are: cloakroom attendant, order handler, food preparer, menu, order. Each class or object has a corresponding lifeline. On each lifeline, the sequences of events that the objects will carry out are represented as rectangles. Between the lifelines, the Sequence Diagram shows the stimuli (material, energy or information) sent between the participants in the interaction. In the functional model, the classes and objects focuses on the representation of the essential products, tasks or interfaces in the organisation, disregarding how they are implemented in practice.

4) The Collaboration Diagram shows the same information as the Sequence Diagram, but without the time sequenced view. It therefore shows the totality of all stimuli flows, and explicitly shows which actors and/or classes exchange stimuli, and what stimulus they exchange. The Collaboration Diagram is more useful for organisation functional architecture development than for elicitation of attributes. For elicitation of attributes, the Sequence Diagram may be used in isolation.

The organisation physical model aims to describe how the organisation is internally designed or implemented through its resources whereas the organisation functional model focuses on the functions, processes, activities and tasks performed by the organisation. Similarly to the organisation functional modelling process, the organisation physical model starts from a use case diagram. However, the object of the physical modelling is the organisation in the use case diagram. In the organisation functional model, the object of modelling were the use cases. Figure 7-8 illustrates the organisation physical modelling process. The numbering in Figure 7-8 represents:

1) The organisation physical model starts with the same diagram used for the functional model. The distinction comes in the following steps when the object of analysis becomes the organisation in Diagram 1 instead of the use cases as in the functional model.

2) Diagram 2 is a Package Diagram. Its use is adapted in this thesis in order to show the organisation structure. Each organisation unit (departments) can be further expanded. Object modelling applies then to each lowest level organisation unit. In the functional model, there was no concern about organisation structure. Use cases were expanded regardless the existence of an organisation unit to perform them.
3) Diagram 3 adapts the Package Diagram from the UML notation in order to prevent the ambiguity that could otherwise occur if processes and resources (represented by object classes or object instances) were mixed in the same Class Diagram or Component Diagram.

4) A Component Diagram is adapted from the UML notation in order to show the structure of processes that can be carried out by the organisation or organisation unit. The functional model showed the use cases regardless of which ones were actually performed by the organisation. The processes package, when expanded through into a Component Diagram, aims to show the use cases that are actually implemented by the organisation.

![Organisation physical modelling diagram](image-url)

*Figure 7-8: The organisation physical modelling*
5) A Class Diagram is an entity-relationship diagram that expands a resource package in order to show the resource classes or instances and their relationships. The organisation information model is included in this resource model.

6) The Deployment Diagram from the UML is adapted in order to model the physical links between the resources in the organisation. The Deployment Diagram in the UML notation is used to represent the processors in the hardware and their physical links.

7) A Statechart Diagram depicts a finite-state-machine that describes how the resource class or instance responds to different external stimuli. These stimuli are the receipt of material, energy or information by instances of the class. Dynamic behaviour is described in terms of a set of potentially nested states, the transitions between these states, the events that trigger these transitions, and actions that are performed, both while transitioning between states, and on entry to, whilst within, and on exit from a state.

8) An Activity Diagram shows the internal behaviour of the resource class whereas the Statechart Diagram provide an external view of class behaviour. The Activity Diagram shows one or more parallel sequences of possible behaviour, distributed along a time line, which in turn may have branches and optional synchronisation.

7.5 Analysis

As already mentioned above, the objective of the analysis processes is to derive requirements and attributes. Relationships among requirements and attributes are then identified and analysed as according to what has been already described in Chapter 6, Section 6.5.

The analysis processes described here mirror the requirements and architecture definition of the systems engineering process as defined by the IEEE-1220-1994 [IEEE, 1995]. They are adapted mainly from the IEEE-1220-1994 standard on systems engineering and, Martin’s systems engineering guidebook [Martin, 1997]. Martin was the leader of the team who developed the EIA-632 standard [EIA, 1997]. Other references considered here are: [Stevens et al, 1998], [Cross, 1989], [Pugh, 1990].

This section aims to demonstrate how product, process and organisation consideration can be taken into account from the beginning of the product development process. This is done through the analysis processes carried out in a structured manner through the modelling approaches described in Section 7.4.

The concurrent engineering facet of the ‘total view framework’ is demonstrated in this section through the simultaneous analysis of product, process and organisation. According to the ‘total view framework’, proposed and described in Chapter 5, the analysis dimension is composed of the requirements, functional and physical analysis processes. These processes relate to other processes in the systems engineering processes and in the product life cycle and also, relate to each other. Systems engineering processes interacting with the analysis processes include: the systems engineering management process, the system integration and verification process, etc. Other life cycle processes interacting with the analysis processes include: production which may receive the layout drawings of product parts to be assembled or integrated or assembly procedures, testing which may feedback changes, etc.
Figure 7-9 illustrates the analysis processes and their interactions with each other. The analysis processes receive inputs from the systems engineering management process, for example. These inputs are management plans, management directions and initial requirements expressing a market need or opportunity. The requirements analysis process identifies stakeholders and their requirements and translates these requirements into technical requirements. The technical requirements are the starting point for functional analysis. Through functional analysis, functional architectures of product, processes and organisations are developed and are the basis for the derivation of functional attributes. The functional architecture is the starting point for physical analysis. Physical analysis develops or identifies the physical architectures of product, processes and organisations from which physical architectures are identified. Normally, every function in the functional architecture has a corresponding technical requirement that justifies its existence. However it is possible to identify necessary functions that have no correspondent in the technical requirements set. Therefore, the technical requirements set will need to be modified. This fact is represented by the requirements loop. Also, every component, task or resource or group of them in the physical architecture must accomplish a given function in the functional architecture. However, some depending on the choice of the components, or tasks or resources, e.g. safety critical elements, the functional architecture and possibly the set of functional attributes will need to be altered. This fact is represented by the design loop. The verification loop represents the need of assurance if a given physical architecture will meet the technical requirements or if the use of a solution element creates a constraint not thought before, for example.

The analysis processes are further detailed in Sections 7.5.1, 7.5.2 and 7.5.3. As indicated in Figure 7-9, Figures 7-10, 7-20 and 7-26 expand the requirements, functional and physical analysis processes, respectively.

7.5.1 Requirements analysis

The requirements analysis process consists of the identification of an initial set of loosely defined stakeholder requirements and its transformation into a complete and consistent set of technical requirements. The set of technical requirements contains conditions and functional and performance
requirements of the end-product, its life cycle processes and their performing organisations. An indication of whether it can be traded-off or not is assigned to each requirement in the requirement set. The resulting set of technical requirements must be traceable to the initial and other identified stakeholder requirements, to any assumption made by the developers and to any established goal of the development organisation.

Various sources ([Blanchard, 1990], [Stevens et al., 1998], [Martin, 1997], [IEEE, 1995], [Ulrich & Eppinger, 1995], [Cross, 1989]) describe a requirements analysis process or equivalent for product development. This section builds on that previous work and aims to adapt a requirements analysis process for the simultaneous and integrated development of an end-product, its life cycle processes and their performing organisations. As such it makes use of the modelling approaches described in Section 7.4 in order to simultaneously analyse requirements for product, processes and organisation.

Figure 7-10 provides an overview of the requirements analysis process. Each of the sub-processes depicted in Figure 7-10 is described in the following paragraphs using as an example the integrated development of a car. The requirements refinement loop assures the fact that technical requirements must map to stakeholder requirements, assumptions or goals.
Define end-product mission/objective

The end-product mission/objective consists of the prime purpose of the system and its operational profile. The mission states in one-sentence what is the system to accomplish. It states the reason of being of the system, its ultimate objective. Consider the example of a car. Its mission or objective can be stated as: ‘to transport private riders on a road’. The end-product mission or objective is the starting point for product modelling. Figure 7-11 places the car mission in a DFD bubble.

Identify potential life cycle processes

Based on the nature of the end-product, whether it is software, hardware, electronic, mechanic or mechatronic, or on existing similar end-products, the main non-operating functions of the system can be identified. These non-operating functions are the object of the end-product’s life cycle processes. Figure 7-12 shows a set of time functions of a Behaviour Diagram (BD) representing an initial list of life cycle processes.

Analyse life cycle process scenarios

Many systems engineering process descriptions such as those in the IEEE-1220 Standard or in [Martin, 1997] include, as part of the requirements analysis process, the identification and definition of operational scenarios that define the range of anticipated uses of the system product. As this thesis proposes the simultaneous and integrated development of product, processes and organisations, it is more appropriate to expand the term operational scenarios to life cycle process scenarios. The term life cycle process scenarios include the system operational and non-operational scenarios.

Scenarios may refer to possible functions embedded in the life cycle process term [Martin, 1997]. For example the life cycle process ‘maintain car system’ in Figure 7-12 may have the scenarios depicted in the time functions of a Behaviour Diagram in Figure 7-13.

Scenarios may also refer to a defined sequence of activities that expands the life cycle process term [Stevens et al., 1998]. Figure 7-14 describes the scenarios for the life cycle process ‘use car system’.
Identify life cycle process performing organisations

Each life cycle process scenario identified is going to be performed by an individual or by an organisation. There may be an organisation that provides a service for 'towing a car', another for 'refuelling', another for 'fixing'. All scenarios in the life cycle process 'use car system' are going to be performed by the driver and the passengers. The life cycle process 'manufacture car system' is going to be performed by a manufacturing organisation which may produce not only that car under development but also many other cars of the same or different types. The same organisation may perform more than one life cycle process or life cycle process scenario. Each organisation performing a life cycle process scenario is represented by a Use Case Diagram (UCD). Figure 7-15 represents an organisation that develops, manufactures, distributes cars, provides after sales support (part of the maintenance life cycle process), and manages all these processes.

![Use Case Diagram](image)

**Figure 7-15: Organisation performing non-operating functions**

Define the scope of the development effort – goals and assumptions

The development of the end-product and its life cycle processes are part of the development effort. Some organisations performing life cycle processes or their scenarios, however, are outside the scope of the development effort since the development agencies cannot include their attributes as part of the trade-off process. However, the requirements from these organisations affecting the end-products and its life cycle processes must be considered when developing the system solution. For example, driving schools, service stations, garages, scrap yards are outside of the development scope of Ford Motor Company. However, all of Ford's business processes must be always adjusting themselves to modifications in the product or its life cycle processes. The development agencies within Ford shall not seek only an optimal product from the user's point of view or the best design for manufacturing, they must seek also the optimisation of the whole Ford Automotive Operations. Ford Automotive Operations include the development, production, distribution and after sales support organisations as anticipated in Figure 7-15. These organisations must then be within the scope of the development effort when Ford develops a car.

Adapting [Ulrich and Eppinger, 1995]'s approach to define the scope of the development effort, it is proposed here to include the definition of business goals, target market(s) for the end-product and assumptions at this stage of requirements analysis. [Stevens et al., 1998] provide further details about these elements and their definitions. Examples of business goals for car development are: car will be model year 2003, greatest market share by the year 2005. Example of target market is: luxury car. Examples of assumptions are: automatic transmission, 4-wheel drive.
Identify stakeholders

The identification of stakeholders is performed by identifying the people or organisations who are affected by the attributes of the end-product, its life cycle processes and their performing organisations within the scope of the development effort. A way of identifying the stakeholders is to separate the system into product, life cycle processes and organisations and investigate who are the people or organisations directly interacting with each of them.

For product, e.g. the car product, a question that can be made, in order to identify stakeholders, is: ‘who are the people who directly interact with the car during its potential life cycle scenarios?’ Candidates are: owner, driver, passengers, passers by, tuner, distributor, dealer, driving instructor, garage mechanic, etc. Observe that the question covers the entire life cycle and not only the end-product use.

For process, i.e., each identified life cycle process scenario, candidate stakeholders are the people or organisations who are: the sources of input, the destination of outputs, the providers of mechanisms, the mechanisms themselves, the sources of control or the controllers themselves. Examples of these stakeholders for the car manufacturing process are: suppliers of parts, managers, machine operators, etc.

For organisation, i.e., each life cycle process performing organisation within the scope of the development effort, stakeholders are the people outside that organisation who can play a role in relation to the business. Examples of organisation stakeholders are stockholders, government, etc. Jacobson et al. (1994) proposes that a good way of finding suitable organisation stakeholders is to name at least two or three people who could serve as the stakeholders in question. For example, if the organisation is specialised in selling cars, one would realise that a stakeholder called ‘the customer’ is, in fact, three different stakeholders: firstly, the normal users of the car (drivers, passengers, etc.); secondly, the buyer of the car, i.e., someone who is competent in purchasing it, but will not be its end user; thirdly, someone who is competent to judge the car in comparison with competing cars.

Product, processes and organisations may have stakeholders in common. The aim, at this stage, is to obtain a list of system stakeholders as complete as possible no matter how each stakeholder interacts with the system.

Define measures of effectiveness

The IEEE-1220 standard defines measures of effectiveness as the metrics by which the customer will measure satisfaction with products produced by the technical effort. Measures of effectiveness reflect overall customer expectations and satisfaction. Key measures of effectiveness may include performance, safety, operability, reliability, and maintainability, or other factors [IEEE, 1995]

This thesis proposes an expansion of the above definition. Measures of effectiveness are the metrics by which stakeholders will measure their satisfaction with the system solution resulting from the development effort. Stakeholders include not only customers. The system solution includes not only the end-product but also its life cycle processes and some of their performing organisations.
A way of identifying measures of effectiveness is to identify the stakeholder concerns towards product, processes and organisation. Concerns may also be used as criteria for grouping requirements. Figures 7-16, 7-17 and 7-18 illustrate simplified versions of the product, processes and organisation requirements models of a car showing the stakeholders concerns towards the system operating and non-operating functions. The modelling activity itself is described in Section 7.4.

**Figure 7-16: Product - DFD representing product, stakeholders and concerns**

**Capture stakeholder requirements**

For each of the stakeholders identified, requirements are captured. Stakeholder requirements can be captured in many ways.

For example, adapting the Yourdon’s approach [Yourdon, 1988], it can be derived a systematic way of capturing stakeholder requirements. It uses the concept of ‘events’. ‘Events’ can describe how stakeholders interact with the product, process and organisation elements of the system. ‘Events’ have: ‘stimulus’ and ‘response’. ‘Stimulus’ will derive the ‘condition’ information related to a requirement. ‘Response’ will derive the ‘function’ information of the requirement. ‘Performance’ information may be included in the ‘response’ frame. Examples of events driving requirements capture are:

For product, e.g., a car:

Event 1:
Stimulus 1: Rider as operator turns the key clockwise.  
Response 1: Vehicle turns on.

For process, e.g., a manufacturing process:

Event 2:
Stimulus 2: Manufacturing equipment supplier offers limited set of manufacturing equipment.  
Response 2: Manufacturing process adapts to manufacturing equipment offered.
Figure 7-17: Processes - BD representing life cycle processes, stakeholders and their concerns

Figure 7-18: Organisation - UCD representing organisation, stakeholders and their concerns
For organisation, e.g., a development organisation:
Event 3:
Stimulus 3: Shareholders limit investment.
Response 3: Development organisation reuses previous designs.

Many authors list sources of stakeholder requirements. Stevens et al. (1998) provides a list of sources of users requirements. Sources of stakeholder requirements can be obtained by replacing the word 'stakeholders' for the word 'users' in Stevens' list: interviews with stakeholders, derivation from business requirements, working in the stakeholder environment, analogous or existing system, change suggestions and problem reports, innovation work, stakeholder group meetings, workshops, studies or descriptive documents, prototypes, new technology, questionnaires, stakeholder modifications of existing systems. Pugh (1991) provides the following additional sources of stakeholder requirements: legislation, patents, trade marks, registered designs and copyright; reports, proceedings and reference books; official and private representative bodies; statistical data; market data publications; specialist libraries; engineering science information.

Organise assumptions, goals and stakeholder requirements

Assumptions, goals and stakeholder requirements can be organised in a hierarchy in which the lower levels detail the upper levels. This organisation approach is detailed in Stevens et al., 1998] and Cross, 1989]. An example of stakeholder requirements hierarchy is:

Req. 1: Provides for personal comfort/convenience
  Req. 1.1: Comfortable front seats
  Req. 1.2: Driving position is fully adjustable
  Req. 1.2.1: Can personalise seat, head rest and arm rest positions

Stakeholder requirements, assumptions and goals can also be classified according to:
• concerns: examples of concerns are: driveability, manufacturability, investment efficiency. They are illustrated as flows in Figures 7-16, 7-17 and 7-18.
• type: condition, function, performance or interface
• compliance level: mandatory, desirable or optional.
• status: to_be_defined, to_be_reviewed, to_be_approved, to_be_deleted, to_be_verified
• allocation (PPO): refers to the requirement allocation to any combination of product, process and organisation. A clue for this allocation is the requirements model that contains the stakeholder corresponding to the requirement.
• constraints (Yes/No) or trade-off ability: indicates whether the requirement is a constraint or not.

Associated with each requirement, assumption or goal, there must be a definition of the way and procedure by which it will be verified.

Figure 7-19 exemplifies some stakeholder requirements.

Define functional requirements, performance requirements and conditions

Stakeholder requirements may not be expressed in objective, quantitative or verifiable terms. However, they, together with assumptions and goals, are the basis from where conditions, functions and required
performance will start to be defined. Examples of these conditions, functions and performance information extracted from stakeholder requirements and organised in a hierarchical format are:

<table>
<thead>
<tr>
<th>Number</th>
<th>Test</th>
<th>Concern</th>
<th>Type</th>
<th>Compliance</th>
<th>Status</th>
<th>PPO</th>
<th>Constraint</th>
<th>Verifiability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S003</td>
<td>Provides for personal comfort/convenience</td>
<td>Comfort</td>
<td>F</td>
<td>MTB</td>
<td>Prod</td>
<td>Y</td>
<td>D</td>
<td>TBD</td>
</tr>
<tr>
<td>S003.001</td>
<td>Comfortable front seats</td>
<td>Comfort</td>
<td>P</td>
<td>MTB</td>
<td>Prod</td>
<td>Y</td>
<td>D</td>
<td>TBD</td>
</tr>
<tr>
<td>S003.002</td>
<td>Driving position is fully adjustable</td>
<td>Comfort</td>
<td>F</td>
<td>DTB</td>
<td>Prod</td>
<td>N</td>
<td>D</td>
<td>TBD</td>
</tr>
<tr>
<td>S003.002.001</td>
<td>Can personalize seat, head rest and arm rest position</td>
<td>Comfort</td>
<td>F</td>
<td>DTB</td>
<td>Prod</td>
<td>N</td>
<td>D</td>
<td>TBD</td>
</tr>
<tr>
<td>S009.007.001</td>
<td>Starts quickly every time whether hot, cold or wet</td>
<td>Driveability</td>
<td>P</td>
<td>MTB</td>
<td>Prod</td>
<td>Y</td>
<td>T</td>
<td>R202</td>
</tr>
<tr>
<td>S009.007.002</td>
<td>Never stalls when cold or hot</td>
<td>Driveability</td>
<td>P</td>
<td>MTB</td>
<td>Prod</td>
<td>Y</td>
<td>T</td>
<td>R202</td>
</tr>
<tr>
<td>S009.007.003</td>
<td>Quick to idle and settle</td>
<td>Driveability</td>
<td>P</td>
<td>MTB</td>
<td>Prod</td>
<td>Y</td>
<td>T</td>
<td>R202</td>
</tr>
<tr>
<td>S009.007.004</td>
<td>Consistent start times</td>
<td>Driveability</td>
<td>P</td>
<td>MTB</td>
<td>Prod</td>
<td>Y</td>
<td>T</td>
<td>R202</td>
</tr>
<tr>
<td>S009.007.005</td>
<td>No overheating</td>
<td>Driveability</td>
<td>P</td>
<td>MTB</td>
<td>Prod</td>
<td>Y</td>
<td>T</td>
<td>R202</td>
</tr>
<tr>
<td>S012</td>
<td>Easy to service</td>
<td>Maintainability</td>
<td>F</td>
<td>MTB</td>
<td>Proc</td>
<td>Y</td>
<td>T</td>
<td>CETP</td>
</tr>
<tr>
<td>S020</td>
<td>Reuse resources from previous vehicle programs</td>
<td>Investment efficiency</td>
<td>F</td>
<td>DTB</td>
<td>Org</td>
<td>N</td>
<td>D</td>
<td>TBD</td>
</tr>
</tbody>
</table>

Figure 7-19: Examples of stakeholder requirements organisation

- Condition C000 'When starting the vehicle';
- Condition C001 'Engine temperature';
- Condition C001.001 'Engine off for two hours' which actually refers to engine temperature when engine is off for two hours. This condition is a child of condition C001;
- Condition C001.002 'Engine just turned off' which actually refers to the engine temperature when the engine was on and just turned off.
- Function F001 'Engine comes up to idle speed', which is a functional requirements saying that the engine must return or go to idle speed;
- Performance P002 'Time to idle speed is 10 seconds', which specifies the actual performance requirements for time to idle speed.

Compile a technical requirements document

The conditions, functions and performance information derived from stakeholder requirements, assumptions and goals are then completed, combined and expressed in objective, quantitative (whenever possible) and verifiable terms. An example of a technical requirement that includes considerations on the conditions, function and performance information selected above is:

Requirement T003.001.001

'After the vehicle has been stopped for two hours with engine off or after just turning the engine off, when starting the vehicle with A/C on or off and at full or no electrical load at temperatures of (4±-1) or (35±/-1) degrees Celsius, at altitudes of sea level or (2000±/-10) meters the engine takes no more than 10 seconds to come up to idle speed and it goes to idle speed at a rate of 100RPM/second.'
Adapting the outputs of the requirements analysis phase proposed by [Stevens et al., 1998] and by the IEEE-1220 standard [IEEE, 1995], the following provides a list of items a technical requirements document may include:

- **goals**: establish the business requirements for the system to be developed.
- **the system primary functions**: end-product mission/objective, life cycle process scenarios and further detailed capability required by stakeholders.
- **stakeholder characteristics**: describe who are affected by the product, processes and organisation attributes and when.
- **performance requirements**: defines the required performance of the system.
- **conditions**:
  - interfaces with end-users, testers, adjusters, repairers and other stakeholders;
  - environment for each of the life cycle process scenarios: weather conditions (e.g. rain, snow, sun, wind, ice, dust, fog), temperature ranges, topologies (e.g. ocean, mountains, deserts, plains, vegetation), biological (e.g. animal, insects, birds, fungi), time (e.g., day, night, dusk), induced (e.g. vibration, electromagnetic, chemical).
  - human factors considerations (e.g., eye movement, reach, ergonomics)
- **constraints**:
  - project constraints: approved specifications and baselines, updated technical and project plans, team assignments and structure, automated tools availability or approval for use, control mechanisms, required metrics for measuring project progress.
  - enterprise constraints: management decisions from a preceding technical review; enterprise general specifications, standards, or guidelines; policies and procedures; domain technologies; established life cycle process capabilities; physical, financial, and human resource allocations to the project.
  - external constraints: public and international laws and regulations; the technology base; industry, international, and other general specifications, standards, and guidelines; and competitor product capabilities.
  - physical characteristics: colour, texture, size, mass, buoyancy, etc.
  - human factors: physical space limits, climatic limits, etc.
- **assumptions**: document each of the assumptions made by the product development organisation.

**Validate the technical requirements**

Consistently with the IEEE-1220-Std [IEEE, 1995], [Martin, 1997] and [Stevens et al., 1998], it must be ensured that the technical requirements are consistent and complete with respect to the stakeholder requirements, business goals, assumptions and identified constraints.

**7.5.2 Functional analysis**

The functional analysis process aims to identify the functional boundary of the system, the functions in the scope provided by the boundary, a functional architecture and the attributes of the elements of the functional architecture. The basic source of functionality is the compiled set of technical requirements
resulting from the stakeholder requirements. For each function resulting from requirements, hazard and failure mode effects and criticality analysis are performed. This may lead to the inclusion of preventive and protective functionality. A behaviour analysis performed in each function will provide the identification of inputs and outputs and how the functions interface each other. A functional architecture is drawn in order to provide a picture of system functions and their interactions. Each function in the system architecture is then described through the use of functional attributes.

The process summarised above is based on the IEEE-1220-Std [IEEE, 1995] and on the work of James Martin [Martin, 1997], the leader of the team who developed the EIA-632 standard [EIA, 1997]. It is however adapted to include the simultaneous functional analysis of product, processes and organisations in order to identify all attributes affecting stakeholder satisfaction. It is performed through the functional modelling approaches presented in Section 7.4.

Figure 7-20 provides an overview of the functional analysis process. The hazard analysis and FMECA loops characterise the fact that additional functions may be necessary in order to provide, for example, hazard monitoring functions, fault detection and recovery behaviour. The following describes each of the sub-processes depicted in Figure 7-20. The description of these processes is expanded using as an example the integrated development of a car.
Identify system boundaries and external functional interfaces

The functional analysis process starts by defining what is within the scope of system functionality and what is outside. In order to do so, the primary functions of the system are stated and the elements in the environment interacting with the system are identified. The primary functions include the end-product mission/objective, the end-product life cycle processes and its scenarios and the business processes which implement these life cycle processes. The elements in the environment are not limited only to people interacting with product, processes or organisations, as in the requirements analysis models. The environment also includes:

- elements interacting with the end-product over its life cycle;
- someone or something playing a role in relation to the business organisation.

The main objective of modelling the environment is to identify the system external interfaces. The functional interfaces are characterised by the data, energy and material crossing the boundaries of the system. The identification of elements in the environment eases the task of identifying the required inputs and outputs of the system.

Figures 7-21, 7-22 and 7-23 illustrates the top-level functional analysis models of a car, its life cycle processes and their performing organisations following the modelling approach described in Section 7.4. Observe in Figure 7-21 that the environment model contains life cycle process enabling products which interact with the end-product.

![Figure 7-21: Product - DFD showing a context diagram of a car](image-url)
Figure 7.22: Process - BD showing a functional analysis of a car life cycle process
Define states and modes

The conditions identified in the requirements analysis process serve as a basis for the identification of the states and modes of operation of the system. States can be derived for example from those conditions expressed as ‘when ...’ or ‘while...’ and define a set of circumstances characterising the system or system element at a given time. Examples of states in a car product are: parked or moving. When moving: accelerating, idle, decelerating. Examples of states for the development process are: waiting for requirements, designing, testing. Examples of states for organisation are: booming, over budget. Modes of operation groups system functionality for a given set of conditions or in a given state. For example, the powertrain control system has different modes of operation for when engine is cranking or running, the throttle is open or closed, the engine is hot or cold, different altitudes. A life cycle process may have different modes of operation, e.g., the Lucas’ concept of runners, repeaters, strangers in product development and in production. An organisation may have different modes of operation when operating in a developed country or in an emerging market, for example. States and modes of operation imply the need for defining control functions. These control functions will enable, disable or trigger other operational or non-operational functions.

Define functions

The traditional approach to integrated development, mainly implemented through the use of concurrent engineering tools, has been to define end-product functionality first and then optimise life cycle processes and organisation. In this thesis, it is proposed to identify operating and non-operating functions of the
system and investigate their interactions from the beginning of the product development process. Operating and non-operating functions can be decomposed into sub-functions. Sub-functions can be identified by the functional requirements embedded in the list of technical requirements resulting from the requirements analysis process or by using the modelling approaches proposed in Section 7.4.

Product functions are illustrated in Figure 7-24 with some identified sub-functions of a car: ‘Provide appearance’, ‘Provide space’, ‘Provide comfort’, ‘Provide visibility’, ‘Provide motion, alert and feedback’, ‘Provide communication, information and entertainment’. Observe in Figure 7-24 the consideration of sub-functions to interface with life cycle process enabling products: ‘Provide service interface’, ‘Provide testing interface’, ‘Provide calibration interface’. Other sub-functions are: ‘Provide input processing’, ‘Provide output processing’, ‘Provide user interface’, ‘Provide energy supply access’, ‘Provide output processing’. The DFD diagram in Figure 7-24 is a child diagram of the context diagram in Figure 7-21, i.e., the function in Figure 7-21: ‘transport private riders on a road’ can be decomposed into the sub-functions in Figure 7-24.

Process functions, e.g. sub-functions of the manufacturing process of a car, would be: purchase parts, position parts, transport parts, assemble vehicle, adjust vehicle, test vehicle.

Organisation functions, e.g. sub-functions of the use case ‘develop car systems’, are: ‘identify strategic intent’, ‘manage vehicle program’, ‘provide manufacturing tooling and facilities.’

**Perform hazard analysis**

Hazard analysis evaluates risk resulting from the frequency and gravity of events in the environment that can affect negatively the operating and non-operating functions of the system and risk reduction from the application of protective and preventive measures. As a result of hazard analysis, additional functions may be proposed in order to detect hazard conditions and implement preventive or protective behaviour. Hazard analysis can be applied not only to the product elements of the system, but also to the process and organisation elements. An example of a product related hazard is: driver has a stroke while driving the car. An example of a manufacturing process hazard is: a calibration error in a measurement equipment. An example of an organisational hazard is: government suddenly devaluates currency.

**Analyse failure mode effects and criticality**

Potential functional failure modes of operating and non-operating functions should be analysed in order to identify the need for fault detection and recovery functions. This procedure may lead to the inclusion of control or poke-yoke functions when the function under analysis is further decomposed. Again, failure modes are not something exclusive of products. Life cycle processes and organisations also may have their failure modes. Example of a manufacturing process failure mode is the fail to deliver the right piece to be assembled. Example of an organisational failure is a person with the inadequate skill for a job. An example of product failure mode is: a flat tyre in a car. Moreover, a failure on a product element may be recoverable through a process or organisation element of the system. An example of this is a car breakdown recovery service, which is essentially a life cycle process related system.
Analyse functional behaviour

The IEEE-1220 Standard [IEEE, 1995] recommends to analyse each subfunction and aggregate of subfunctions to:

- determine the responses (outputs) of the function(s) to stimuli (inputs);
- understand the functional behaviour of subfunctions under various conditions and check the integrity of the functional arrangement logic;
- identify and define states and modes for which subfunctions exhibit different behaviours;
- identify and define a functional time line for each operational and non-operational scenario.

STD, BD and Sequence Diagrams in the product, process and organisation functional modelling, respectively, support the functional behaviour modelling.

Define functional interfaces

[Martin, 1997] recommends to define the start/end states and inputs/outputs for each function. This ensures that all state transitions are completely defined within functions, and required inputs and outputs are provided. Identify the interface items that trigger each function. This can be done by connecting the functions in the DFDs (product functional model), in the BDs (process functional model) and Use Cases and Sequence Diagrams (organisation functional model) through flows of information, material or energy. Figure 7-24 illustrates a DFD representing the functional decomposition of a car with the functional interfaces defined by the flows of information, material and energy among the functions. For example, functional interfaces between the functions ‘provide comfort’ and ‘provide motion, alert and feedback’ are: movement, noise and forces.

Establish a functional architecture

The functions and functional interfaces identified above compose a functional architecture. The functions can be further detailed through the behaviour modelling. The IEEE-1220 standard [IEEE, 1995] recommends the establishment of a functional architecture, appropriate to the layer of development, to define lower-layer functional and performance requirements from which physical solutions will be determined. Figure 7-24 illustrates a simplified functional architecture of a car.

Verify functional architecture

The resulting functional architecture is then verified in order to assess the functional architecture completeness in satisfying the technical requirements. When functional architecture elements are not upward traceable to validated technical requirements, it must be determined if non-required functions and/or performance requirements were introduced during functional analysis, or whether valid functional and/or performance requirements were introduced and need to be reflected in the technical requirements set.

Define functional attributes

Functional attributes can be generated and attached to each of the functions or activities in the product, process and organisation functional models. Basis for the identification of functional attributes are:
Figure 7-24: Product – car functional decomposition
- performance requirements identified in the requirements analysis process and cross-referenced to functions in the functional analysis models;
- inputs and outputs in any of the functional analysis models;
- timing and resource utilisation which are more explicitly represented in the process functional models but can also be considered in the product and organisation models. For example: fuel consumption in a car product and investment efficiency in a car organisation;
- states and modes of operation;
- reliability, maintainability and risk attributes derived from failure mode effects and criticality and hazard analysis of product, process and organisation.

Examples of functional attributes corresponding to some of the functions in the models of Figures 7-21, 7-22, 7-23 are provided in Figure 7-25. Figure 7-25 shows the correspondence between the elements in the functional models and the functional attributes elicited to the corresponding function. Figure 7-25 illustrates the correspondence between the inputs and outputs in the functional model (see Figure 7-24) and the attributes elicited.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Provide comfort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provide motion feedback</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Regulation, form, environment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Force, form, diet</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Movement, mechanics, noise, and forces</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Atmosphere, conditioning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timing, to, environment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliability, design</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage organisation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 7-25: Links between functional attributes and functional model elements**
7.5.3 Physical analysis

The systems engineering process described by the IEEE-1220-Std [IEEE, 1995] uses the term ‘synthesis’ to refer to the process of defining system product solutions and identifying subsystems to satisfy the requirements of the verified functional architecture. Synthesis translates the functional architecture into a physical architecture that provides an arrangement of elements, their decomposition, interfaces (internal and external), physical constraints, and designs. [Martin, 1997] includes the generation of alternative physical solutions as part of the synthesis process. Alternate configurations, or architectures, are developed and evaluated against the requirements. Prototypes or models may be constructed for more than one architecture to support trade-off analysis.

The physical analysis process proposed in this thesis is essentially a modelling activity. It aims to identify the component elements of a physical architecture of the product, its life cycle processes and their performing organisations, the interactions among these elements and the attributes that characterise each element and interaction. Physical analysis can therefore be understood as the analysis of the implementation of products, processes and organisation constituents of the whole system. Physical analysis can be done either during the synthesis process of products, processes and organisations, in support of it, or after it, in order to model the resulting physical architecture. Product, processes and organisations are analysed in an interactive and iterative manner. For example, a decision on the shape of a part may affect the sequence by which that part must be machined. Also, the availability of a resource may determine the existence or not of a feature in a product and affects the schedule of the manufacturing process.

The physical analysis process uses the modelling approaches proposed in Section 7.4.

As mentioned above, the IEEE-1220-Std and [Martin, 1997] describe a product synthesis process. This thesis extracts from those references the activities covered by the modelling approaches proposed and adapts them to include not only product but also processes and organisations.

Physical analysis is supposed to be applicable for the choice of a solution or for the analysis of a previously defined solution. Figure 7-26 provides an overview of the physical analysis processes including the activities extracted and adapted from the references mentioned. These activities are described in the following paragraphs. Examples of activities which are essentially part of the synthesis process as described by the IEEE-1220-Std and were not considered for the physical analysis process: assess standardisation opportunities, assess off-the-shelf availability, assess make or buy alternative, develop models & fabricate prototypes, finalise design.

![Figure 7-26: Overview of the physical analysis process](image_url)
Identify system physical elements

System physical elements are:
- the product, its subsystems or components depending on the layer of the system breakdown structure;
- actual life cycle processes, activities and tasks corresponding to each product element;
- organisation, department, section, team or resources corresponding to each life cycle process element.

Allocate functions to system elements

Identify which functions, in the resulting functional architecture of products, processes and organisations, are performed by which system elements. For example, for the product physical modelling, an architecture flow diagram shows how the functional flows are allocated to each physical element.

Figure 7-27 shows a DFD representing a car and its functional interactions with the environment. Figure 7-27 is identical to the one in Figure 7-27 apart from the fact that in the former the physical end-product, car system, is a replacement for the mission/objective statement in the latter, 'transport private riders on a road'.

Figure 7-27: Product - DFD representing a car and its functional relationships with the environment

Figure 7-28 shows the identified subsystems in the layer below and their functional interactions.
Figure 7.28: Product - an example of an architecture flow diagram of a car
By comparing the inputs and outputs of the product elements in Figure 7-28 with those of the functions in Figure 7-24, the functional decomposition of a car, the relationships between functions and physical elements can be identified (see Figure 7-29).

**Identify physical interfaces**

Physical interfaces are the physical links connecting the system with its environment and the system elements among themselves. Product physical interfaces are the physical connections through which energy, material and information flows, identified in the functional analysis, are exchanged between two physical elements. Figure 7-30 shows an FBD representing the physical interfaces of car with its environment.

Process physical interfaces are:

- the time relationships between process physical elements;
- actual data, material or energy exchanged between process physical elements.

Organisation physical interfaces are:

- the organisation links between organisation physical elements;
- the relationships and physical connections among resources.

**Establish a physical architecture**

The physical architecture depicts the structure of the system elements in terms of its constituent elements and the physical interfaces among them. The system physical architecture may include: CAD drawings, schematics, electric/electronic circuit charts, process flow charts, organisation charts and the diagrams presented in the physical modelling approaches in Section 7.4.

For product, Figure 7-31 shows an FBD representing the architecture of a car, i.e. its subsystems and their physical connections. The diagram in Figure 7-31 corresponds to an expansion of the central bubble in Figure 7-30.

For process, Figure 7-32 represents an example of a process physical model. It is a behaviour diagram describing the development process adopted by Ford, the FPDS (Ford Product Development System). FPDS' sub-processes are depicted on the diagonal of Figure 7-32. They are: 'identify pre-strategic intent', 'manage vehicle program', 'design appearance', 'engineer vehicle', 'provide manufacturing/tooling facilities', 'design powertrain', 'sourcing'. The diagram shows that decisions made at the 'identify pre-strategic intent' phase highly affect the subsequent sub-processes. For clarity...
reasons, not all inputs and outputs exchanges were shown in the diagram. An example of input to the whole FPDS process are 'inputs/cycle plan assumption'. An example of output is 'powertrain'.

![Diagram of a car and its physical interfaces with its environment.]

**Figure 7-30: Product - FBD representing a car and its physical interfaces with its environment**

**Verify physical architecture**

Physical verification is performed for the purpose of:

- assure that the elements of the lowest level of the physical architecture are traceable to the verified functional architecture (see Design Loop in Figure 7-9);

- assure that the physical architecture elements satisfy the validated technical requirements set (see Verification Loop in Figure 7-9).

**Derive physical attributes**

Each element of the product, process and organisation physical models can be further described through their association with physical attributes or with other sources of physical attributes such as: CAD drawings, circuit charts, material information, prototype information, product breakdown structure, process flow charts, organisation charts, design documentation in general. Figure 7-33 exemplifies the direct links between elements of the physical models in Figures 7-31 and 7-32 and physical attributes.
Figure 7-31: Product - FBD representing an architecture interconnect diagram of a car
Having justified, proposed and described the 'concurrent structured analysis method' that supports the 'total view framework' and its fundamental concepts, the following chapters demonstrate how the method can be implemented using commercially available computer tools through examples of applications from the automotive industry. Chapter 8 describes the use of computer tools for implementing the method and consequently demonstrating the feasibility of the framework. Chapters 9 and 10 provide the examples of application.
Part IV
Implementation
Chapter 8
Overview of examples and use of tools

This chapter aims:

- to provide an overview of the examples presented in the following chapters 9 and 10. The examples refer to the application of the ‘concurrent structured analysis method’ to automotive subsystems development.
- to describe the criteria for selection and the use of commercially available tools for implementing the ‘concurrent structured analysis method’ and, consequently, demonstrating the ‘total view framework’.

8.1 Examples overview

Three examples of application have been developed to demonstrate different aspects of the framework and method. They are: the diesel PCS, the gasoline PCS and a seat height adjust mechanism, hereafter referred as SHM. All of them are considered within the context of the integrated development of the whole product – the car. As such, the links between the car model elements and the model elements developed in these examples are preserved and pursued. Also, although these examples are named by the subsystem products, they include not only product models but also process and organisation models through the requirement, functional and physical analysis processes. Figures 8-1, 8-2 and 8-3 present the scope of the diesel PCS, gasoline PCS and SHM examples, respectively.

Figure 8-1 shows the decomposition of a car, its life cycle processes and their associated organisations. The decomposition of the car goes through powertrain, PCS, module hardware, microcontroller, PCU, EGR control until the EGR control software modules. The life cycle processes decomposition focuses on the development process starting from the FPDS (Ford Product Development System) until the software changing and configuration management process. Although Figure 8-1 lists other life cycle processes at all product decomposition levels, the focus on the development process modelling was considered enough for demonstration purposes. The same applies for the organisation decomposition which focused on development organisation decomposition. The organisation decomposition started with the whole Ford organisation, responsible for meeting car stakeholder requirements, and ended with the Diesel Sub-System Projects group, responsible for delivering software modules. The objectives of the diesel PCS example are:

- to demonstrate the proposed modelling approach for the requirements, functional and physical analysis of product, processes and organisation;
- to identify functional and physical attributes from functional and physical models;
- to identify the relationships among requirements and attributes and their propagation.

Figure 8-2 differs from Figure 8-1 in the decomposition from the PCS level and in the development organisation. The shown gasoline PCS development organisation is the Ford/AVT/CAPE/PCSE in 1995 whereas the diesel PCS development organisation in the Visteon organisation, which is a tier 1 supplier. Visteon is a Ford owned company for subsystems development, created in 1997. This example, the
gasoline PCS, focuses on the Inlet Air Control feature software modules, the customer calibration process and the gasoline features and strategy groups. It intends to demonstrate the clustering approach for complexity management described in Chapter 6, Section 6.5.6.

The analysis processes in the PCS examples concentrate essentially on software subsystems and development processes and organisation. In order to demonstrate the application of the framework for the development of hardware subsystems, manufacturing and assembly processes and, organisation, the SHM example has been developed. In Figure 8-3 the decomposition of a car goes through 'interior/climate control', 'seating', 'front seat springs, frame, tracks and mechanisms' up to the 'front seat height adjust mechanism'. At each of these subsystem levels, the corresponding life cycle processes and their associated organisations have been listed. This example demonstrates the 'total view framework' on the SHM product, its assembly process and the H. R. Adcock Ltd. organisation, a tier 2 supplier. This example also demonstrates the applicability of the framework when product subsystems are developed by different companies, which is often the case for complex products development.

8.2 Tools

The demonstration of the 'total view framework' for the integrated development of a passenger car and its subsystems made use of the off-the-shelf software tools illustrated in Figure 8-4. Those tools are:

- Cradle™ 3.2: is a systems and software engineering environment that provides through life support from requirements capture to system implementation with supporting configuration management, project control, and document generation capabilities [3SL, 1998]. In this work, Cradle™ is used for requirements, functional and physical analysis; product, process and organisation modelling; attributes management; hierarchy, traceability and impact relationships identification.

- Chaco-2.0: is a software package designed to partition graphs [Hendrickson & Leland, 1995]. Chaco was conceived to develop parallel computing architecture. This thesis adapts Chaco to integrated product development. In this thesis, Chaco is used to demonstrate how closely together system elements should be considered in order to maximise cohesion and minimise coupling, which are the criteria used in this work to manage complexity.

- Excel™ 97: is the Microsoft™ spreadsheet tool [Microsoft Corporation, 1998]. It is used in this work for the presentation in a matrix format of the impact relationships between requirements, functional and physical attributes. Excel™ is also used to calculate a clustering quality index and to compare the results of the partitioning before and after using Chaco.

The following sections detail the use of these tools in the context of this thesis.

8.2.1 Cradle

An overview of Cradle with a brief description of all its modules is provided in Appendix B. The appendix focuses on those aspects that are used in this thesis. Further details can be found in Cradle manuals [3SL, 1998]. The following sub-sections concentrate on the justification of the choice of Cradle to carry out this implementation and on how Cradle is actually used for that purpose, showing the adaptations made.
### Product

- **Car**
  - Powertrain
  - Accessories
  - Body/exterior
  - Chassis
  - Electrical/electronics
  - Interior/climate control

- **Powertrain control system**
  - Engine
  - Transmission
  - Driveline axle

- **Module Hardware**
  - Sensors
  - Actuators

- **CPU**
  - A/D Converter
  - Low Speed I/O
  - Event Processor Array
  - CAN Controller
  - SPI Handler

- **EGR control**
  - Fan Control
  - Air Conditioning Control
  - Control Lift Pump
  - Cool Plug Control
  - Diagnostic Bulk Light Handling
  - Diesel Fuel Flow
  - SCP Instrument Cluster Handling
  - Port Deactivation Control
  - Fueling
  - Modal Control
  - FEVONA
  - High Speed Digital Output Handler
  - Low Speed Digital Output Drivers
  - Analogue Input Handling
  - Serial Peripheral Interface Handler
  - SCP Communications
  - EEPROM Handler
  - Low Speed Digital Input Handler
  - High Speed Digital Input Handler

- **EGR control valve position**
- **EGR control throttle position**
- **EGR calculation of valve and throttle operation**

### Processes

- **Ford Product Development System**
  - Ford Production System
  - Order to delivery
  - After sales service
  - Training
  - Operation
    - Start
    - Idle
    - Low speed
    - High Speed Extreme
    - Collision
  - Service
    - Refueling
    - Check up
    - Maintenance
  - Repair
    - Towing
    - Fix
  - Recycling
  - Disposal

- **Powertrain PCS**
  - Assembly
  - Testing
  - Integration
  - Operation
  - Service
  - Repair
  - Recycling
  - Disposal

- **PCS development**
  - PCS assembly
  - PCS testing
  - PCS integration
  - PCS service
  - PCS repair
  - PCS recycling
  - PCS disposal

- **Customer requirements phase**
- **System requirements phase**
- **Architectural design phase**
- **Detailed design phase**
- **Construction and build**
- **Workstation based testing**
- **Testbench based testing**
- **Dyno/vehicle based testing**
- **Transfer**

- **Changing and configuration management**

### Organisation

- **Product development**
  - Automotive strategy office
  - Public affairs
  - Marketing and sales
  - Purchasing
  - Design
  - Quality
  - Process leadership
  - Finance
  - Technical affairs
  - Manufacturing

- **Visteon (Tier 1)**
  - CEO
  - Visteon operations
  - Business strategy
  - Global marketing
  - Sales and service
  - Finance
  - Supply and logistics
  - Human resources
  - Public affairs

- **PCS division**
  - Exterior systems division
  - Electronic systems division
  - Glass systems division
  - Interior systems division
  - Climate control systems division
  - Chassis systems division
  - Chief engineer - small and medium
  - Quality
  - Chief engineer - truck
  - Chief engineer - large and luxury
  - Controller
  - Advanced technology

- **Components and applications**
  - Systems and technologies
  - - Feature development A
  - - Feature development B
  - - Gasoline sub-system projects
  - - Gasoline sub-systems technologies
  - - Gasoline features
  - software and strategy
  - - Software integration
  - and tools support
  - - Diesel sub-system projects
  - - Diesel sub-system technologies

---

*Figure 8-1 – The diesel PCS example*
Figure 8-3: The SHM example
8.2.1.1 Why Cradle?

Multi-notational modelling capability

The 'total view framework' requires the modelling of product, process and organisation throughout the requirements, functional and physical analysis processes. Therefore, a tool that supports the total view approach must provide different modelling notations in order to cope with the necessarily broad modelling scope.

In Chapter 7, it has been proposed and justified:

- the Yourdon notation and Function Block Diagrams for product modelling;
- IDEF0 and behaviour diagrams for process modelling;
- Use Case and UML notation for organisation modelling.

There are tools in the market that support each of the above mentioned modelling notations isolatedly. For example, Cadre's Teamwork supports the Hatley-Pirbhai notation (an enhancement of the Yourdon notation) for requirements analysis and software architecture modelling. Vitech's CORE and RDD 100 support Function Block Diagrams, Behaviour Diagrams, N2 diagrams and translations from and to IDEF models. Rational Rose supports Use Case Models and OMT, Booch and UML object modelling notations that can be used for implementing the Jacobson's approach to enterprise modelling.

Cradle provides all but the IDEF0 notations, proposed in Chapter 7, in the same software environment. As a consequence of this, any element in any model can be treated as an independent entity that can be
linked to any other element in any other model. This capability avoids potential translation and compatibility problems. In this thesis, Cradle’s behaviour diagrams are adapted to include the IDEF0 notation.

Methodology free
As a consequence of Cradle’s multi-notational capability, its use does not impose the adoption of a specific development methodology. For example, Cadre’s Teamwork adopts the Hatley Pirbhai methodology for structured analysis and design. CORE and RDD 100 use the SREM/DCDS approach. Rational Rose uses OMT, Booch or UML methodology based on the object orientation paradigm.

Being methodology-free makes Cradle adequate for the demonstration of a novel development framework such as the 'total view framework'.

Scope of Cradle within systems engineering process
Cradle’s capabilities cover the whole systems engineering process from requirements capture to physical architecture definition. This makes easier the task of linking requirements to functional and physical models and their corresponding attributes.

Integration of stakeholders and sub-contractors into the systems engineering process
A fundamental aspect of integrated development (IPD, IPPD, IPPED) is the team approach for development organisation. This team approach does not have to be limited to one site or one company.

On the contrary, for complex product development, the notion of integrated team must go beyond company’s boundaries. In the current industry environment, it is becoming less likely to any one company to own the development of the entire complex product and its subsystems. Being so, the integration of some stakeholders, e.g. customers, and suppliers into the development team is not only desirable but also needed.

Cradle can integrate the stakeholders into the development team by [3SL, 1998]:

• providing the stakeholder with on line access into the project’s central database;
• providing the stakeholder with a copy of the database and a read-only copy of the SEE used to create it.

Central to the advantageous integration of the stakeholder into the SEP is the notion that the stakeholder is there as a read-only user, who inspects, reviews, advises, decides between alternatives and authorises deliverables without the need of formal meetings. The only modifications made to the project database by a stakeholder are the addition of annotations.

Cradle can integrate the suppliers into the development team by [3SL, 1998]:

• allowing them to add lower levels of decomposition to requirements hierarchies;
• allowing them to add lower levels of detail to specific areas of analysis, architecture or design models;
• allowing them to add cross references between existing items and those that it creates, such cross references optionally being typed to identify them;
- allowing them to form part of the review process for (some of) the items produced by the prime-contractor;
- allowing the prime-contractor to form part of the review process for all items produced by the supplier.

8.2.1.2 Cradle use

This section highlights how Cradle is used in the context of integrated product development as proposed in this thesis. This section concentrates on the implementation solutions that are peculiar to this thesis.

Project setup

Figure 8-5 illustrates Cradle's project setup to implement key elements of the 'total view framework': requirements, attributes, relationships, stakeholders and development agencies. Requirements and relationships are pre-defined elements of any Cradle project setup. Attributes, stakeholders and development agencies are created through the use of Systems Notes in Cradle. The project setup for pre-defined items and for System Notes are described below, following the numbering in Figure 8-5:

1) Project setup menu: is used to define category codes for the categorisation of pre-defined information items and user-defined System Notes. Also, frame types are setup to establish the format of frames.
Frames can then be created for each information item and can adopt any one frame type previously defined. The creation of user-defined System Notes is also made during project setup. Relationship characteristics, types, reasons and rationale can be created through the definition of cross-reference (X-Ref) parameters.

2) Category code definition: presents the list of category codes defined. The category codes are defined at the beginning of the project setup and they are not information item specific. Cradle's pre-defined category codes are category 1, category 2, category 3 and category 4. Categories defined in order to demonstrate the implementation of the 'total view framework' are, for example: concerns, func/phys (provide indication if the item of information belongs to functional or physical analysis), PPO (product, process, organisation or any combination of those), layer (position in the system breakdown structure).

3) Category code setup: exemplifies the list of values created for the category code 'concerns'. Each category code has its own particular list of possible values it may assume. Appendix D presents the category code setup for the other category code defined.

4) Frame type setup: presents the definition of frame types. Examples of frame types are: analysis reports, QFD entries, requirements elements, attribute elements. Frame types must be setup before the frame setup for any information item.

5) Pre-defined items type setup: presents the list of Cradle's pre-defined information items. The definition of frames and categories for any of the pre-defined information items is started up from this window.

6) Assign categories (to pre-defined items): shows the categories assigned to requirements, for example: concerns, PPO, tradeoff_ability. Each item of information can have categories assigned to it.

7) Frame setup (for pre-defined items): shows the frames created to provide additional information to requirements, for example: verifiability, ownership, FMECA.

8) System Note type setup: shows the additional information items created for the implementation of the 'total view framework': stakeholders, development agencies and attributes.

9) Assign categories (to System Notes): shows the categories assigned to attributes, for example: func/phys, PPO, layer. Any System Note type can have categories assigned to it.

10) Frame setup (for System Notes): shows the frames created to provide additional information to attributes, for example: factor, value, tolerance and unit.

11) Cross reference parameters: allows the definition of link types and groups, the way to perform re-entrancy and recursion and relationship elements such as reason and rationale. The principles of recursion and rationale are illustrated in the Appendix B. This window 11 shows that the re-entrancy as source and destination must be set to 'No' and 'Yes', respectively. Recursion must be set as VIA TYPE 'Same & Different'. Explanation can be found below.

12) Link type setup: creates the hierarchy, impact and traceability link types.


Appendix D contains a more complete list of project settings.
Stakeholders and development agencies as System Notes

Cradle provides a pre-defined set of items that may be cross-referenced. These items are: source statements within source documents, requirements, events, process specifications in the essential model, process specifications in the implementation model, module specifications in the implementation model, instances of user-defined types of system note. System notes are user-defined items of information.

Cradle provides originator and source-org. as default frames of requirements and process specifications, respectively. Stakeholders and development agencies would normally be in these default frames. However, as frames, they cannot be cross-referenced to other items or have other frames attached to them.

Since the 'total view framework' proposes a shift from a customer-driven development to a stakeholder-driven development, it is important to identify which stakeholder may be affected by a change in an attribute, for example. This kind of link can only be directly identified if stakeholders can be cross-referenced.

Also, one key aspect of integrated development is team formation. This thesis proposes impact relationships among attributes as criteria for team formation. As the impact among attributes is identified and the links between attributes and development agencies are identified, the links among development agencies can be inferred by transitivity. This is only possible if development agencies can be cross-referenced.

In order for stakeholders and development agencies to be cross-referenced, they need to be implemented in Cradle as System Notes.

Functional and physical attributes as System Notes

Functional and physical attributes are not part of Cradle’s pre-defined information items list. Using only Cradle’s pre-defined items, functional and physical attributes could be created as frames of the pre-defined information item ‘specification’. However, as frames, attributes cannot be linked to other information items or even to other attributes. In order to be able to be cross-referenced, it is necessary that functional and physical attributes be set-up as System Notes.

Besides the concept of ‘frames’, the closest Cradle gets to ‘attributes’ is through the concept of ‘performance parameters’. Performance parameters are in fact a default ‘frame’ of the ‘specification’ of the Implementation Model DFD and FBD elements. Performance parameters store the performance characteristics for a diagram symbol within that symbol’s textual definition. In the ‘total view framework’, attributes are not only implementation driven or physical. They can also be functional. Also, DFDs and FBDs do not contain all notations necessary to model product, process and organisations. Therefore attributes must be able to be linked to ‘specifications’ of elements in other diagrams (not only DFDs and FBDs). Being created as System Note types, attributes can be linked to specification in the functional (essential) or physical (implementation) models of product, process and organisation.
Functional and physical attributes as different System Note types

One may argue that 'attributes' could be created as one System Note type and 'functional' and 'physical' would be set as category values. However in doing so, the way Cradle implements cross-reference transitivity would generate undesirable cross-reference re-entrancy. For example: Suppose that functional attribute A is affected by physical attributes B and physical attribute C and that B and C do not impact each other. If physical and functional attributes are only different category values of the same System Note type, by cross-reference transitivity Cradle would inform that they affect each other regardless of the re-entrancy parameter definition (see Window 11 in Figure 8-5). If, on the other hand, they are different System Note types and the re-entrancy parameters are set to 'No' and 'Yes' (as indicated in Figure 8-5, window 11), Cradle would, correctly, not 'see' a transitive cross-reference between B and C.

Rationale for the definition of re-entrancy and recursion cross-reference parameters

As shown in Figure 8-5, window 11, re-entrancy parameters are both set as 'No' and recursion parameters are set as 'Same & Different'.

Setting recursion as 'Same & Different' means that transitive relationships between requirements and physical attributes can be automatically identified if it is known the relationships between requirements and functional attributes and between functional and physical attributes. If recursion was set to 'Same only', for investigating transitivity Cradle would consider only relationships among functional attributes or among physical attributes, not between those types.

Setting re-entrancy parameters as indicated in Figure 8-5, window 11 means that Cradle can be used to indicate the coupling between, for example, functional attributes given that they are affected by the same physical attribute or the coupling between requirements affected by the same functional attribute. This is particularly useful for the investigation of compliance to Suh's first axiom [Suh, 1990].

Requirements capture

Cradle provides means to capture requirements from source documents or to follow the systematic process mentioned in Section 7.5.1 (item: 'Capturing stakeholder requirements') using the concept of 'events'. 'Events', like 'requirements', are pre-defined information items in any Cradle Project Database. Figure 8-6 illustrates how Cradle can be used for editing 'events'. 'Events' can describe how stakeholders interact with the product, process and organisation elements of the system. 'Events' in Cradle have two default frames: 'stimulus' and 'response'. 'Stimulus' will derive the 'condition' information related to a requirement. 'Response' will derive the 'function' information of the requirement. 'Performance' information may be included in the 'response' frame.
Requirements structuring

Figure 8-7 illustrates the fact that Cradle allows stakeholder requirements to be organised in a hierarchy that reflects increasing levels of understanding and detail about the stakeholder requirement. Notice in the bottom right window in Figure 8-7 the category values assigned to that particular requirement: driveability, mandatory, condition.
Requirements analysis

Figure 8-8 illustrates the use of Cradle to extract condition, function and performance information from stakeholder requirements.

In Figure 8-8:
- Window 1: presents Cradle toolset menu showing the 'requirements' option;
- Window 2: presents the requirements list;
- Window 3: presents condition C000 'When starting the vehicle';
- Window 4: presents condition C001 'Engine temperature';
- Window 5: presents condition C001.001 'Engine off for two hours' which actually refers to engine temperature when engine is off for two hours. This condition is a child of condition C001;
- Window 6: presents function F001 'Engine comes up to idle speed', which is a functional requirements saying that the engine must return or go to idle speed;
- Window 7: presents performance P002 'Time to idle speed is 10 seconds', which specifies the actual performance requirements for time to idle speed;
- Window 8: is the condition C001.002 'Engine just turned off' which actually refers to the engine temperature when the engine was on and just turned off.

Figure 8-8: Analysis of stakeholder requirements using Cradle
Compilation of technical requirements

Technical requirements can be compiled using and complementing the information resulting from the requirements analysis - conditions, functions and performance. Figure 8-9 illustrates the use of Cradle for the compilation of a technical requirement from the conditions, functions and performance identified above. The requirement is a child of requirement T003.001: Cranking time, which is a child of requirement T003: Starting.

Figure 8-9: An example of technical requirement in Cradle

Appendix D provides a list of some technical requirements. These technical requirements concentrate on driveability, fuel economy and emissions requirements that drive the diesel PCS development (Chapter 9, Section 9.3).

Links from stakeholders to technical requirements or vice-versa

Cradle can provide traceability from stakeholders, through possibly events, to stakeholder requirements and from these to technical requirements, through conditions, functions and performance information. This can be done through the concept of cross-references. Consider the example in Figure 8-10 where a technical requirement (AT) ‘is traceable to’ a condition (AC), a functional requirement (AF) and a performance requirement (AP), those ‘are traceable to’ a ‘stakeholder requirement’ (AS) and that ‘is traceable to’ a ‘system note type stakeholder’ (Customer). Cradle stores ‘direct’ cross-reference information and can, by using that information, identify transitive links between a technical requirement and a stakeholder. Figure 8-11 shows the ‘windows’ generated by Cradle showing the direct cross-references created and an example of a transitive cross-reference automatically inferred by Cradle.

Figure 8-10: Example of traceability from technical requirement to stakeholders
In Figure 8-11:

- Window 1: is the main Cradle menu;
- Window 2: is Cradle toolset;
- Window 3: is the cross-reference viewer showing the direct cross-references from requirement AT to requirements AC, AF and AP. The cross-reference is a link of type ‘TRACEABILITY’.
- Window 4: is the cross-reference viewer showing the direct cross-reference from requirement AF to requirement AS. The cross-reference is a link of type ‘TRACEABILITY’.
- Window 5: is the cross-reference viewer showing the direct cross-reference from requirement AS to stakeholder ‘Customer’. The cross-reference is a link of type ‘TRACEABILITY’.
- Window 6: is the cross-referenced items viewer showing the transitive cross-reference from requirement AT to stakeholder ‘Customer’. The cross-reference is a link of type ‘TRACEABILITY’.

**Separation between hierarchy links and traceability links**

The fact that link types can be attributed to cross-references isolates, for example, the hierarchical links within a requirements structure from traceability links as shown above. On the other hand, Cradle also allows link types to be grouped in link groups so that cross-references of different types can be used for transitivity inference. Figure 8-12 illustrates the differences between link types and link groups.
'HIERARCHY' and 'TRACEABILITY' are link types. When those two link types are grouped in the 'STRUCTURE' link group, the resulting set of cross-references is the union of the sets of cross-references of each individual link type in the group.

Figure 8-13 illustrates the effect of link groups on cross-reference transitivity. Consider, for example, that the functional requirement AF in Figure 8-10 has a 'HIERARCHY' link with a requirement AF.1. Figure 8-13 shows that when the cross-references considered are in the cross-reference group 'STRUCTURE' composed of 'HIERARCHY' and 'TRACEABILITY' links, the requirement AF.1 is transitively 'traceable to' the stakeholder 'Customer'. The traceability links were exemplified in Figure 8-11.

Adapting Cradle’s modelling capability

Figure 8-14 shows which Cradle’s diagrams were used for the demonstration of the ‘total view framework’. The use of the diagrams depends on the type of analysis and on the integration element to be modelled. The diagrams shown in the list in Figure 8-14 are the main diagrams to be used. Other diagrams may also be used. For example, Data Flow Diagram can also be used for process modelling, although it has a more limited capability than the Behaviour Diagram. The Behaviour Diagram models time line and Data Flow Diagram does not. Care must be taken when using the same type of diagram to model product, process and organisation elements in the same analysis type at the same level of the system breakdown structure. In this situation, the numbering of different diagrams may coincide.

<table>
<thead>
<tr>
<th>Analysis Type</th>
<th>Integration Element</th>
<th>Requirements</th>
<th>Functional</th>
<th>Physical</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Product</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hardware</td>
<td>Data Flow Diagram</td>
<td>Data Flow Diagram</td>
<td>Function Block Diagram</td>
</tr>
<tr>
<td></td>
<td>Software</td>
<td></td>
<td>State Transition Diagram</td>
<td>Data Flow Diagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Entity Relationship Diagram</td>
<td>State Transition Diagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Structure Chart</td>
</tr>
<tr>
<td></td>
<td>Process</td>
<td>Behaviour Diagram</td>
<td>Behaviour Diagram</td>
<td>Behaviour Diagram</td>
</tr>
<tr>
<td></td>
<td>Organisation</td>
<td>Use Case Diagram</td>
<td>Use Case Diagram</td>
<td>Use Case Diagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sequence Diagram</td>
<td>Package Diagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Collaboration Diagram</td>
<td>Class Diagram</td>
</tr>
</tbody>
</table>

Figure 8-14: List of Cradle’s modelling notations used
A description of those notations can be found in Appendix C.

Figure 8-15 lists the necessary adaptations to some of Cradle's diagrams in order to implement the 'total view framework'. State Transition Diagrams, Structure Charts, Subsystem Diagrams and Object Class Diagrams are used as originally conceived.

<table>
<thead>
<tr>
<th>Diagram</th>
<th>Adaptation required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Flow Diagram</td>
<td>For requirements analysis, a central 'data process' represents the product system element, 'discrete data flows' represent concerns and their orientation are not important and, 'terminators' represent the stakeholders that interact with the product. For physical analysis of hardware, 'data process' represent physical parts and 'data flows' represent functional flows (data, energy or material). For functional analysis and physical analysis of software it is used as originally conceived.</td>
</tr>
<tr>
<td>State Transition Diagram</td>
<td>No adaptation required.</td>
</tr>
<tr>
<td>Entity Relationship Diagram</td>
<td>No adaptation required.</td>
</tr>
<tr>
<td>Function Block Diagram</td>
<td>'Equipment' represents physical parts of a product and 'links' represent the physical connectors (e.g. wires, screws). If attributes are to be assigned to physical connectors, these have to be represented as 'buses' which can be cross-referenced in Cradle. 'Links' are defined in Cradle using Data Dictionary entries which cannot be cross-referenced.</td>
</tr>
<tr>
<td>Structure Chart</td>
<td>No adaptation required.</td>
</tr>
<tr>
<td>Behaviour Diagram</td>
<td>For requirements analysis, 'time functions' represent life cycle processes, 'data links' represent concerns, and 'time or discrete items' represent stakeholders that interact with the life cycle processes. For functional and physical analysis of process elements, the BD notation is adapted to represent the IDEF0 notation. 'Time functions' in BD represent 'activities' in the IDEF0 notation. 'Discrete item', 'time item' and 'discrete functions' in BD can represent 'inputs', 'outputs', 'mechanism' and 'control' in the IDEF0 notation. If any 'input', 'output', 'mechanism' and 'control' are to have attributes assigned to them, they must be represented as 'discrete functions' which can be cross-referenced in Cradle. 'Time and discrete items' in Cradle are defined as data dictionary entries which cannot be cross-referenced.</td>
</tr>
<tr>
<td>Use Case Diagram</td>
<td>For requirements analysis, 'system' represents the organisation to be analysed, 'actors' represent the stakeholders and 'links' represent their concerns. For functional analysis, it is used as originally conceived.</td>
</tr>
<tr>
<td>Sequence Diagram</td>
<td>Objects may represent the conceptual resources necessary to perform a given use case. Those may include human resources such as manager, accountant, operator or machines such as driller, lathe, etc. Messages in the original application can be used to represent material or energy status as well as information.</td>
</tr>
<tr>
<td>Collaboration Diagram</td>
<td>Like in the sequence diagram, objects in this application may represent human or capital resources. Messages may represent material, energy as well as information.</td>
</tr>
<tr>
<td>Package Diagram</td>
<td>Used for representing organisation structure. For example, a given organisation structure can be set initially as two packages, such as: functional organisation and program organisation. Each package can be further deployed into the organisation units. Different aspects of the organisation unit can be treated separately by the use of different packages such as processes and resources. Processes can be further deployed by the use of the component diagram and resources can be further deployed by the use of the class diagram.</td>
</tr>
<tr>
<td>Component Diagram</td>
<td>In this application, this diagram is used to depict the structure of processes in the organisation unit. Components may represent processes, activities or tasks. Component dependency may represent the composition relationship between a process and its component activity or between an activity and its component tasks.</td>
</tr>
<tr>
<td>Class Diagram</td>
<td>Used to represent classes of resources and the relationships among them.</td>
</tr>
<tr>
<td>Deployment Diagram</td>
<td>Can be used to represent a plant layout or the physical position of resources in relation to each other and how they are linked together.</td>
</tr>
<tr>
<td>Statechart Diagram</td>
<td>No adaptation required.</td>
</tr>
<tr>
<td>Activity Diagram</td>
<td>No adaptation required.</td>
</tr>
</tbody>
</table>

*Figure 8-15: Adaptations required on Cradle notations*
Process specification numbering

Process specifications in Cradle specify the functions or the physical parts to which the attributes belong. Therefore process specifications have to be uniquely identified because every attribute must be created and cross-referenced to one and only one process specification. When creating symbols that will be specified by process specifications, care must be taken in order to avoid coincidence with the number of any other symbol already created or likely to be created. An idea is to assign a '1' to the first digit of a product model symbol number, e.g., 10, 11, 12, 101; a '2' to the first digit of a process model symbol number, e.g., 20, 21, 22, 210; a '3' to the first digit of an organisation model symbol, e.g., 30, 31, 32, 302, 320.

Attributes management

Functional and physical attributes are not included in Cradle's pre-defined set of items of information. In the Yourdon approach to systems analysis and design, for example, these attributes are supposed to be embedded in the specifications. Although it might be enough for impact analysis in software systems development, it is certainly not the case in general systems. In those, attributes perform a key role in identifying change impacts. A change in one attribute in a specification changes the specification but may not affect other specifications related to it. That is the reason to capture the relationships between attributes and not only between specifications.

In order to have stand-alone attributes, they are managed in Cradle as System Notes. Being included as types of System Notes, attributes can be cross-referenced to themselves and to other items. As System Notes, attributes will be treated by Cradle as any other type of items of information, such as diagrams, events, requirements or specifications.

Figure 8-16 illustrates how attributes are managed by Cradle. Window 1 is a DFD diagram showing a functional model of a car. Window 2 is the process specification (PSPEC) corresponding to the function 'Provide_motion_alerting_feedback'. The definition of the function on Window 1 is the PSPEC in Window 2. Clicking twice on the 'bubble' on Window 1 brings the PSPEC (on Window 2) to the screen. As the DFD diagram is part of the functional model, a functional attribute must be associated with that function in the model and with the corresponding PSPEC. Window 3 illustrates the functional attribute Number 7, 'Performance/ fuel economy', an attribute of the function 'Provide_movement_alerting_feedback'. The functional attribute is associated with the PSPEC through a cross-reference of the type 'TRACEABILITY'. The association of the PSPEC with the function it refers to in the diagram is automatic. Cradle provides automatic navigation from PSPEC to diagram symbol and vice-versa. Window 4 shows that attributes can also be structured in a hierarchy with lower level attributes providing a more detailed definition of the upper level attribute. The hierarchical links among attributes in Cradle are represented by a cross-reference of link type 'HIERARCHY'.
Cross-references implementing traceability, hierarchy and impact links using Cradle

The above paragraphs provided examples of two types of links implemented in Cradle: TRACEABILITY and HIERARCHY links. Those are the 'default' applications of Cradle's cross-referencing functionality, i.e., the applications for which cross-references were conceived in Cradle in the first place.

In order to use Cradle for impact analysis, cross-references of a link type IMPACT are created. IMPACT links are normally applicable to the lower level attributes and/or requirements in each attribute/requirement hierarchy.

Figure 8-17 provides the structure of links that can be implemented by Cradle in order to carry out the 'total view framework'. Examples of traceability and hierarchy links were provided in Figures 8-11, 8-12 and 8-13. Impact relationships are captured in Cradle but are represented using MS-Excel. Examples of impact relationships in a matrix format can be found in Chapter 9, Sections 9.2 and 9.3 and in Chapter 10.
Cradle's cross-references are, by definition, bi-directional. Being so, the direction of impact cross-references has to be inferred from the items being linked. Figure 8-17 shows how the direction of an impact cross-reference shall be interpreted: ‘physical attributes’ impacts ‘functional attributes’, ‘functional attributes’ impacts ‘technical requirements’ and so on. Cradle users input the impact cross-references shown in Figure 8-17. Cross-references among items of the same type (other than physical attributes) can be inferred by transitivity.

### 8.2.2 Chaco-2.0

[Hendrickson & Leland, 1995] describes the capabilities and operation of Chaco 2.0, a software package designed to partition graphs. Chaco 2.0 allows for recursive application of several methods for finding small edge separators in weighted graphs. These methods include inertial, spectral, Kerninghan-Lin and multilevel methods in addition to several simpler strategies. Each of these approaches can be used to partition the graph into two, four or eight pieces at each level of recursion. In addition, the Kerninghan-Lin method can be used to improve partitions generated by any of the other algorithms. [Hendrickson & Leland, 1995] provides a brief description of these methods, along with references to relevant literature.

Chaco was developed in a parallel computing context but has been used in many different settings (e.g. database organisation, sparse matrix envelope reduction, matrix reordering, DNA sequencing, chromosomal mapping, large systems optimisation).
The most generic problem Chaco addresses can be expressed as following: 'given a graph G with 'n' weighted vertices and 'm' weighted edges, divide the vertices into 'p' sets in such a way that the sum of the vertex weights in each set is as close as possible, and the sum of the weights of edges crossing between sets is minimised'. Unfortunately, even in the case where p=2 and the edge and vertex weights are uniform, this graph partitioning problem is NP-complete [Garey et al., 1976]. Hence there is no known efficient algorithm to solve the problem generally, and it seems unlikely that such an algorithm exists. Therefore, it is necessary to resort to heuristic solutions in which balance may be partially compromised or (more typically) the minimisation is approximate.

Chaco's choice was based on:
- its flexibility
- adequacy to hardware available (IBM workstation running AIX 4.1 operating system)

Chaco's flexibility is determined by:
- the number of partitioning algorithms it uses;
- the possibility of entering graphs without vertices and/or edges weighting;
- the possibility of partitioning among unbalanced sets of vertices;
- the capability of reordering matrices (without partitioning).

The algorithms in Chaco are based on inertial, spectral, Kerningham-Lin (KL), and multilevel principles in addition to several simpler strategies. The simpler strategies provide an initial partitioning before the other more complex methods can be used. These simpler strategies were not used in this thesis because the analysis processes implemented in Cradle already provide an initial partitioning. The inertial method uses geometric information which do not easily map to Chaco's intended application in this thesis. The multilevel-KL is used for partitioning large graphs (~10000 vertices). The algorithms used in this thesis are the KL and the spectral. Chaco allows the KL to look for unbalanced partitions by setting the KL _IMBALANCE parameter to a larger value than its default of zero. The spectral graph algorithms can be used for vertices sequencing. If the SEQUENCE parameter is TRUE (or nonzero), a Fiedler vector will be sorted and written to the file whose name is specified by the parameter SEQ_FILENAME. The Fiedler vector contains an alternative numbering to vertices. Vertices connected by an edge will tend to be assigned numbers that are close to each other.

The uses of Chaco in the context of this thesis are:
- Functional and physical architecture. The most obvious way by which Chaco can be used in the context of systems engineering is for product, process and organisation architecture. Cradle's diagrams such as DFD, FBD, BD and SQD are graphs in their own right. Processes, equipment, time functions and objects can be considered vertices in these diagrams, respectively. Edges are the flows, links or connectors between these symbols. Criteria for edge weight can be derived in terms of the type of data, energy or material flowing between symbols. Vertex weight can represent the cost of an equipment or resource, the complexity of a process. Chapter 9, Section 9.4 illustrates this application of Chaco.
- Attributes and requirements clustering. Once physical attributes at any level of the system breakdown structure have been identified along with the relationships among them, Chaco can be used to cluster them. In graph notation, attributes are vertices and the relationships among them are the edges. Vertex
weights may represent, for example, the technical difficulty to achieve an attribute level. Edge weights may represent relationship strengths. As physical attributes are linked to functional attributes and these to requirements, by transitivity relationships among functional attributes and among requirements can be identified. Chaco can again be used to cluster functional attributes and requirements. Chapter 9, Section 9.4 proposes this application of Chaco.

Team formation. As proposed in Chapter 5, Figure 5-1, the result of integrated development is not only a product, but also, its life cycle processes and one or more of their performing organisations. Obviously, for complex products, the development activities have to be performed by teams of people. Considering that people or teams of people when working in the integrated development of a complex product exchange attributes, a graph can be constructed to represent this communication flow. Each vertex in the graph represents a person or a team of people. Each edge represents a set of attributes that link these groups. Vertex weight can represent the number of people in the team represented by the vertex. Edge weight can be calculated by how the attribute is going to be used, whether it will be part of a decision process or just information input for further processing. Chapter 9, Section 9.4.5 proposes this application of Chaco.

8.2.3 Excel

Excel is used in this thesis in three ways:
- as an alternative representation of the cross-references identified in Cradle (for examples, see Chapters 9 and 10);
- as a representation of partitioning before and after using Chaco (for examples, see Chapter 9, Section 9.4);
- to calculate a complexity index for the evaluation of functional and physical architecture, attributes and requirements clustering and team formation (for examples, see Chapter 9, Section 9.4).

This chapter provided an overview of the examples developed and to be presented in the following Chapters 9 and 10. Having presented the rationale behind the use of tools in this chapter, the following chapters present examples of the actual utilisation of those tools in order to demonstrate the 'concurrent structured analysis method' and the 'total view framework'. Chapter 9 presents the diesel and gasoline PCS examples and Chapter 10 presents the seat height adjust mechanism (SHM) example.
Chapter 9
The PCS examples

This chapter aims:

• to demonstrate the deployment of requirements and attributes within layers and between layers of the product breakdown structure of a car as proposed in Chapter 6, Section 6.5.5;

• to demonstrate the ‘total view framework’ and the ‘concurrent structured analysis method’ using the systems engineering environment tool (Cradle) introduced in Chapter 8, Section 8.2.1, applied to a diesel PCS subsystem;

• to demonstrate the complexity management approach proposed in Chapter 6, 6.5.6, using a graph partitioning algorithm (Chaco-2.0) introduced in Chapter 8, Section 8.2.2, applied to a gasoline PCS subsystem.

9.1 Introduction

In Chapter 6, it was proposed the integration among product, process and organisation elements through the concepts of requirements, attributes and relationships. Section 6.5.5 proposed a matrix representation of the relationships among requirements, attributes and their sources. The relationships propagate horizontally from stakeholders, through stakeholder requirements, technical requirements, functional attributes to physical attributes and development agencies. Also, requirements and attributes propagate vertically down the product breakdown structure through allocation, partitioning or decomposition. This chapter demonstrates those ideas by applying them to a car and to the next level down to a powertrain.

Some powertrain requirements are allocated down to the diesel PCS subsystem. From these requirements the ‘concurrent structured analysis method’ described in Chapter 7 was implemented. The focus is on to demonstrate the analysis processes through the modelling approaches proposed in Chapter 7. Also the adaptations to Cradle notational capabilities are demonstrated through the diesel PCS example. Those adaptations were described in Chapter 8, Section 8.2.1.2.

As the car and powertrain examples demonstrate how the ‘total view framework’ addresses the issue of integration and the diesel PCS example demonstrates how it addresses the issue of scope, the issue of complexity management is demonstrated via the gasoline PCS example. The approach to complexity management proposed in this thesis focuses on a complexity analysis procedure that uses Chaco-2.0 and Excel, introduced in Chapter 8, Sections 8.2.2 and 8.2.3. The complexity analysis procedure also uses Cradle to obtain a graph-like version of the system. Chaco-2.0 is used for partitioning the resulting graph and Excel is used to calculate a complexity index. The complexity index is based on the work of Hitchins (1992) presented in Chapters 2, Section 2.2.4 and on the ideas proposed in Chapter 6, Section 6.5.6.

This chapter is structured as following: Section 9.2 presents the car and powertrain examples, Section 9.3 presents the diesel PCS example and Section 9.4 presents the gasoline PCS example.

9.2 The car and powertrain examples

This section aims to illustrate the kind of matrices that can be obtained as a result of the application of the ‘concurrent structured analysis method’ within the ‘total view framework’. The items of information in the
matrices are a result of the analysis processes described in Chapter 7. Those items of information are retrieved in Cradle as described in Section 8.2.1.2. The matrices are a way of representing the 'impact' cross-references generated and retrieved in Cradle. The matrices are created in Excel. The rationale supporting the matrices presented in this section is explained in Chapter 6, Section 6.5.5.

Figures 9-1, 9-2 and 9-3 provide examples of the relationships among requirements and attributes of a car. Figures 9-5, 9-6 and 9-7 show the deployment of these requirements and attributes to the next level in the car system breakdown structure for the development of a powertrain. Figure 9-4 illustrates how a passenger car can be broken down in its component subsystems.

Figure 9-1 shows the relationships between some stakeholder requirements and their corresponding technical requirements for a car. Some stakeholder requirements are further explained through a second and third level requirements hierarchy. For example, the stakeholder requirement about 'value for the money' is further expanded into 'low operating cost' and 'good fuel economy'. 'Good fuel economy' is further expanded into 'competitive fuel economy', 'uses non-premium fuel', 'multi-fuel capability', 'emission system does not hurt gas mileage or performance', 'maintains good fuel economy'. The stakeholder requirements above the requirement 'vehicle must comply to emissions regulations' are from the Ford document 00.00QFD-D02-1, QFD primary/secondary/tertiary wants template European & U.S. cars and light trucks [Ford Motor Company, 1995c]. The others are regulations, life-cycle-related and organisation requirements. Some of the stakeholder requirements are translated into technical requirements. Only some technical requirements are identified in this example. They include:

- two regulatory requirements for emissions and fuel economy,
- one technical requirement for driveability specifying how the requirement shall be verified,
- one life cycle process goal saying that vehicle tuning time must not exceed 30 minutes and
- one assumption referring to where the product will be manufactured.

The stakeholder requirements versus technical requirements matrix is filled with 1's or empty cells. A '1' in a cell indicates that the stakeholder requirement in the cell row 'is translated by' the technical requirement in the cell column. The inexistence of such a relationship is indicated by an empty cell. For example, the technical requirement 'Driveability total weighted demerits (...) driveability test procedure.'

Figure 9-1 also shows the relationships between the sources of stakeholder requirements, the stakeholders, and the stakeholder requirements. The 'stakeholder requirements versus stakeholders' matrix cells are filled with '1' if the corresponding stakeholder in the column 'is the source of' the stakeholder requirement in the row. The cells are left empty otherwise.

Figure 9-1 also shows the relationships between the sources of technical requirements and the technical requirements. Sources of technical requirements are the people or organisations who are part of the development agencies set and translate stakeholder requirements into technical requirements. The 'development agencies versus technical requirements' matrix cells are filled with '1' if the corresponding development agency in the row 'is the source of' the technical requirement in the column. The cells are left empty otherwise.
Figure 9-1: Example of a partial matrix ‘stakeholder requirements versus technical requirements’ for a car
Although Figure 9-1 shows self-interactions matrices of technical requirements, these relationships can only be identified after the relationships between technical requirements and functional attributes and among functional attributes are identified. The technical requirement self interaction matrix is obtained by transitivity. The development agencies self interaction matrix, by its turn, can only be filled in after the technical requirement self interaction matrix is filled in.

Figure 9-2 shows the relationships between technical requirements and functional attributes. The existence of a relationship between a functional attribute and a technical requirement, indicated by a '1' in the technical requirements versus functional attributes matrix, means that that functional attribute 'verifies the accomplishment of' or 'implements' the related technical requirement. It can be observed in Figure 9-2 that the functional attribute 'vehicle weight' is not linked to any technical requirement listed. This occurs because the goal that requires vehicle weight is not included in this partial technical requirements list. It should however be included in a more complete list. 'Vehicle weight' is included in this list of functional attributes to demonstrate attribute partitioning and because it affects fuel economy.

The functional

Figure 9-2: Example of a partial matrix 'technical requirements versus functional attributes' for a car
attributes corresponding to the driveability technical requirement were derived from the Coordinating Research Council, Inc. Report No. 591, 1993 CRC Driveability Workshop [CRC, 1994]. This matrix also exemplifies a typical life cycle process functional attribute, 'vehicle tuning time', and a typical organisation functional attribute, 'productivity'. The matrix includes two attribute elements in order to help the description of the functional attributes: unit and direction. The upward arrow indicates 'the greater or higher the better' and the downward arrow indicates 'the smaller or lower the better'. Although not represented in Figure 9-2, other possible value for attribute direction is 'target is better'. Air fuel-mixture ratio is an example of functional attribute where 'target is better'. Its target value is 14.7:1.

Figure 9-2 shows also an alternative way of filling in the cells of the functional attributes self-interaction matrix. This alternative way uses '+' and '-' in the cells to indicate the existence of correlation between attributes. '+' indicates that changing one attribute in its desired direction the other will change in its desired direction as well, i.e., improving one attribute, the other will improve. '-' indicates that changing one attribute in its desired direction the other will change against its desired direction, i.e., improving one attribute, the other will degrade.

As mentioned in Chapter 6, Section 6.5.5, functional attributes shall be chosen from the functional model of a system in such a way they are as much independent as possible from each other. However, as they are related to physical attributes, it may happen that the same physical attribute affects different functional attributes. These attributes would then be said as dependent on each other. Although Figure 9-2 presents a functional attribute self-interaction matrix, such matrix can only be filled in after the identification of the dependencies between functional and physical attributes and among physical attributes. These dependencies are also a consequence of dependencies among lower level functional attributes.

Figure 9-2 also lists the development agencies that are the sources of functional attributes. These development agencies work for concept readiness prior to any product program launch. In the example, these development agencies are those which are not vehicle program specific and which work primarily with research or advanced development. Also, in this case, the development agencies self-interaction matrices are obtained by transitivity of the relationships between development agencies and functional attributes and among functional attributes.

Figure 9-3 shows the relationships between functional attributes and physical attributes of a car. The list of physical attributes reflects the physical architecture adopted, i.e., the partitioning of the system into subsystems. The life cycle processes and organisation characteristics are also included among the physical attributes as the system is composed by product, process and organisation elements. Powertrain, chassis, body, electric, interior characteristics refer to size, length, width, diameter, area, volume, material, etc. Life cycle processes characteristics refer mainly to the timing, cost, risk related attributes of the lowest level decomposition of life cycle processes activities. Organisation characteristics refer to the characteristics of the resources used in the organisation in order to accomplish the organisation function, such as: number of employees, number of a given type of machines, resource availability, etc.
The relationships between a functional attribute and a physical attribute in Figure 9-3 indicate that the physical attribute 'implements' or 'affects' the functional attribute, fuel economy, for example, is a functional attribute affected by powertrain, chassis, body, interior, electric physical characteristics. The functional attribute 'vehicle mass', for example, is the sum of powertrain, chassis, body, interior and electric subsystems masses. This is the reason to say that 'vehicle mass' is affected by powertrain, chassis, body, interior characteristics. 'Vehicle tuning time' is affected by powertrain, chassis and body characteristics as these subsystems need to be adjusted when integrated and is also affected by the FPS (Ford Production System) characteristics as the production process determines when and how the vehicle is tuned.

The physical attribute self-interaction matrix indicates the existence of relationships among physical attributes. In order to identify the existence of relationships among product physical attributes a physical architecture of the product with subsystems and physical connections can be considered. For example, as powertrain and chassis are connected, there are powertrain dimensions that affect chassis dimensions. The analysis in depth of these product physical relationships is the realm of more specific disciplines such as Dimensional Control (see [Jeffreys & Leaney, 1998]). The relationships among product physical
attributes and process physical attributes are the basis for the development of design for X concurrent engineering tools (see [Huang, 1996]). For example, the time to assemble a part is affected by the part symmetry [Boothroyd & Dewurst, 1994]. In Figure 9-3, the operation process characteristics are affected by the interior and powertrain characteristics. Also the order to delivery process characteristics, such as the width of the container, is a result of the body external dimensions. The relationships between product, process and organisation physical attributes can also be exemplified by using [Boothroyd & Dewurst, 1994] as different assembly resources, e.g. manual or automatic assembly, may be considered depending on factors such as product characteristics, production volume. In Figure 9-3, the organisation characteristics taken into consideration is the whole Ford Automotive Operations whose business processes are FPDS, FPS, After sales service, Order to delivery. This is the reason why these processes characteristics are related to the organisation characteristics.

The sources of physical attributes are the development agencies who actually design a vehicle meeting a vehicle program requirements. Considering the Ford Motor Company example the source of a vehicle design are the program specific organisations such as the PMTs (Program Module Teams). The design of the development and production processes is derived from the project management activity. Requirements and attributes shown in Figures 9-1, 9-2 and 9-3 are deployed down to the subsystems at the immediate lower layer of the system breakdown structure. Figure 9-4 illustrates how a passenger car can be broken down in its component subsystems. Observe that associated with each product component subsystem there are the corresponding life cycle processes and their performing organisations.

Figure 9-4 also illustrates how some of the vehicle functional attributes are deployed into its subsystems’ functional attributes. Fuel economy, for example, is decomposed into powertrain, chassis and body functional attributes. Vehicle mass is partitioned among all of the vehicle component subsystems. Vehicle tuning time results from powertrain calibration time, chassis tuning time and body tuning time.

In order to exemplify requirements and attributes deployment to subsystems, Figures 9-5, 9-6 and 9-7 show the ‘stakeholder requirements versus technical requirements’, ‘technical requirements versus functional attributes’, ‘functional attributes versus physical attributes’ matrices, respectively, for a powertrain.

In Figure 9-5, stakeholder requirements and technical requirements related to driveability, and exhaust emissions were directly allocated to the powertrain. Although the fuel economy related stakeholder requirements were kept the same, their corresponding technical requirement come as goal to be achieved by the powertrain subsystem. Also, some stakeholders and stakeholder requirements that have not appeared before for the car, are elicited for the powertrain. They exemplify the local stakeholders mentioned in Section 6.5.5. These are especially those related to powertrain life cycle processes and their performing organisations. For example, calibrators are stakeholders specific to powertrain. ‘Easy to calibrate’ is a powertrain specific stakeholder requirement and it maps to a calibration timing technical requirement. This calibration timing however affects the tuning time technical requirement for the car.
Also, similarly to the organisation requirement for a car assuming where the car would be manufactured, there is a requirement for the powertrain assuming where the powertrain would be manufactured.

Figure 9-6 shows the powertrain specific functional attributes presented in Figure 9-4 deployed from the car functional attributes and the emissions and driveability functional attributes directly allocated to the powertrain. Also specific powertrain life cycle processes and organisation functional attributes are elicited.

Following the same rationale for the car physical attributes elicitation shown in Figure 9-3, the list of powertrain physical attributes, shown in Figure 9-7, reflects the physical architecture adopted, i.e, the partitioning of the powertrain into subsystems: engine, transmission, driveline axle, PCS, life cycle processes and organisations. PCS characteristics may contain software specific characteristics such as lines of code, number of modules, number of global variables, etc.

Section 9.3 shows the next level of deployment of requirements and attributes through the diesel PCS example.
**Development agency**

**Self-interaction matrix.**

‘1’ means development agency's 'source' of technical requirement. Nothing, otherwise.

**Cross-interaction matrix.**

‘1’ anticipates that meeting one technical requirement may affect the other. Nothing, otherwise.

**Technical requirements**

'1' means that technical requirement 'translates' stakeholder requirement. Nothing, otherwise.

**Stakeholders**

- Cross-interaction matrix
- 1st level
- 2nd level
- 3rd level

**Technical requirements**

- Cross-interaction matrix
- 1st level
- 2nd level
- 3rd level

---

**Figure 9-5:** Example of a partial matrix 'stakeholder requirements versus technical requirements' for the powertrain
9.3 The diesel PCS example

This section focuses on demonstrating the use of Cradle modelling capabilities for the simultaneous analysis of product, processes and organisations. The diesel PCS example demonstrates the implementation of the 'concurrent structured analysis method' for a(n) (essentially) software subsystem. In Section 7.5 the analysis processes were demonstrated by using a car, its life cycle processes and their performing organisations as examples. Figure 7-28, for example, illustrates the physical decomposition of a car into its five subsystems: powertrain, chassis, body, interior and electric. Expanding the 'powertrain_system' in Figure 7-28 would show the powertrain subsystems and their physical interfaces. The powertrain subsystems are: engine, transmission, driveline and the PCS. This section, then, continues the analysis process initiated in Section 7.5 for a car. As illustrated in Chapter 8, Figure 8-1, in this example, the diesel PCS is decomposed down to the EGR FEATURE MODULES (software modules). The systems and subsystems to be analysed along the deployment process are the diesel PCS and the EGR control system.
Figure 9-7: Example of a partial matrix 'functional attributes versus physical attributes' for the powertrain

The EGR feature controls the EGR valve (the valve can be open, closed, or anywhere in between) that introduces exhaust gas into the intake manifold to reduce combustion temperature. A position sensor tells the module hardware where the EGR valve is. Two solenoids control vacuum to the EGR valve. The normally closed control solenoid allows manifold vacuum to the EGR valve when energised. The normally open vent solenoid allows atmospheric pressure into the vacuum line when not energised. EGR control affects: NOx and CO emissions during hot start and stabilised drive; fuel economy during hot...
start, stabilised drive and highway drive; driveability during idle conditions. The main objective of the EGR control is to reduce NOx, compromising fuel economy and driveability.

The requirements, functional and physical analysis processes are highly iterative and the sequence showed here is just for exemplification purposes. The analysis processes followed are as close as possible to those proposed in Chapter 7. For the product analysis processes, the requirements were captured and the functional and physical models were developed by the New Diesel team. For processes and organisation, requirements and models were derived from interviews and documents at Ford-Dunton-UK. The results of the analysis processes are requirements, functional attributes and physical attributes and their relationships. These are, as already exemplified in Section 9.2 for a car and powertrain, documented in matrices using Excel™.

9.4.1 Requirements analysis

Figure 9-8 illustrates the use of a DFD (Data Flow Diagram) in Cradle for the product requirements analysis model with the New Diesel Control System in the central 'bubble' surrounded by the stakeholders affected by the PCS product attributes. The arrows represent the stakeholders concerns towards the product.

![Figure 9-8: Product The diesel PCS requirements model](image)

Figure 9-9 illustrates the use of a BD (Behaviour Diagram) in Cradle for the process requirements analysis model. The rectangles represent the PCS life cycle processes (e.g. develop, evolve, purchase components, assemble, integrate into powertrain, calibrate powertrain). The rounded rectangles represent the stakeholders affected by the process attributes. These stakeholders are the sources of inputs (on the left of the central rectangles), receivers of outputs (on the right), sources of control (above the
central rectangle) and the sources of mechanisms (below). The arrows represent the stakeholder concerns towards the life cycle processes.

Figure 9-10 illustrates the use of a UCD (Use Case Diagram) in Cradle for the PCSE organisation requirements model. The rectangle represents the organisation, the bubble (use case) in the rectangle represent the life cycle process performed by the organisation. The human figures (actors) represent the stakeholders affected by the PCSE organisation attributes. The arrows represent concerns. In the figure, for clarity reasons, the concerns are named beside the actors and between "*'s.

Figure 9-9: Process_The diesel PCS life cycle processes requirements model

Figure 9-10: Organisation_The PCSE organisation requirements model
For each of the stakeholder concerns above requirements are captured and translated into technical terms as proposed in Section 7.5.1. These requirements can also be managed by Cradle as described in Section 8.2.1.2.

### 9.3.2 Functional analysis

Figure 9-11 illustrates the use of a DFD in Cradle essential model for the product functional analysis model of the diesel PCS. The central 'bubble' represents the PCS and the surrounding rectangles represent the elements in the environment interacting with the PCS. The arrows in this example represent mainly information flow although they can also represent energy and material.

Appendix D, Section D.4 provides a set of DFDs and STDs (State Transition Diagrams) representing the PCS functional decomposition down to the level where the EGR flow control function is supposed to be performed. Figure 9-12 illustrates a third level DFD in the PCS functional decomposition hierarchy. Observe in Figure 9-12 that the functions ‘Oxygen_Controller’ and ‘Determine_Oxygen_Available’ are related with the EGR flow control function.

![Figure 9-11: Product The diesel PCS product functional model (top level 1)](image)

Figure 9-13 illustrates the use of a BD in Cradle essential modelling for the life cycle processes functional analysis model. The emphasis of the process functional analysis is the identification of life cycle processes and their component functions. The information, material and energy these functions may exchange are not the focus at this stage of the analysis process. The rectangles in Figure 9-13 represent process functions and the rounded rectangles represent inputs, outputs, control or mechanisms depending whether they are positioned on the left, right, above or below a central rectangle, respectively.
Figure 9-12: Product PCS functional decomposition (level 3)

Figure 9-13: Process PCS life cycle processes (top level of functional decomposition)

The development process is decomposed up to the fourth level in order to describe the 'improve feature' function which is very much affected by the product and organisation attributes, as discussed in Chapter 4. Figure 9-14 illustrates the process 'improve feature'.

Appendix D, Section D.5 provides a set of BDs representing the processes at different levels of the development process functional decomposition hierarchy.
Figure 9-14: Process the process 'improve feature' at the fourth level of process functional decomposition

Figure 9-15 illustrates the use of an UCD in Cradle essential model for the functional model of the PCSE organisation. The UCD represents use cases in the PCSE organisation and their interactions with the 'actors' outside the organisation. Those interactions are represented by the arrows in Figure 9-15.

Figure 9-15: Organisation the PCSE organisation functional model
9.3.3 Physical analysis

Figure 9-16 illustrates the use of DFDs in Cradle implementation model to describe hierarchically the physical structure of the diesel PCS. As the focus of the PCSE work is the software development and hardware specification, Figure 9-16 shows sensors and actuators, which are in fact PCS components, as being terminators (represented by rectangles). The arrows in the diagram represent signal, data or control flows between the EEC or electronic module and the sensors and actuators.

The EEC-module is further expanded in successive DFDs. Three levels below the level shown in Figure 9-16, all features (software subsystems) that implement the PCS functionality are represented. This is illustrated in Figure 9-17.

For each feature a functional analysis is performed and the feature is structured into its constituent software modules. In Cradle, the feature functional analysis can be represented by lower level DFDs and the feature structure by STCs (Structure Charts). Figure 9-18 illustrates a DFD representing the functional analysis of the EGR_control feature. This DFD is actually a result of the expansion of the EGR_control 'bubble' in Figure 9-17 and is two levels lower than the one shown in Figure 9-17. Figure 9-19 shows the corresponding STC representation of the software structure, a physical analysis of the feature.

Appendix D, Section D.6, shows a more complete set of Cradle DFDs that represents the PCS product physical analysis. Appendix D, Section D.7, shows a DFDs and STDs in Cradle representing the functional analysis of the EGR_Control feature. Appendix D, Section D.8, contains the STCs representing the physical analysis of the EGR_Control feature.
Figure 9-18: Product functional analysis of the EGR control feature

Figure 9-19: Product structure of the EGR software modules
Figure 9-20 illustrates a description of how the evolution process for diesel PCS software is actually implemented. Observe that Figure 9-20 includes not only the functions or activities performed in the process but also inputs, outputs, control and mechanisms. This consists of an adaptation of Cradle's BD notation to the IDEF0's ICOM notation (see Appendix A for details on IDEF0).

Figure 9-21 and 9-22 illustrate the use of Cradle's UML notations for the physical modelling of the PCSE organisation. Figure 9-21 shows a Package Diagram (PD) showing the functional groups within the PCSE organisation. It shows the organisation structure. Figure 9-22 shows a Class Diagram showing the resources in the PCSE organisation and how they relate to each other. Both diagrams can be developed in Cradle as expansions of the PCSE organisation as shown in the UCD in Figure 9-15.

9.3.4 Requirements, attributes and relationships

Figure 9-23 shows some PCS requirements, some functions extracted from Cradle's PCS product essential model (see Figure 9-12 and Appendix D, Section D.4), and the traceability links between them. The '1's in the matrix represents the traceability links between requirement and functions. Examples of other requirements not shown in Figure 9-23 are:

- WCR (Worldwide Customer Requirements); MISRA Guidelines; 'The system shall be designed using structured methods for software development'; 'Reuse as far as possible l4 control system'; 'As it is a low volume program, development cost should be minimised and traded against simpler solutions that may increase variable cost to produce the most cost effective overall solution'; 'The system has a short life architecture'; 'The system shall be designed with sufficient confidence to protect for reasonable changes in program requirements for the identified applications'; 'The system shall not be compromised by flexibility for future requirements'; 'The is New Diesel installations is intended to be a minimum impact replacement to the gasoline engine installation'; 'The system architecture shall be an ECU interfaced to a Injection Control Unit (ICU) via a FIEONA type protocol'; 'FIEONA will require redesign to support the is5 configuration'.

Examples of other functions can be found in Cradle's essential model in Appendix D, Section D.4.

Figure 9-24, 9-25, 9-26 and 9-27 shows the functions in the PCS product essential model, the PCS components (hardware and software) at various levels of the PCS breakdown structure and the traceability links between functions and components. Focus was given to the functions and components related to the EGR_control feature. Therefore only components affecting that feature were listed in the figures. The complete physical decomposition from where the physical components were extracted is in Appendix D, Section D.6. Figures 9-16 and 9-17 contain the physical components listed in the matrices in Figures 9-24 and 9-27, respectively.

Figure 9-28 illustrates the impact links between the functional and physical attributes of the EGR_control feature, the PCS evolution process and the diesel PCS development organisation. The functional models used for deriving the functional attributes in Figure 9-28 are in Figures 9-13, 9-15 and 9-18, for the evolution process, the development organisation and the EGR_control feature, respectively. The physical models used for deriving the physical attributes in Figure 9-28 are in Figures 9-19, 9-20 and 9-22 for the EGR_control feature structure, the diesel PCS evolution process and the diesel subsystems projects group, respectively.
Figure 9-20: Process_diesel PCS evolution process physical analysis
Figure 9-21: Organisation_PCSE functional units

Figure 9-22: Organisation_resources in the PCSE organisation for diesel PCS development
### Figure 9-23: Product-Traceability links between requirements and functions of the diesel PCS product

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Engine function</th>
<th>Engine running</th>
<th>Engine cranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>- ngine running</td>
<td>- Engine cooling</td>
<td>- Coolant control</td>
<td>- Fuel control</td>
</tr>
<tr>
<td>- Coolant temperature</td>
<td>- Coolant temperature</td>
<td>- Coolant temperature</td>
<td>- Coolant temperature</td>
</tr>
<tr>
<td>- Coolant temperature</td>
<td>- Coolant temperature</td>
<td>- Coolant temperature</td>
<td>- Coolant temperature</td>
</tr>
<tr>
<td>- Coolant temperature</td>
<td>- Coolant temperature</td>
<td>- Coolant temperature</td>
<td>- Coolant temperature</td>
</tr>
<tr>
<td>- Coolant temperature</td>
<td>- Coolant temperature</td>
<td>- Coolant temperature</td>
<td>- Coolant temperature</td>
</tr>
</tbody>
</table>

### Figure 9-24: Product-Traceability links between PCS functions and PCS components

<table>
<thead>
<tr>
<th>PCS functions</th>
<th>PCS components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
</tbody>
</table>

### Figure 9-25: Product-Traceability links between PCS functions and EEC module components

<table>
<thead>
<tr>
<th>PCS functions</th>
<th>EEC module components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
</tbody>
</table>

### Figure 9-26: Traceability links between PCS functions and microcontroller components

<table>
<thead>
<tr>
<th>PCS functions</th>
<th>Microcontroller</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
</tbody>
</table>

### Figure 9-27: Traceability links between PCS functions and software subsystems (features)

<table>
<thead>
<tr>
<th>PCS functions</th>
<th>Software subsystems (features)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
<tr>
<td>Determine oxygen available</td>
<td>Oxygen sensor</td>
</tr>
</tbody>
</table>
Figure 9-28: Product, process, and organisation - Impact links between functional and physical attributes of the EGR control feature
9.4. The gasoline PCS example

This section aims:

- to propose a complexity analysis procedure in order to address the complexity management issue;
- to demonstrate a novel use of Chaco-2.0, for product restructuring;
- to discuss the advantages of restructuring through the gasoline PCS example;
- to propose further uses for the complexity analysis procedure.

The 'total view framework', presented in Chapter 5 addressed the issue of scope by defining that product, process and organisation should be considered as elements of the balanced system solution resulting from the integrated development effort. Chapter 6 proposed the concepts of requirements, attributes and relationships as an approach to integrate those elements. Chapter 6, Section 6.5.6 proposed a way to manage complexity through the analysis of clustering (that could be) of requirements, attributes, functions, activities, resources or physical parts based on the strength of their relationships seeking to maximise cohesion and minimise coupling. The idea underlining the clustering approach is to monitor complexity as it grows. The paradigm proposed is to identify, earlier in the product development life cycle, opportunities for restructuring product, process and organisation to minimise complexity and with it, lead time, cost and risk. It is proposed that complexity should be continuously monitored and restructuring should continuously take place to minimise it. This is an alternative for the paradigm of drastic reengineering.

The clustering approach contributes to improve the visibility of relationships. As already demonstrated in the previous sections, Sections 9.2 and 9.3, the size of the relationship matrices can be potentially very large. Re-ordering the elements in the matrices in clusters would allow a more focused analysis of relationships. The size of matrices has been a disadvantage of QFD use, for example.

As the size of the relationship matrices can be very large it is highly recommended to use computer aid to perform the partitioning into clusters. Traditional partitioning algorithms such as those used for Group Technology purposes and reviewed in [Kusiak, 1990] do not fit to the large size of matrices expected from the total view applications. Therefore large graph partitioning algorithms are adopted. This section demonstrates the use of Chaco-2.0, described in Chapter 8, Section 8.2.2, for restructuring a product architecture of a subsystem of a gasoline PCS software application.

This section is structured as following: Section 9.4.1 introduces the object of the complexity analysis procedure, a software subsystem of a gasoline PCS application. Section 9.4.2 describes the complexity analysis approach and procedure. Section 9.4.3 presents and analyses the results obtained. Section 9.4.4 draws some conclusions based on the results. Section 9.4.5 proposes other applications for the complexity analysis procedure within the 'total view framework'.

9.4.1 Background

The complexity analysis procedure to be presented in Section 9.4.2 aims to analyse a gasoline PCS software subsystem that resulted from the evolution over the period 1978-1996 and to identify and justify
improvement opportunities. The analysis compares the way Ford’s partitions the software with the partitioning performed by a graph partitioning algorithm, Chaco-2.0 (see Chapter 8, Section 8.2.2).

The object of analysis is the idle speed control (ISC) subsystem, part of the overall PTEC software strategy (see Chapter 4, Section 4.2.4) used for developing the gasoline PCS software. The overall PTEC strategy is documented in a strategy document containing software code written in C. The strategy document is called KBAD0. Section 12 of the strategy document refers to the ISC subsystem and is called ISC_KBAD0. The document release date is 17/10/1996.

As mentioned in Chapter 4, Section 4.2.4, the PTEC software strategy was developed by Ford in an attempt to improve software reuse. The PTEC software strategy consists of 700KBytes of software code written in C. It contains all software subsystems that can be used to compose a given gasoline PCS application software. Depending on the application, software subsystems are removed from the original PTEC strategy in order to comply with the specific requirements of that particular application. In theory, all the gasoline software applications could be derived from the PTEC strategy.

The PTEC strategy was a result of the gasoline PCS software evolution over the period 1978-1996. This example provides an indication that continuous modifications at the bottom level of the product breakdown structure may lead to a non-optimised partition of the product causing increased and unnecessary complexity (in terms of coupling and cohesion).

Section 12 of the PTEC strategy document describes the adaptive air bypass idle speed control system. The ISC system is designed to regulate the duty cycle of an air bypass solenoid as necessary to obtain the desired engine speed for all idle operating conditions (base idle, hi-cam, various accessory loads) and provide for a dashpot action. Predicted airflows for the different load states at idle are adaptively corrected to minimise the impact of hardware variability. Acceptable idling performance is achieved by a careful balance of bypass air solenoid control and feedback spark control. The amount of airflow through the air bypass is controlled by the solenoid position, which is in turn determined by the solenoid duty cycle (ISCDTY). The objective of the idle speed control strategy is to determine ISCDTY. Feedback spark control is the subject of another section of the strategy document and is not approached in this thesis.

Section 12 is divided into 5 sections, each one representing a feature (or software subsystem), as illustrated in Figure 9-29.

<table>
<thead>
<tr>
<th>Document Section</th>
<th>Feature Label/Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.1</td>
<td>IACEX_V3 IACEX</td>
</tr>
<tr>
<td>12.2</td>
<td>IACDN_NO_VERSION_LEVEL</td>
</tr>
<tr>
<td>12.3</td>
<td>IACFB_NO_VERSION_LEVEL</td>
</tr>
<tr>
<td>12.4</td>
<td>IACFF_NO_VERSION_LEVEL</td>
</tr>
<tr>
<td>12.5</td>
<td>IACHW_IDLE_AIR_CONTROL_HEATED_WINDSHIELD</td>
</tr>
</tbody>
</table>

*Figure 9-29: The feature sections*

The strategy document sections and subsections reflect the way Ford partitioned the ISC strategy.
Figure 9-30 presents a block diagram description of the idle speed control strategy. It also illustrates the distribution of functionality among 4 of the strategy document sections. Section 12.5 is not mentioned in the figure but it would be included in the ‘desired engine speed calculation’ and considered as part of Section 12.2.

Each of these subsections is further sub-divided into two lower levels as indicated in Figure 9-31:

<table>
<thead>
<tr>
<th>Level 2 section</th>
<th>Number of level 3 sub-sections</th>
<th>Number of level 4 sub-sections</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.1</td>
<td>8</td>
<td>13</td>
</tr>
<tr>
<td>12.2</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td>12.3</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>12.4</td>
<td>9</td>
<td>14</td>
</tr>
<tr>
<td>12.5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Total number of sub-sections</td>
<td>33</td>
<td>45</td>
</tr>
</tbody>
</table>

In order to identify how Ford partitioned the software into sections and sub-sections, a Structure Chart (STC) was developed in Cradle, representing the structure of the strategy document. Section F.1, in Appendix F, contains the ISC KBAD0 strategy document structure.

9.4.2 The complexity analysis approach and procedure

The approach to the complexity analysis is to decompose the system (in this example, a software) down to its elementary units (in this example, software units) and investigate at the very bottom level of the system breakdown structure the exchange of data among those units. The units are then clustered together. The way Ford partitioned the whole set of software units is compared to the partitioning performed by Chaco-2.0. The results of the partitioning by both approaches are evaluated according to a complexity index. The complexity index is obtained by the clustering quality evaluation approach proposed by Hitchins (1992) (see Chapter 2, Section 2.2.4)
The tools used for carrying out the complexity analysis procedure are:

- Cradle™ for modelling a given system architecture using DFD and STC notations;
- Chaco-2.0 for partitioning purposes;
- Microsoft Excel™ for the calculation of the complexity index.

The complexity analysis procedure followed in this example consists of the following steps. The steps are indicated with a generic title. The description under each title corresponds to what has been actually done in this example.

1) Obtain the system breakdown structure

Since the input to the analysis is a document containing software written in C, it was necessary to translate that document into a more graph like format. It was decided then to consider the whole Idle Speed Control software as a system. In order to identify the system breakdown structure it was necessary to investigate the call of subroutines in the software. This can be easily investigated with the use of an STC in Cradle. The resulting STCs are depicted in Appendix F (Section F.2)

2) Translate the structure into a graph format

Once the hierarchical system structure was identified, a hierarchy of data flow diagrams (DFDs) could be developed. The top-level DFD was developed with the help of Ford documents showing the other subsystems the ISC interacted with. The lower level diagrams were obtained by using the STCs in Appendix F (Section F.2) and the strategy document. The resulting DFDs are depicted in Appendix F (Section F-3).

DFDs decomposition did not stop at the modules identified in the strategy document. Each level 4 subsection (e.g. 12.x.x.x) of the strategy document corresponded to a software module. However software modules could contain many software units. These software units could be easily identified by the comments in the strategy document. Also, for simplicity, nested IFs were considered as one software unit. These software units constituted the bottom level of the ISC system breakdown structure.

3) List and weigh the graph vertices

Those software units became the vertices of the graph to be partitioned by Chaco-2.0, their data exchange corresponded to the edges linking those vertices. Appendix F (Section F-4) shows a list of vertices ordered according to Ford’s strategy section numbering. Vertex weights were assigned to each vertex. Vertex weights corresponded to the number of lines of code in each software unit.

4) Assess graph edge weights

As each of the bottom level DFD ‘bubbles’ were described by a Process Specification (PSPEC) in Cradle, Cradle reporting capability ‘Spec List’ could be used to list the inputs and outputs of each software unit at the bottom level of the system breakdown structure. That report provided by Cradle is in Appendix F (Section F.5). As Cradle provides reports in ‘text’ format, they can be read by word processing tools such as Microsoft™ Word™. The information in the word document was rearranged and converted to be used by Microsoft Excel. In Excel, each input/output or flow was classified as data or control information, by assigning a D or C to an appropriate field. Data information was the one used for
calculations purpose and control information was the one used for decisions purposes in the IFs of the software. Data information was represented by plain arrows in the DFDs and control information was represented by dashed arrows in the DFDs. Other information associated to each flow in Excel is: INPUT TO and OUTPUT FROM. Appendix F (Section F.6) shows the list of flows and their associated information in an Excel file. The D/C classification was used to assign weights to the edges of the graph composed by the bottom level software units (bottom level ‘bubbles’ in the DFD hierarchy). The work of Dhama (1995) on quantitative models of cohesion and coupling proposes that control information should weight twice as data information for cohesion and coupling assessment. Therefore, in this example, the weight of an edge linking two vertices is calculated by: (number of data flows linking the two vertices) + 2 x (number of control flows linking the two vertices). Appendix F (Section F.7) shows an Excel spreadsheet listing the vertices, their neighbours and the weight of the edges linking a given vertex to its neighbours (the neighbour of a vertex 1 is a vertex linked by an edge to vertex 1).

5) Format Chaco-2.0 input files
Chaco-2.0 input files must have an appropriate format so they can be interpreted by Chaco-2.0.

% This is the format of the graph input file
Number-of-vertices Number-of-edges {1}[1][1]
{Vertex-number} [Vertex-weight] neighbour (edge weight)

Appendix F (Section F.8) contains Chaco-2.0 input files. The first line of the input file contains three integers: the number of vertices in the graph (101 in this example), the number of edges in the graph (180, in this example), a third parameter with up to 3 digits (111 in this example). The third parameter indicated that the analysis would include weights on vertices and edges and, that the input file would have a line per vertex neighbour.

6) Set Chaco-2.0 parameters
In order to control the computation performed by Chaco-2.0, different values can be assigned to some parameters. To the KL_IMBALANCE parameter was assigned the value ‘0’ for balanced partitions and a value ‘1’ for unbalanced ones. The default value of this parameter is ‘0’ as Chaco generally tries to keep the vertex weight sums in the partitions it generates as nearly equal as possible. For performing the partitioning, a value ‘FALSE’ (default) was assigned to a SEQUENCE parameter, for sequencing a value ‘TRUE’ was assigned to that parameter. In this example, sequencing followed partitioning.

7) Partition and sequence (using Chaco-2.0)
Using Ford’s partitioning indicated by the section numbering of the strategy document only sequencing was calculated using Chaco-2.0. An example of Chaco’s output file containing the sequence can be found in Appendix F (Section F.9).

The 101 vertices were also partitioned using two algorithms implemented by Chaco: imbalanced multi-level KL and spectral bisection. Partitions sizes of 4, 8, 16 and 32 sets were chosen. An example of Chaco’s partitioning output file can be found in Appendix F (Section F.10)

8) Present partitioning results (using Excel)
Square matrices indexed by the graph vertices were implemented in an Excel spreadsheet. The matrices’ cells were filled in with edge weights, following an N^2 chart format (see Appendix C). Unlike the
traditional N² chart format, the matrix diagonal does not contain the vertex names but the vertex weights. To reorder the matrix columns and rows the Excel's sorting mechanism was used.

9) Calculate a complexity index

The approach to the calculation of a complexity index was based on the work of Hitchins (1992) presented in Chapter 2, Section 2.2.4. The formula used to calculate the complexity index is, for a given square matrix:

\[ \text{complexity index} = \sum \text{number of matrix cells} \times (\text{matrix cell value}) \times (\text{matrix cell distance from matrix diagonal}) \]

The calculation procedure was implemented in a 'macro' using Excel. The macro coded in Visual Basic is presented in Appendix F (Section F.11).

10) Present the results

Appendix F (Section F.12) contains the results obtained, summarised in the following:

- Ford's partitioning with Chaco's sequencing, 5 sets, complexity index: 4706, see Figure 9-32;
- Imbalanced multi-level KL, 4 sets, complexity index: 3607;
- Balanced spectral bisection, 4 sets, complexity index: 3394, see Figure 9-33;
- Ford's partitioning with Chaco's sequencing, 32 sets, complexity index: 5889;
- Imbalanced multi-level KL, 32 sets, complexity index: 3820;
- Balanced spectral bisection, 32 sets, complexity index: 4408.

Figure 9-32 and 9-33 present the matrices resulting from Ford's partitioning in 5 sets and Chaco's balanced spectral bisection partitioning in 4 sets, respectively. Notice the complexity index at the top left corner of the matrices.

9.4.3 Analysis of the results

Figure 9-34 illustrates a comparison of Ford's partitioning with the partitioning performed by the algorithms implemented in Chaco-2.0. Partitioning into 4 and 32 sets were chosen because the number of level-2 and level-3 sub-sections in the Ford's strategy document was 5 and 33 (see Figure 9-31), respectively, and because Chaco can only partition a set into 'powers of 2' sets. Figure 9-34 shows that the algorithmic partitioning provides an improvement of 23 to 35% in the complexity index if compared with Ford's partitioning. The improvement was observed with different partitioning algorithms, with either balanced or imbalanced partitioning, with either 4 or 32-set partitioning.

Figure 9-35 shows the number of software units originally in Ford's partitions that are in each set of the algorithmic partitioning for the 4-set partitioning. Figure 9-35 indicates that there is a correspondence between both partitions. Sets 0, 1, 2 and 3 in the algorithmic partitioning (spectral bisection) correspond to sections 12.2, 12.4, 12.3 and 12.1, respectively, in Ford's strategy document.

As the algorithmic partitioning proposes a migration of 6 software units from section 12.1 to 12.3 and 7 software units from section 12.4 to 12.1, it was investigated the characteristics of the interactions between the mentioned sections (12.1 and 12.3 and, 12.4 and 12.1). Figure 9-36 provides a list of these interactions.
Before Chaco's partitioning

Figure 9-32: Ford's partitioning into 5 sets and Chaco's sequencing
After Chaco's partitioning

Figure 9-33: Balanced spectral bisection partitioning into 4 sets
Figure 9-34: Complexity indices calculated from Ford’s partitioning compared with algorithmic partitioning

<table>
<thead>
<tr>
<th>Algorithmic partitioning (see Figure 9-33) total: 101 units</th>
<th>Set 0</th>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>total: 34 units</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12.2/5 total: 36 units</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12.4 total: 24 units</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12.3 total: 20 units</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12.1 total: 21 units</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(software unit 47 and 48)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(software units 57 and 64)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(software units 55, 59 and 62)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(software unit 67)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(software units 8, 9, 10, 11, 13, 14)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 9-35: Correspondence between the number of software units in Ford’s and in the algorithmic partitioning

Figure 9-36: Relationships between section 12.1, 12.3 and 12.4 vertices which swap partitions

It is observed that the outputs of software units originally in 12.4 are being used as control inputs by many software units in 12.1. Those software units in 12.4 are those which calculate the final value of the overall theoretical output of the units in that section. The algorithmic partitioning proposes that those final calculations should be performed where the results are going to be used. Also control and data information generated by software units in 12.1 are used by software units in 12.4. That information is not related to the expected output of 12.1 but with checking and updating data. The algorithmic partitioning proposes that these activities be performed where the resulting information is going to be used.
The relationships between 12.1 and 12.3 are of three types: section 12.1 provides feedback information used as control inputs by section 12.3, section 12.1 calculates feedforward control information to be used by 12.3 and section 12.3 calculates control information to be used by a diagnostics function in section 12.1. In this case, the algorithmic partitioning proposes to send the diagnostics software unit to section 12.3, to move the control calculations to 12.3 and to calculate the final value of the theoretical 12.1 output in section 12.3.

Figures 9-37 and 9-39 are N2 chart representations of the block diagram in Figure 9-30 with some diagonal cells grouped together to reflect the adopted partitioning. Figure 9-38 and 9-40 are a graphical representation of the resulting matrices in Figures 9-32 and 9-33, respectively. Figures 9-32 and 9-33 refers, respectively to:

- Ford's partitioning with Chaco's sequencing, 5 sets (only 4 sets were used for this analysis);
- Balanced spectral bisection, 4 sets.

![Figure 9-37: Theoretical N2 chart for Ford's partitioning](image)
Figure 9-38: N2 chart after Ford's partitioning and sequencing using Chaco-2.0

Figure 9-39: Theoretical N2 chart for algorithmic partitioning
A comparison of Figure 9-37 with Figure 9-38 reveals the existence of coupling where it was not anticipated in the theoretical modelling and the existence of strong and medium coupling where it was expected a weaker coupling. The existence of coupling below the N2 chart diagonal shows the existence of feedback not anticipated in the theoretical model. The coupling stronger than expected above the N2 chart diagonal is due to the use of expected feedforward outputs as control parameters by the subsequent partitions. Figure 9-41 shows the list of cells that were expected to be empty in the theoretical model (Figure 9-37) and are, in reality, filled (Figure 9-38).

Figure 9-41: Non-expected relationships between Ford's partitions in Figure 9-32
The same can be said about the comparison of Figure 9-39 with Figure 9-40. However, strong coupling between partitions is more often in Figure 9-38 than in Figure 9-40. Figure 9-42 shows the list of cells that were expected to be empty in the theoretical model (Figure 9-39) and are, in reality, filled (Figure 9-40).

**Figure 9-42: Non-expected relationships between algorithmic partitions in Figure F-58**

### 9.4.4 Concluding remarks

- The gasoline PCS example analysed here has shown that the way Ford partitions its software can be improved by at least 28% in a complexity index that reflects cohesion and coupling levels.

- The greatest contributions of the algorithmic partitioning is the uncoupling of Set 0 (corresponding to Section 12.2) and the de-coupling between Sets 2 and 3 (corresponding to Sections 12.3 and 12.1, respectively). Entire software modules such as the 12.1.5.1 (isc_timers) and 12.1.7.1 (isc_fmem) were suggested to move from section 12.1 to section 12.3. This would provide a better mapping between the theoretical functional architecture and the actual physical architecture.

- This move is significant if one considers that Ford-PCSE is organised by feature groups and that each section in the strategy document corresponds to a feature. If the actual software modules allocated to different features have a higher coupling that they could have (if re-arranged) than unnecessary communication overheads are being incurred.

- Given that Ford-PCSE is organised by feature groups, strong coupling between features (software subsystems) may have an implication in the communication overhead between development groups. Therefore, a better partitioning may also means savings in the communication overhead.

- The current interface complexity of control strategies (see Figure 4-20) in the gasoline PCS suggests that a better partition can be obtained if the whole PTEC strategy software is revisited by using the approach above. With a better partition development time can be shortened and at least the communication cost between development groups can be reduced.

Generalising the remarks above, it can be said:
• A lower complexity index indicates less coupling and higher cohesion between the partitions or modules of a system solution. It is expected that each partition correspond to a function or to a cohesive set of functions in the functional model. A better correspondence between functions and partitions has the advantage of easing the maintenance or evolution stages of the product life cycle.

• It has been observed that what happens, in practice, is that the evolution of a system occurs from its physical architecture. However, if the changes in the physical architecture do not keep a mapping to the functional architecture, there is no assurance of a logical grouping of parts in the product. As a consequence, cohesion gets lower and coupling tends to grow to accommodate non-foreseen interface problems. Therefore, complexity grows and with it, lead time, cost and effort.

• The investigation of the evolution of the complexity index can be used to call for a review in the functional model or concept. Reallocation of units between partitions may even indicate the need to modify the functional architecture. Ideally, each function should correspond to one partition so that a required modification in the functionality would be obtained by just modifying software units in that partition without affecting other partitions. In reality, that is not always possible and can be made worse if attention is not given to the allocation of new created software modules to the appropriate partition.

• An example of collateral effects of non-monitoring a complexity index or of not keeping the links between physical architecture and functional architecture is the fact that, in general, companies are organised according to the functional architecture of the product. Physical partitions allocated to be developed by these organisational units must correspond as much as possible to the functionality the organisation units are supposed to deliver. Otherwise, unnecessary coupling is being created between organisation units and with it communication overheads and development time increase.

• The complexity analysis procedure proposed in this section provides a way of continuously monitoring complexity growth so that it can be avoided to get to a point where reengineering is the only way out.

9.4.5 Other applications within the ‘total view framework’

Some direct applications of the complexity analysis procedure within the ‘total view framework’ are the re-structuring of self and cross interaction matrices of requirements, functional attributes, physical attributes and development agencies. A total system solution may be sought, for example, seeking to minimise the complexity index. The axiomatic design theory [Suh, 1990], for example, provides the theoretical support for the application on cross-interaction matrices between functional and physical attributes.

Similarly to the analysis above which used DFDs as graph notation, BDs can be used to analyse and optimise process architecture following the same approach above. [Kusiak, Larsson & Wang, 1994] provides some examples using a triangularisation algorithm for process scheduling.

The example developed in this section (Section 9.4) demonstrated the use of a complexity analysis method applied to a product architecture (software in this case). However, the analysis approach above
can be used in a broader context within the 'total view framework', e.g., for team architecture and overall system optimisation.

**Team architecture**

Consider, for example, a simplified and very hypothetical model of the whole system development organisation as shown in Figure 9-43.

Figure 9-43 shows the flow of requirements and attributes among organisation units at a given level of the system breakdown structure. The decomposition of the organisation units follows the product breakdown structure until it gets to a point where one individual is responsible for the processing of a requirement and its transformation into a product, process and/or organisation attribute. A vertex in the graph representing the bottom level organisation would be a person or an information processing resource (e.g. a computer) and an edge would be a requirement or attribute. Vertex weight would reflect the processing capability of the vertex. If all vertices are people, it can be assumed the same weight for all of them. Edge weight can reflect the importance of a requirement to a customer or the technical difficulty in obtaining an attribute. Partitioning can be done in levels (e.g. using multi-level KL algorithm in Chaco-2.0) and a hierarchy of teams would be obtained seeking minimisation of inter-team communication and maximisation of intra-team communication.

The matrices resulting from the analysis processes in the 'total view framework' provide the vertices (sources of 'what' and 'how'), the edges (requirements and attributes) and the feedforward links between vertices (relationships, i.e., a source of 'what' provides a 'what' item to a source of 'how'). Feedback links cannot be inferred from the matrices and have to be provided.

**Overall system optimisation**

Michelena & Papalambros (1995) propose to solve a large optimisation problem by dividing it into smaller and more manageable sub-problems. The large optimisation problem would have its equations translated into a graph format in such a way that dependent variables would become the vertices in the graph and independent variables the edges linking the vertices. The resulting graph would then be input to a graph partitioning algorithm that divides the graph into smaller and as uncoupled as possible partitions.

A similar approach can be adopted by using the information provided by the analysis processes in the 'total view framework'. Functional attributes are allocated, partitioned or decomposed (see Section 6.5.5) into lower levels in the system breakdown structure. At a chosen level in the system breakdown structure, the identified functional attributes of all product, process and organisation subsystems at that level are made the vertices of a graph. Physical attributes linked to the functional attributes by impact cross-references in Cradle can be easily identified through the use of Cradle's reporting capability (e.g.
Xref summary or Xref list reports). As in the procedure described above for the gasoline PCS, Cradle reports can be imported into an Excel spreadsheet.

Functional attributes would then be partitioned into smaller sets using Chaco-2.0 and the complexity index as a criterion for identifying the best partitioning. As physical attributes represent the edges linking the functional attributes, they would also be automatically clustered together by the partitioning procedure. The result of the process is smaller matrices linking functional and physical attributes. These matrices can be used for sub-problem optimisation, where the coupling between partitions is very weak.

This chapter demonstrated the applicability of the 'total view framework' and 'concurrent structured analysis method' for software products – the gasoline and diesel PCS. Chapter 10 demonstrates the applicability of the framework and method to a mechanical product, the seat height adjust mechanism (SHM)
Chapter 10
The SHM example

This chapter aims:

• to demonstrate the applicability of the ‘total view framework’ and ‘concurrent structured analysis method’ to an essentially hardware product, a mechanical product, the seat height adjust mechanism (SHM) and to manufacturing and assembly processes and organisations;
• to demonstrate the utilisation of a commercial systems engineering environment, Cradle, for the SHM;
• to demonstrate the applicability of the concepts of requirements, attributes and relationships for an integrated development approach applied to the SHM.

10.1 Introduction

This example was first introduced in Chapter 8, Section 8.1. The PCS examples concentrated on software subsystems of the PCS and on their development process and organisation. The SHM example, on the other hand, aims to demonstrate the applicability of the ‘total view framework’, ‘concurrent structured analysis method’ and the concepts of requirements, attributes and relationships to a mechanical product and its manufacturing and assembly processes and organisations.

Figure 8-3, in Chapter 8, situates the SHM in the system breakdown structure of a car together with the SHM life cycle processes and their performing organisations. Focus will be given to the SHM manufacturing and assembly processes and to the H.R. Adcock Ltd. organisation responsible for developing and producing the SHM.

The SHM is developed and manufactured by H.R. Adcock Ltd, a 2nd Tier supplier to the Ford Motor Company. H.R. Adcock Ltd supplies the SHM to Johnson Controls Automotive, who manufactures the complete seat for Ford. The SHM is currently being produced to be incorporated on the new Ford Focus, a replacement for the Escort.

In the car, the SHM is positioned underneath the driver’s seat. Its interface with the user is a handle which, when rotated, will change the seat height roughly in proportion to the input rotation. The seat will stop at a desired position within a specified span. The SHM is connected to the seat frame in order to transmit to the seat the height adjustment required by the user. The SHM is conceived in such a way it requires the same amount of effort from the user to adjust the seat position independent on its initial position or on the user weight, within the specified height span.

Physically, the SHM consists of a cylindrical axe, called spindle, connected, at one end to a handle mechanism (interface with user) and at the other end to the seat frame. The frame end of the spindle is mobile. It is connected to a nut moving on a threaded region on the spindle. The nut converts the rotational movement input by the user into a translational movement transmitted to the frame. The SHM is pre-loaded to keep user effort low and constant. The pre-load is obtained by a pre-compressed spring held between the frame end of the spindle and a trunnion, which is by its turn, held by a thrust washer shoulder-rolled on the spindle.
Figure 10-1 provides an illustration of the SHM (see STAGE 3 in the figure) and shows some of its components and how they are placed during the assembly process.

**STAGE 1**

**STAGE 2**

**STAGE 3**

Figure 10-1: SHM, its components and assembly process

The SHM is a good example of integrated development as the product, its production process and the H.R. Adcock Ltd organisation were all developed by a small team coached by Neil Adcock, who is skilled in mechanical design, manufacturing engineering and management. Neil Adcock developed the product around a shoulder rolling manufacturing process for which his company retains an industry patent. Also the development considered the fact that the product unit cost should not be superior to a given limit.

This example was developed using:
- 2 visits to H.R. Adcock Ltd;
- 2 interviews with Neil Adcock;
- documents provided by H.R. Adcock Ltd. (see Appendix E, Section E.1);
- customer requirements list (see Appendix E, Section E.1);
- a sample of the product;
- the expertise of a manufacturing engineer, Lee Barnett.

This example aims to demonstrate the applicability of the 'total view framework' to an essentially hardware product, a mechanical product in this example. This example aims to demonstrate, that although the requirements, functional and physical modelling methods and tools were imported from the software arena, they are also applicable to a hardware product.
As product, process and organisation were already well established, this example was developed through a reverse engineering exercise. There was no attempt to simulate the iterations undertaken by Neil Adcock's team when developing the product. Therefore the physical models were developed first and then the functional models. The general sequence followed for the development of this example was:

1) modelling;
2) requirements and attributes identification;
3) relationships identification.

Cradle was used for modelling and requirements documentation. Excel was used for attributes and relationships identification.

This chapter is structured as following: Sections 10.2, 10.3, 10.4 demonstrates the modelling approaches introduced in Chapter 7 and implemented using Cradle for the requirements, functional and physical analysis of product, process and organisation; Section 10.5 demonstrates the requirements and attributes capture and the identification of their relationships; Section 10.6 documents the lessons learned with this example.

### 10.2 Requirements analysis

The development of the requirements analysis models consisted of identifying the stakeholders interacting with product, life cycle processes and organisations. The organisation selected was H.R Adcock, responsible for the development and production of SHMs. Although H.R. Adcock develops and produces the SHM, the life cycle process modelling focused only on manufacturing and assembly production processes. Therefore, development processes such as the manufacturing and assembly processes set up are not included in the scope of modelling presented in this chapter.

Figure 10-2 presents the product requirements analysis model. It consists of a Data Flow Diagram (DFD) with the product main function in a central bubble. The central bubble is surrounded by rectangles representing the stakeholders who actually interact with the product over its life cycle. In practice, the stakeholder who provides the requirements of most impact to the H.R. Adcock work is the seat manufacturer (Johnson Controls). The arrows linking the stakeholders to the product mission (central bubble) express the stakeholder concerns. For example, the car manufacturer is likely to be worried with weight, cost, functionality adequacy.

Figures 10-2 does not imply that the H.R. Adcock will capture the requirements corresponding to all the stakeholder concerns shown in the figure. Also, Figure 10-2 does not imply that the H.R. Adcock organisation will interact with all those stakeholders to get their requirements, for example. The figure aims to present a totality of sources of requirements, some will be pursued, some not. For example, the car manufacturer requirements are likely to come via the Tier 1 supplier, the seat manufacturer. As a Tier 2 supplier, H.R. Adcock Ltd. is not expected to interact with the car manufacturer directly. It is also important to remember that the SHM is part of an upper layer system, the seat, which is supposed to have undergone this process at their layer of integrated development. Some of the requirements captured for the seat at its layer would then be allocated, partitioned or decomposed to the SHM.
During the requirements analysis process, not only the product operational mission is identified but also the product life cycle processes. Figure 10-3 lists the life cycle processes identified for the SHM. They include: development, material purchase, bough-in parts purchase, production, delivery, integration into seat, use and disposal. All those processes are at the same building block (see Chapter 5), and consequently the same layer of integrated development as the SHM. Therefore those life cycle processes are from a Tier 2 perspective.

Figures 10-4 and 10-5 exemplify the requirements analysis of the assembly and manufacturing processes, respectively. Although they are represented as different figures, the requirements analysis for assembly
and for manufacturing are part of the same diagram in Cradle. These processes are sub-processes of the production process in Figure 10-3. The stakeholders surrounding the life cycle process (or sub-process) represent the sources of inputs, outputs, control or mechanisms and who are affected by the life cycle process attributes. Concerns are written alongside the stakeholders.

Figure 10-4: Process – The SHM assembly requirements model

Figure 10-5: Process – The SHM manufacturing requirements model
Figure 10-6 illustrates a Use Case model representing the organisation requirements model for the H.R. Adcock Ltd organisation. The stakeholders affected by the organisation attributes surrounds the organisation. Concerns are identified alongside each stakeholder.

Figure 10-6: Organisation – Adcock Ltd. organisation requirements model

10.3 Functional analysis

The functional modelling of the SHM actually started by identifying the functions of each individual component of the SHM as illustrated in Figure 10-7. The functions were then grouped into interface, rotation, translation, end-stop, pre-load functions. These groups of functions were then regrouped into functions of higher level of abstraction. The functional decomposition of the SHM is shown in Figures 10-8 and 10-9. Appendix E, Section E.2 shows lower level decompositions, completing the product functional model.
Figure 10-7: Functions of the SHM individual components

Figure 10-8: Product - Functional context diagram of the SHM
Figure 10-9: Product – Functional decomposition of the SHM

Figure 10-10 shows a Behaviour Diagram representing a top-level functional model of the SHM life cycle processes. Differently from the product functional model, the process functional model was developed from top to bottom modelling a theoretical process. The emphasis of the process functional model was the modelling of the production process and its manufacturing and assembly sub-processes. It can be observed in Figure 10-10 that the purchase of raw material and bought in parts were considered as processes outside the production process. Figure 10-11 decomposes the production process into a generic set of manufacturing and assembly processes.
Figure 10-10: Process – the SIHM life cycle processes top level functional model
Figures 10-12 and 10-13 illustrate theoretical, generic or textbook-like process models of the manufacturing and assembly processes. Firstly, the mainstream set of activities was generated, e.g.: select, position, process. Secondly, verification, validation and review processes were added. Thirdly, exception handling were dealt with. A more complete process functional model can be found in Appendix E, Section E.3.

Figure 10-14 shows a Use Case Diagram (UCD) representing a functional model of the H.R. Adcock Ltd. organisation. Please notice that the business processes in Figure 10-6 are decomposed into the use cases shown in Figure 10-14: develop_SHM, produce_SHM, sell, deliver, purchase, manage organisation. Figure 10-14 shows the exchange of information, material and energy between stakeholders in the environment and the use cases in the organisation.
Figure 10-12: Process – The manufacturing_part process functional model
Figure 10-13: Process – the Assemble_SHM process functional model
Figures 10-15, 10-16 and 10-17 illustrate different criteria for decomposing the use cases in Figure 10-14. The use case chosen for decomposition is the 'Produce_SHM'. Figure 10-15 decomposes the use case 'Produce_SHM' into sub-functions, such as manufacturing and assembly. Figure 10-16 decomposes the use case 'Manufacture_parts' into possible scenarios such as: normal operation, late arrival of raw material, late arrival of consumables, processing equipment breakdown. Figure 10-17 decomposes the use case 'Manufacture_parts' using as criteria the parts to be manufactured, for example: manufacture_spindle, manufacture_bobbin, manufacture_ferrule, manufacture_tube. Although Figure 10-17 only shows the interactions between stakeholders and the 'Manufacture_spindle' process, those relationships are valid for the other use-cases shown.

Figure 10-18 and 10-19 decompose the use case 'Manufacture_tube' emphasising sequence and flow, respectively. At this functional analysis stage, it is not intended to model business or organisation processes as they currently are, but to model ideal processes. Figures 10-18 and 10-19 serve the purpose of exemplifying the organisation functional analysis model proposed in Chapter 7, Section 7.5.2.
Figure 10-15 – Organisation – the 'Produce_SHM' function decomposition

Figure 10-16: Organisation – the 'Manufacture_parts' function scenarios
Figure 10-17: Organisation – other 'Manufacture_parts' function scenarios

Figure 10-18: Organisation – Sequence Diagram for modelling the Manufacturing_tube function
10.4 Physical analysis

Figure 10-20 is a product physical context diagram of the SHM. It is a DFD showing the physical connections between the SHM and the elements in the environment which are physically connected to the SHM, such as: handle mechanism and seat frame.
Figure 10-20: Product—the SHM physical context diagram

Figure 10-21 is a Function Block Diagram (FBD) representing the physical decomposition of the SHM into its physical parts and the physical connections among them. The diagram can be called an 'architecture interconnect diagram', using the terminology adopted by [Hatley & Pirbhai, 1988].

Figure 10-22 is a Data Flow Diagram (DFD) representing an expansion of the functional context diagram shown in Figure 10-8. Differently from the functional decomposition shown in Figure 10-9 that shows functions and flows, Figure 10-22 shows the actual physical components and their exchange of material, energy and/or information. Again, following [Hatley & Pirbhai, 1988], Figure 10-22 can be seen as an 'architecture flow diagram'.

Figure 10-23 shows a Behaviour Diagram (BD) representing the actual manufacturing and assembly processes to produce one SHM. The processes in Figure 10-23 are further expanded in order to show each process in greater detail. A complete process physical model can be found in Appendix E, Section E.4. For example, not all activities represented in Figure 10-23 have a complete set of ICOM (Input, Control, Output, Mechanism) symbols. However, the lowest level diagrams contain those symbols.
Figure 10-21: Product – SHM’s architecture interconnect diagram

Figure 10-22: Product – SHM’s architecture flow diagram
Figure 10-23: Process – the SHM manufacturing and assembly process
Figure 10-24 shows the same use case diagram shown in Figure 10-14. In the organisation functional analysis the emphasis was in understanding the organisation functions represented by the use-cases. In the organisation physical analysis the emphasis is on identifying and modelling the resources that constitute the organisation, in this case the H.R Adcock Ltd. The organisation is represented in Figure 10-24 by the big central rectangle. Again, emphasis will be given to the resources used for the SHM production.

Figure 10-24: Organisation – the H_R_Adock limited organisation Use Case Model

Figure 10-25 is a Package Diagram showing the departments of the H.R.Adock Ltd organisation. There is no formal division between engineering and manufacturing but in practice those were the main divisions identified in the organisation. Figure 10-25 suggests that a hierarchy of Package Diagrams can model an organisation structure.

The Package Diagram can also be used to provide different views of a given entity. In Figure 10-26, a Package Diagram is used to initiate the development of two views of the 'manufacturing' organisation: the processes and the resources views. Another possible view could be the information view (see [Vernadat, 1996] for details). Again focus is given to the resources and process views as the main sources of organisation physical attributes.

Figure 10-27 is an expansion of the 'resources' symbol in Figure 10-26. It depicts a Class Diagram that lists and organises the resources. Relationships between different types of resources represented by the classes are also identified. Resources are initially divided into human and capital resources. Capital resources include stock and processing equipment. Processing equipment include all machinery in the shop floor such as lathes, tapping machine, thread roller.
Figure 10-25: Organisation – Package Diagram showing organisation departments

Figure 10-26: Organisation – Package Diagram showing different views of the manufacturing department
Figure 10-27: Organisation – a Class Diagram representing a resources model
Each resource can have its behaviour modelled by a Statechart Diagram (SCD) and by an Activity Diagram (ACD). The SCD models how the resource or class of resources responds to external stimuli. Figure 10-28 shows an example of an SCD of a tube-lathe. Figure 10-28 considers two super-states of a tube-lathe: ON and OFF. For the ON superstate, for example, the tube-lathe can be ON waiting or processing. It also could be OFF because of maintenance, calibration or because it was simply switched off. The SCD in Figure 10-28 shows the states and superstates mentioned above and the conditions for transitions between states.

![Statechart Diagram of a tube-lathe](image)

**Figure 10-28: Organisation — a Statechart Diagram describing the behaviour of a resource 'Tube_Lathe'**

The ACD depicts a set of component activities that are required to accomplish the behaviour of a resource or class of resources. Figure 10-29 shows an example of an ACD modelling the resource 'Tube_Lathe'.

Figure 10-30 illustrates a Component Diagram (CPD) expanding the 'Processes' view in Figure 10-26. The CPD shows a hierarchy of tasks in order to accomplish the use case Produce_SHM or the manufacturing activity.
Figure 10-29: Organisation - an Activity Diagram describing operations performed by a resource 'Tube_Lathe'

Figure 10-30: Organisation - a Component Diagram describing a hierarchy of manufacturing processes

Figures 10-31 and 10-32 show the various resources in the SHM manufacturing plant and how they interrelate.
Figure 10-31 is a Collaboration Diagram that can be expanded from the ‘Resources’ package in Figure 10-26. The diagram shows mainly the flow of work in process in the plant.

Figure 10-32 is a Deployment Diagram (DPD) illustrating how the resources are physically connected, i.e., how the flow of work-in-process can be implemented. It can be observed from Figure 10-32 that the main connection between machines is performed by an operator who has to transport work-in-progress from machine to machine.

The diagrams in Figures 10-31 and 10-32 for organisation physical modelling are analogous to the ‘architecture flow diagram’ (see Figure 10-22) and the ‘architecture interconnect diagram’ (see Figure 10-21), respectively, for product physical modelling.
10.5 Requirements, attributes and relationships

From source documents and from the analysis models, requirements and attributes are captured. For requirements, the models provide who are potential sources of requirements and how to classify requirements by concerns. For attributes, the models provide the elements that will be described by the attributes. Once requirements and attributes are listed, the relationships among them are identified.

Appendix E, Section E.5 contains a Cradle report showing a list of the requirements identified for the SHM. Figure 10-33 illustrates links between some of the requirements and the stakeholders identified during the requirements analysis process.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Stakeholders</th>
</tr>
</thead>
<tbody>
<tr>
<td>z SHMs per yd</td>
<td>23.2</td>
</tr>
<tr>
<td>Minimise weight</td>
<td>23.2</td>
</tr>
<tr>
<td>Minimise cost</td>
<td>23.2</td>
</tr>
<tr>
<td>Minimise risk of involuntary lowering</td>
<td>23.2</td>
</tr>
<tr>
<td>Minimise risk of seat not going up</td>
<td>23.2</td>
</tr>
<tr>
<td>Minimise risk of seat not going down</td>
<td>23.2</td>
</tr>
<tr>
<td>Geometric constraints</td>
<td></td>
</tr>
<tr>
<td>Front bumper bush diameter 12.0 to 12.8 mm</td>
<td>23.2</td>
</tr>
<tr>
<td>Front bumper bush diameter 25.3 to 25.8 mm</td>
<td>23.2</td>
</tr>
<tr>
<td>Rear lighting hole diameter 8.04 to 9.34 mm</td>
<td>23.2</td>
</tr>
<tr>
<td>Rear lighting slot width 3.1 mm minimum</td>
<td>23.2</td>
</tr>
<tr>
<td>Op/centre dimension over front and rear location 236.8 mm</td>
<td>23.2</td>
</tr>
<tr>
<td>Min thickness M30 x 6.00</td>
<td>23.2</td>
</tr>
<tr>
<td>Min depth 12.2 mm minimum</td>
<td>23.2</td>
</tr>
<tr>
<td>Functional constraints</td>
<td></td>
</tr>
<tr>
<td>Stroke-stop to stop 54.3 mm</td>
<td>23.2</td>
</tr>
<tr>
<td>Torque with pre-load 2.5 Nm maximum</td>
<td>23.2</td>
</tr>
<tr>
<td>Handle interface—sheet metal for handling</td>
<td>23.2</td>
</tr>
<tr>
<td>Design for a minimum number of parts</td>
<td>23.2</td>
</tr>
<tr>
<td>Minimise handling: design for handling presentation</td>
<td>23.2</td>
</tr>
<tr>
<td>Minimise subassemblies</td>
<td>23.2</td>
</tr>
<tr>
<td>Emphasise standardisation</td>
<td>23.2</td>
</tr>
<tr>
<td>Undertake engineering change in batches</td>
<td>23.2</td>
</tr>
<tr>
<td>Provides for personal driving position is fully adjustable</td>
<td>23.2</td>
</tr>
<tr>
<td>Can personalise seat, head rest and arm rest position</td>
<td>23.2</td>
</tr>
<tr>
<td>Easy to see, understand reach and operate controls and displays</td>
<td>23.2</td>
</tr>
<tr>
<td>Reachable from driving position</td>
<td>23.2</td>
</tr>
<tr>
<td>Controls are oriented properly for natural hand movement</td>
<td>23.2</td>
</tr>
<tr>
<td>Low effort</td>
<td>23.2</td>
</tr>
<tr>
<td>Low effort</td>
<td>23.2</td>
</tr>
<tr>
<td>Controls provide quality feel and sound when operated</td>
<td>23.2</td>
</tr>
<tr>
<td>Smooth, precise operation</td>
<td>23.2</td>
</tr>
<tr>
<td>Excellent visibility around</td>
<td>23.2</td>
</tr>
<tr>
<td>Command, sealing position</td>
<td>23.2</td>
</tr>
<tr>
<td>Level 1</td>
<td>23.2</td>
</tr>
<tr>
<td>Level 2</td>
<td>23.2</td>
</tr>
<tr>
<td>Level 3</td>
<td>23.2</td>
</tr>
</tbody>
</table>

Legend: 1 - indicates that stakeholder is source of requirements; empty cell — otherwise

Figure 10-33: Requirements versus stakeholders matrix

Functional attributes are derived from the models developed during the functional analysis process. Figure 10-34, 10-35 and 10-36 provides examples of generation of product functional attributes from product, process and organisation functional models, respectively. The functional attributes are associated to elements of the models, e.g., functions. Appendix E, Section E.6 contains a more complete set of functional models with a draft set of functional attributes.
Figure 10-34: Example of derivation of product functional attributes

- grip interface force resistance
- energy transfer efficiency
- frame displacement versus rotational handling effort
- angular displacement versus frame displacement

Figure 10-35: Example of derivation of process functional attributes

Legend:
- candidate functional attributes
- number of parts
- quantity
- quality
- number of sub-assemblies
- time
- cost
- quality
- number of level of subassemblies
- depends on levels of subassemblies
- number of subassemblies
- candidate functional attributes
The functional attributes are then organised in a hierarchy corresponding to the hierarchy of functions they relate to. Figure 10-37 shows a matrix relating requirements and functional attributes. The functional attributes obtained are depicted in the top rows of the matrix. Notice that they are organised as a hierarchy following the hierarchy of functions in the functional models. The relationships between the functional attributes in a hierarchy may have one of the following meanings:

- Higher level functional attributes 'are obtained from' lower level ones;
- Higher level functional attributes 'depends on', 'is affected by' lower level ones;
- Higher level functional attributes 'are a function of' lower level ones.

For example, consider the product functional attribute ‘SHM - Effort versus seat position characteristic curve’. It can be calculated by the composition of three lower level attributes: '45 - Force-to-frame x frame displacement'; '44- angular displacement x frame displacement', '47 - frame displacement x rotational handling effort'. (The number refers to the corresponding function identification in the functional model). The '44-angular displacement x frame displacement' attribute is, by its turn, affected by, 'rotational energy loss' and 'angle versus displacement characteristic curve'.

The choice of process functional attributes focused on attributes that would affect the 'Production time' of one sample of an SHM. Again, it follows the process functional model hierarchy. According to Figure 10-37, in the hierarchy of functional attributes, 'Production time' is a function of 'average waiting time for raw material arrival', 'mean time between failure of manufacturing equipment', 'time to manufacture the longest lead time part'. This last one, by its turn, is a function of 'number of parts requested for that [sub] assembly' and 'time to manufacture part unit', for example.
The organisation functional attributes in Figure 10-37 are presented in only one level, as they were all extracted from the top level UCD shown in Figure 10-36. The focus for obtaining organisation functional attributes was the production organisation related attributes such as: balance, utilisation, productivity. The production organisation has a basic function of producing a specified number of SHMs in a given period of time for a calculated cost.

The relationships between requirements and functional attributes intend to identify what functional attribute implements a given requirement. The functional attribute is engineered to meet one or more requirements. The existence of a link between requirements and functional attributes indicates that a change in the functional attribute may affect the requirement and transitorily the stakeholder satisfaction. On the other hand if it is necessary to focus in a given requirement to improve stakeholder satisfaction, a matrix such as the one presented in Figure 10-37 shows where to focus in the concept of product, process and organisation to achieve that goal.

It can be observed, in Figure 10-37, that the requirements from the user such as ‘low effort’, ‘smooth and precise operation’ and functional constraints map onto product functional requirements. Geometric constraints found no correspondence to the list of functional attributes provided. A way of verifying that a geometric constraint is being considered is mapping then directly to physical attributes. Another way is having attributes of an interface function such as ‘compliance to constraints’. These would then be related to physical attributes.

Typical life cycle process requirements such as ‘minimise set-up and interventions’, ‘design for a minimum number of parts’, relate to the process functional attributes. Also requirements such as schedule and cost are also affected by process function attribute as for example, ‘raw material waiting time’ may affect schedule. The time to produce one SHM affect cost because of, for example, the labour involved in the activity.

Quantity, schedule and cost requirements are typical organisation requirements because they will determine the amount of resources that has to be allocated. That is the reason why most of the organisation functional attributes are related to those requirements.

Figure 10-38 presents a self-interaction matrix of functional attributes. The relationships in the matrix indicate correlation between functional attributes. The existence of a relationship indicates that changing one attribute the other one related to it will change. The relationship is bi-directional. This is the reason for only half matrix to be presented. The matrix does not tell whether the correlation is positive or negative or whether it is strong or weak. It just indicates the existence or not of a correlation between two functional attributes.

The main diagonal of the matrix is filled with black to indicate the default correlation between identical functional attributes. A desired characteristic of attributes, as mentioned in Chapter 6, Section 6.4.4, is to be functionally independent. In such an ideal situation only the main diagonal of the matrix would be filled in. A matrix, such as the one in Figure 10-38, can be used to evaluate the quality of a set of functional attributes regarding functional independence.
It can be observed in Figure 10-38 that product functional attributes are correlated only to other product functional attributes. They are not correlated to process or organisation functional attributes. This is expected to be true for the conceptual stage of product development when no design consideration is yet formulated. At the design stage, the interactions between product, process and organisation attributes become clearer and can be used to infer the existence of relationships among functional attributes. As only time related functional attributes were captured for process,

Physical attributes are derived from the product, process and organisation physical models. The approach adopted for capturing physical attributes was:

- For product: key physical characteristics of each part of the SHM were captured. It was intended to capture at least one characteristic per part. Examples are: 'bobbin shoulder length', 'spring material', 'body half shape';
- For process: as, for functional attributes, the focus was on time attributes. The 'time to produce one SHM' was then decomposed using the model in Figure 10-23. The total time to produce one SHM can be calculated using two basic operators: MAX operator that delivers the maximum value in a set of attributes and + which delivers the sum of two attribute values. The MAX operator can be used for concurrent activities and the + for sequential ones. Expansions of the model (see Appendix E, Section E.4) were used to derive the attributes of the 'Assemble_assembly_3' and 'Manufacture_tube' processes.
- For organisation: the basic source of attributes was the class diagram, presented in Figure 10-27 because it listed the resources used. Examples of organisation physical attributes are: number of operators, number of machine types, number of tool types. As it was chosen not to go beyond the first level of attributes, the expansions of a given class as exemplified in Figures 10-28 and 10-29 were not used. Functional attributes such as ‘balance’, ‘capacity’, ‘utilisation’ can have their physical attributes counterpart derived from diagrams such as the ones shown in Figures 10-31 and 10-32. An example of a physical attribute that can be derived from Figure 10-32 is 'the size of the basket to transport work-in-progress'.

Like with the functional attributes, physical attributes were captured in a hierarchy format indicating either that a higher level attributes can be calculated from lower level ones or that a higher level subsystem attributes can be obtained from lower level ones.

Figure 10-39 illustrates the matrix that captures the relationships between functional attributes and physical attributes. Physical attributes can be found at the top of the matrix, on the header of each column.

The relationships captured in Figure 10-39 indicate that a given physical attribute implements a functional attribute. The functional attribute is a function of the physical attribute or is affected by it in any way.

Figure 10-39 shows that the physical attributes tend to impact functional attributes of the same type. Thus product physical attributes tend to impact product functional attributes, process physical attributes tend to impact process functional attributes and organisation physical attributes tend to impact
organisation functional attributes. An exception to this trend is the existence of relationships between process and organisation attributes. Some organisation functional attributes, such as qualification of operators, characteristics of a given type of machine and characteristics of a given type of tool, can impact the production time of one SHM (top-level process functional attribute). Care must be taken when choosing process and organisation attributes in order to assure their independence (as much as possible) and to assure that they are representing distinctively process and organisation objectives. A process is related to the description of the life cycle of a product. An organisation is related to the resources used to implement a given life cycle process. The key attribute for process is lead-time. The key attribute for organisation is productivity.

Figure 10-39 shows that for the relationships with process physical attributes ('time 9- time to produce one SHM') only 'Assemble_assembly_3 time' and 'Manufacture_tube_time' were considered.

Figure 10-40 shows the matrix that captures the relationships between functional attributes and physical attributes. These relationships indicate whether the physical attributes correlate or not. When they correlate, a change in one attribute will cause a change in the related one.

Differently from all other previous matrices, the matrix in Figure 10-40 shows that, at the design level, there are relationships among physical attributes of different types. It can be seen on the matrix that there are relationships between product and process physical attributes, product and organisation physical attributes and, process and organisation physical attributes.

Examples of relationships between product and process physical attributes are those relating assembly time with the symmetry, shape, size of a given part. For example, time to assemble assembly_5 is correlated to front trunnion bush diameters. A complete reference about those type of relationships is provided by [Boothroyd, Dewhurst & Knight, 1994] on product design for manufacture and assembly.

Examples of relationships between product and organisation physical attributes exist because a characteristic of a part may require an operator with special qualification, a machine with a special capability or a tool with a particular peculiarity. For example, the bobbin shoulder length may determine the qualification of the operator or the characteristic of a given machine or the characteristic of a given tool.

Examples of relationships between process and organisation physical attributes because the resources characteristic may affect the time to produce one SHM. Examples of these resource attributes are again machine type, qualification of operator, etc. Undoubtedly they affect both lead-time and productivity.

According to Figure 10-40, at the physical, design or implementation level, there are relationships among physical attributes of product, process and organisation. Figure 10-39 illustrated the relationships between functional and physical attributes. Therefore, transitively, it can be said that there are relationships among functional attributes of different types – product, process and organisation. The fact that product, process and organisation are coupled at the design level makes them coupled at the conceptual level as well. As a consequence, changing a physical attribute considering only the functional attribute directly linked to each may cause the need for further iterations of the development process. If all functional
Represent that a functional attribute on a column impacts the requirement on the row. The requirement is met through the engineering of that functional attribute.

<table>
<thead>
<tr>
<th>Functional Attribute</th>
<th>Requirement 1</th>
<th>Requirement 2</th>
<th>Requirement 3</th>
<th>Requirement 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute 1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute 2</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute 3</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Attribute 4</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Figure 10-37: Requirements versus functional attributes matrix for the SHM
There is no correlation between the functional attributes on the rows and columns. Comparing the matrices may help in the analysis of the attributes as well.

There is no correlation between the functional attributes on the rows and columns.

Figure 10-38: The functional attributes self-interaction matrix for the SHM
attributes affected can be considered beforehand, the solution can be thought from the conceptual or functional level and can be done in a more structured fashion. This is very different from the traditional approach, adopted in the automotive industry, for example, of changing at the design level and adapting for coping with unexpected behaviour. This traditional approach, as shown in Chapter 4, is likely to lead to increased complexity. The structured approach is a way of managing complexity.

10.6 Lessons learned

The SHM example was developed to demonstrate the applicability of the ‘total view framework’ for a hardware product. Attributes are extracted from the product, process and organisation models and are used to describe product, process and organisation. In a typical mechanical engineering environment product models could be CAD drawings. Those too can be used for generating attributes. Attributes, like models or drawings, are a description means. Differently from models, drawings or specifications, attributes have granularity. They are much more specific and they provide a universal language that can be used to describe not only product but also process and organisations. As such, capturing the relationships between attributes can be a way of integrating product, process and organisation.

Traditional structured analysis from where the ‘concurrent structured analysis method’ was developed is in most of the applications used for software development. For software development, specifications are generated as a result of the analysis process. Specifications are in general codified algorithms that represent the lowest level description of a function in the software. They may include, however, attributes such as memory, execution time, synchronisation. For hardware, one specification, however, would contain almost exclusively attributes. These attributes individually may affect many others in different specifications.

The SHM example illustrated the use of attributes for describing a hardware product and their usefulness in providing integration between product, process and organisation.

The following are a list of some lessons learned from the SHM example.

Requirements

- Requirements available were, essentially, a list of physical attributes to be found in the final product and the quantity, schedule and cost required.
- The use of the requirements model allowed the identification of stakeholders and concerns before the actual requirements capture. Stakeholders are the sources of requirements and concerns are the way they can be classified. Also concerns can be used as measures of effectiveness when evaluating the final result of the development effort.

Requirements versus functional attributes relationship pattern

- Product functional attributes tended to address functional and user requirements;
- Process functional attributes tended to address schedule and life cycle process requirements;
- Organisation functional attributes tended to address quantity, schedule and cost attributes.

Functional attributes self-interaction pattern

Direct relationships among functional attributes tend to be kept within each particular type of attributes — product, process and organisation.
Functional attributes versus physical attributes relationship pattern

Direct relationships between functional and physical attributes tend to be kept within each particular type of attributes – product, process and organisation.

Physical attributes self-interaction pattern

There are relationships of physical attributes of different types. Product attribute are correlated with process and organisation attributes. Process attributes are correlated to organisation attributes. This is an expected result from the many publications on concurrent engineering.

Transitive relationships

As physical attributes of product, process and organisations are interrelated, it can be inferred, by transitivity, that their functional attributes are also correlated. These functional attributes would then be coupled by design.

Modelling for attributes

The most useful models for the sake of generating attributes were those that contained all elements the attributes should describe. For example: DFDs in the product functional model; FBDs in the product physical model; BD in the process models; UCD in the organisation functional model and CD in the organisation physical model. DFD contains product functions. FBD contains product components. BD contains processes, activities or tasks. UCD contains organisation functions. CD contains organisation resources.

Process versus organisation modelling

Care must be taken for avoiding confusion and duplication of work when modelling process and organisation. For example, when developing the Sequence Diagram in the organisation functional model the objective is to identify generic resources (e.g. manager, driller) whose roles would then be fulfilled by the resources in the organisation physical model. The process physical model has the objective of identifying the actual sequence of activities that is performed. The process models describe the life cycle processes of a sample of a product. The organisation models describe the resources that must be used to perform that life cycle process not necessarily for one sample of a product but potentially for many products of that kind. The process models are used for timing attributes. The organisation models are used for productivity attributes. In the 'total view framework', all product life cycle processes must be modelled, but only some organisations of interest will be modelled.

Sequence versus concurrency

Although the way the example was developed and presented may imply a sequence, the product, process and organisation models can and should be developed concurrently. Differently from this example that was developed by two people, the actual modelling activity must be performed by a team of specialists, each modelling his own speciality related to product, process and organisation.

Quality of models and quality of requirements and attributes

The quality of models is determined by the quality of the information available about the object to be modelled. The quality of models determines the quality of the requirements and attributes obtained. The more complete the list of stakeholders identified in the requirements models, the more complete the list of requirements will be. The hierarchy of attributes, the level of detail they cover, their independence are functions of the models developed.
The need for a systematic approach for generating attributes
As presented in the SHM example, attributes are obtained just by choosing some parameters to describe an element of a model. There is no systematic approach that regulates the choice. What would be a better set of attributes? How to obtain a set of attributes with the desired characteristics outlined in Chapter 6, Section 6.4.4? It is necessary to develop a work, just it has been recently done for requirements, for attributes as well. Attributes are important elements in a product evolution.

Quality of relationships dependent upon requirements and attributes quality
If attributes are chosen in such a way they are very independent, the relationships will not include dependencies among attributes of the same type. In such a case focus will be given to relationships among attributes of different types - product, process, organisation.

Big size of resulting matrices despite simplifications
The SHM is a relatively simple product if compared with a car. Even though, many simplifications had to be made for the elaboration of this example. Simplifications included the following:
• only the production life cycle process was expanded in the modelling and had attributes captured;
• for capturing lower level process physical attributes only one assembly process (out of six) and one manufacturing process (out of six) were considered.
• only the production organisation was expanded in the modelling and had attributes captured;
• only one example of resource, a machine was modelled;
• organisation attributes were captured at the top level only.

Despite the simplifications, the relationship matrices were of a large order. This suggests that clustering may have to precede the matrix presentation. A big matrix can be presented just to show inter-cluster relationships and small matrix clusters used for relationship analysis. The great amount of work necessary to follow the concurrent structured approach suggests that a company adopting it must cope with the high overhead to be incurred. This is certainly not the case of the SHM developer and producer. But it can be the case of complex product developers such as Ford, BAe, for example.

The SHM example, together with the PCS examples presented in Chapter 9, demonstrated a way of obtaining a total view with commercially available tools. However, they also demonstrated how complicated and lengthy the process for obtaining requirements, attributes and relationships can be. It can be concluded from these examples the need to adapt the ‘total view framework’, and its fundamental concepts, to the constraints of a real industrial environment. Although the ‘total’ is desirable and it was demonstrated how to get there, the examples suggest that difficulties may arise when trying to obtain the total view within real industry timescale and budget. The challenge remains for a more pragmatic approach that still uses the concepts stressed in this thesis. This is object for further work.

In the light, also, of the experience gained in implementing the ‘total view framework’ and ‘concurrent structured analysis method’ through the examples developed in the last three chapters, the next chapter puts together the issues discussed in this thesis. Besides it evaluates the framework and method against needs, trends and existing approaches for coping with complexity in the integrated development of complex products.
Part V
Evaluation
Chapter 11
Evaluation and discussion

This chapter aims:

• to draw together the topics covering the aims of this thesis – framework (Chapters 5 and 6), method (Chapter 7) and implementation (Chapters 8, 9 and 10);
• to evaluate the framework and method against the trends in the automotive and space industries. The trends were highlighted in Chapter 3.
• to evaluate the framework and method against the needs identified during the gasoline and diesel PCS case studies. The case studies are reported in Chapter 4.
• to compare the framework and method with general approaches to manage complexity presented in Chapter 2.
• to compare the framework and method with integrated development, concurrent engineering and systems engineering. These concepts were reviewed in Chapter 2.
• to perform a critical analysis of the thesis.

11.1 Introduction

The topics covering the aims of this thesis – framework, method and implementation - are grouped in three broad sections, Sections 11.2, 11.3 and 11.4, respectively. The evaluation of the framework and method against trends identified in the automotive and space industries is presented in Section 11.5. The evaluation of the framework and method against the needs identified during the gasoline and diesel PCS case studies is presented in Section 11.6. In Section 11.7, the framework and method are compared with general approaches to manage complexity. Sections 11.8, 11.9 and 11.10 compare and identify the relationships of the framework and method with integrated development, concurrent engineering and systems engineering, respectively. Section 11.11 performs a critical analysis of the thesis: delineating its scope, stating what has been addressed and what has not; highlighting where there is room for improvement and; emphasising its contribution.

11.2 The ‘total view framework’

In this section, the needs of complex product development identified in this thesis are presented. The concepts embedded in the ‘total view framework’ addressing the needs are outlined.

11.2.1 Needs

The need for continuous changes in product, process and organisation in order to maintain manufacturing business competitiveness, if not appropriately managed, may cause complexity to escalate unnecessarily and with it cost, lead-time, and effort increase. This fact is even more dramatic for products already intrinsically complex such as modern automobiles, aeroplanes and satellites.

In Chapter 2, complex products are defined as:

• implementing many different functions with multiple modes of operation;
• having multidisciplinary technology implementation;
• being highly integrated (tightly coupled);
• having strong coupling between independently specified objectives of the product (e.g. weight, quality, cost, performance);
• having multi-phased life cycle processes (e.g., manufacturing, integration, testing);
• needing a very large, focused, labour-intensive, multi-skilled and multidisciplinary workforce.

Warfield (1976) states that complexity implies that one individual can, at best, do no more than contribute to a solution. Therefore one condition to prevent complexity escalation is to have a team of knowledgeable individuals, who together can encompass all aspects of the problem domain. Teams, in the product development arena, are at the core of the integrated development approach, whatever name it assumes (IPD, IPPD, IPPED, etc.).

The move towards integrated development was observed, in Chapter 3, when analysing the trends in terms of product development approach used by two industries of complex products — the space and the automotive industries.

The space industry is being hit by shrinking public funding and, at the same time, increasing commercial interest. The product development, traditionally driven by performance and risk avoidance, is now taking into consideration life cycle cost, development lead-time and risk management, besides performance.

The traditional document-driven systems engineering process is giving place to a more trade-off driven approach. The rigorously sequentially phased development process with many reviews is incorporating some principles of concurrent engineering.

The automotive industry is in a business environment of increasing competitiveness, increasing customer awareness, increasing variety and tightening of requirements. This is leading to an increasing complexity of the product which the traditional component approach cannot cope with. Concurrent engineering was already being used by the automotive industry and now it is being part of a move towards systems engineering. Chapter 3 describes FPDS (Ford Product Development System) as an example of this trend.

Concurrent engineering was used as an enhancement to the traditional component approach. Many examples can be found in the literature of concurrent engineering techniques being used to enhance manufacturing and assembly of automotive components. Under systems engineering, the focus is on the total vehicle system. The car manufacturer is starting to see the vehicle as a way of meeting customer needs and those needs as the main engineering driver. Traditionally, the car has been seen as an assembly of components. There was the belief that the optimisation of the parts would lead to the optimisation of the whole vehicle.

Concurrent engineering and systems engineering are ingredients of integrated development, according to the DoD regulation 5000.2-R which defines IPPD (Integrated Product and Process Development) [DoD, 1996].

The scope of integrated development has been ever increasing since IPD [Andreasen & Hein, 1987] including product, process, enterprise, etc. as shown in Chapter 2.

The need for integrated development and its scope was also indicated by the needs identified in the gasoline and diesel PCS case studies presented in Chapter 4.
The gasoline PCS case study analysed the evolution of the gasoline PCS, especially its application software, over the period 1978-1995 in terms of stakeholders, requirements, functions, components, development process and organisation. The case study shows that the number of stakeholders increased, the variety and tightening of requirements increased, the number of functions jumped from 3 to the order of tens and the variety of components increased. The software development approach was to modify existing versions of the software in a reactive manner without investigation of change effects to the PCS as a whole or to upper level systems. If the vehicle passed in the exhaust emissions test, the application software would be approved. The software, as a consequence, grew in complexity in various aspects. It implemented many more functions. It became very expensive and difficult to maintain as very little structure survived in the final code. The variety of software versions increased dramatically to implement all required applications. In 1995, the gasoline PCS developers decided to produce one only piece of application software code that could be easily modified to implement any future application. Also the development process started to include versioning and bookshelving processes. The development organisation was modified to include algorithm and software developers in the same functional group. Those measures, especially the generic code, were an attempt to improve reuse, ease of calibration and development time as the complexity of the gasoline PCS had reached a critical level. The gasoline PCS case study highlights some needs to manage complexity while developing a complex product:

- maintain traceability from stakeholders to their requirements, from these to functions, from these to implementation elements, so that a change in one of them can be quickly responded by the others;
- consider not only the product as the solution of the complex problem but also its life cycle processes and their performing organisations;
- keep visibility of the interactions or relationships among the various elements of the solution implementation, including not only the product elements but also the process and organisation elements.

The diesel PCS was first developed in 1994 using a structured approach. The approach included the use of a systems engineering process, a structured analysis technique, computer tools for early requirements capture and analysis, functional analysis and physical analysis, configuration management, co-located team. The result was a code much easier to maintain. On average, a diesel application can be developed in half of the time with half of the people than a gasoline one with the same level of functionality. The diesel PCS met the requirements of functionality, execution time, memory, ease of calibration, reuse and shorter development time. Those requirements were not met by the application software alone but also by the structured analysis process, by a calibration guide, by a co-located team, each addressing a different requirement. The diesel PCS provided an example where the system solution was not only composed by the product, but also by its life cycle processes (development process) and their performing organisation (co-located team).

The needs identified from the trends in the space and automotive industries and from the case studies can be summarised in three main issues:

- **scope** - the need to identify up-front in the integrated development of a complex product what is the scope of the system solution delivered by an integrated development process;
• integration – the need to integrate requirements, functions and implementation elements throughout the integrated development process and to have an integrated solution;

• complexity management – the need to manage complexity throughout the integrated development process considering that the system solution will be in a very dynamic environment and, as a consequence, will be highly likely to change.

11.2.2 Scope

The trends of product development in the automotive and space industries point towards integrated development using systems engineering and concurrent engineering approach. The case studies suggest the need to include product, process and organisation in the scope of the system solution. This evidence provides an indication of what should be included in the scope of an integrated development framework.

The scope of the system solution resulting from the integrated development effort encompasses the product, its life cycle processes and [some of] their performing organisations. The effort of integrated development ends up producing not only the product but also its life cycle processes and some of their performing organisations. Product life cycle processes and their performing organisations constrain, or are constrained by, product development in such a way that they had better be considered as part of the resulting product development effort from the outset. Based on this assumption a generic model for integrated development was proposed in Chapter 5.

The generic model considers also a time dimension as it recognises that the solution is likely to change over time to accommodate to an ever increasingly dynamic market environment. Therefore, the generic model serves not only for modelling the development of a new product but also of an evolving existing product. According to the model (see Figure 5.1 in Chapter 5):

• Stakeholders have needs. Some of these needs are perceived and considered as requirements by the product development organisation. (In this thesis, 'product development organisation' was also called 'development agency' to refer to the fact that, for complex products, it is unlikely that there is only one organisation developing all product parts, components and subsystems. Therefore the term 'development agency' may also refer to tier suppliers);

• Based on the requirements, the development agency develops the product, its life cycle processes and [some of] their performing organisations. (The scope of the development process, in terms of the life cycle process performing organisation, may vary from industry to industry. In the automotive industry, some life cycle process performing organisations such as scrappers, dealers and service stations, are not generally considered as part of the vehicle development effort. They should be considered as stakeholders. In the space industry, a satellite development may include all satellite life cycle performing organisations.);

• Product, process and organisation have attributes;

• Those attributes affect stakeholder satisfaction or dissatisfaction;

• The cycle restarts as stakeholders have new requirements or as other stakeholder needs are perceived by the product development organisation.

The generic model proposes that:
• the integrated development of complex products must be a requirements driven process;
• the integrated development of complex products integrates product, life cycle processes and associated organisations (or at least the development organisation) development;
• the integrated development of complex products searches for a balanced solution trading-off product, processes and organisation attributes as the emergent properties of a whole integrated system.

In order to move from requirements to attributes in the integrated development model, this thesis proposes a ‘total view framework’. The ‘total view framework’ defines the elements within the scope of the integrated development process.

The ‘total view framework’ consists of the application of the systems engineering process concurrently to a product, its life cycle processes and [some of] their performing organisations at all layers of the product breakdown structure.

The elements of the ‘total view framework’ are placed on three dimensions: the integration dimension, the analysis dimension and the structure dimension. The integration dimension contains product, process and organisation elements to be integrated. The analysis dimension contains the core sub-processes of the systems engineering process [IEEE-1220-Std-1994]: requirements analysis, functional analysis and physical analysis. The structure dimension refers to the hierarchy layers of the complex product breakdown structure (see Figure 5-2 in Chapter 5).

The dimensions of the ‘total view framework’, analysis, integration and structure, are set to manage the three factors, pointed by [Hitchins, 1996], through which complexity emerges: variety, connectedness and disorder, respectively. Each framework dimension addresses mainly one complexity factor. Analysis refers to a separation of the whole into its component elements, an examination of these elements and their relationships, and a follow-on decision relative to a future course of action [Blanchard, 1998]. Integration means putting together heterogeneous elements to form a synergistic whole [Vernadat, 1996]. Structure is the way a set of elements are put together or organised. A structure contains elements, relationships among them and rules that guide their organisation. Simon (1969) and Doran (1996) state that hierarchical structures emerge naturally with the complexity growth of a set of elements and relationships.

The scope of the ‘total view framework’ encompasses the scope of systems engineering (SE) and that of concurrent engineering (CE). The scope of systems engineering, according to the EIA-632 standard and the IEEE-1220-Std-1994, includes the product development through requirements analysis, functional analysis and physical analysis and only requirements analysis for process development, at all levels of the product breakdown structure. The scope of concurrent engineering includes product, process and organisation but only at the physical analysis stage of the systems engineering process and at the bottom level or component level of the product breakdown structure. (see Figures 5-3 and 5-4 in Chapter 5)

11.2.3 Integration
The ‘total view framework’ sets up the elements necessary for integrated development and puts together systems engineering and concurrent engineering within the same framework. However, it is still necessary to identify the means of integration among product, process and organisation. Integration of
product, process and organisation takes place through the concepts of requirements, attributes and relationships, stressed in Chapter 6.

Requirements and attributes represent the interface between the stakeholders in the problem domain and the system solution elements in the solution domain. As mentioned in Section 6.2, stakeholders have requirements towards the system. The system has attributes affecting stakeholder satisfaction or dissatisfaction. Requirements and attributes may refer to product, process and organisation serving as a unifying language among different elements of the system solution. Relationships between product, process and organisation such as those described by many concurrent engineering methods and tools can be captured through the identification of the relationships between requirements and attributes and among attributes.

The importance of requirements and attributes were stressed in Chapters 3 and 4. In Chapter 4, one of the main lessons learned from the diesel PCS case study in developing a complex product was the early capture and analysis of requirements. In Chapter 3, the systems engineering process embedded in FPDS treats attributes as the means of communicating between layers of the vehicle breakdown structure and as the basis for verifying the product.

The key means of integration documented in the literature on integrated development and also in Chapter 2 is teamwork. In order to incorporate teamwork into the approach to integration adopted in this thesis, the relationship matrices include also the relationship between requirements and their sources and between attributes and their sources. Sources of requirements and attributes are the people or groups, in the development agencies, responsible for capturing or developing them. Relationships among these people or groups can also be captured.

The relationship matrices adopt a QFD-type format and as such intend to be an efficient means of communication between stakeholders and development agencies and among development agencies at different layers of the product breakdown structure. The benefits of QFD in terms of avoiding late changes, reducing cost, shortening development time, improving product quality indicate the potential of the integration approach proposed in this thesis. Those benefits are yet open to be fully exploited.

11.2.4 Complexity management

The identification of the relationships among requirements and attributes provide an indication of the level of coupling in the system solution and, therefore, can be used as a complexity management tool. The rationale for using level of coupling for complexity management comes, again, from the case studies and from the literature.

The gasoline PCS addresses the requirements for functionality, execution time, memory, reuse, ease of calibration, development time, all through the product alone. This strategy led to one more version of the application software, now a generic one. The improvements in terms of reuse and development time were only marginal. For example, the strategy reduced the number of base software versions from 30 to 25 but resulted in a product with enormous interface complexity among software subsystems. Negligible impact happened in terms of the other requirements.
The diesel PCS case study has shown the benefits to uncouple the requirements by addressing them by not only product elements, but also process and organisation elements. Functionality and execution time were addressed by the application software product, ease of calibration by a calibration guide, reuse by the structured analysis techniques, development time by teamwork.

Many references highlight the importance of uncoupled development.

[Suh, 1990] states in his Axiom 1 that “in an acceptable design, the design parameters and the functional requirements are related in such a way that specific design parameter can be adjusted to satisfy its corresponding functional requirement without affecting other functional requirements”. Suh then separates design into three groups: uncoupled, coupled and decoupled designs. An uncoupled design is the one that satisfies the above axiom.

Genrikh Altshuller, the developer of the Theory of Inventive Problem Solving, states that “a problem becomes a creative one when an attempt to improve system’s parameters by conventional means leads to deterioration of other parameters”. The approach to solve the problem tries to separate the ‘contradicting’ parameters in time, in space and by system transformations. System transformations include combination of systems, combination of system and anti-system, separation of contradictory properties between system and its sub-systems. [Fey et al., 1994]

Stevens et al. (1998) propose a structured approach for coping with complexity in complex product development in which user requirements, systems requirements and elements of the architecture design can be depicted over tree diagrams and related to each other. They also propose to keep track of the relationships among components, functions and requirements as necessary to cope with complexity.

The case studies and the literature review suggest that in the continuous product evolution cycle, as attributes may have some sort of dependency among themselves, to meet a new requirement that affects one attribute may not be as simple as it seems. Many other attributes may be affected. Requirements associated with these other attributes would then be also affected. Therefore, the change of an attribute could affect the whole satisfaction or dissatisfaction figures.

The early identification of the relationships among attributes and between requirements and attributes provides a picture closer to reality in terms of the amount of work that needs to be done to evolve an existing system. For a complex product, due to the great number of potential relationships, it is much easier to overlook them and, as a consequence, to only realise problems later in the product life cycle, when the cost of a change is much higher.

As mentioned above, relationships are represented in matrices. As for a complex product, the number of requirements and attributes can be very large, clustering is also an ingredient of the complexity management approach. This thesis uses metrics proposed by [Hitchins, 1992] that evaluates the quality of clustering based on the criteria of maximising cohesion within a cluster and minimising coupling between clusters. The metrics are used for evaluating the complexity of a solution. In Chapter 6, it is proposed that complexity metrics be used for functional and physical architecture improvement, design evaluation and team formation.
For evaluating the level of coupling, and using it to manage complexity, the following procedure is proposed in this thesis:

- to identify relationships between requirements and attributes;
- to identify relationships among attributes;
- to identify relationships among requirements by transitivity;
- to cluster requirements in order to minimise a complexity metrics.

The better solution would be the one with lower complexity, because it would reflect less coupled requirements.

11.3 The ‘concurrent structured analysis method’

Warfield (1976) states that structure is a necessary ingredient for coping with complexity. The diesel PCS case study used structured analysis method and tools to develop the diesel application software. Structured analysis was conceived, in the late 1970s, for software development to address, mainly, the productivity, reliability and maintainability issues.

The diesel PCS development used the Hatley Pirbhai methodology, a structured analysis methodology, through the use of a CASE tool called Teamwork [Cadre, 1995]. The methodology and tool were used for software development and for the analysis of hardware requirements. Although the development focused on product operation requirements with little consideration of other life cycle process and organisation requirements, the use of a structured analysis approach addressed many of the unwritten needs of the gasoline PCS development in terms of development time, reuse and ease of calibration.

In Chapter 7, it is proposed a method that extends the diesel PCS structured analysis approach to include hardware, life cycle process and organisation analysis, addressing not only operation requirements but also and simultaneously the non-operation ones. To fulfill the scope of the ‘total view framework’, the method supports the requirements, functional and physical analysis that is performed concurrently for product, process and organisation. The method is called ‘concurrent structured analysis method’.

The ‘concurrent structured analysis method’ demonstrates that, although large, the scope of the ‘total view framework’ is approachable. The outputs of the method are the requirements and attributes used for integration of product, process and organisation. Once requirements and attributes are known, the relationships between them and among attributes are captured. A basic principle behind this structured analysis method is the identification of requirements and attributes and of the relationships among them, early in the integrated development of complex products. The advantages of the application of this principle were learned from the diesel PCS case study.

The traditional structured analysis methodologies derive functional and module specifications for software. In the method proposed, the aim is to derive functional and physical attributes. In this case, attributes are treated as individual items of information as opposed to packages of information in the specifications. Individual attributes can be related to each other and to requirements making feasible the approach to manage complexity described above and in Chapter 6.

With the ‘concurrent structured analysis method’ it is not intended to prescribe another method of structured analysis. It is intended to demonstrate that, with the existing available methods, it is possible
to develop the product, its life cycle processes and their performing organisations in an integrated manner as elements of a whole integrated system.

The 'concurrent structured analysis method' consists of three analysis processes: requirements, functional and physical analysis. These processes are adapted, mainly, from modern systems engineering standards, IEEE-1220-1994 [IEEE, 1995] and EIA-632 [EIA, 1997]. These standards focus on product development and requirements capture for life cycle processes. They are adapted in order to provide requirement, functional and physical analysis to product, process and organisation. The analysis processes are performed through the simultaneous modelling of product, process and organisation from the outset.

The 'concurrent structured analysis method' is not an open-loop process. Iterations are likely to occur. However, one should prefer many iterations with short cycles to few iterations with large cycles. The ability to solve problems soon after they are identified and preferably very early in the product development life cycle will determine the efficacy of the integrated and concurrent development.

Sections 11.3.1 and 11.3.2 provide further details on the analysis processes and on the modelling approaches used.

11.3.1 The analysis processes

The objective of the analysis processes is: to generate the requirements, functional attributes and physical attributes.

Requirements analysis is triggered by the identification of some initial and obvious stakeholders and their needs expressed by stakeholder requirements. The requirements analysis process then identifies other stakeholders, their concerns and their requirements. As part of the analysis process, in the stakeholder requirements set, functions, performance, conditions, constraints, assumptions and goals are identified. These are then stated in clear, unambiguous, and measurable technical terms constituting the technical requirements set. Examples of adaptation from standardised requirements analysis procedure are:

- up-front identification of potential life cycle processes for the end-product to be;
- analysis of life cycle process scenarios instead of the traditional analysis of operational scenarios;
- up-front identification of life cycle process performing organisations.

Functional analysis translates the technical requirements into a functional architecture, from which functional attributes are derived. Functional attributes describe each element in the functional architecture. The functional architecture describes the functional arrangements and sequencing of sub-functions resulting from decomposing the set of system functions to their sub-functions. Functional analysis is performed without consideration of a design solution. Adaptations from standardised functional analysis procedures occur only in terms of their expansion to include processes and organisations. For example: states and modes are identified for product, process and organisation; hazard analysis is performed for product, process and organisation; failure mode effects and criticality are analysed for product, process and organisation.

Physical analysis translates the functional architecture into a physical architecture from which physical attributes are derived. Physical attributes describe each element on the physical architecture. The physical architecture provides an arrangement of elements, their decomposition, interfaces (internal and external), physical constraints, and designs. The IEEE-1220-Std [IEEE, 1995] uses the term 'synthesis'
to refer to the process of defining system product solutions and identifying subsystems to satisfy the requirements of the verified functional architecture. Synthesis translates the functional architecture into a physical architecture that provides an arrangement of elements, their decomposition, interfaces (internal and external), physical constraints, and designs. (Martin, 1997) includes the generation of alternative physical solutions as part of the synthesis process. Alternate configurations, or architectures, are developed and evaluated against the requirements. Prototypes or models may be constructed for more than one architecture to support trade-off analysis. The physical analysis process proposed in this thesis is essentially a modelling activity. It aims to identify the component elements of a physical architecture of the product, its life cycle processes and their performing organisations, the interactions among these elements and the attributes that characterise each element and interaction. Physical analysis can therefore be understood as the analysis of the implementation of products, processes and organisation constituents of the whole system. Physical analysis can be done either during the synthesis process of products, processes and organisations, in support of it, or after it, in order to model the resulting physical architecture. Product, processes and organisations are analysed in an interactive and iterative manner. For example, a decision on the shape of a part may affect the sequence by which that part must be machined. Also, the availability of a resource may determine the existence or not of a feature in a product and affects the schedule of the manufacturing process.

The requirements, functional and physical analysis process are carried out through modelling. Requirements and attributes are derived from the resulting models.

11.3.2 Modelling
Requirements, functional and physical analysis processes are carried out through the simultaneous modelling of product, process and organisation. The modelling techniques used result from an expansion of software development techniques such as structured analysis and design ([Yourdon, 1988], [DeMarco, 1981], [Marca, 1987]) and object oriented techniques ([Booch, 1996], [Rumbaugh, 1997], [Jacobson, 1994]). Modelling was chosen as a structured way to obtain requirements and attributes.

Again, there is no intention in this thesis to be prescriptive in determining the notations used for modelling product, process and organisation. The objective of modelling is to obtain requirements, functional attributes and physical attributes of product, process and organisation. The modelling approaches used in this thesis were selected in order to provide a total view of product, process and organisation, and, therefore, to make possible that a complete set of requirements and attributes would be derived. In order to provide a total view, the approaches chosen must be able to cover all necessary views to model product, process and organisations. Many authors describe which views are necessary for modelling product, process and organisations and these views are outlined below. In summary, a list of these views is:

- for product: functional modelling: activity, data and state [Calvez, 1993]; physical modelling [Stevens et al., 1998]: behaviour, structure and layout;
- for process [Curtis et al., 1992]: functional, behavioural, organisational and informational;
- for organisation [Vernadat, 1996]: function, information, resource and structure.

The objective of the method proposed here is to get a set of requirements and attributes as complete as
possible. If it is perceived that repeated sets of attributes are coming from a given model, that model can be made redundant. The method, approaches and notations must serve the purpose of deriving requirements and attributes. Otherwise, it will be modelling for the sake of modelling.

For product modelling, the Yourdon approach was used. The Yourdon approach was developed for software development. As modern complex products contain also hardware elements, the Yourdon approach needs to be extended. The work of Hubka and Eder (1988), formulating a generic theory of technical systems, shows that it is necessary to model also:

- energy and material couplings in addition to only information (data) couplings;
- physical connections in addition to only functional logical connections;
- functional and physical attributes, instead of specifications, are associated to each element of the functional and physical models, respectively.

The product requirements analysis model is not part of the traditional Yourdon approach. A DFD is used to develop a context diagram with the end product mission/objective in a central bubble surrounded by the stakeholders (rectangles) who interact with the product. The flows linking the central bubble to the stakeholders represent the stakeholders concerns (e.g. functionality, assemblability) towards the product. The direction of these flows is irrelevant.

For process modelling, it is used the SADT approach (Marca & McGowan, 1988) with IDEF0 notation adapted on a Behaviour Diagram (Alford, 1990). The Behaviour Diagram notation includes a timeline so that sequence, concurrency, alternative, loops can be represented in the model. This cannot be done in the original IDEF0 notation. The process functional analysis model represents a set of ideal activities in the process. The physical process model represents a set of real activities in the process. The process requirements analysis model does not exist in the original SADT methodology. It is represented by an adapted IDEF0 notation in which the function of the process to be modelled is linked to the stakeholders who are sources of inputs, control and mechanisms and to the destination of outputs. The direction of the links is irrelevant and these links represent the concerns of these stakeholders towards the process under modelling.

For the organisation modelling, it is used in this thesis is based on the approach proposed by Jacobson (1995) extended by using the UML (Unified Modelling Language) notation for the object model (Texel and Williams, 1997). The functional view is covered by Jacobson's Use Case modelling approach. The other views are covered by the UML notations, for example: class diagram for information and resource view, collaboration and deployment diagrams resource view; package diagram for organisation view.

Key attributes derived from the modelling approaches above are:

- performance, reliability and product physical characteristics for product;
- lead time and risks for processes;
- productivity, risks and resources physical characteristics for organisation.

The attributes follow the hierarchical structure from the diagrams from where they are extracted.
11.4 The implementation

The 'concurrent structured analysis method' and the 'total view framework' were demonstrated through examples of application from the automotive industry. The examples were developed using: a commercial systems engineering environment, Cradle-3.2; a graph partitioning algorithm, Chaco-2.0, acquired from the SANDIA Laboratory in the USA; and an electronic spreadsheet, Microsoft Excel-97. Cradle was used for modelling, requirements management, attributes management and relationships management. Chaco-2.0 was used for clustering. Excel was used for relationship representation and calculation of a complexity index.

Chaco-2.0's original application is the identification of the best architecture of processors in parallel computing. The application proposed in this thesis is a novel application of Chaco-2.0. The idea was just to replace the processors in the original application by requirements, attributes, functions, parts, etc. in this application. The connections between processors were replaced by relationships, function flows, physical connections in this application. The use of Chaco-2.0 is justified by the fact that the relationship matrices resulting from the 'concurrent structured analysis method' are very large (e.g. hundreds to thousands attributes in a car).

Cradle was selected because:

- it has multi-notational capability;
- it does not impose a specific development methodology;
- it covers the whole systems engineering process from requirements capture to physical architecture definition;
- it allows integration of stakeholders and sub-contractors into the systems engineering process.

Cradle allows the inclusion of stakeholders and development agencies, functional and physical attributes as items of information and the use of cross-references to represent different types of relationships. The items of information and relationships are represented in Excel and Chaco is used for clustering of the relationship matrices in Excel. Clustered matrices are again represented in Excel. Excel provides the means to derive a complexity index.

Three examples of application were developed to demonstrate different aspects of the framework and method. They are: the diesel PCS, the gasoline PCS and a seat height adjust mechanism, referred as SHM. Although they are subsystems, all of them are considered within the context of the integrated development of the whole product - the car. As such, the links between the car model elements and the model elements developed in these examples are preserved and pursued. Also, although these examples are named by the subsystem products, they include not only product models but also process and organisation models through the requirement, functional and physical analysis processes.

The diesel PCS example:

- demonstrated the proposed modelling approach for the requirements, functional and physical analysis of product, processes and organisation;
- identified functional and physical attributes from functional and physical models;
- identified the relationships among requirements and attributes and their propagation.
The gasoline PCS demonstrated the implementation of a complexity management approach.

The analysis processes in the PCS examples concentrate essentially on software subsystems and development processes and organisation. In order to demonstrate the application of the framework for the development of hardware subsystems and manufacturing and assembly processes and organisation, the SHM example was developed. This example demonstrates the 'total view framework' on the SHM product, its assembly process and the H. R. Adcock Ltd. organisation, a Tier 2 supplier. This example also demonstrates the applicability of the framework when product subsystems are developed by different companies, which is often the case for complex products development.

11.4.1 The diesel PCS example

The diesel PCS example demonstrates the use of the relationship matrices for deploying down requirements and attributes from the car layer down to the powertrain layer and then to the PCS layer. Matrices with the relationships between development agencies and requirements and attributes were also demonstrated. The examples also demonstrated the fact that the deployment can be done by allocation, partition or decomposition.

For the PCS, the 'concurrent structured analysis method' was implemented in Cradle demonstrating the applicability of the 'total view framework'. Requirements, functional and physical models were developed in Cradle for product, process and organisation. Attributes were captured from the functional and physical models. Relationships between requirements and attributes and among attributes were captured in matrices presented in Excel.

The diesel PCS example focused on the EGR_control software subsystem. The relationship matrices developed followed the structure of the functional and physical models. It was demonstrated the traceability links from requirements, to functions and from these to software and hardware subsystems. The matrices followed the same structure of the physical models, capturing the traceability relationships from the functions at the bottom level of the product functional structure with the component or software subsystems at each level of the product breakdown structure.

As process and organisation models were developed only at the layer of the EGR_control software subsystem, a relationship matrix integrating product, process and organisation was developed. The integration strategy was done by capturing the relationships among functional and physical attributes of product, process and organisation. It was observed from this example that functional attributes tend to map to physical attributes of the same type (product, process or organisation). To investigate the coupling between product, process and organisation it was still necessary to investigate the relationships among physical attributes. This was done in the SHM example.

11.4.2 The gasoline PCS example

The modelling and analysis processes provide a structured way to move from stakeholder requirements to functional and physical attributes of product, processes and organisations. In those processes, the relationships among requirements, among attributes and between requirements and attributes were captured. Also, development agencies can be linked to requirements and attributes providing information
that can be later used for team formation, for example. In order to make the information gathered and structured in the process useful for, for example, evolving a current system, it is necessary to provide visibility of the captured relationships. The diesel PCS example showed that the potential order of the matrices can be enormous. Thus, it is necessary to constantly re-arrange models, requirements and attributes in clusters so that the analysis of relationships can be more focused and so that any structure modification can be done sooner rather than later (e.g. during a reengineering process). Benefits can be foreseen if an objective criterion can be used for guiding the re-structuring process. The gasoline PCS example proposes and demonstrates a re-structuring procedure that uses as quality criterion the minimisation of a complexity index. The complexity index is calculated in such a way that reflects coupling minimisation and cohesion maximisation. The re-structuring procedure is named 'complexity analysis procedure' and is applied to the Idle Speed Control software strategy code. The Idle Speed Control is a subsystem of the gasoline PCS.

As the size of the relationship matrices can be very large it is highly recommended the use of computer aid to perform the partitioning into clusters. Traditional partitioning algorithms such as those used for Group Technology purposes and reviewed in [Kusiak, 1990] do not fit the large size of matrices expected from the total view applications. Therefore large graph partitioning algorithms are adopted. The gasoline PCS example demonstrates the use of Chaco-2.0, a graph partitioning algorithm originally developed for parallel computing applications.

The Idle Speed Control strategy code analysed is the result of 20 years of evolution of the software code. The complexity analysis procedure consists of decomposing the software down to its elementary software units and investigating at the very bottom level of the software breakdown structure the exchange of data among those units. The software units are depicted on the row and columns of a matrix. The cells in the matrix contain numbers reflecting the data exchange between units. The units are then clustered together in order to maximise cohesion and minimise coupling by using Chaco-2.0. The units are also clustered together according to Ford's partitioning. The complexity index is then calculated for both partitioning approaches. Chaco's partitions resulted in an improvement of 23 to 35% in the complexity index if compared with Ford's partitioning. This means that the reactive evolutionary approach adopted by the gasoline PCS group led to 23 to 35% more coupling among software units than it was really necessary. This also means bigger interface complexity than necessary, bigger organisational complexity, larger communication overheads to develop the PCS, more costs and longer development time.

Both partitioning approaches and the coupling between partitions were also compared with the theoretical coupling between functions in the Idle Speed Control functional model. Each partition was mapped to a function. The result indicated that Chaco's partitioning was closer to the theoretical functional partitioning than the Ford's one. This is an indication that the actual software could be partitioned in a more modular way. Ford's partitioning was making the functions more coupled than they needed to be. This is critical if one remembers that Ford PCSE, the group responsible for the PCS development, is organised by feature with each function corresponding to a feature. More coupling between functions or partitions suggests more communication overhead, increasing cost and development time.

For a better mapping between the functions in the theoretical model and the physical software units in the resulting partitions it may be necessary review the functional model or concept. Reallocation of units
between partitions may even indicate the need to modify the function initially expected from a given initial partition. Ideally, the theoretical functional model should map exactly to the physical model so that modification in the functionality would be obtained by just modifying software units in a product partition without affecting other partitions. In reality, that is not always possible and can be made worse if attention is not given to the allocation of new created software modules to the appropriate partition.

Some direct applications of the complexity analysis procedure within the 'total view framework' are the re-structuring of self and cross interaction matrices of requirements, functional attributes, physical attributes and development agencies. A total system solution may be sought, for example, seeking to minimise the complexity index. The axiomatic design theory [Suh, 1990], for example, provides the theoretical support for the application on cross-reference matrices between functional and physical attributes.

Similarly to the analysis above, the same procedure can be used to analyse and optimise process architecture. [Kusiak, Larsson & Wang, 1994] provides some examples using a triangularisation algorithm for process scheduling.

This example just demonstrates the use of a complexity analysis method. Although it was applied only on a product architecture (software in this case), the analysis approach can be used in a broader context within the 'total view framework', e.g., for team architecture and overall system optimisation, besides the applications already mentioned above.

11.4.3 The SHM example
The seat height adjust mechanism (SHM) example demonstrated the applicability of the ‘total view framework’ and ‘concurrent structured analysis method’ to an essentially hardware product, a mechanical product.

This example demonstrates that although the requirements, functional and physical modelling methods and tools were imported from the software arena, they are also applicable to a hardware product.

As product, process and organisation were already well established, this example was developed through a reverse engineering exercise. There was no attempt to simulate the iterations undertaken by when the product was actually developed. Therefore the physical models were developed first and then the functional models. The general sequence followed for the development of this example was: 1) modelling; 2) requirements and attributes identification; 3) relationships identification. Cradle was used for modelling and requirements documentation. Excel was used for attributes and relationships identification.

Attributes were extracted from the models and were used to describe product, process and organisation. Not all processes and organisations were modelled, just the manufacturing and assembly processes and the producer organisation. In a typical mechanical engineering environment, product models could be CAD drawings. Those too can be used for generating attributes. Attributes, like models or drawings, are a way of describing a product. Differently from models, drawings or specifications, attributes have granularity. They are much more specific and they provide a universal language that can be used to describe not only product but also process and organisations. As such, capturing the relationships
between attributes can be a way of integrating product, process and organisation. And, at the end of the day, attributes are the changing elements during product, process and organisation evolution. Therefore, having the links among attributes and between requirements and attributes provide a way of performing change impact analysis.

Traditional structured analysis from where the ‘concurrent structured analysis method’ was developed is in most applications used for software development. For software development, specifications are generated as a result of the analysis process. Specifications are in general codified algorithms that represent the lowest level description of a function in the software. They may include, however, attributes such as memory, execution time, synchronisation. For hardware, one specification, however, would contain almost exclusively attributes. These attributes individually may affect many others in different specifications.

The SHM example illustrated the use of attributes for describing a hardware product and their usefulness in providing integration between product, process and organisation.

Some lessons learned with the SHM example are:

- most useful model for generating attributes were those containing all elements attributes should describe. For example: data flow diagrams containing a hierarchy of product functions; behaviour diagrams containing hierarchy of process activities; use case models containing hierarchy of organisation functions; class diagram containing resources and their relationships;
- care must be taken for avoiding confusion and duplication of work when modelling process and organisation. Process models model the life cycle of a product being modelled. The emphasis of process models is activities of a product life cycle. The key attribute derived from the process models is time. Organisation models focus on the resources performing the life cycle process. For economic reasons, resources may be used to perform the life cycle process for more than one product. The organisation models must take it into consideration. Key organisation attributes is productivity. This mapping from product, process and organisation to specific attributes can also be derived from the work [Clark & Fujimoto, 1991]. There, Clark & Fujimoto propose some parameters for evaluating product development performance: product quality, development lead time and organisation productivity. Those parameters map to product, process and organisation, respectively;
- the quality of models is determined by the quality of the information available about the object to be modelled. Models should be elaborated by the people actually designing the product, process or organisation or who have expertise in the product, process or organisation development;
- the quality of models determines the quality of requirements and attributes obtained. If the models are incomplete or incorrect, so will be the requirements and attributes obtained from them;
- there is still the need for researching some guidelines to generate attributes from the models in a more systematic way. Attributes were derived from the models just by thinking which attributes could be related to a given function, activity, component or resource. There is no guarantee that the set of attributes chosen is the best one;
- the quality of relationships depends on the quality of the requirements and attributes captured. If for example the definition of an attribute is not clear or if the attribute is not specific, it may be difficult to relate to a single requirement, for example:
• relationships among product, process and organisation were more evident during physical analysis. Relationships among requirements and among functional attributes of product, process and organisation have to be identified by transitivity. The interactions among product, process and organisation attributes in the physical analysis models are in fact a concurrent engineering principle. This thesis proposes to investigate how these interactions propagate to link functional attributes or requirements. The resulting level of coupling among functional attributes or requirements can be used to evaluate the complexity of a solution;

• despite the many simplifications made, the matrices resulting from the SHM example were very big (order around 150). The effort necessary to implement the SHM example suggests that the overhead necessary is not compatible to the size of a Tier 2 supplier. Only a Tier 1 supplier or the car manufacturer itself could afford going through this process.

11.5 Evaluation against the trends in the complex product industry

Chapter 3 describes the trends affecting product development in the space and automotive industries. These trends are summarised in this section and it is analysed how the framework and method address them.

11.5.1 Space industry

Space products are well known for their complexity. They are multidisciplinary products, accomplish many functions and are composed by a great number of interacting parts. The traditional development approach used in the space industry is a sequentially phased program with many review points, so that to move from one phase to the next the partial deliverables must be approved during the review process. Also, traditionally, space products have been a result of publicly funded projects with great emphasis on performance and risk avoidance. However, decreasing levels of public funding to finance space programs, increasing commercial interest in space and advances in information technology are forcing space industry to consider life cycle cost and development lead time among their measures of effectiveness when developing a product. In order to reduce cost and lead time the space industry started to consider, for example: reuse of proven concepts and technologies between programs, synergies between military and commercial applications, use of commercial off the shelf solutions, risk management instead of risk avoidance, up-front analysis and simulation instead of comprehensive testing. Also the traditional document passing systems engineering philosophy started and used by the space industry is opening space to a more trade-off based systems engineering.

NASA for example uses a spiral model to describe its systems engineering process instead of the traditional waterfall model. The spiral model is characterised by a strong emphasis in capturing requirements up-front in the system development cycle. The requirements are successively detailed, translated into concepts, traded-off and translated into a design solution. At each turn of the spiral the system is understood in greater detail, increases its resolution and moves further deep from the problem to the solution domain. The use of the spiral model is particularly useful in high-risk developments because design evolves as detailed requirements emerge.
The latest NASA programs have been developed under the flag of 'faster, cheaper, better' space missions. It relies on earlier integration between system development and verification and on off-line technology development.

The 'total view framework' and the 'concurrent structured analysis method' are developed in support of the trends in the space industry.

The 'total view framework' recognises the need to consider more than one stakeholder and the requirements analysis method identifies those stakeholders and capture their requirements very earlier in the integrated development process. The latest space programs, e.g. International Space Station, involve partnership between government and commercial organisations, international organisations, academia, etc. Thus it is necessary to identify stakeholders and their requirements up front. Requirement analysis enhances requirements completeness as they first identify all possible stakeholders affected by product, process and organisation attributes. Product stakeholders are those affected by the product attributes over the product life cycle not only during its operation. Process stakeholders are those who are the sources of inputs, outputs and controls or the receivers of outputs of the life cycle process activities. Organisation stakeholders are those people outside the organisation that have vested interest in it or play a role towards the organisation or are affected by the organisation attributes.

The 'total view framework' and the matrices resulting from the concurrent structured analysis processes identify potential relationships among attributes that could be further investigated through simulations, for example. Also the models and matrices resulting from the analysis processes may serve as a repository of known relationships to be re-used in future programs avoiding unnecessary duplication of work. Even if it is necessary to perform tests to identify relationships, these would be registered in the matrices and testing could be reduced for future programs.

They also address the objectives of a 'faster, cheaper and better' space program within a manageable risk.

- **faster:** because requirements and constraints for the product, its life cycle processes and [some of] their performing organisations are captured early in the product development process. This reduces late changes in the product due to the late arrival of requirements. This also allows, for example, life cycle process enabling product and organisation resource solutions to be investigated earlier and concurrently to product development. Also as the relationships among the various attributes of product, process and organisation are investigated and identified beforehand and as the developers linked to those attributes are also identified, people do not need to wait until the next review meeting to discuss eventual modifications. They can go to the meeting with solution proposals already discussed. This speeds up the development process. A shorter product development is a consequence of the visibility of interactions among many stakeholders, requirements, functions, product and processes, which can be implemented in the Cradle's product data management module.

Change propagation can be analysed through the use of cross-references and quantified through the use of performance modelling in Cradle (this was not done in this work and can be object of further work). Also the project repository provided by Cradle can be totally or partially re-used for future projects leading to a shorter time to orbit for future space missions.
• cheaper: the reduction of late changes also reduces the development cost. Lower life cycle cost results from addressing life cycle process requirements and being able to model the performance of life cycle processes even at the pre-specification stage of the product. For example, using Cradle for the early modelling of a satellite and its operation process, identifying their respective attributes and investigating their relationships leads to balanced solutions regarding design margin definitions which will greatly affect the operations cost of the mission.

• better: because the product is developed to requirements and verified against requirements and the 'total view framework' traces the links between product attributes and requirements, product compliance to requirements is expected. As earlier trade-offs can be made due to the awareness and visibility of relationships among product, process and organisation attributes, a more balanced solution can be achieved optimising the overall satisfaction of the set of stakeholders. Better product quality results from the fact that the product is developed and verified against requirements that can be traced to the original mission stakeholders' requirements. In the current space mission implementation environment, stakeholder requirements include not only operational and functional requirements but also specific market requirements, commercial partner requirements, international partner requirements and life cycle process requirements. Attributes corresponding to these requirements are assigned to the product, process and organisation models. Performance analysis based on attributes such as quality, cost, risk and schedule can take place very early in the product life cycle by using Cradle, for example. To assure, for example, that requirements that most impact quality from a users perspective are being addressed, Cradle allows the creation of requirements categories, e.g. primary and secondary — primary being those requirements imposed by the users and secondary, by the standard way of proceedings while defining a space mission. Secondary requirements are the ones that will be traded off in order to find a balanced solution in terms of cost, risk and schedule. Primary requirements are the ones that must be met in order to obtain a high quality product.

• manageable risks: as risks can be considered as attributes associated with the functions or states in the functional analysis models, the physical attributes and their relationships affecting those risks will be identified and from there decisions can be made regarding, for example, redundancies and target values of physical attributes. Also these risks can be associated with other attributes in such a way that it would be clear the attributes to be compromised in order to reduce risk, for example. To manage risks, contrasting with the traditional risk avoidance approach of space mission implementation, Cradle performance modelling module can evaluate cost savings and the impact in product quality in terms of the amount of risk accepted.

11.5.2 Automotive industry
The automotive industry faces an environment of increasing number of stakeholders, requirements becoming more numerous and stringent, customer demanding for increasing product functionality while keeping the price constant, increased number of options available to implement required functionality. This led to increasing product complexity which cannot be any longer satisfactorily managed by the traditional component approach used by the automotive industry.
In order to cope with the increasing complexity of the automobile, the product development process in the automotive industry is going through a number of changes. The changes are listed below and the way the 'total view framework' and 'concurrent structured analysis method' can enhance them is explained:

- more structured requirements capture and analysis: the 'concurrent structured analysis method' proposes a requirements analysis process based on structured analysis techniques.

- anticipation of life cycle process requirements: the life cycle process requirements are considered in the 'concurrent structured analysis method' when during the product requirements analysis the stakeholders identified are those who are affected by the product attributes during the product life cycle process not only during its operation. Also, requirements are captured for the life cycle processes during the process requirements analysis. Product, process and organisation requirements analyses are carried out concurrently at the beginning of the integrated development process.

- greater automation and integration of CAD, CAM, CAE, PDM and rapid prototyping tools: the 'total view framework' provides a framework of what should be integrated from the outset in the system solution to be developed. The attributes that are identified during the analysis processes can come from CAD drawings for example. Attributes can feed CAE models to have their relationships quantified. Cradle, the software tool used to implement the 'concurrent structured analysis method' is a PDM tool in itself. Therefore, the 'total view framework' does not intend to be a replacement for tools such as CAD, CAM, CAE, PDM and rapid prototyping but to provide a framework where those tools could then be used for the substantiation of a whole integrated system solution.

- hierarchical system structure and tier-supplier approach: one of the dimensions of the 'total view framework' is structure. Differently, though, from a traditional system breakdown structure, the 'total view framework' considers on each layer of the system hierarchy not only the product elements but also the corresponding life cycle process and organisation elements. Therefore, suppliers may include not only those providing product subsystems but also those providing life cycle process enabling products and organisation resources. The tier supplier structure can be reflected in the product breakdown structure.

- engineering analysis up-front: another dimension of the 'total view framework' is the analysis dimension, from which result the product, process and organisation requirements and attributes. The functional and physical analyses provide a set of attributes whose values can be calculated through engineering analysis. Those attributes can also be used for an early performance analysis in order to reach early agreement of specifications among stakeholders. Cradle, for example, has performance modelling capability that can be used for that purpose.

- empowerment of project teams: the contribution of the 'concurrent structured analysis method' to teamwork is the fact that it proposes to associate to each technical requirement and to each attribute the people in the development agencies actually responsible for obtaining that requirement or attribute. As relationships among attributes are identified, attributes can be clustered and to those clusters would correspond a team of people responsible to deliver that set of attributes in a balanced solution.
reuse: the results of the analysis processes, i.e., models, requirements, attributes and relationships, can be reused from vehicle program to vehicle program. Care must be taken however to keep track of the modifications carried out from program to program, to extend the physical modifications to the functional and requirements models and, to extend the modifications to the upper levels of the product breakdown structure. These measures keep the analysis structured and help to manage complexity growth in future programs.

FPDS (Ford Product Development System) is the approach Ford adopted to shift from the traditional component approach to implement the changes listed above towards systems engineering. Although, one of the FPDS supporting concepts is concurrent engineering, FPDS is still very much product focused. An example of this fact is that, from the 15 attributes chosen to verify total vehicle quality, 13 are product, 1 is process (customer life cycle) and 1 is organisation (product/process design compatibility) attributes. The ‘total view framework’, on the other hand, seeks a balanced system solution composed of product process and organisation elements.

C3P is the strategy Ford uses to integrate CAD/CAM/CAE/PIM information. The information contained in the C3P repository includes packages of information such as drawings, models, specifications, reports, requirements documents. The ‘total view framework’ proposes to use three fundamental information items: requirements, attributes and relationships. The advantage of the approach included in the ‘total view framework’ is to have more granularity of information, so that individual attributes or individual requirements may be related to each other or elicited separately. The drawings, models and specifications provided by C3P contain many attributes in one document what makes it more difficult to relate individual items of information to each other. In order to implement the ‘total view framework’, it is proposed the PIM tool to be part of a systems engineering environment (SEE) that supports the product, process and organisation modelling in a hierarchical structure. CAD/CAM/CAE tools would be used as source of physical attributes (CAD), performance analysis of process attributes (CAM) or investigation and quantification of relationships (CAE).

11.5.3 Both industries
Both industries, although through different paths, move towards integrated development. The space industry now adopts a more trade-off-based systems engineering and concurrent engineering as part of their development process. The automotive industry shifts from its traditional component approach towards systems engineering. Concurrent engineering is being in use for 10 years in the automotive industry where it, so far, enhanced the component approach. The ‘total view framework’, as a systems engineering and concurrent engineering approach to the integrated development of complex products, meets the trends towards integrated development of both industries.

1 Customer life cycle includes serviceability/repairability, recycleability, durability/reliability, corrosion, on-going quality.

2 Product/process design compatibility includes investment efficiency, reusability, commonality, carryover, complexity, manufacturing life-cycle cost.
11.6 Evaluation against the needs highlighted in the case studies

The gasoline PCS case study highlighted some needs in order to manage complexity while developing a complex product:

- to maintain traceability from stakeholders to their requirements, from these to functions, from these to implementation elements, so that a change in one of them can be quickly responded by the others;
- to consider not only the product as the solution of the complex problem but also its life cycle processes and their performing organisations;
- to keep visibility of the relationships among the various elements of the solution implementation, including not only the product elements but also the process and organisation elements.

The 'total view framework' and the 'concurrent structured analysis method' were developed in order to meet those needs:

- as mentioned above, the 'total view framework' has three fundamental items of information. They are: requirements, attributes and relationships. Relationships are classified in three types: hierarchy, traceability and impact relationships. Traceability links are the ones linking stakeholders, to functions, to implementation elements. Impact links are the ones linking, mainly, requirements, functional attributes and physical attributes. These account for change impact analysis.
- the integration dimension of the 'total view framework' includes the product, its life cycle processes and their performing organisations as elements of the system solution.
- the models and the matrices resulting from the 'concurrent structured analysis method' are an attempt to provide visibility of relationships. As the matrices tend to be very large for a complex product, it is proposed the use of clustering algorithms to deal with matrices of a more manageable order. Chapter 9, Section 9.4 demonstrates the use of clustering.

The gasoline PCS is an example of the traditional component approach used for product development in the automotive industry. According to that approach, very early in the vehicle program development process, the product was broken down into its constituent parts (e.g. 400 parts) and for each of those parts optimisation was pursued. The assumption was that the optimisation of each individual part would lead to the optimisation of the whole vehicle. Very little work was done in terms of requirements and functional analysis. Therefore, the general trend was that the product kept a very static concept since it was first conceived, with incremental evolutions on individual parts.

The gasoline PCS application software is the result of 20 years of incremental evolution. One existing software application is modified to meet the requirements of a new vehicle program. Modifications were performed directly in the software without consideration of their effects on the higher level [sub]systems: PCS, powertrain, vehicle. Modifications were implemented without links to the higher level requirements motivating them or to functional analysis. All modifications were implemented based on a set of software change requests. The modified software was tested in the vehicle. If the vehicle met the emissions standards, the modifications were kept. Otherwise, new modifications were required or the specification changed. The result was a product that was very expensive to maintain and an explosion in the number of application versions to meet basically the same functionality. Therefore, although the resulting software
met the functionality required and memory and execution time constraints, it was failing to meet reuse and development time targets established by the Ford corporation. The way Ford was organised with strategy groups as functional departments while software groups were part of the vehicle program organisations contributed very much to the increase of variety in software versions and complexity of the software product itself. Also the institutionalised change driven development process trying to address reuse, development time, ease of calibration, functionality and constraints, all through the product alone made the product unnecessarily more complex.

The diesel PCS group addressed reuse by using an structured approach to software development, development time by putting a co-located team to develop the software, ease of calibration by providing a calibration guide for powertrain calibrators, functionality and constraints by the software product itself.

The philosophy behind the approach adopted by the diesel PCS group was, in general terms, to identify requirements and to adopt as independent as possible solutions to each requirement or requirement set.

The ‘total view framework’ extends the approach used by the diesel PCS group to formally include in the structured analysis method, not only the product, but also its life cycle processes and [some of] their performing organisations. The aim of capturing requirements, attributes and relationships, fundamental elements of the ‘total view framework’ and of the ‘concurrent structured analysis method’, is to investigate how a given product-process-organisation solution couples the initially uncoupled requirements and functional attributes. Less coupled solutions mean also solutions that are less complex and easier to evolve. In Chapter 9, Section 9.4 it is proposed a method that can be used to compare different solutions in terms of a complexity index. If physical attributes are affecting each other and they are ‘implementing’ functional attributes, by transitivity, those functional attributes will affect each other. The same rationale can be used to identify coupling among requirements. As product, process and organisation are all part of a whole system solution, the ‘total view framework’ includes them as object of the application of a structured analysis method, with their requirements and attributes being subject to the coupling investigation.

The problem with the traditional component approach used by the automotive industry and for the gasoline PCS development is that when changes are made directly at the bottom level of the product breakdown structure without concerns to requirements and functions, couplings may be being created among functions and among requirements. These couplings increase the overall system complexity and, as a consequence, reduce its capacity to evolve. In the short term, one may think he has got a shorter development time through reuse. In a longer term, the product, process or organisation will end up having to be reengineered. The ‘total view framework’ and the ‘concurrent structured analysis method’ intend to keep the couplings created when evolving a product in sight through the matrices resulting from the analysis processes. At each evolution cycle, when trying a solution that will increase coupling, the system partition can be reviewed and a new functional concept can be proposed. This approach would avoid the reengineering shock, by incremental, not only physical changes, but also conceptual ones if necessary.
11.7 Relationship and comparison with general complexity management approaches

This section aims to relate the work done in this thesis with the concepts and approaches to complexity management reviewed in Chapter 2. The work done in this thesis is referred as 'total view framework' but this includes the 'concurrent structured analysis method' as well.

The 'total view framework' and the definition of complex products

[Prasad, 1996]’s understanding of the term ‘complex product’, as already mentioned in Chapter 2, Section 2.2.2, considers aspects involving product requirements, functionality and components, manufacturing process and product development organisation. The ‘total view framework’’s analysis and integration dimensions include all of those aspects and more as it proposes requirements, functional and physical analysis to be applied not only to a product, but also to all of its life cycle processes (including manufacturing) and to some of their performing organisation (at least the product development one).

The ‘total view framework’ and the definition of complexity

The dimensions of the ‘total view framework’ comply with the definition of complexity proposed by [Hitchins, 1996] as illustrated in Figure 11-1. The ‘total view framework’ is developed over three basic dimensions: analysis, structure and integration. These dimensions are set to manage the three factors through which complexity emerges [Hitchins, 1996]: variety, disorder, connectedness. Figure 11-1 illustrates the fact that each framework dimension addresses mainly one complexity factor.

![Complexity factors and framework dimensions](image)

**Figure 11-1:** Complexity factors and framework dimensions

Analysis refers to a separation of the whole into its component parts, an examination of these parts and their relationships, and a follow-on decision relative to a future course of action [Blanchard, 1998]. In dealing with variety, analysis is used to identify the whole set of elements, separate these elements into groups using some criteria of similarity and taking different decisions depending on the characteristics of each group of elements.
Integration means putting together heterogeneous components to form a synergistic whole [Vernadat, 1996]. Integration is then an immediate response to the connectedness factor. Systems are characterised by properties that are not explained by any of its component parts taken in isolation. These properties are only explained by the interactions among these component parts. Interactions among component parts characterise the connectedness factor. Component parts with stronger interactions would have to be integrated within a subsystem. These subsystems would then be integrated to compose the whole system.

Structure is the way a set of elements are put together or organised. A structure contains elements, relationships among them and rules that guide their organisation. Therefore structure addresses mainly the disorder factor. Examples of systems structure are: hierarchies and heterarchies. Simon (1969) and Doran (1996) state that hierarchical structures emerge naturally with the complexity growth of a set of elements and relationships. However, there is a strong on-going trend within space, IT and organisation systems for heterarchical structures, i.e., systems where functions are performed by intercommunicating peer systems [Grant, 1997]. Examples of such systems are: global positioning satellite system, distributed parallel computing, object-oriented systems, computer network systems, advanced distributed simulation system, multi-agent systems, Java applets, holonic organisation.

The ‘total view framework’ and complexity escalation
Due to the fact the ‘total view framework’ proposes to carry out product, process and organisation requirements analysis up-front, it is expected to have a good understanding of the problem to be solved very early during the integrated development process. In the case of evolving an existing product, for example, as requirements are linked to attributes and these attributes linked to the people who provide them in the development agencies, a group of knowledgeable specialists can be formed at any time a set of attributes must be modified to meet the requirements. This avoids the situation illustrated in Figure 2-1 in which a person or team would be overwhelmed by the problem scope.

The ‘total view framework’ and complexity metrics
The ‘total view framework’ uses the complexity metrics proposed by [Hitchins, 1992] (see Chapter 2, Section 2.2.4). Chapter 9, Section 9.4 illustrates the use of the metrics.

The complexity metrics, within the ‘total view framework’ scope, can be used for: re-structuring of self and cross interactions matrices of requirements, attributes and development agencies; team architecture and; overall system optimisation (see Chapter 9, Section 9.4.5). A complexity index can be used as criteria for obtaining a system partitioning according to requirements clusters for example. The team architecture can follow such a criteria as well.

The complexity index was used in Chapter 9, Section 9.4 to analyse the architecture of the gasoline PCS software and how it could be improved. It is proposed in this thesis those other applications proposed above to use the same approach and procedure as the one demonstrated in Chapter 9.

The ‘total view framework’ and interpretive structural modelling
The ‘total view framework’ provides all the information necessary to obtain an interpretive structural model of requirements or of attributes. In the case of attributes, an interpretive structural model would
organise attributes in a (di)graph format identifying the levels of attributes based on their relationships with each other. This could help to use only the lowest level attributes on the matrices resulting from the analysis processes. This is a matter for further work.

**The ‘total view framework’ and Pugh’s total design**

[Pugh, 1991] defines total design as the systematic activity necessary, from the identification of the market/user need, to the selling of the successful product to satisfy that need — an activity that encompasses product, process, people and organisation.

Although Pugh acknowledges the need to encompass product, process, people and organisation, the methods described by Pugh focus on developing the product. The ‘total view framework’ includes not only the product but also its life cycle processes and [some of] their performing organisations within the development effort scope. The total design activity model (see Figure 2-4 in Chapter 2) proposed by Pugh is covered by the ‘total view framework’ analysis processes: the market and specification phases are covered by requirements analysis; the concept design phase is covered by the functional analysis; the detail design phase is covered by physical analysis. The preparation for the manufacture and sell phases is considered during the life cycle process development which is concurrent to the product development process. Therefore, the ‘total view framework’ encompasses the scope of Pugh’s total design. The same can be said about the [Clausing, 1994]’s total quality development which is in essence an extension of Pugh’s total design.

**The ‘total view framework’ and total quality control**

[Feigenbaum, 1991] defines total quality control as being an effective system for integrating the quality development, quality maintenance and quality improvement efforts of the various groups in an organisation so as to enable marketing, engineering, production, and service at the most economical levels which allow for full customer satisfaction.

The above definition states that the scope of total quality control is bounded to an organisation and the product life cycle processes it performs. Also, the objective of the organisation is to allow for full customer satisfaction at the most economical levels.

The ‘total view framework’ proposes a broader scope and a broader objective than total quality control:

- it includes all life cycle processes, not only those performed or related to a specific organisation. This follows the trend in the automotive and space industries of considering cost of ownership and life cycle cost as quality parameters. Also, the current increasing environment awareness leads organisations to consider aspects of the life cycle such as disposal, recycling, debris recovery (space industry).

- it includes all stakeholders affected not only by the product but by all product life cycle processes and by [some of] their performing organisations. The result of the integrated development effort within the ‘total view framework’ is a whole balanced system solution considering all identified stakeholders being affected by the system attributes — the system being composed of product, process
and organisation elements. Of course the customer is a key stakeholder but other relevant ones are: government, stockholders, competitors, performers of life cycle processes.

The ‘total view framework’ and total systems approach
The total systems approach proposed by [Hatley & Rushton, 1997] is included in the product requirements, functional and physical analysis processes proposed in Chapter 7, therefore is part of the ‘total view framework’. The word ‘total’ in the total systems approach refers to the fact that the product functional and physical models were developed taking into account the whole product life cycle process. All elements in the environment interacting with the product during its life cycle were considered in the models.

The ‘total view framework’ and business process reengineering
The ‘total view framework’ intends to provide visibility of the relationships among product, process and organisation elements. Instead of getting to a point where reengineering will be needed, the ‘total view framework’ proposes to keep track of the relationships among attributes and see how the modifications on these attributes and new relationships being created will affect the partition of the whole system, including the business organisation. Also as requirements and attributes are linked, clusters of requirements can be identified from related clusters of attributes and the organisation could then be structured taking into consideration those requirements clusters. That would then be a practical way of guaranteeing that everyone in the organisation is working to meet a requirement. Since requirements are expected to change with time, so would the organisation. Providing that the organisation is tuned with the market, its continuous incremental changes based on requirements modifications would guarantee continuous organisation competitiveness and avoid the need for reengineering.

Anyhow, if business process reengineering is required, the ‘total view framework’ provides a method through which it could be done, taking into consideration not only the organisation itself, but also its products and their product life cycle processes. The organisation modelling approach proposed in this thesis is an extension of the one proposed in [Jacobson et al., 1995] for business process reengineering.

The ‘total view framework’ and holistic enterprise evolution
In terms of organisation modelling philosophy, the ‘total view framework’ is very similar to the holistic enterprise evolution [Yeh, 1997]. Yeh proposes three dimensions for managing change in an organisation: activity dimension, infrastructure dimension and co-ordination dimension. The activity dimension is the functional modelling of the organisation and corresponds to the organisation functional model in the ‘total view framework’. The infrastructure dimension is the resources modelling of the organisation and corresponds to the organisation physical model. The co-ordination dimension is the information modelling of the organisation, which in theory would be included in the organisation physical model. The ‘total view framework’ does not explicitly refer to an information model, although it captures all product, process and organisation information required to produce an information model. Generally and simplistically speaking, an information model consists of an entity relationship diagram applicable for modelling product, process or organisation. However it is a flat model and as such can get very complicated when modelling complex systems. For the ‘total view framework’ it was chosen to work
with modelling approaches that allowed hierarchical representation. Information modelling can be done with the set of notations used for organisation physical modelling and can be object of further development. It was chosen not to do it in this thesis because the focus of organisation physical modelling was to model resources, their relationships and attributes.

The 'total view framework' and holonic manufacturing systems

The matrices resulting from the 'concurrent structured analysis method' and their clustering provides a way of continuously updating the product, process and organisation structures to cope with the changes dictated by the environment (e.g. market). The clusters in the matrices, be they clusters of requirements, attributes, functions, components or development agencies, are calculated to be the most cohesive and the less coupled as possible. This would give them the characteristics of holons in terms of being autonomous, distributed and co-operative. If the organisation is structured according to these clusters it is believed that it will be more adaptable to rapid change. The existence of distributed holons does not invalidate the idea of hierarchy. Holons are stable intermediate forms before a hierarchy is formed [Simon, 1981] [Koestler, 1967]. Therefore, as described above, the 'total view framework' monitors the origination of holons and the modification of existing ones.

The 'total view framework' and ZOPH

ZOPH is a German acronym for goal system, product system, process system and agent system. It is a proposal of information model for the product development organisation (see Chapter 2, Section 2.2.1). Each system contains additional information in the form of attributes. It uses the same modelling notation for all systems and is implemented through a relational database.

Like the 'total view framework', ZOPH recognises that the elements to be integrated are product, process and organisation (named as agents). This corresponds to the integration dimension of the 'total view framework', although the process and organisation considered in the 'total view framework' are not only the product development process but also all product life cycle processes and not necessarily only the product development organisation.

ZOPH is implemented in a relational database. The 'total view framework' is implemented in a systems engineering environment that can provide a broader range of notations suitable to the peculiarities of product, process and organisation modelling requirements and capabilities such as configuration management not typically implemented in relational databases.

The 'total view framework' and product families

The 'total view framework' was not developed with product family in mind and further work would be necessary to make it more suitable for the development of product families. However some aspects identified in this thesis are also supportive of the product family idea. For example, the diesel PCS was developed in such a way that different sets of requirements were addressed by different elements in the whole diesel PCS system. The product itself met functionality and constraint requirements, calibration guide met the ease of calibration requirements, the structured approach met the reuse requirements, a team co-location met the development time requirements. The strategy of producing a product family
instead of a general purpose product is based on the same principles of making each product more suitable to meet a given set of requirements and consequently a different niche market.

The idea of clustering is also related to the idea behind product families. Some clusters of requirements and attributes would be common to all products in the family. Others would be used to differentiate the products in the family.

The models, requirements, attributes and matrices resulting from the utilisation of the 'concurrent structured analysis method' for one product can be easily re-used for other products in the family. Cradle, the SEE used to implement the 'total view framework' and the 'concurrent structured analysis method' have, for example, export and copy capabilities to allow inter-project re-use of information.

The 'total view framework', modularity and integration
The clustering method using as a clustering quality criteria the complexity index proposed in Chapter 9, Section 9.4 aims to find a balance between modularity and integration. It intends to find the best set of clusters minimising coupling between clusters and maximising cohesion within clusters. Further work may investigate the links between modularity and clusters of requirements and attributes.

11.8 Comparison with integrated development
The words 'integrated' and 'development' are often used in a phrase with one or more words between them, e.g., 'product', 'product and process', 'product, process and enterprise' and so on. This thesis proposes the generic term 'integrated development' and defines it as being the process of developing or evolving a product, its life cycle processes and their performing organisations in a concurrent and integrated manner seeking a balanced solution to meet stakeholder requirements (see Chapter 5, Figure 5-1). The premise is that when the developer ends the design of a product many elements of the product life cycle and of the organisations performing the life cycle processes are automatically determined by the constraints imposed by the product design. Traditionally, if the product design does not fit to the way its life cycle is implemented or if there is no resource to, for example, implement a given feature on the product, the product design is modified. This increases development time, cost as a lot of the product development is wasted and reveals poor product quality as many stakeholder requirements were neglected when developing the product. Therefore this thesis proposed a 'total view framework' to support integrated development. The framework has three dimensions: integration, analysis and structure. The integration dimension includes the elements to be integrated as a result of the development effort: product, process and organisation.

Many elements of the integrated development approaches reviewed in Chapter 2, Section 2.3, are embedded in the 'concurrent structured analysis method' proposed and demonstrated in this thesis:

- concurrency: the product, its life cycle processes and their performing organisations are modelled and analysed simultaneously throughout the requirements, functional and physical analysis processes. This method was described in Chapter 7 and demonstrated in Chapter 9 and 10 for automotive subsystems: diesel PCS and seat height adjust mechanism (SHM), respectively. The fact that the 'concurrent structured analysis method' was implemented in Cradle which works on a network
allows that developers working at different elements of product, process and organisation to work simultaneously.

- integration: integration between product, process and organisation is achieved through the concepts of requirements, attributes and relationships introduced in Chapter 6. Requirements for product, process and organisations are captured early in the integrated development process. Attributes are captured along the functional and physical analysis stages. Relationships between requirements and attributes and among attributes are captured. These relationships may happen even between attributes of different types (e.g. product, process and organisation). Thus, it is possible to investigate, for example, what organisation attributes may affect a product characteristic. Also relationships such as those between product symmetry and difficulty to assemble can also be identified. This was exemplified in Figures 10-37 to 10-40 in Chapter 10. The concepts of requirements and attributes were implemented as items of information in Cradle and the relationships as cross-reference. Cradle provided, then, automatically, links between any two chosen items of information.

- teamwork: in order to identify the relationships between requirements and attributes or among attributes, it is proposed in Chapter 6, Section 6.5.5 and demonstrated in Chapter 9, Section 9.2 a generic matrix format. The matrix also links requirements and attributes to development agencies. Development agencies may not be only organisations, they may be people in the development organisations who are responsible for a given requirement or attribute. The clustering approach proposed and demonstrated in Chapter 9, Section 9.4 can be applied to the definition of a team architecture as suggested in Section 9.4.5.

- use of information technology: although not mentioned in this thesis, the implementation of the 'concurrent structured analysis method' is not supposed to be carried out by one person or a small functional group in the organisation. It is intended that the SEE (systems engineering environment), e.g. Cradle, contain all project information in a central repository and that that information can be accessed and modified by the appropriate people in the development agencies via computer network. These people would produce the analysis models and would communicate to each other in order to identify the relationships among attributes. Cradle has the capability of creating a team structure, providing different access rights and defining user skills. More thoughts about an appropriate organisation to implement the 'concurrent structured analysis method' is object of further work.

11.9 Comparison with concurrent engineering

The 'total view framework' recognises that the result of the product development effort is not only the product itself but it is also, even if not planned, the product life cycle processes and their performing organisations. The fact is that the non-consideration of life cycle process requirements and organisation or resources constraints early in the development process will lead to more iterations of product concept and design. This contributes to increase product development time, increase costs and compromises competitiveness.

Like the 'total view framework', concurrent engineering also involves simultaneous consideration of product, process and organisation elements up front in the development process. [Prasad, 1996a]
proposes an information model integrating product, process and organisation (called PPO) to be used for product development.

The contribution of this thesis to concurrent engineering is to propose the concept of concurrent engineering to be applied, not only at the design or implementation stages of development, but also at the conceptual (functional) and requirements stages. Also, it is proposed here to consider concurrent engineering not only at the bottom layer of a product breakdown structure but also at system and subsystems layers. In summary the ‘total view framework’ extends the concept of concurrent engineering horizontally to include not only design or physical implementation but also concept and requirements, and vertically applying concurrent engineering at all layers of the product breakdown structure.

In order to demonstrate the feasibility of the ‘total view framework’, a ‘concurrent structured analysis method’ was presented in Chapter 7. The ‘concurrent structured analysis method’ consists of using well established modelling notations to perform requirements, functional and physical analysis of product, process and organisation, simultaneously.

Of particular interest to the concurrent engineering concept in terms of considering ‘from the outset, all elements of the product life cycle from concept to disposal, including quality, cost and user requirements’ is the requirements analysis method.

The requirements analysis method includes identifying all stakeholders affected by the product, its life cycle processes and their development organisation, identifying their concerns towards product, process and organisation, and capturing their requirements. This is done through the use of models containing the product or its mission in a central symbol surrounded by their stakeholders. For product, a Data Flow Diagram is used to consider the product in a central bubble surrounded by the stakeholders affected by the product attributes or interacting with the product over its life cycle (e.g., see Figures 7-16, 9-8 and 10-2 in Chapters 7, 9 and 10, respectively). For process, it is used a Behaviour Diagram with each life cycle process represented by a central rectangle surrounded by the stakeholders who are the sources of inputs, mechanisms or controls or the receivers of outputs (e.g., see Figures 7-17, 9-9 and 10-4 in Chapters 7, 9 and 10, respectively). For organisation, it is used a Use Case Diagram with the organisation in a central rectangle surrounded by all stakeholders playing a role towards the organisation (e.g. see Figures 7-18, 9-10 and 10-6, respectively). The central symbols are linked to the stakeholders by arrows representing stakeholder concerns towards product, process and organisation. These concerns are used to categorise the requirements captured. Those requirements will guide all following development stages.

The seat height adjust mechanism example (see Chapter 10) demonstrated that the links between product, process and organisation become explicit after physical analysis. The resulting physical attribute self-interaction matrix has shown clusters of interactions between product and process attributes, product and organisation attributes that had not become evident during functional analysis and requirements analysis. As functional attributes are related to physical attributes, it is possible to infer relationships among functional attributes, not captured during functional analysis, by transitivity. As these functional attributes are related to requirements, it is possible to identify the relationships among requirements by transitivity. Therefore, when evolving a product or even at the next iteration of the development process it is possible to have a picture of the scope of change necessary before any implementation is yet made.
As the 'concurrent structured analysis method' is implemented in a SEE (systems engineering environment) Cradle, a computer software tool, working on a network, the exchange of information among developers and between developers and subcontractors and suppliers is very simplified.

Cradle provides many project management capabilities, such as change control, baselining, reviewing, approval, registering. Taking the project management aspects for granted, this thesis focused on the engineering aspects of the 'total view framework'.

The demonstration of the 'total view framework' through the use of Cradle also facilitates computer integration with other computer-based support to concurrent engineering. Cradle can manage the configuration of information from CAD/CAM/CAE tools which can be called from within Cradle. This was not demonstrated in this thesis but is part of Cradle's capability.

The information resulting from the modelling-based analysis processes during the 'concurrent structured analysis method' is captured in Cradle and can be used for the concurrent engineering formal techniques (e.g. QFD, GT, VE). Although not straightforward, the document generation in Cradle can in theory be used to compose documents such as QFD, GT or VE matrices. This objective was not pursued in this thesis and can be object of further work. However, the choice of Cradle allows this sort of integration of formal concurrent engineering techniques.

In the following, the relationships between the work done in this thesis and some formal concurrent engineering techniques are discussed.

QFD
In this thesis, the matrices that capture the relationships between requirements and attributes and among attributes are inspired in the QFD matrix format. The only difference, in terms of format, is the inclusion of the development agency versus attribute matrix right below the roof of the traditional 'house of quality' matrix.

Whereas QFD jumps from the product characteristics straight to parts characteristics, the 'total view framework' proposes to deploy requirements from layer to layer until getting to the bottom layer of the product breakdown structure. This adapts QFD to complex product development. This was also an enhancement proposed by EQFD [Clausing, 1994] (see Chapter 2, Section 2.2.6.3), when it proposed that total systems expectations be deployed down to the subsystem level and then to the piece-part level.

In terms of contents, whereas QFD considers product attributes in the first two matrices and process attributes in the last two matrices (see Figure 2-15), relationships, in this thesis, are considered among product, process and organisation always in the same matrix. In QFD, product attributes are a function of process attributes. QFD, then, identifies, for example, what process attribute should be addressed in order to modify a product attribute and, therefore, affect customer satisfaction or dissatisfaction. This implies sequence: the product is designed with required attributes, how to set up the manufacturing process to produce those product attributes? The approach adopted by the total view framework seeks concurrency and is based on the hypothesis that product, process and organisation attributes may affect each other. Their relationships should be considered from the outset in order to provide a balanced solution not only to the customers, as in QFD, but also to other stakeholders.
QFD is viewed as an efficient way of communicating requirements down to the developers and implementers. Advantages in terms of shorter product development, less engineering changes, cheaper product with better quality are well reported in the literature. The same advantages are expected from the relationships implementation in the ‘total view framework’.

The information resulting from the concurrent structured analysis processes can be used to develop $Q^2$FD (Quantitative QFD, see Chapter 2, Section 2.4.4.1) matrices [Maier, 1996]. Requirements, functional and physical attributes and relationships are available from the concurrent structured analysis processes and are documented in Cradle or in a spreadsheet format. The relationships identified can be viewed as preliminary ones to be investigated further and quantified in order to compose the $Q^2$FD matrix. The clustering approach demonstrated in Chapter 9, Section 9.4 can be used to provide smaller and more focused $Q^2$FD matrices.

**Taguchi methods**

Product, process and organisation are integrated through the concepts of requirements, attributes and relationships. Some elements of attributes proposed in Chapter 6, Section 6.4.3 are common to Taguchi’s approach such as factor, value (level in the Taguchi methods) and tolerance. Also some attribute characteristics in Section 6.4.4 such as independence, hierarchy and minimum number of attributes are also proposed by Taguchi [Fowlkes & Criveling, 1995]. Definitions and developments related to signal attributes, noise attributes and control attributes deserve a closer look to the ones who want to go deeper in the emerging field of attributes engineering. This is object of further work.

The relationships presented in the relationship matrices proposed in Chapter 6, Section 6.5 and demonstrated in Chapter 9, Section 9.2 can be investigated, further qualified or even quantified by the use of Taguchi methods. Also attributes can be broken into factor, level and tolerance to have their relationships investigated and presented in the relationship matrices proposed.

**Axiomatic design**

The idea of functional independence heralded by axiomatic design [Suh, 1990] is used in this thesis as a means of managing complexity. The use of this idea was reinforced by the approach used by the diesel PCS group: distributing the requirements among various independent elements of the system solution instead of using the product to meet all requirements. The implementation of this idea in this thesis starts with the relationship matrices between requirements and attributes and among attributes. (Chapter 6, Section 6.5 and Chapter 9, Section 9.2). Then clustering was proposed in Chapter 6, Section 6.5.6 and demonstrated in Chapter 9, Section 9.4 as a means to reduce coupling between clusters of requirements, functions, functional attributes, parts or physical attributes. In Chapter 9, Section 9.4, it is demonstrated the use of clustering for finding a better software architecture in terms of maximising a complexity index. There, it is then proposed to apply the same approach for clustering functional attributes which would measure the functional independence proposed in axiomatic design.
Theory of inventive problem solving
The identification of relationships and attributes, proposed in Chapter 6, Section 6.5 and demonstrated in Chapter 9, Section 9.2 and 9.4, provides an indication of potential areas of contradiction to be further exploited by an approach such as TRIZ (Chapter 2, Section 2.4.4.4).

Design for assembly
Not only design for assembly but also other design for X methods provide the requirements of a given life cycle process (represented by the X) towards the product. The seat height adjust mechanism example (Chapter 10) illustrated the use of such requirements (see Figure 10-33) and also the relationships between life cycle process physical attributes and product physical attributes in Figure 10-40. Those relationships were anticipated by the requirements.

Group technology
The relationships between organisation physical attributes (resources attributes) and product physical attributes (parts characteristics) captured in the matrices proposed in Chapter 6 and demonstrated in Chapter 9 contain the necessary information for the development of GT charts.

Value engineering
The impact relationship between product functional attributes and product physical attributes or the traceability relationship between product functions in Cradle’s essential model and product parts in Cradle’s implementation model can be used to provide information to Value Engineering charts. The difference from the traditional VE approach is that, using the framework proposed in this thesis, Value Engineering could be applied at each layer of the product breakdown structure.

FMEA and hazard analysis
FMEA and hazard analysis are part of the concurrent structured functional analysis method proposed in Chapter 7, Section 7.5.2. However the FMEA and hazard analysis procedures proposed in this thesis are extended to include not only product but also process and organisation.

11.10 Comparison with systems engineering
The IEEE-1220-Std-1994 defines ‘systems engineering’ as: “an interdisciplinary collaborative approach to derive, evolve, and verify a life cycle balanced solution that satisfies customer expectations and meets public acceptability.” The definition of customer in the standard is broad. It includes developers, manufacturers, testers, distributors, operators, supporters, trainers, disposers and the general public. Therefore it can be replaced by a more generic term, ‘stakeholders’. Traditional systems engineering focused essentially on product development. In the 1990s systems engineering acquired a broader scope including the product and its life cycle processes in the definition of system. However, only requirements are captured for the life cycle processes during the systems engineering process.

The ‘total view framework’ proposes a broader scope for systems engineering. It includes in the system solution the product, its life cycle processes and [some of] their performing organisations. All these
elements are in the scope of the development effort and are object of the application of the systems engineering process.

The analysis dimension of the 'total view framework' contains the requirements, functional and physical analysis processes which are the core of the systems engineering process as defined by the IEEE-1220-Std-1994 standard.

The structure dimension of the 'total view framework' is the hierarchy dimension composed by the layers of the system breakdown structure. It is a principle of systems engineering that 'the elements of a system may themselves be systems, and every system may be part of a larger system in a hierarchy.'

The 'total view framework' can be defined in terms of systems engineering; it consists of the application of the systems engineering process concurrently to a product, its life cycle processes and [some of] their development organisations at all layers of the product breakdown structure.

The 'total view framework' contributes to bring systems thinking into practice as it encompasses product, process and organisation elements. For example, Ford Motor Company's FPDS manuals defines systems thinking as being: thinking first of optimising the whole Ford organisation, then a vehicle program, and then a particular job that one is doing. The 'total view framework' uses systems thinking to deliver a balanced system solution to stakeholders.

The 'concurrent structured analysis method' is developed from a compilation of various systems engineering process from various sources (e.g. IEEE-1220, EIA-632, [Stevens et al., 1998])

The implementation of the 'total view framework' uses a commercial software tool classified as a systems engineering environment.

The 'total view framework' and the 'concurrent structured analysis method' follows the same trend that is guiding the evolution of systems engineering standards, methods and tools: a broader scope. Standards are evolving towards a broader scope of the definition of the term 'system' and a more generalised definition of the term systems engineering. Methods and tools are evolving from simply design tools to complete development and management tools. Cradle, the tool chosen for the implementation of the 'total view framework' includes requirements management, functional analysis, physical analysis and, even, project management capability. Therefore it can provide links between elements of the requirements, functional and physical models allowing traceability and impact analysis.

The 'concurrent structured analysis method' uses structured analysis techniques to move from requirements to the physical elements of product, process and organisation. The use of structured techniques have the advantage of providing: high maintainability; effective partitioning; extensive use of graphics; differentiation between logical and physical considerations; existence of a logical model; requirements partitioning before specification; interfaces traceability; systematic description of logic. All these characteristics are incorporated into the 'concurrent structured analysis method'.

All notations used for the implementation of the method are from the software engineering arena and are adapted to product, process and organisation modelling. Cradle, the systems engineering environment used, is also a CASE (computer aided software engineering) tool. Software engineering has greatly contributed for the revival of systems engineering during the 1990s.
11.11 Critical analysis

This section performs a critical analysis of the thesis: delineating its scope, stating what has been addressed and what has not; highlighting where there is room for improvement and; emphasising its contribution.

This thesis can be seen as part of a broader research area: the management of complexity when developing or evolving complex products to meet requirements from a very dynamic market place. Other topics within this same line of research include: modularity, product families, evolvability. These are all state-of-the-art research topics, especially in the product development arena.

The scope of this thesis includes product development, integrated development, systems engineering and concurrent engineering. This thesis is a research in the field of engineering. Therefore, managerial aspects related to integrated development (e.g. teamwork) and to systems engineering (e.g. configuration management) were approached only superficially in this work.

The main contributions of this thesis are:

- to justify, propose and demonstrate the adoption of a broader scope for systems engineering, a traditional discipline used for developing complex products. The scope proposed include not only the product but also, its life cycle processes and their performing organisations;
- to justify propose and demonstrate the applicability of concurrent engineering not only to the design stages of product development but also to the earlier stages of requirements and functional analysis and using it as an instrument to manage complexity. Concurrent engineering was also applied at all levels of the product breakdown structure, not only to the lowest or component level as usual;
- to integrate systems engineering and concurrent engineering in the same integrated development framework, called the 'total view framework';
- to justify, propose and demonstrate the integration of product, process and organisation through the concepts of requirements, attributes and relationships;
- to justify, propose and demonstrate a procedure for complexity management based on the level of coupling between functions, functional attributes or requirements obtained by investigating the level of coupling between physical parts or attributes;
- to propose and demonstrate a 'concurrent structured analysis method' that implements the integrated development framework. The 'concurrent structured analysis method' is a requirements driven method and perform requirements analysis, functional analysis and physical analysis, simultaneously on product, process and organisation. It uses structured analysis techniques;
- to propose and demonstrate a way by which a total view of product, process and organisation can be provided. The way chosen was modelling product, process and organisation by using modelling notations originally applied to software development. The notations were chosen to cover all views necessary to model product, process and organisation according to specialised literature;
- to adapt software development notations for hardware, process and organisation modelling;
- to demonstrate how commercially available tools can be used to implement the 'total view framework' and the 'concurrent structured analysis method' on software and hardware (mechanic) automotive subsystems.
This thesis has not addressed the following issues:

- organisation for the actual implementation of the ‘total view framework’ in a company environment. Such an organisation would include a centralised repository of information being shared by people involved in the development of product, process and organisation at all levels of the product breakdown structure;

- project management. The control functions of the systems engineering process were kept outside the scope of this thesis. Considerations on how to document, plan, review, control, decide during an integrated development project that uses the ‘total view framework’ were not specifically addressed in this thesis. The ‘total view framework’ is essentially an engineering framework. However, Cradle, the systems engineering environment software package used for implementing the framework, contains project management capability.

Room for improvement exists in the following aspects of this thesis:

- a complete example including a product, all of its life cycle processes and [some of] their performance organisations was not developed. The only objective with the examples developed in this thesis was to demonstrate the framework and method. There was no intention to produce complete examples.

- the size of the relationship matrices resulting from the ‘concurrent structured analysis method’ can be very big (of the order of hundreds or thousands) challenging the pragmatism of the approach. Better criteria for attributes selection and clustering may help to focus on the important issues. Further work in the area of attribute engineering is necessary.

- the implementation of the ‘concurrent structured analysis method’ is a labour intensive activity and is unlikely to be suitable to Tier 2 supplier-type of company (e.g. SHM producer). The overhead necessary to produce the models developed in this work is not compatible with the size of a Tier 2 supplier-type of company. A way of identifying requirements and attributes compatible with the company’s resources must be pursued. One should remember that the objective of the method was to get to a complete (or as complete as possible) set of requirements and attributes.

- the demonstration of the complexity analysis procedure, in Chapter 9 for the gasoline PCS example, included only product parts (software modules) and product functions. A broader example including product, process and organisation elements can be developed following the same procedure. For the sake of demonstrating the procedure and how it accomplished its objectives, the work performed in the thesis was considered enough.

Having drawn together the issues covered by this thesis, evaluated this work against the needs, trends and other approaches for coping with complexity whilst developing complex products and performed a critical analysis, the following chapter draws the conclusions of this thesis.
Chapter 12
Conclusions and further work

This chapter:

• draws the conclusions of the thesis;
• summarises a critical analysis of the thesis;
• identifies opportunities for further work.

12.1 Conclusions

In the light of the objectives of Chapter 1 and discussion and evaluation in Chapter 11, the following main conclusions are drawn from the work documented in this thesis:

Needs: the literature review and the case studies were used to characterise the needs for a 'total view framework' for the integrated development of complex products.

Concept: a 'total view framework' that encompasses systems engineering and concurrent engineering and the concepts of requirements, attributes and relationships provided the elements to meet the needs.

Design: a 'concurrent structured analysis method' was developed by reviewing structured analysis techniques and standardised systems engineering processes and adapting them to fit the elements and the structure of the 'total view framework'.

Implementation: a commercially available systems engineering environment, a graph partitioning software and a commercial spreadsheet tool were adapted and used to implement three examples from the automotive industry: the diesel PCS, the gasoline PCS and the seat height adjust mechanism (SHM). The examples demonstrate the utilisation of the 'concurrent structured analysis method' within the 'total view framework'.

Evaluation: framework, method and implementation were evaluated against the identified needs.

The following sections expand the conclusions listed above.

12.1.1 The needs for a 'total view framework'

The review on complexity and complexity management approaches, in Chapter 2, identified:

• The need to encompass all aspects of a complex problem in order to manage complexity escalation.
• The need to use integrated development to have a team of knowledgeable individuals who together can encompass all aspects of the problem.
• The need to accommodate in a product development framework the broader scope that integrated development is assuming including process and organisation aspects as indicated in the review on integrated development.

The trends in the space and automotive industry, in Chapter 3, indicated:

• The need to adapt the traditional document-passing, performance-focused systems engineering approach used in the space industry to a more trade-off based systems engineering approach that seeks
a balanced solution for life cycle cost, development lead-time and risk management, besides performance requirements.

- The need to incorporate concurrent engineering principles into the traditionally rigorously sequentially phased space program.

- The need to move from the traditional component approach to a systems engineering approach in the automotive industry keeping the principles of concurrent engineering already in use in that industry.

The gasoline and diesel PCS case studies documented in Chapter 4 highlighted:

- The need to maintain traceability from stakeholders to their requirements, from these to functions, from these to implementation elements, so that a change in one of them can be quickly responded by the others.

- The need to consider not only the product as the solution of the complex problem but also its life cycle processes and their performing organisations.

- The need to keep visibility of the interactions or relationships among the various elements of the solution implementation, including not only the product elements but also the process and organisation elements.

- The need to capture requirements early in the development process.

- The need to use a structured approach when moving through the requirements analysis, functional analysis and synthesis stages of the product development process.

In Chapter 11, all needs listed above were grouped into three main issues:

- **Scope** - the need to identify up-front in the integrated development of a complex product what is the scope of the system solution delivered by an integrated development process;

- **Integration** – the need to integrate requirements, functions and implementation elements throughout the integrated development process and to have an integrated solution;

- **Complexity management** – the need to manage complexity throughout the integrated development process considering that the system solution will be in a very dynamic environment and, as a consequence, will be highly likely to change.

### 12.1.2 The conception of the framework and its main elements

In order to address the scope issue:

- A generic model for the integrated development process (see Figure 5-1 in Chapter 5) was developed based on the assumption that the effort of integrated development results in not only the product but also its life cycle processes and their performing organisations.

- The generic model consists of: stakeholders have needs; some of these needs are perceived and considered as requirements by the product development organisation (also called ‘development agency’); based on the requirements, the development agency develops the product, its life cycle processes and [some of] their performing organisations; product, process and organisation have
attributes; those attributes affect stakeholder satisfaction or dissatisfaction; the cycle restarts as stakeholders have new requirements or as other stakeholder needs are perceived by the product development organisation.

- The integrated development of complex products must be a requirements driven process.
- The integrated development of complex products integrates product, life cycle processes and associated organisations (or at least the development organisation) development.
- The integrated development of complex products searches for a balanced solution trading-off product, processes and organisation attributes as the emergent properties of a whole integrated system.

The ‘total view framework’ (see Figure 5-2 in Chapter 5) was conceived to provide the elements and structure for the integrated development process:

- The ‘total view framework’ consists of the application of the systems engineering process concurrently to a product, its life cycle processes and [some of] their performing organisations at all layers of the product breakdown structure.
- The elements of the ‘total view framework’ are placed on three dimensions: the integration dimension, the analysis dimension and the structure dimension. The integration dimension contains product, process and organisation elements to be integrated. The analysis dimension contains the core sub-processes of the systems engineering process [IEEE-1220-Std-1994]: requirements analysis, functional analysis and physical analysis. The structure dimension refers to the hierarchy layers of the complex product breakdown structure (see Figure 5-2 in Chapter 5).
- The scope of the ‘total view framework’ encompasses the scope of systems engineering (SE) and that of concurrent engineering (CE). The scope of systems engineering, according to the EIA-632 standard and the IEEE-1220-Std-1994, includes the product development through requirements analysis, functional analysis and physical analysis and only requirements analysis for process development, at all levels of the product breakdown structure. The scope of concurrent engineering includes product, process and organisation but only at the physical analysis stage of the systems engineering process and at the bottom level or component level of the product breakdown structure.

In order to address the integration issue:

- Integration of product, process and organisation takes place through the concepts of requirements, attributes and relationships, documented in Chapter 6.
- Requirements and attributes represent the interface between the stakeholders in the problem domain and the system solution elements in the solution domain.
- Requirements and attributes may refer to product, process and organisation serving as a unifying language among different elements of the system solution.
- Relationships between product, process and organisation such as those described by many concurrent engineering methods and tools can be captured through the identification of the relationships between requirements and attributes and among attributes.
• Requirements must be captured and analysed early in the product development cycle as learned from the diesel PCS case study.

• Attributes may serve as a means of communicating between layers of the product breakdown structure and as the basis for verifying the product.

• This thesis proposes the use of matrices capturing the relationships between sources of requirements and requirements and, sources of attributes and attributes in order to capture the relationship between the people involved in product development.

• The relationship matrices adopt a QFD-type format and as such intend to be an efficient means of communication between stakeholders and development agencies and among development agencies at different layers of the product breakdown structure.

In order to address the issue of complexity management:

• The level of coupling among requirements or among functional attributes can be used for complexity management.

• For evaluating the level of coupling, and using it to manage complexity, the following procedure is proposed in this thesis: to identify relationships between requirements and attributes; to identify relationships among attributes; to identify relationships among requirements by transitivity; to cluster requirements in order to minimise a complexity metrics.

• Rationale for using level of coupling as a complexity indicator comes from the gasoline and diesel PCS case studies, from the Suh's work on axiomatic design and from Altshuller's work on the theory of inventive problem solving.

12.1.3 The design of a method within the framework

• This thesis proposes a method that extends the diesel PCS structured analysis approach to include hardware, life cycle process and organisation analysis, addressing not only operation requirements but also and simultaneously non-operation ones. The method is called 'concurrent structured analysis method' (see Figure 7-2 in Chapter 7 for the method overview).

• The method supports the 'total view framework' by performing requirements, functional and physical analysis concurrently for product, its life cycle processes and their performing organisations at all layers of the product breakdown structure.

• The outputs of the method are the requirements and attributes used for integration of product, process and organisation. Once requirements and attributes are known, the relationships between them and among attributes are captured. The identification of requirements and attributes and of the relationships among them is done early in the integrated development of complex products.

• The requirements, functional and physical analysis processes are adapted mainly from modern systems engineering standards, IEEE-1220-Std-1994 and EIA-638.

• Examples of adaptation from standardised requirements analysis procedure are: up-front identification of potential life cycle processes for the end-product to be; analysis of life cycle process
scenarios instead of the traditional analysis of operational scenarios; up-front identification of life cycle process performing organisations.

- Adaptations from standardised functional analysis procedures occur only in terms of their extension to include processes and organisations.

- Physical analysis can be understood as the analysis of the implementation of products, processes and organisation constituents of the whole system. Physical analysis can be done either during the synthesis process of products, processes and organisations, in support of it, or after it, in order to model the resulting physical architecture.

- Requirements, functional and physical analysis processes are carried out through the simultaneous modelling of product, process and organisation. The modelling techniques used result from an expansion of well-established software development techniques.

- The modelling approaches used in this thesis were selected in order to provide a total view of product, process and organisation, and, therefore, to make possible that a complete set of requirements and attributes would be derived.

- The set of views necessary to produce a total view was available in the literature:
  - for product: functional modelling: activity, data and state [Calvez, 1993]; physical modelling [Stevens et al., 1998]: behaviour, structure and layout;
  - for process [Curtis et al., 1992]: functional, behavioural, organisational and informational;
  - for organisation [Vernadat, 1996]: function, information, resource and structure.

- For product modelling, the Yourdon approach is adapted. Adaptations include:
  - energy and material couplings in addition to only information (data) couplings;
  - physical connections in addition to only functional logical connections;
  - functional and physical attributes, instead of specifications, are associated to each element of the functional and physical models, respectively;
  - for requirements analysis, a DFD is used to develop a context diagram with the end product mission/objective in a central bubble surrounded by the stakeholders (rectangles) who interact with the product. The flows linking the central bubble to the stakeholders represent the stakeholders concerns (e.g. functionality, assemblability) towards the product. The direction of these flows is irrelevant.

- For process modelling, IDEF0 notation is adapted on a Behaviour Diagram following the SADT methodology. The process requirements analysis is represented by an adapted IDEF0 notation in which the function of the process to be modelled is linked to the stakeholders who are sources of inputs, control and mechanisms and to the destination of outputs. The direction of the links is irrelevant and these links represent the concerns of these stakeholders towards the process under modelling.

- For the organisation modelling, the approach proposed by Jacobson (1995) is extended by using the UML (Unified Modelling Language) notation for the object model. The UML notation is adapted for organisation modelling.
scenarios instead of the traditional analysis of operational scenarios; up-front identification of life cycle process performing organisations.

- Adaptations from standardised functional analysis procedures occur only in terms of their extension to include processes and organisations.

- Physical analysis can be understood as the analysis of the implementation of products, processes and organisation constituents of the whole system. Physical analysis can be done either during the synthesis process of products, processes and organisations, in support of it, or after it, in order to model the resulting physical architecture.

- Requirements, functional and physical analysis processes are carried out through the simultaneous modelling of product, process and organisation. The modelling techniques used result from an expansion of well-established software development techniques.

- The modelling approaches used in this thesis were selected in order to provide a total view of product, process and organisation, and, therefore, to make possible that a complete set of requirements and attributes would be derived.

- The set of views necessary to produce a total view was available in the literature:
  - for product: functional modelling: activity, data and state [Calvez, 1993]; physical modelling [Stevens et al., 1998]: behaviour, structure and layout;
  - for process [Curtis et al., 1992]: functional, behavioural, organisational and informational;
  - for organisation [Vernadat, 1996]: function, information, resource and structure.

- For product modelling, the Yourdon approach is adapted. Adaptations include:
  - energy and material couplings in addition to only information (data) couplings;
  - physical connections in addition to only functional logical connections;
  - functional and physical attributes, instead of specifications, are associated to each element of the functional and physical models, respectively;
  - for requirements analysis, a DFD is used to develop a context diagram with the end product mission/objective in a central bubble surrounded by the stakeholders (rectangles) who interact with the product. The flows linking the central bubble to the stakeholders represent the stakeholders concerns (e.g. functionality, assemblability) towards the product. The direction of these flows is irrelevant.

- For process modelling, IDEF0 notation is adapted on a Behaviour Diagram following the SADT methodology. The process requirements analysis is represented by an adapted IDEF0 notation in which the function of the process to be modelled is linked to the stakeholders who are sources of inputs, control and mechanisms and to the destination of outputs. The direction of the links is irrelevant and these links represent the concerns of these stakeholders towards the process under modelling.

- For the organisation modelling, the approach proposed by Jacobson (1995) is extended by using the UML (Unified Method Language) notation for the object model. The UML notation is adapted for organisation modelling.
12.1.4 Examples of method implementation

- The implementation of the examples were carried out by adapting a commercial systems engineering environment, called Cradle and a graph partitioning software, called Chaco-2.0. An Excel spreadsheet was also used. Figure 8-4 in Chapter 8 provides an overview about the way those tools were used.

- Cradle was selected because:
  - it has multi-notational capability;
  - it does not impose a specific development methodology;
  - it covers the whole systems engineering process from requirements capture to physical architecture definition;
  - it allows integration of stakeholders and sub-contractors into the systems engineering process.

- Cradle’s adaptations include:
  - extension of Cradle’s default set of items of information to include functional attributes, physical attributes, stakeholders and development agencies;
  - adaptation of Cradle’s modelling notations from software development notations to model hardware, processes and organisations;
  - creation of different relationship types to cover hierarchy, traceability and impact relationships, using Cradle’s own capability.

- Cradle was used for modelling, requirements management, attributes management and relationship management.

- Chaco-2.0 is a graph partitioning software originally developed for parallel computing architecture. This work represents a novel application for Chaco. In this work it was used to identify the best software architecture in terms of minimisation of coupling and maximisation of cohesion. This work also proposes to use Chaco-2.0 for clustering requirements and attributes based on the strength of their relationships.

- Excel was used for the representation of the relationship matrices and for the calculation of a complexity index.

- The ‘concurrent structured analysis method’ and the ‘total view framework’ were demonstrated through three examples from the automotive industry: the diesel PCS, the gasoline PCS and the seat height adjust mechanism (SHM).

- The diesel PCS example, documented in Chapter 9, demonstrates the use of the relationship matrices for deploying down requirements and attributes from the car layer down to the powertrain layer and then to the PCS layer. Matrices with the relationships between development agencies and requirements and attributes were also demonstrated.

- The diesel PCS example demonstrates the use of Cradle for the requirements, functional and physical analysis of product, process and organisation for the development of a software subsystem — the EGR control system.
• The gasoline PCS example, documented in Chapter 9, proposes and demonstrates a re-structuring procedure that uses as quality criterion the minimisation of a complexity index. The complexity index is calculated in such a way that reflects coupling minimisation and cohesion maximisation. The re-structuring procedure is named ‘complexity analysis procedure’ and is applied to the Idle Speed Control software strategy code. The Idle Speed Control is a subsystem of the gasoline PCS that evolved over the period 1978-1995.

• The gasoline PCS example demonstrated the use of Chaco-2.0, a graph partitioning algorithm originally developed for parallel computing applications.

• The complexity analysis procedure provided evidence that the way Ford partitions the Idle Speed Control software could be improved. The partition reflects the allocation of software modules to a given software subsystem or feature. Partitioning becomes important if one remembers that PCS development department is organised by feature and that unnecessary coupling means unnecessary communication overhead cost that could be avoided.

• Chaco’s partitioning was closer to the theoretical functional partitioning than the Ford’s one. This is an indication that the actual software could be partitioned in a more modular way. Ford’s partitioning was making the functions more coupled than they needed to be.

• The gasoline PCS example suggests that monitoring a complexity index may avoid future need for re-engineering.

• The thesis also proposes the use of the complexity analysis procedure for re-structuring of self and cross interaction matrices of requirements, functional attributes, physical attributes and development agencies. Other applications include process architecture, team architecture and overall system optimisation.

• The seat height adjust mechanism (SHM) example, documented in Chapter 10, demonstrated the applicability of the ‘total view framework’ and ‘concurrent structured analysis method’ to an essentially hardware product, a mechanical product.

• The SHM example illustrated the use of attributes for describing a hardware product and their usefulness in providing integration between product, process and organisation.

• Some lessons learned with the SHM example are:
  • most useful model for generating attributes were those containing all elements attributes should describe;
  • care must be taken for avoiding confusion an duplication of work when modelling process and organisation;
  • the quality of models is determined by the quality of the information available about the object to be modelled;
  • the quality of models determine the quality of requirements and attributes obtained;
  • there is still the need for researching some guidelines to generate attributes from the models in a more systematic way;
  • the quality of relationships depends on the quality of the requirements and attributes captured;
relationships among product, process and organisation were more evident during physical analysis. Relationships among requirements and among functional attributes of product, process and organisation have to be identified by transitivity;

• despite the many simplifications made, the matrices resulting from the SHM example were very big (order around 150). The effort necessary to implement the SHM example suggests that the overhead necessary is not compatible to the size of a Tier 2 supplier. Only a Tier 1 supplier or the car manufacturer itself could afford going through this process.

• the example suggests the need for a move towards a more pragmatic approach that takes into account the constraints of the real manufacturing business environment.

12.1.5 Evaluation of framework, method and implementation

The evaluation against the trends in the space industry indicates that:
• The ‘total view framework’ and the ‘concurrent structured analysis method’ support the trends in the space industry.
• The ‘total view framework’ recognises the need to consider more than one stakeholder and the requirements analysis method identifies those stakeholders and capture their requirements very early in the integrated development process.
• The ‘total view framework’ and the matrices resulting from the concurrent structured analysis processes identify potential relationships among attributes that could be further investigated through simulations, for example. This complies with the trend to do more analysis up-front and less testing.
• The framework and method address the objectives of a ‘faster, cheaper and better’ space program within a manageable risk.

The evaluation against the trends in the automotive industry indicates that:
• The ‘concurrent structured analysis method’ supports the move towards a more structured requirements capture and analysis processes.
• The ‘concurrent structured analysis method’ accomplishes the need to anticipate life cycle process requirements.
• The ‘total view framework’ provide a framework for the integrated use of CAD, CAM, CAE and PDM tools.
• The structure dimension of the ‘total view framework’ complies with the trend towards a hierarchical vehicle structure and tier-supplier approach.
• The analysis dimension of the ‘total view framework’ complies with the trend towards more engineering analysis up-front.
• The contribution of the ‘concurrent structured analysis method’ to teamwork is the fact that it proposes to associate to each technical requirement and to each attribute the people in the development agencies actually responsible for obtaining that requirement or attribute. As relationships among attributes are identified, attributes can be clustered and to those clusters would correspond a team of people responsible to deliver that set of attributes in a balanced solution.
• The results of the analysis processes, i.e models, requirements, attributes and relationships, can be reused from vehicle program to vehicle program.
The evaluation against the trends in both industries indicates that:

- The 'total view framework', as a systems engineering and concurrent engineering approach to the integrated development of complex products, meets the trends towards integrated development of both industries.

The evaluation against the needs highlighted from the gasoline case study indicates that:

- The 'total view framework', through the concept of relationships meets the need for traceability and impact analysis.
- The integration dimension of the 'total view framework' meets the need for considering product, process and organisation as elements of the system solution.
- The models and matrices resulting from the 'concurrent structured analysis method' address the need for visibility of relationships.

The evaluation against the needs highlighted by the diesel PCS case study indicates that:

- The 'concurrent structured analysis method' meets the need for structure highlighted by the diesel PCS by using structured analysis techniques.
- The concepts of requirements, attributes and relationships are used to integrate product, process and organisation in the system solution.
- The concept of relationships allow continuous evaluation of the level of coupling of a given solution.

The results of a comparison of the framework and method with general approaches to manage complexity, documented in Chapter 9, include the following:

- The dimensions of the 'total view framework' – analysis, integration and structure - correspond to the factors through which complexity emerges [Hitchins, 1996] – variety, connectedness and disorder -, respectively.

- The 'total view framework' has a broader scope than Pugh's total design. Whereas the 'total view framework' includes product, process and organisation within the development effort, Pugh's total design includes only the product.

- The 'total view framework' has a broader scope and a broader objective than total quality control. Total quality control focuses on the product and processes performed by a given organisation. The 'total view framework' includes all life cycle processes, not only those performed by a specific organisation of interest. The 'total view framework' focuses on stakeholders whereas total quality control focuses on customers.

- The concepts of requirements, attributes and relationships are used to monitor the level of coupling of a solution that includes product, process and organisation in order to promotes incremental changes avoiding to get a point where dramatic reengineering is needed.

The results of a comparison of the framework and method with integrated development, documented in Chapter 9, include the following:

- This thesis proposes a generic model and a framework for integrated development including product, process and organisation in its scope. Besides the 'concurrent structured analysis method' supports
many elements of the integrated development approaches reviewed in Chapter 2, such as concurrency, integration, teamwork and use of information technology.

The results of a comparison of the framework and method with concurrent engineering, documented in Chapter 9, include the following:

- The ‘total view framework’ extends the concept of concurrent engineering horizontally, from only design or physical implementation to include also concept and requirements, and vertically applying concurrent engineering at all layers of the product breakdown structure.
- The ‘concurrent structured analysis method’ consists of using well established modelling notations to perform requirements, functional and physical analysis of product, process and organisation, simultaneously.
- The requirements analysis method includes identifying all stakeholders affected by the product, its life cycle processes and their development organisation, identifying their concerns towards product, process and organisation, and capturing their requirements. Those requirements guide all following development stages.
- Relationships between product, process and organisation are more evident after the physical analysis. Transitivity, though, the relationships among functional attributes and among requirements can be inferred.
- The exchange of information among developers and between developers and subcontractors and suppliers is very simplified, as the ‘concurrent structured analysis method’ is implemented using Cradle.
- The demonstration of the ‘total view framework’ through the use of Cradle also facilitates computer integration with other computer-based support to concurrent engineering.
- The information resulting from the modelling-based analysis processes during the ‘concurrent structured analysis method’ is captured in Cradle and can be used for the concurrent engineering formal techniques (e.g. QFD, GT, VE).

The results of a comparison of the framework and method with systems engineering, documented in Chapter 9, include the following:

- The ‘total view framework’ proposes a broader scope for systems engineering. It includes in the system solution the product, its life cycle processes and [some of] their performing organisations. All these elements are in the scope of the development effort and are object of the application of the systems engineering process.
- The analysis dimension of the ‘total view framework’ contains the requirements, functional and physical analysis processes which are the core of the systems engineering process as defined by the IEEE-1220-Std-1994 standard.
- The structure dimension of the ‘total view framework’ is the hierarchy dimension composed by the layers of the system breakdown structure. It is a principle of systems engineering that ‘the elements
of a system may themselves be systems, and every system may be part of a larger system in a
hierarchy.'

- The 'concurrent structured analysis method' is developed from a compilation of various systems
  engineering process from various sources (e.g. IEEE-1220, EIA-632, [Stevens et al., 1998])

- The implementation of the 'total view framework' uses a commercial software tool classified as a
  systems engineering environment.

- The 'total view framework' and the 'concurrent structured analysis method' follows the same trend
  that is guiding the evolution of systems engineering standards, methods and tools: a broader scope.

- All notations used for the implementation of the method are from the software engineering arena and
  are adapted to product, process and organisation modelling. Cradle, the systems engineering
  environment used, is also a CASE (computer aided software engineering) tool. Software engineering
  has greatly contributed for the revival of systems engineering during the 1990s.

### 12.2 Results from critical analysis

#### 12.2.1 Contribution

- To justify, propose and demonstrate the adoption of a broader scope for systems engineering, a
  traditional discipline used for developing complex products. The scope proposed include not only the
  product but also, its life cycle processes and their performing organisations.

- To justify propose and demonstrate the applicability of concurrent engineering not only to the design
  stages of product development but also to the earlier stages of requirements and functional analysis
  and using it as an instrument to manage complexity. Concurrent engineering was also applied at all
  levels of the product breakdown structure, not only to the lowest or component level as usual.

- To integrate systems engineering and concurrent engineering in the same integrated development
  framework, called the 'total view framework'.

- To justify, propose and demonstrate the integration of product, process and organisation through the
  concepts of requirements, attributes and relationships.

- To justify, propose and demonstrate a procedure for complexity management based on the level of
  coupling between functions, functional attributes or requirements obtained by investigating the level
  of coupling between physical parts or attributes.

- To propose and demonstrate a 'concurrent structured analysis method' that implements the integrated
  development framework. The 'concurrent structured analysis method' is a requirements driven
  method and perform requirements analysis, functional analysis and physical analysis, simultaneously
  on product, process and organisation. It uses structured analysis techniques.

- To propose and demonstrate a way by which a total view of product, process and organisation can be
  provided. The way chosen was modelling product, process and organisation by using modelling
  notations originally applied to software development. The notations were chosen to cover all views
  necessary to model product, process and organisation according to specialised literature.
• To adapt software development notations for hardware, process and organisation modelling.
• To demonstrate how commercially available tools can be used to implement the ‘total view framework’ and the ‘concurrent structured analysis method’ on software and hardware (mechanic) automotive subsystems.

12.2.2 Issues not addressed
• Organisation for the actual implementation of the ‘total view framework’ in a company environment. Such an organisation would include a centralised repository of information being shared by people involved in the development of product, process and organisation at all levels of the product breakdown structure;
• Project management. The control functions of the systems engineering process were kept outside the scope of this thesis. Considerations on how to document, plan, review, control, decide during an integrated development project that uses the ‘total view framework’ were not specifically addressed in this thesis. The ‘total view framework’ is essentially an engineering framework. However, Cradle, the systems engineering environment software package used for implementing the framework, contains project management capability.

12.2.3 Topics for improvement
• A complete example including a product, all of its life cycle processes and [some of] their performance organisations was not developed. The only objective with the examples developed in this thesis was to demonstrate the framework and method. There was no intention to produce complete examples.
• The size of the relationship matrices resulting from the ‘concurrent structured analysis method’ can be very big (of the order of hundreds or thousands) challenging the pragmatism of the approach. Better criteria for attributes selection and clustering may help to focus on the important issues. Further work in the area of attribute engineering is necessary.
• The implementation of the ‘concurrent structured analysis method’ is a labour intensive activity and is unlikely to be suitable to Tier 2 supplier-type of company (e.g. SHM producer). The overhead necessary to produce the models developed in this work is not compatible with the size of a Tier 2 supplier-type of company. A way of identifying requirements and attributes compatible with the company’s resources must be pursued. One should remember that the objective of the method was to get to a complete (or as complete as possible) set of requirements and attributes.
• The demonstration of the complexity analysis procedure, in Chapter 8 for the gasoline PCS example, included only product parts (software modules) and product functions. A broader example including product, process and organisation elements can be developed following the same procedure. For the sake of demonstrating the procedure and how it accomplished its objectives, the work performed in the thesis was considered enough.
12.3 Opportunities for further work

Quantitative relationships. The relationships in this thesis are treated as binary relationships. There is only indication of whether there is a relationship between two items of information or not. Thus there is an opportunity for further work in quantifying those relationships in order to have a picture closer to reality of the impact of changes in the values of all attributes involved. A starting point to such a research can be the Quantitative QFD or Q$^2$FD [Maier, 1996].

Performance modelling. The quantitative approach may not have only the matrix format. Cradle has the capability of creating performance parameters for each model symbols in DFDs and in FBDs in Cradle's implementation model. These performance parameters accept values, tolerances and formula so that the impact of a change in one parameter can be investigated in another parameter. These performance parameters can also be fed into Cradle by other analysis tools, e.g. CAE, MatLab. Performance modelling could be done very early in the product life cycle and could seek balance among product, process and organisation parameters.

Attributes engineering. In this thesis, functional and physical attributes were captured from functional and physical models, respectively, and followed the hierarchical structure of the model symbols. However no systematic method was used to assure that that was the best set of attributes to be gathered. Similarly to the field of requirements engineering, attributes engineering would provide concepts, framework, approach, method and tools for attributes capture and structuring. Attributes engineering would be especially useful for products evolving incrementally over time. With a good set of attributes, the product evolution can be followed and planned. Better criteria for attributes selection and clustering may help to focus on the important issues. The issue of attribute independence can be further investigated using as a starting point the work [Fowlkes & Creveling, 1995] on robust design. Attribute structuring can have its basis on the work [Warfield, 1976] on interpretive structural modelling (see Chapter 2, Section 2.3.2).

Product family. The 'total view framework' was developed for projects supposed to produce one product. However, product families are becoming a strategic approach to meet a more diverse range of requirements and therefore increase market share. To accommodate the work presented in this thesis to include the scope of a product family is object for further work. A starting point is the work [Lam, 1999] on a product family approach to systems engineering.

Organisation for the 'total view framework'. The issue of how to organise resources in a company in order to carry out the 'total view framework' was not addressed in this thesis. Elements of the organisation would be a central repository (e.g. Cradle) of information with distributed access over a network and specialised people working in a team environment. Similarly to many other approaches to product development, the 'total view framework' requires an appropriate organisation for its success.

Automatic document generation. The information gathered to produce the models and matrices of the 'total view framework' can be re-used and formatted to compose QFD, GT, VE matrices. There is no need to capture further information to produce those charts. Cradle has a document generation capability that needs to be further investigated, and possibly, further developed, to accommodate to the legitimate needs of providing different views with the same set of information.
**Complexity management.** The idea of using the level of coupling among requirements and among functional attributes as a way of managing complexity and deciding on the evolution path of a system needs further investigation. The works [Suh, 1990] on axiomatic design and of Altshuller on the theory of inventive problem solving (see Chapter 2, Section 2.4.4.4) is a starting point for this investigation. An example of application using the relationship matrices of requirements and attributes, proposed in this thesis, needs to be developed. The relationships between modularity and clusters of requirements and attributes needs further investigation.

**Integration of simulation and engineering analysis tools into the framework.** The integration of simulation and engineering analysis tools with the systems engineering environment (e.g. Cradle) must be pursued in order to provide further visibility of the effect of changes in the lower levels of the product breakdown structure and to investigate the necessary changes driven by upper level requirements. One approach for an integration architecture is the one adopted by [Lu et al., 1993] proposing a layered framework for a system workbench integrating and facilitating teams.

**Integration of CAD/CAM/CAE tools within the systems engineering environment.** Physical integration between a systems engineering environment (e.g. Cradle) and CAD/CAM/CAE tools is already possible, considering that the packages of information provided by those tools can be managed by the systems engineering environment. What must be sought, however, is logical integration through attributes, for example. The links through attributes would provide information such as: 'what are the requirements that are going to be affected when a shape of part changes from X to Y?' As it is today, packages of information (e.g. CAD drawings) couples many requirements as they are in reality a package of attributes.

**Pragmatic approach.** The 'total view framework' was developed for coping with a very dynamic environment, where the speed of change requires the knowledge of the relationships among all the elements that affect stakeholders satisfaction or dissatisfaction so that complexity can be managed. To cope with change, however, it is necessary to speed up the process of getting to the total view. Otherwise, when finishing all the models and having captured all the requirements, attributes and relationships, it would be necessary to start again because a new business scenario may have taken place. The issue of pragmatism was not addressed by this work although it is a key factor for selling the ideas of this thesis. Open issues are: How to adapt the total view to the constraints of tier suppliers, for example, without loosing sight of the total? Having the picture of the total view for a given product, how to prioritise what to focus on? How to concentrate on the strongest relationships?
References

3SL, 1998. Cradle v3.2 manuals, Barrow-in-Furness


Ascent Logic, 1995. RDD-100 manuals.


Bate, R. et al., 1995. A systems engineering capability maturity model, version 1.1, CMU/SEI-95-MM-003, November.


Booch, G., 1993. *Object-oriented analysis and design with applications*. Addison-Wesley, Reading, MA.


Cadre, 1995 *TeamWork manuals*


Chamberlain, R.G. & Shisko, R., 1992 "Fundamentals of systems engineering at NASA". IN:
Proceedings of the INCOSE'92. Seattle, WA, USA.


DoD, 1996 U.S. Department of Defense, Regulation 5000.2-R. Mandatory procedures for major defence acquisition programs and major automated information system acquisition programs, March 15.


Dube, C. 1998 “LPD 17: adding people and profit to IPPD”. IN: Proceedings of the INCOSE '98. INCOSE, Seattle, WA, USA.


Fararoy, S. et al., 1995. Accurate train localisation in open space and tunnels.


Ford Motor Company, 1995c Complete vehicle – quality function deployment (QFD). 00.00QFD-D02, October, 31 pages


Ford Motor Company, 1995h “QFD primary/secondary/tertiary wants template European & U.S. cars and light trucks” IN: Worldwide design standard, Oct 1995 00.00QFD-D02-1


Ford Motor Company, 1996d. *FOBVX8::BROOKSPR Job 930 ISC_KBAD0 17/10/1996 (Ptech – Idle Speed Control Strategy)*.


Ford Motor Company, 1997b. *AVT PCSE organisational planning*. Issue No. 5, 14-Jan-97, 2 pages


Hitchins, D., 1997. ©CADRAT - a tool for synthesis, clustering, structure and architecture (not published)


Loureiro, G., 1994. *QFD auxiliado por computador em abordagens de engenharia simultânea (Computer aided QFD in a concurrent engineering approach).* MSc Thesis, Instituto Tecnológico de Aeronáutica (ITA), São José dos Campos, SP, Brasil.


McMunigal, J. E. et al., 1990. “TQM and Japanese optimisation techniques.” IN: Third Annual European Conference on Japanese Optimisation Technology (Cambridge University, October 2)


Responsible office: HS/Program Operations Division, NASA HQ, Washington, DC (February 27,
1990)


Proceedings of the 4th International INCOSE Congress.

New York. Volume 1, Number 1, 1998, pp. 82-9. ISSN 1098-1241.


Paulk, M.C. et al., 1993 SW-CMM: capability maturity model for software, Version 1.1, Software
Engineering Institute, February.

INCOSE’92. INCOSE, Seattle.

Percivall, G.S., 1994. “System architecture for evolutionary system development”. IN: Proceedings of
the INCOSE’94. Seattle, WA, USA.


• Pugh, S., 1991. Total design: integrated methods for successful product engineering. Addison-Wesley


IN: Proceedings of the 28th signal, systems and computers conference, ASILOMAR, IEEE,


Schilke, N. A., 1994. (General Motors Corporation, USA) "Engineering for the customer". SAE paper 945232.


Schlenoff et al., 1996. Unified process specification language: requirements for modelling process. NIST, Gaithersburg, USA, NISTIR 5910.


TD Technologies, 1995. Slate user manual


Appendix A
Structured and object oriented analysis methods
This appendix aims to provide an overview of important structured and object-oriented analysis methods which have been influencing many others since the 1970s. The methods reviewed in this appendix are:

- SADT (Structured Analysis and Design Technique) and IDEF0 notation;
- SA/SD (Structured Analysis/Structured Design);
- HPM (Hatley Pirbhai Methodology);
- The Unified Process and the UML (Unified Modelling Language) notation.

A.1 SADT and IDEF0
SADT (Structured Analysis and Design Technique) [Marca & McGowan, 1988] was initially developed between 1969 and 1977 by Softech [Ross, 1985]. It aims to develop system descriptions centred on the concepts of modelling. It covers the requirements analysis phase (determining what the system will do), the functional analysis phase (defining functions and their interfaces) and the specification documentation phase. Although primarily designed for software engineering, it has been applied to a large variety of application domains including electronic system design, financial system analysis, budget construction and tracking cycles, security systems, or curriculum development in education. [Ross, 1985]

The power of SADT as a communication and analysis tool for business users and engineers was acknowledged in 1978 by the United States Air Force developing the IDEF suite of modelling techniques. SADT has been selected as the basis for IDEF0, an activity based modelling language. IDEF0 can model systems consisting of men, machines, data, materials, products, etc. [Vernadat, 1996]

A.1.1 Notation
The notation is based on two types of constructs:

- the IDEF0 ICOM function box (or SADT diagram) to represent activities (see Figure A-1);
- arrows to connect activities (see Figure A-2).

The IDEF0 function box represents an activity or group of activities perceived as a whole, which transforms inputs into outputs under the influence of a control using the mechanisms provided. The inputs (I) are objects to be processed or transformed by the function. They can be information objects or physical objects. The controls (C) are defined in the form of information objects. They are used to activate/regulate/synchronise the function. Examples of control objects can be orders, constraints, manufacturing plans, schedules, management directives, or regulations. The mechanisms (M) can represent information and/or physical resources. Examples include data files,
databases, machines, plants, software systems, and human operators. The outputs (O) are objects processed or transformed by the function.

1-way arrow

2-way arrow

Bundle

Spreading

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A

A
A.2 SA/SD

The SA/SD (structured analysis/structured design) methodology was developed by Yourdon (1989) influenced by the work of DeMarco (1978). The methodology can be used for system or application specification and for describing the organisation of a system in a hierarchical form, e.g. for designing a software program architecture.

A.2.1 Model

The methodology is based on the use of two models: the essential model and the implementation model. The essential model is composed of two parts:

- the environment model which describes the environment within which the system will operate;
- the behavioural model of the system to be designed.

The implementation model describes the solution in three hierarchical levels: processor, task and modules. The processor level of the implementation model describes the application processors, relations and interfaces. Each processor will be mapped to a set of functions in the essential model. The task level corresponds to a decomposition of the processor into tasks and their relationships providing a software architecture. The module level specifies each task decomposition.

A.2.2 Notations

The essential model use DeMarco’s data flow diagram (DFD) enhanced with the distinction between process functions and control functions. Process functions are expanded into other DFD and control functions are expanded into state transition diagrams (STD). The models also use entity-relationship diagrams for specifying data and application entities. A process function at the bottom of the hierarchy is described by a process specification (PSPEC). Any data item is further described in a data dictionary.
The processor and task levels use DFDs. The module level specifies each task decomposition using a structure chart.

A.2.3 Method

To obtain the environment model:

- Identify elements in the environment (outside the scope of the system) interacting with the system;
- Identify what data is exchanged between system and environment;
- Elaborate a context diagram (DFD) with the system, elements in the environment and data interaction.
- List the events describing stimulus from the elements in the environment and system response.

To obtain the behavioural model:

- Each system response will generate a function with output in a DFD;
- Data exchange among functions are identified;
- The functions are regrouped into successive upward levels (bottom-up approach);
- STD, entity relationship diagrams, process specification, data dictionary entries complete the model.

To obtain the implementation model:

- Identify necessary processors as support for the essential model. This is obtained by reorganising the essential model.
- Reorganisation of the transformation diagram for each processor, in order to show the tasks, interfaces, data management and data scheduling.
- Define the implementation schema for each task in the form of a hierarchy of modules (Structure Chart)

A.3 HPM

The HPM methodology [Hatley & Pirbhai, 1988] was built on the work performed in the 1970's by [Yourdon, 1979] and [DeMarco, 1978], who developed structured analysis techniques to improve the system specification process. The techniques are a collection of guidelines and graphical communication tools that allow the systems analyst to develop a functional specification that is easily read and understood. It develops the system functional specification in a top-down manner, starting with general and proceeding to the specific. The HPM methodology was developed in the mid 1980's on a commercial avionics project at Smith's industries (Grand Rapids, MI) and Boeing. It was documented in 1988.

Extends traditional structured analysis and design to include models for real-time behaviour and system architecture. Provides a consistent format for system and software specification, easing the transition from system design to software design.

A.3.1 Model

The methodology develops a system specification model composed by (as illustrated in Figure A-5):

- a system requirements model: captures what problem the system is to solve. It is built as a technology-independent model. It is an integrated model of two aspects: information processing (functional) behaviour and control (state) behaviour. These aspects are represented by process and control models respectively. It is built as a layered set of DFDs and CFDs (control flow diagrams).
with associated PSPECs and CSPECs (control specifications). Each successive level of diagrams and specifications expresses a refinement of the higher-level diagrams.

- a system architecture model: captures how to solve the problem. It is a technology-dependent model of the system's structure. It shows the physical entities that make up the system; the information flow between these physical entities; the channels on which this information flows. It consists of a layered set of AFDs (architecture flows diagrams) and AIDs (architecture interconnect diagrams) and their associated AMSs (architecture module specifications) and AISs (architecture interconnect specifications). Each successive layer refines the configuration defined by the higher-level diagrams.

![Diagram](image.jpg)

*Figure A-6: Diagrammatic representation of the HPM (Source: [Hatley & Pirbhai, 1988])*

### A.3.2 Notations

**System requirements model:**

The **process model** is represented by DFD (Data Flow Diagrams), process specifications and requirements dictionary, borrowed directly from structured analysis. The DFD is the primary tool depicting functional requirements. It partitions these requirements into component functions, or processes, and represents them as a network interconnected by data flows. Its main purpose is to show how each process transforms its input data flows into output data flows, and to show the relationships between those processes. Two additional tools support the DFD: a process specification (PSPEC or process specification) for each primitive process, and a requirements dictionary (RD) that specifies every data flow. Processes that are not primitive are broken down into more detailed DFDs and do not have PSPECs.

The **control model** informs under what circumstances the system will perform the functions represented in the DFDs. It uses two other tools: CFD (Control Flow Diagram), which show the flow of control signals in our system, and control specifications (CSPECs or control specifications), which indicate how the control processing takes place.

CSPECs contain diagrammatic and tabular representations of finite state (FS) machines. The control signals flowing to and from the CSPEC bars on the CFDs are the inputs and outputs of these FS
machines. CSPECs contain basically two types of diagrams: a state transition diagram (STD) and a process activation table (PAT). STDs show the states of the system and how they are influenced by control signals. They respond to events represented by control flows and show the corresponding action that the system must take. Events and actions are represented on STDs as "Event/Action" labels on each of the transitions between the states. PATs show the circumstances under which the processes on a DFD are enabled and disabled. Like the DFDs and PSPECs, the CFDs and CSPECs are supported by the requirements dictionary, which contains the definition of all control signals in the system. Processes that are not controlled from a CSPEC are "data triggered", that is, they are enabled each time there is sufficient data at their inputs to perform the specified function. The functions of the system, as represented in a DFD, are modelled independently of when they are supposed to be enabled or disabled.

System Architecture Model:
The AFD (Architecture Flow Diagram) is the primary tool depicting the system architecture. It shows the physical partitioning of the system into its components pieces, or modules, and the information flow between them. Its main purpose is to allocate the functional processes of the requirements model to physical units of the system, and to add more processes as needed to support the new physical interfaces. The requirements model data processes, PSPECs and CSPECs are divided into groups that are allocated to these architecture modules. AFDs are supported by architecture module specifications (AMSS) and an architecture dictionary (AD). Module specifications define the inputs, outputs, and processes allocated from the requirements model for each architecture module. The AD contains the data and control flow definitions of the requirements dictionary, plus the allocation of these flows to architecture modules.

In addition to the physical entities and the information flow between them, we need to capture how the information flows from one architecture module to another, and for this we use the architecture interconnect diagram (AID). The AID shows the actual physical information channels between the AFD's architecture modules and to and from the environment. The AID is supported by architecture interconnect specifications (AISs), textual specifications that specify the characteristics of the communication channels. The architecture dictionary also captures the allocation of data and control flows to specific interconnect channels.

A.3.3 Method
The process consists first, of searching for all technology independent specifications. The model is then enhanced by interfaces: inputs, outputs, user and maintenance and safety aspects. Finally the system architecture is deduced with the two parts, the hardware and software. (see Figure A-6)
A.4 The Unified Process and the UML notation

The Unified Process and the Unified Modelling Language (UML) is the result of the joint effort of Grady Booch, Jim Rumbaugh and Ivar Jacobson the creators of the Booch design method, object modelling technique (OMT) and use-case modelling, respectively. In 1997, the UML became the standard modelling language for object-oriented development with the formal support of the Object Management Group (OMG) and its various member companies. [http://www2.aw1.com/cseng/titles/O-201-89542-0/techniques/index.htm]

The Unified Process is a generic software development process framework that can be specialised for a very large class of software systems, for different application areas, different types of organisations, different competence levels, and different project sizes. Jacobson, Booch & Rumbaugh (1999) use the phrases ‘component-based’, ‘use-case driven’, ‘architecture-centric’ and ‘iterative and incremental’ to define the Unified Process. Being component-based means that the software is made up of components interconnected via well-defined interfaces. Architecture provides the structure in which to guide the work in the iterations, whereas use cases define the goals and drives the work of each iteration. A use-case is a description of a set of sequence of actions, including variants, that a system performs that yields an observable result of value to a particular actor. The Unified Process uses the UML notation.

A.4.1 The UML notation

Vocabulary

The UML has the following categories of vocabulary: things, relationships and diagrams. Figure A-7 illustrates the elements under each category. Figures A-8 to A-18, extracted from [Jacobson, Booch & Rumbaugh, 1999], illustrate each of these elements. Definitions of each of these elements can be found in [Jacobson, Booch & Rumbaugh, 1999]

Figure A-7: The vocabulary of the UML (Source: [Jacobson, Booch & Rumbaugh, 1999])
Definitions and graphical notation

Class: a description of a set of objects that share the same set of attributes, operations, relationships, and semantics. (see Figure A-8)

Interface: a collection of operations that are used to specify a service of a class or a component. (see Figure A-8)

Collaboration: a society of classes, interfaces, and other elements that work together to provide some cooperative behaviour that is bigger than the sum of all the elements; the specification of how an element, such as a use case or an operation, is realised by a set of classifiers and associations plying specific roles used in a specific way. (see Figure A-9)

Use case: a description of a set of sequence of actions, including variants, that a system performs that yields an observable result of value to a particular actor. (see Figure A-9)

Active class: a class whose instances are active objects, i.e., objects that own a process or thread and can initiate control activity. (see Figure A-10)

Component: a physical and replaceable part of a system that conforms to and provides the realisation of a set of interfaces. (see Figure A-10)
Node: a physical element that exists at run time and that represents a computational resource, generally having at least some memory and often times processing capability. (Figure A-10)

Interaction: a behaviour that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose. (see Figure A-11)

State machine: a behaviour that specifies the sequences of states an object goes through during its lifetime in response to events, together with its responses to those events. (see Figure A-12)
Package: a general purpose mechanism for organising elements into groups. (see Figure A-13)

Note: a comment attached to an element or a collection of elements. (see Figure A-14)

Dependency: a semantic relationship between two things, in which a change to one thing (the independent thing) may affect the semantics of the other thing (the dependent thing). (see Figure A-15)

Association: a structural relationship that describes a set of links, where a link is a connection among objects; the semantic relationship between two or more classifiers that involves the connections among their instances. (see Figure A-16)

Generalisation: a specialisation/generalisation relationship, such that objects of the specialised element (the subtype) are substitutable for objects of the generalised element (the supertype). (see Figure A-17)
Extensibility mechanism: one of three mechanisms (stereotypes, tagged values, and constraints) that can be used to extend the UML in controlled ways. (see Figure A-18).

Models

The Unified Process repeats over a series of cycles. To carry out the next cycle efficiently, the developers need all the representations of the software product (see Figure A-19):

- a use case model with all the use cases and their relationships to users;
- an analysis model, which has two purposes: to refine the use cases in more detail and to make an initial allocation of the behaviour of the system to a set of objects that provides the behaviour.
- a design model that defines (a) the static structure of the system as subsystems, classes, and interfaces and (b) the use cases realised as collaborations among the subsystems, classes, and interfaces.
- an implementation model, which includes components (representing source code) and the mapping of the classes to components.
- a deployment model, which defines the physical nodes of computers and the mapping of the components to those nodes.
- a test model, which describes the test cases that verify the use cases.
- the representation of the architecture.
A.4.1 Process

The Unified Process repeats over a series of cycles making up the life of a system, as illustrated in Figure A-7. Each cycle concludes with a product release to customers. Each cycle consists of four phases: inception, elaboration, construction, and transition. Each phase is further subdivided into iterations.

Figure A-20: The system life cycle as approached by the Unified Process (adapted from [Jacobson, Booch and Rumbaugh, 1999])

Figure A-21 lists the workflows—requirements, analysis, design, implementation and test—in the left-hand column. The curves approximate the extent to which the workflows are carried out in each phase. An iteration in any phase goes through all the five workflows.
In the **inception phase**, a vision for the end product is developed and the business case for the product is presented. A simplified use case model indicates the basic functionality of the product for each of its major users. A draft architecture is developed containing the most crucial subsystems. In this phase, the most important risks are identified and prioritised, the elaboration phase is planned in detail, and the whole project is roughly estimated.

In the **elaboration phase**, most of the product's use cases are specified in detail and the system architecture is designed. The architecture is expressed as views of all models of the system, which together represent the whole system. This implies that there are architectural views of the use-case model, analysis model, design model, implementation model and deployment model. The view of the implementation model includes components to prove that the architecture is executable. The result of this phase is an architecture baseline. At the end of this it is possible to plan all the activities, estimate the resources and assess the risks to complete the project.

In the **construction phase** the product is built and the bulk of the required resources is expended. The architecture of the system is stable but tolerates minor changes. At the end of this phase, the product contains all the use cases that management and the customer agreed to develop for this release.

In the **transition phase** a small number of experienced users tries the product and reports defects and deficiencies. Developers then correct the reported problems and incorporate some of the suggested improvements into a general release for the larger user community. The transition phase involves activities such as manufacturing, training customer personnel, providing help-line assistance, and correcting defects found after delivery. The maintenance team often divides these defects into two categories: those with sufficient effect on operations to justify an immediate delta release and those that can be corrected in the next regular release.

![Figure A-21: Workflows distribution over the Unified Process phases (Source: [Jacobson, Booch and Rumbaugh, 1999])](image)
References


Appendix B
Cradle overview

This appendix provides an overview of Cradle, version 3.2.

Figure B-1 illustrates Cradle modules. For each of these modules, this appendix describes briefly its capability.

B.1 Cradle – Project Data Management module
This module includes configuration management, cross-referencer, text and graphics reporters, workgroup management, project database, system notes.

B.1.1 Cradle project database (PDB)
A Cradle project is a set of information stored by Cradle in a project database (PDB). Cradle is a client/server product. The Cradle Database Server runs as a background process on a nominated workstation and provides access to PDBs. Cradle toolset is the client for the CDS. The Cradle toolset consists of a set of software tools to implement Cradle’s functionality. Cradle’s toolset can be grouped into three areas:
• specifically associated with producing an analysis (Essential Model);
• specifically associated with producing a system design (Implementation Model);
• common to both these activities, or part of either activity.

Figure B-2 shows the organisation of Cradle's project database.

Project information can be categorised as either system information or project data.

Main categories of system information are:
• CMS information: Cradle's Configuration Management System (CMS) is used to control the changes in the information held in the PDB. There are a number of different pieces of CMS information held in the PDB, including Change Tasks (CHTs), Change Requests (CHRIs), Log Notes, and CMS Event Records.
• project parameter: information about the project itself, and the way that Cradle has been configured.
• user profile: description of the users who can access the information. A user profile contains details about the user, such as a username, a team name, and the rights and privileges that they have been given.

Project data is one of three types:
• non-model or common information that exists outside the models but is nevertheless project information;
• essential model that is the implementation-independent definition of the system requirements;
• implementation model that is a design-specific view of the system.

Common project data can be:
• source document: a large document that is a source of requirements;
• system note: a generic object stored in the PDB. It can be used to contain a wide variety of user-definable information;
• event: an external trigger from the environment which the system must respond to.
• cross-reference: a logical link between two pieces of, usually dissimilar, information.

The essential and implementation models contain different sets of diagrams and definitions.

Diagram project data is stored in a range of diagram types:
• architectural modelling: Function Block Diagram (only implementation model)
• functional modelling: Data Flow Diagram, Entity Relationship Diagram, State Transition Diagram, Structure Chart (only implementation model) and Data Structure Diagram;
• behaviour modelling: Behaviour Diagram;
• object oriented modelling: Use Case Diagram and UML notation.

Definition project data are:
• data dictionary entry: textual description of a piece of data that is manipulated by the system;
• process specification: describes the function that a process performs;
module specification (implementation model only): describes the function that the module of code will perform;

terminator description: describes a terminator (e.g. sensor, actuator, person)

Each item of information (diagram, requirement, event or specifications) has an identity, a type, some associated details, and a set of frames. Any type of information may be held in a frame, such as: text, CAD drawings, spreadsheets, databases, CNC tool data, planning information, encapsulated postscript. Any operation (e.g. cross referencing) on the item applies to all frames within it.

B.1.2 Workgroup management

A Cradle user is defined on a per-project basis. When a project is created there is only one user, MANAGER. This account, or User Profile, is the account used by the Project Manager to perform all administrative operations, such as adding more users, creating the project structure, etc. A User Profile is defined by the manager of the project. Changing User Profiles is therefore usually beyond the scope of a user's privileges. User Profiles contain user security clearance, user skills, user access privileges and tools in the Cradle toolset that the user can use. Skills are the mechanism that is used to restrict information to those users who are intended to see it. When a frame type is defined, a skill may be associated with that frame type.

The Project Manager also creates the project structure. The project structure is determined by the users that the manager creates. There are three types of projects:

- Single person project: a single user using Cradle for a project. The user's team name is set to PROJECT.
- Group project: a set of users in a project. Their profiles' team names are all set to PROJECT.
• Team project: a set of teams in a project, each team with a number of members (each of whose profiles' team names are set to TEAM A, TEAM B, etc., as appropriate), and possibly a number of users at project level, such as QA user (whose profiles' team names are set to PROJECT).

Items of information in the Cradle PDB can have user-definable security classification. The user has a user-definable security clearance. A user can only access information that has a classification which is the same or lower than their clearance.

When a user wants to access a piece of information, the item of information, the user profile and the project structure are taken into account. The access right granted is one of: NO ACCESS, READ_ONLY, READ_WRITE.

B.1.3 System Notes
Before version 2.0, the only way that information which was not, for example, a requirement, event or specification, could be stored was as a frame (either pre-defined or user-defined) in one of these types of item. The user was left to decide which of the pre-defined set of information types was the most suitable. Such choices were often not easy and tended to distort the semantics of the item that was being used to store the information. System Notes were introduced to solve this problem, and are Cradle's project-extensible area of the lifecycle. System Notes can be imported, exported, reported, passed through the Configuration Management System (CMS), stored within the project baselines, and included in CMS Change Tasks (CHTs). System Notes can be reported by user-defined documents as the information type included within the component document sections of Document Diagrams (DCDs). System Notes can be cross referenced to each other. They can also be cross referenced to and from requirements, events, Essential Model Process Specifications (Pssecs), Implementation Model Pssecs, and Implementation Model Module Specifications (Mspecs).

B.1.4 Cross referencer
Cross references are relationships between two items of information. Cradle cross-references may be bi-directional, many-to-many, and/or transitive. Each cross-reference consists of:

• the identity of its 'from' part;
• the identity of its 'to' part;
• a set of optional attributes: reason, rationale, reference, note;
• a record of its creator, creation date, last modifier, and last modification date.

The following types of information may be linked by cross-references:

• Source statements within source documents;
• Requirements;
• Events;
• Process specifications in the Essential Model;
• Process specifications in the Implementation Model;
• Module Specifications;
• Instances of user-defined types of System Note.
Cross-references can be classified into link types which may belong to link groups. Names of link types and link groups are project-specific.

During the project setup, the user is allowed to control recursion between different types of System Notes when traversing cross-references. This is done using the VIA TYPES menu on the Cross Reference Parameters popup (please see Figure B-3).

The principle of recursion and re-entrancy in transitive cross-references to System Notes is illustrated in Figures B-4 and B-5, respectively.
B.1.5 Configuration Management System (CMS)
Cradle supports the concept of drafts. When an item of information (diagram, requirement, event or specifications) is created, its draft is set as A. Subsequent drafts are allocated sequential letters. Cradle provides mechanism by which this draft information can be submitted for approval, reviewed, approved, rejected, registered and baselined. While any item of information remains as draft it is not controlled by the CMS. Once it is registered into an open baseline, it becomes controlled by the CMS and its version changes to I. As changes are made to this information, subsequent to it being initially captured into the CMS, the version will be incremented.

Approved information can be collectively stored in a baseline which is a version of the information. Formally approved information can be registered into an open baseline. At a later stage, the baseline is closed, at which point the version represented by the baseline is defined, and:

- information cannot be added to or removed from the baseline;
- the baseline cannot be re-opened;
- the version number of all information in the baseline is incremented when the information is registered into a baseline.

Information that has been formally approved and baselined may need altering at a later date. Information that has been formally rejected will need modification before it is re-submitted. The CMS provides a mechanism for formally controlling requests for changes and the change tasks themselves.

The CMS contains knowledge of the relationships between individual items of information. For example, when submitting a diagram, the CMS creates a package of the diagram, including, e.g. associated requirements, Data Dictionary (DD) definitions, process and module specifications. Each requirement, specification and DD entry has frames containing information such as spreadsheets, CAD drawing, simulations. Any change to the information in the frames is also detected and advised by the CMS.

B.1.6 Graphics and Text Reporters
Graphics reporter provides the means to produce printed output from the diagrams held within the project.

Text reporter provides the means to produce printed output from the text stored in the Cradle project, even if this text information is related to text in diagrams. An example of a text report related to a diagram is the Diagram IO List. This report lists the inputs to and outputs from any PROCESS or FUNCTION symbol in a diagram.

B.2 REQ - Requirements management module
Source documents are the stakeholder documentation that describes what stakeholders want from the system. Cradle’s Source Document Management allows documents to be registered and have requirements extracted from them, but it also allows documents to have many versions, therefore allowing requirements to evolve as the system develops. While Cradle is capable of managing the changes in the source documents that are presented to the project, it is up to the project members to assess the impact that the changes in the source document are going to have on the information in the project.

To assess the impact of the changes in the source document, Cradle generates a comparison report from which it is possible to identify lines added to or modified in a source document or lines deleted from a
source document. The use of cross-references provides the other information items that are affected by
the changes in the source document.

Requirements can be extracted from source documents or created as isolated entities. In order to process
requirements Cradle provides functionality to:
• capture requirements from source documents;
• edit, name and number requirements;
• support requirements hierarchies;
• navigate through requirements using many different search definitions, enabling the user to check for
duplication of requirements, non-compatibility;
• cross reference requirements to other items in the project;
• categorise requirements;
• attach related information to individual requirements without affecting the original text of the
requirement (e.g. database, graphs, spreadsheets, tables, diagrams);
• navigate directly between requirements and other items in the project database (PDB).

The pre-defined requirements properties in Cradle are: the actual textual information that makes up the
requirement; originator; reference; category codes.

B.3 SYS - System Modelling
Cradle separates system modelling into an Essential Model and an Implementation Model, following
Yourdon’s terminology [Yourdon, 1989]. The Essential Model is a logical or functional model. It
models what the system is supposed to do, its functions. The Implementation Model is a physical or
structural model. It models how the system is implemented or structured to accomplish those functions,
its architecture. Figure B-3 lists the modelling notations that Cradle supports. Elements in the Essential
and Implementation Models are detailed using data dictionary, process specification, module specification
and terminator descriptions (see Figure B-2).

Cradle has a consistency checker that operates on all items of information that can be created within the
Essential and Implementation Models. The consistency checker performs a number of standard checks on
selected project data, and produces a report. The rules that the consistency checker uses to check
information are designed to ensure that the information about a system is complete, and consistent.

The consistency checker operates within Essential and Implementation Models. Cradle has a comparator
to perform: checks between any pair of Data Flow Diagrams (DFDs), checks between fragments of the
same or different DFDs, comparisons between the Essential and Implementation Models.

Cradle also allows the Essential and Implementation Model STDs to be animated. The animation means
that the set of states, conditions, and actions in the STD are traversed, or exercised, in one of a variety of
sequences, and at each stage in the execution of the STD’s finite-state-machine, the corresponding effect
on the STD’s parent Data Flow Diagram (DFD) is graphically displayed.
<table>
<thead>
<tr>
<th>Modelling Type</th>
<th>Diagram</th>
<th>Description</th>
<th>Essential Model Implementation Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architectural</td>
<td>Function Block Diagram</td>
<td>Shows how different parts of the system interact with other system components</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data Flow Diagram</td>
<td>Shows data flowing through the processing elements of a system, how the system interacts with the environment, and how the system is controlled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Entity Relationship Diagram</td>
<td>Shows how data elements of a system (the entities) are related to each other.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>State Transition Diagram</td>
<td>Shows the system as a finite-state-machine. The diagram shows all the states that the system may exist in, and what must happen in the system or the environment for the system to change from one state to another.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Structure Chart</td>
<td>Shows the structure of the data that the system manipulates.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data Structure Diagram</td>
<td>Graphically represents the structure of the data that the system manipulates</td>
<td></td>
</tr>
<tr>
<td>Functional</td>
<td>Behaviour Diagram</td>
<td>Shows the behaviour of part of the system with respect to the surrounding environment.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use Case Diagram</td>
<td>Captures the part of the system which is of interest. Describes the system and its environment. The system is made up of all relevant system functions or processes, the use cases. The environment is made up of the actors that are external to the system and communicate with use cases within the system.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Package Diagram</td>
<td>Creates an overall structure to an UML model. It captures the logical functionality, the process (scheduleable units, availability, fault tolerance), the component (compilation units, dependencies, software management), the deployment (processors and links, performance, fault tolerance) views. Each view in the package can be further decomposed.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sequence Diagram</td>
<td>Identifies the actors and classes involved in the interaction, with a lifeline for each. Between the lifelines, it shows the messages sent between the participants in the interaction.</td>
<td></td>
</tr>
<tr>
<td>Unified Method</td>
<td>Collaboration Diagram</td>
<td>Shows the totality of all messages flows, and explicitly shows which actors and/or classes exchange messages, and what messages they exchange.</td>
<td></td>
</tr>
<tr>
<td>Language notation</td>
<td>Class Diagram</td>
<td>Shows classes of objects and their relationships with other classes.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Statechart Diagram</td>
<td>Depicts a finite-state-machine that describes how the class responds to different external stimuli. These stimuli are the receipt of messages by instances of the class, in the form of calls of the class' operations.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Activity Diagram</td>
<td>Shows one or more parallel sequences of possible behaviour, distributed along a time line, which in turn may have branches and optional synchronisation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Component Diagram</td>
<td>In the component view, it shows the dependencies between compilation units, typically including source files, and the runtime components of the final system. In the process view, it shows the dependencies between processes in terms of runtime components.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Deployment Diagram</td>
<td>Shows the physical processors in the system, the processes that are allocated to them, and, to a limited extent, how these processors are interconnected.</td>
<td></td>
</tr>
</tbody>
</table>

**Figure B-3: Modelling notations supported by Cradle**

**B.4 PERF – Performance Modelling module**

Cradle works essentially at the pre-specification stage. However it provides a performance modelling capability. Performance modelling allows any part of the Implementation Model to be assessed in terms of the characteristics that it needs in order to be viable when built. This allows the effects of performance requirements and assumptions on the design to be studied. The performance modelling functionality provided is based on instrumenting symbols on state models with performance data expressed as sets of
Performance Parameters (PPs). A state model is a set of one or more Implementation Model Data Flow Diagrams (DFDs) and/or FBDs.

B.5 SWE – Software Engineering module
As a computer aided software engineering (CASE) tool, Cradle has the capability of automatically generating software code. Cradle can generate code in ADA, Pascal and ANSI C. Cradle generates code from structure charts and their associated data dictionaries and module specifications. Cradle can also reverse engineer code in a given file. Given the code, Cradle generates the corresponding structure chart.

B.6 DOC – Document Generation module
A Cradle document is a collection of formatted pages which contain information retrieved from either a Cradle Project Database (PDB), external host operating system files, or both.

Cradle’s Document Composer is a collection of tools that can be used to produce customised documents. The Document Composer provides complete control over the document and page layout, and the document content. Project data is read from the PDB at the time that a document is printed. The Document Composer can produce reports containing both test and graphics.

To produce a document that are three tools within Cradle that must be used. They are:

- **Document Editor**: is used to design the document structure and content in terms of existing page templates. The structure is represented in a diagram that shows the order in which sections of a document appear. Sections can also be split into subsections, and so on.

- **Template Editor**: is used after the document has been split into sections, and is used to specify the layout of the document.

- **Document Printer**: is used to print the newly created document.

Reference
Appendix C
System modelling notations

This appendix aims to describe the system modelling notations used in the thesis. These notations include the N\(^2\) chart and the notations provided by Cradle, a systems engineering environment software package developed by 3SL. The notations provided by Cradle include the non-UML and the UML notations. Although the package diagram is included under UML notation it is not an explicit diagram type in UML. For each notation it is intended to provide a definition, not necessarily from a Cradle perspective, and how a notation may relate to others. The various elements of each notation are also defined. The notations described in this appendix are:

<table>
<thead>
<tr>
<th>Cradle notations</th>
<th>UML notations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The N(^2) chart</strong></td>
<td>Use case diagram (UCD)</td>
</tr>
<tr>
<td>Non-UML notations</td>
<td>Package diagram (PD)</td>
</tr>
<tr>
<td>Data flow diagram (DFD)</td>
<td>Sequence diagram (SQD)</td>
</tr>
<tr>
<td>Entity relationship diagram (ERD)</td>
<td>Collaboration diagram (COD)</td>
</tr>
<tr>
<td>State transition diagram (STD)</td>
<td>Class diagram (CD)</td>
</tr>
<tr>
<td>Data structure diagram (DSD)</td>
<td>Statechart diagram (SCD)</td>
</tr>
<tr>
<td>Behaviour diagram (BD)</td>
<td>Activity diagram (ACD)</td>
</tr>
<tr>
<td>Function block diagram (FBD)</td>
<td>Component diagram (CPD)</td>
</tr>
<tr>
<td>Structure chart (STC)</td>
<td>Deployment diagram (DPD)</td>
</tr>
</tbody>
</table>

C.1 The N\(^2\) chart

The N\(^2\) chart provides a structured method for the definition of functional interactions and interfaces. The chart itself is a graphical presentation of all of the functions within a system (subsystem, task, etc.), together with the one-way interactions between each of these functions ordered in a fixed co-ordinate matrix format. The chart gets its name from the fact that for N functions there are N squared intersections or squares on the diagram, each of which may contain a function or function interface. The number of possible interfaces for the system is equal to N\(^2\)-N, where N is the number of functions within the system. Since both functions and function interfaces occupy a square of the diagram, the total number of squares graphically is equal to the square of the number of functions involved.

Figure C-1 illustrates a simplified N2 chart which contains all system functions (F1 through F4) on the diagonal axis and all system internal function interfaces in the remaining matrix locations (N\(^2\)-N, where N is a 4, is equal to 12). Each interface square (dotted line) represents a one-way interface between two internal functions. The square labelled F1→F4, for example, represents the one-way interface between function 1 (output) and function 4 (input). All function outputs are defined in the squares which are in the horizontal row of the function, while all function inputs are defined in the squares which are in the vertical column of the function. External inputs and outputs are defined on the top and bottom areas and the side areas respectively. The system illustrated in Figure C-1 has two external inputs (to F1 and F4) and two external outputs (from F1 and F4). One of the primary purposes of the N\(^2\) Chart is to indicate...
where interactions and interfaces do not exist. In Figure C-1, the squares below and to the right of function 3 are empty, indicating that there are no interfaces between functions F3 and F4. The table at the right hand side of the figure defines the basic rules for the $N^2$ Chart.

**Figure C-1: The $N^2$ chart (Source: [Lano, 1979])**

**C.2 Data flow diagram (DFD)**

A DFD shows the structure of functions, their decomposition and their inter-relationships. A single function is split into sub-functions. Relationships and flows between the functions can be expressed. These are often produced as hierarchical drawings which also decompose interfaces and control flows. DFDs allow complex functions to be decomposed and perform some basic consistency checks, such as whether the interface decomposition is consistent. [Stevens et al., 1998]

A DFD consists of: control processes, data processes, flows, stores, terminators and equipments. The functionality of the data processes is described by a textual Process Specification (PSPEC) and/or by another DFD referred to as a child DFD. The functionality implemented by control processes can be described textually as a PSPEC and/or by a State Transition Diagram (STD). Flows and stores represent data movement, and are defined in the Data Dictionary (DD).[3SL, 1998]

Figure C-2 shows an example of a DFD whose elements are defined, as following [3SL, 1998]:

**Data process:** an active entity that is capable of processing information.

**Data store:** a named repository of data acting as a buffer between the real world and the system, or within it.

**Control process:** an active entity that is capable of processing control or status signals and exercising control over the processes in the system.

**Control store:** a named buffer of a control or status signal, between the real world and the system.

**Comment:** a textual annotation to some part of a diagram.

**Terminator:** an element of the real world that is outside the system but interacts with it.

**Discrete data flow:** a named flow of data.

**Continuous data flow:** a named flow of data that is continually present, such as an analogue voltage.
Discrete control flow: a control or status signal that carries no data, and is either present or not.

Continuous control flow: a control or status signal that carries no data but is significant when in one of two levels. These flows may be raised or lowered.

Boundary point: a connection point between a flow of data or control, and other DFDs.

Equipment: an active physical entity that is capable of processing information.

Library equipment: an active physical entity that is capable of processing information, reused in this, or other, systems and therefore considered to be off-the-shelf functionality.

C.3 Entity-relationship diagram (ERD)

An entity is a "thing that can be distinctly identified", e.g., customers, suppliers, parts, products. A relationship is "an association among entities". The entity-relationship diagram (ERD) defines a partial model of the system by the entities within the system and the relationships between them. [QSS, 1994]

While the DFD models the active processing of information by the system, the ERD models the static relationships amongst this information that are preserved and maintained by the system. The ERD shows how items of data relate, statically, to each other. ERDs cannot exist in a hierarchy, instead, either a single ERD is produced for the entire system analysis or design (when the ERD is considered to relate to the entire DFD hierarchy), and/or ERDs can be produced as companions to specific DFDs that contain a large quantity of stored data, and contain the processes that create, update, or otherwise maintain stored data. [3SL, 1998]

Figure C-3 shows an example of ERD, whose elements are defined as following [3SL, 1998]:
Data object: an item of data operated upon the system.

Relationship: a relationship between one or more data objects, as one of: one-to-one, one-to-many, many-to-many.

Sub/supertype relationship: a relationship between (normally) three or more data objects, used in the situation where one data object contains data elements common to two or more other data objects (such as an object ‘machine’ containing data items ‘price’, ‘weight’, and ‘size’, and two other data objects, ‘lathe’ an ‘drill’, containing data elements unique to each machine type).

Connection symbol: a link between a data object and a relationship.

Sub/supertype connection: a link between data objects and relationships or sub/supertype relationships.

Associative connection: a link between an unnamed relationship and a named data object that indicates that the relationship contains data in addition to the relationship between the data objects.

Comment: a textual annotation to some part of a diagram.

C.4 State transition diagram (STD)

The state transition diagram (STD) models the different operational states of a system (or part of a system), plus the events that trigger the transition from one state to another.[Stevens et al., 1998]

Whereas process specifications (PSPECs) are used to specify data processes in a DFD, STDs specify control processes. Control processes enforce sequencing over environmental control stimuli to the system and the internal operation of the system.[3SL, 1998]

Figure C-4 shows an example of STD whose elements are defined in the following [3SL, 1998]:

![Diagram of a state transition diagram (STD)](image)
State: a named state of the STD's finite-state-machine.
Condition: the condition which must be true for the associated transition to occur. An event flow name. One or more conditions can be associated with a transition. Keywords AND, OR, and NOT can be used to convey a logical expression.
Action: an action performed by the finite-state-machine when making a transition between two states. Either an event flow name or to enable, disable, or trigger a named transformation.
Initial transition: the transition initially performed by the finite-state-machine when it, or the system, is activated.
Transition: a transition between two states.
Boundary point: a connection point for the initial transition to enter the initial state.
Comment: a textual annotation to some part of a diagram.

C.5 Data structure diagram (DSD)
The data structure diagram (DSD) has its origins back in the work of Jackson (1975) who believed a data structure to have four basic configuration: sequence, selection, repetition and elementary. The data structure diagram shows the composition of data.

DSDs provide a graphical alternative to the composition of a data dictionary entry. A DSD contains data items, and shows the decomposition of data items into lower-level data items. The data items whose composition are represented in a DSD can be either flows in a DFD or data objects in an ERD.[3SL, 1998]

Figure C-5 shows an example of DSD whose elements are defined in the following [3SL, 1998]:

Figure C-4: Example of STD showing its elements (Source: [3SL, 1998])
Data object: an item of data in the system's Data Dictionary

Iterative data object: an item of data that appears as an iterative (repeated) component of another, higher-level, data item.

Selection data object: an item of data that appears as an optional (selected) component of another, higher-level, data item.

Inter-item component connector: the means of interconnecting data items.

Boundary point: a means to graphically subdivide a DSD.

C.6 Behaviour diagram (BD)

The behaviour diagram (BD) notation was developed by merging the concepts of functional flow block diagrams (FFBD) [Malcolm, 1993], flow notations such as IDEF0 and DFDs, graph models of computation, and hierarchy of control concepts. The result is an 'executable' notation which may be used to define the intended behaviour of a system. Being 'executable' means that BDs can be used as a specification to a discrete event simulator to yield the times of the outputs from the times of the inputs and the duration of the functions. The foundation of the BD notation is the concept of 'items' processed by 'functions'. An 'item' may have contents, but arrives logically as a unit at an identifiable moment in time. The 'item' can be used either to represent a physical thing or a set of data. Some 'items' (called state 'items') contain the partially processed results of previous 'functions' operating on previous input 'items', and are passed on to subsequent 'functions'. 'Items' entering or exiting the system boundary are the 'observables' of the system. The decomposition of a 'function' is represented in a 'response net', or RNet, that defines how the contents of the input 'items' are used to determine which output 'items' will be emitted and their contents and which exit will be selected. 'Function nets', or F Nets, describe sequences, selections, iterations, loops, concurrencies and replications of 'functions'. [Alford, 1992]

BDs are used to illustrate the behaviour of functions in the system as a time sequenced set of functions. While DFDs show what the system is, BDs show what it does and when. BDs and DFDs can be thought
of as alternative views of the interactions between the same set of system functions. BDs show the interaction between system functions by distributing them along a time line to depict a processing sequence. The time line may branch into two or more parts at a node which must subsequently converge. Nodes are used to create sequential, conditional, concurrent, replicative, and iterative processing constructs. [3SL, 1998]

Figure C-6 shows an example of BD whose elements are defined in the following [3SL, 1998]:

![BD Diagram]

*Figure C-6: Example of BD showing its elements (Source: [3SL, 1998])*

**Comment:** a textual annotation to some part of a diagram.

**Time function:** a series of functions occurring over time in a pre-defined order. A time function may contain lower-level time functions in a lower-level BD.

**Shared function:** a function that may be reused in this, or other, BD and therefore considered to be off-the-shelf functionality. All uses of the shared function ‘X’, on any BD, reference the same definition and/or BD for ‘X’

**Discrete item:** an item at the lowest level of detail of interest.

**Time line start:** represents the start of a time sequence.
Time line end: represents the end of a time sequence.

Parallel node: acts as the starting point for a set of concurrent operations. Has two or more outgoing branches, none with a label. The set of operations distributed along each branch are, collectively, considered to occur in parallel with the operations along all other branches. Within a branch, the processing of individual operations is sequential, unless that branch itself contains other nodes. All branches converge at a matching parallel node. The processing along all branches is considered to be synchronised by this node, so that the processing along some branches may be held until the processing along other branches has completed.

Replicate node: creates a set of replicants of those of its outgoing branches that have a label. The number of these replicants is specified in the replicate node's label. There must be one unlabelled outgoing branch containing operations that control the replicants. All the outgoing branches converge into a second replicate node that marks the end of the replication.

Iteration node: causes its outgoing branch to be processed iteratively. At the end of the iterative processing section of time line, there is another iteration node which may or may not have a label; if it does, this label must match that of the iteration node at the start of the loop. From this second iteration node, a time line loop branches back to the first iteration node. This time line loop contains the iteration condition, either at its start or at its end, depending on the type of loop that is required.

End loop node: causes a jump in the sequence of control to the outgoing branch of the loop node with the specified label, such that control continues after the specified loop.

Goto node: causes a jump in the sequence of control along the time line of the label node in the same BD that has the matching label. There may be many goto nodes with the same label.

Label node: marks the destination of a goto node with a matching label. There may not be more than one label node with the same label.

Selection node: acts as an exclusive-OR operation. Has two more outgoing time line branches, each with a label. The node effectively routes the flow of control along one of these outgoing branches, depending on the exit condition of the upstream operation. All the time line branches must subsequently converge at a matching destination selection node. As an alternative, an operation may have multiple, labelled, outgoing branches, which converge at a selection node. This effectively subsumes the first selection node within the operation. The nodes are intended to create sequential, conditional, concurrent, replicative, and iterative processing constructs.

Exit diagram node: causes all processing in the diagram to stop, and returns the flow of control to the exit of the BD's parent function with the same name as the node's incoming branch.

Event node: causes the flow of control to begin at the start of the time line in the BD specified in the node's label.

Validation node: serves simply as a means of reporting timing statistics from the time line (when timing information is available for BDs) where the information reported is identified by the node's label.

Time line: represents the flow of time from object to object.

Time line loop: used in a loop or iteration structure to return to a loop or iteration node and begin another repetition.

Data link: represents the flow of data both into and out of functions.

Trigger data link: represents the flow of data into a function, that triggers the function to execute.
C.7 Function block diagram (FBD)

Functional block diagrams (FBDs) are used to show the organisation of, and flows of data between, pieces of system functionality. In this respect, they are similar in character to DFDs. However, FBDs are specifically intended to show the physical elements of a system and how they are interconnected. The concept for using FBDs is to show the hierarchy of the physical parts of the system. [3SL, 1998]

Figure C-7 shows an example of FBD whose elements are defined in the following [3SL, 1998]:

![Diagram of FBD](image)

**Figure C-7: Example of FBD showing its elements (Source: [3SL, 1998])**

- **Comment**: a textual annotation to some part of the diagram.
- **Boundary point**: a connection point between a link and other FBDs.
- **Terminator**: an element of the real world that is outside the system but that interacts with it.
- **Bus**: a piece of equipment for the transfer of data.
- **Equipment**: an active physical entity that is capable of processing information.
- **Link**: represents a physical link between equipments, buses, and terminators.
- **Library equipment**: an active physical entity that is capable of processing information, reused in this, or other, systems and therefore considered to be off-the-shelf functionality.

C.8 Structure chart (STC)

The structure chart (STC) depicts the interaction between software modules (procedures, functions, subroutines, processes, or tasks). The relationships between modules are shown by calls. The existence and direction of the call indicates that a module has the means to call the other module and that, at runtime, the module may call the other module zero or more times. Typically, the STC contains a hierarchy of modules with class used to show how one module will call another. As module calls normally involve the passage and return of parameters (or arguments), the STC depicts these with couples. Data and control couples are provided that indicate the passage or return of an item of data or some value that controls the operation of the recipient module (such as a condition for a loop or test). The orientation of the couple indicates which module is sending the parameter, and which is receiving it. [3SL, 1998]

Figure C-8 shows an example of STC whose elements are defined in the following [3SL, 1998]:

![Diagram of STC](image)
Module: a module of software.

Inclusion module: a module of software that in the final source code appears within the source code of another module.

Transaction centre: a module that selectively calls one of the set of lower-level modules connected to it on the basis of some condition (it acts like an if-then-else).

Library module: a module from a library whose implementation is not defined by the project.

Off-page connector: a named object that essentially breaks a single call into two calls that are on different STCs. Off-page connectors are associated with each other if they have the same name. The user can traverse from an off-page connector on one STC to the associated off-page connector on another STC at the same or at a different level.

On-page connector: a named object that essentially breaks a single call into two calls that are on the same STC. On-page connectors are associated with each other if they have the same name. The user can traverse from an on-page connector on one part of an STC to the associated on-page connector on another part of an STC.

Shared data area: a repository of data shared by one or more modules, used to represent either data common to several modules, or preserved between invocations by a single module (such as local static data in the language C), or accessed whenever invoked by a single module.

Global data area: a repository of data shared by all modules that connect to it, used to represent some piece of implementation technology (software and/or hardware based) that stores information in a manner that is independent of any module using that information.

Call: indicates that a module could call another module, and calls it zero or more times at runtime.

*Figure C-8: Example of STC showing its elements (Source: [3SL, 1998])*
Invocation: indicates that a module calls another module that thereafter executes in an independent manner; used for processes and tasks.

Iterative call: indicates that one module iteratively calls another module, for example a call within a loop.

Data couple: an item of data either passed to, or returned by, a module.

Control couple: a control or status value or flag that affects the operation of its recipient module.

Dialogue couple: an item of data passed to, an updated by, a module.

Function return: an item of data returned by a module.

Boundary point: a connection point between a call an other STCs.

Comment: a textual annotation to some part of a diagram.

**C.9 Use-case diagram (UCD)**

The use case diagram (UCD) is a diagram that shows a set of use cases and actors and their relationships; use case diagrams address the static use-case view of a system. [Jacobson, Booch & Rumbaugh, 1999]

Each use case contains one or more possible sequences or flows of events between the system and the environment: a normal sequence, and sequences that relate to alternate cases or error handling situations. A scenario is a particular route through the normal, alternate, or error use case flows, and is normally chosen sensibly to highlight a particular alternate sequence or to describe a particular exceptional case. Lower level UCDs from the use cases in the top-level UCD to: decompose a complex use case into a set of simpler use cases; define the set of scenarios for each use case; share use cases, between use cases in a UML model, and between models. In the top-level UCD, the system shown as a single rectangular system symbol expands into a top-level package diagram (PD). [3SL, 1998]

Figure C-9 shows an example of UCD whose elements are defined in the following [3SL, 1998]:

![Image of a Use-case diagram](Source: [3SL, 1998])
Comment: an explanatory note for the diagram.

System: representation of the entire system, contain the entire UML model.

External actor: a part of the environment that stimulates the system or is stimulated by it.

Use case or scenario: a class of interactions between one or more actors and the system.

Shareable use cases (uses): a common set of behaviour that is shared by one or more use cases, in one or more UML models.

Actor-use case relationship: link between actors and the use cases in which they are involved.

Uses relation: shows the use case referenced by a use case.

Extends relation: shows the use case of which particular use cases are a specialisation (if any).

C.10 Package diagram (PD)

Package diagrams (PDs) exist to create and overall structure to a UML model through the use of packages. The system in the top-level UCD is decomposed into a top-level PD containing four packages. Any package may be decomposed into one or more other packages shown on a lower-level PD. The model will consequently contain a hierarchy of PDs, in four sub-trees, all rooted under the top-level PD [3SL, 1998]:

- In the model PD sub-tree, a level is reached where packages expand to class diagrams (CDs) that contain the definitions of one or more classes. This sub-tree essentially defines the static structure of the UML model. The classes on this CD may appear on other CDs, but such appearances will identify the class as being part of another package.

- In the components PD sub-tree, a level is reached where packages expand to CPDs that describe the dependencies between compilation units and the mapping between classes and compilation units.

- In the processes PD sub-tree, a level is reached where packages expand to CPDs that describe dependencies between processes (individual executable units that can be scheduled for execution by a processor) and components.

- In the processors PD sub-tree, a level is reached where packages expand to DPDs that detail the processor(s) within the system, the links between them, and the mapping between processes and these processors.

Figure C-10 shows an example of PD whose elements are defined in the following [3SL, 1998]:

Comment: an explanatory note for the diagram.

System package: a connection of elements in a UML model, part of one of the four alternate views shown in a UML model, i.e., logical view (functionality); process view (scheduleable units, availability and fault tolerance); component view (compilation units, dependencies and software management); deployment view (processors and links, performance, fault tolerance).

Global package: a collection of elements that is outside the strict scope of the system but contains functionality or behaviour that is used in the system, normally specifically in its implementation (such as windowing system or I/O driver packages).

Package dependency: relationships between packages that indicate which packages a particular package is dependent upon for it to be able to exhibit its specified behaviour.
C.11 Sequence diagram (SQD)

A sequence diagram (SQD) is an interaction diagram that emphasises the time ordering of messages. [Jacobson, Booch, Rumbaugh, 1999]

Use case scenarios are described by either a SQD or a collaboration diagram (COD), or both. Both diagram styles show the details of the interaction of the environment with the appropriate part(s) of the system (classes and their object instances), in terms of a time sequenced exchange of messages. The time sequencing is made graphically explicit by the SQD, whereas the COD focuses on clearly showing the totality of message transfers between all of the actors and classes/object instances involved in the scenario. An SQD identifies the actors and classes (or object instances) involved in the interaction, with a timeline (lifeline) for each. Between the lifelines, the SQD shows the messages sent between the participants in the interaction [3SL, 1998].

Figure C-11 shows an example of SQD whose elements are defined in the following [3SL, 1998]:

**Comment:** an explanatory note for the diagram.

**External actor:** a part of the environment that stimulates the system or is stimulated by it.

**Object instance:** an object that is created, performs actions, and/or is destroyed during the lifeline.

**Lifeline node:** a connection point in a lifeline.

**Object deletion:** indicates the deletion of an object.

**Lifeline script:** a label for the actions being performed over the time of the lifeline, which is attached to a particular segment of the lifeline.

**Lifeline:** a representation of the existence of an object over a particular period of time.

**Activation:** the period during which an object is performing an action, either directly or through a subordinate procedure. The activation time may include time when it has control information on a stack, but during which control resides in something that is called. Also known as the focus of control.

**Computing activation:** the period during which an object activation is actually computing (i.e. it is the top item on the stack).
Synchronous message: an instantaneous communication between objects that conveys information, with the expectation that an action will be initiated as a result.

Flow of control message: a message that passes the focus of control from one object to another.

Asynchronous message: a communication between objects, taking a measurable time, that conveys information, with the expectation that an action will be initiated as a result.

**C.12 Collaboration diagram (COD)**

A collaboration diagram (COD) is an interaction diagram that emphasises the structural organisation of the objects that send and receive messages; a diagram that shows interactions organised around instances and their links to each other. [Jacobson, Booch & Rumbaugh, 1999]

The COD shows the same information as the SQD, but without the time sequenced view. It therefore shows the totality of all the message flows, and explicitly shows which actors and/or classes exchange messages, and what messages they exchange.

Figure C-12 shows an example of COD whose elements are defined in the following [3SL, 1998]:

Comment: an explanatory note for the diagram.

External actor: a part of the environment that stimulates the system or is stimulated by it.

Object instance: an object that is created, performs actions, and/or is destroyed during the lifeline.

Multi-object instance: a set of objects.

Interaction link: an indication that the object instances and actors collaborate and have message exchanges.

Synchronous message: an instantaneous communication between objects that conveys information, with the expectation that an action will be initiated as a result.

Flow of control message: a message that passes the focus of control from one object to another.
Asynchronous message: a communication between objects, taking a measurable time, that conveys information, with the expectation that an action will be initiated as a result.

C.13 Class diagram (CD)

A class diagram (CD) is a diagram that shows a set of classes, interfaces, and collaborations and their relationships; class diagrams address the static design view of a system; a diagram that shows a collection of declarative (static) elements. [Jacobson, Booch & Rumbaugh, 1999]

A CD contains classes which have a name and is defined by attributes and operations. A class may appear on one or more CDs, and potentially more than once on any such CD if desired. A class is said to be defined in the package that contains it, but may be declared within (i.e. used within) any other package that the designer requires. Classes exhibit relationships with other classes. These relationships range from a simple association, through a part-of aggregation or composition relationships, to kind-of inheritance relationships. These relationships are shown in CDs by appropriate types of connection symbol, linking the classes together. [3SL, 1998]

Figure C-13 shows an example of CD whose elements are defined in the following [3SL, 1998]:

Comment: an explanatory note for the diagram.
Class: a set of objects with similar structure, behaviour, and relationships.
Boundary class: a class whose primary role is to interface with the system environment, a stereotype of class.
Control class: a class whose primary role is to co-ordinate and manage the system behaviour, a stereotype of class.
Entity class: a class whose primary role is to manage a set of data objects, a stereotype of class.
Template class: a class with one or more unbound formal parameters.
Utility class: a class that supplies utility behaviour, a stereotype of class.
Relationship node: a symbol used to join classes in tertiary or higher relationships, in particular for association classes.
Association: a bi-directional relationship between classes that involves connections among their instances. Associations may have names, source and destination labels, and source and destination roles.

Unidirectional association: a unidirectional association between classes.

Aggregation: a special form of bi-directional association that specifies a whole-part relationship between the aggregate (whole) and a component part.

Unidirectional aggregation: a unidirectional aggregation.

Composition: a form of aggregation with strong ownership and coincident lifetime as part of the whole.

Unidirectional composition: a unidirectional composition.

Inheritance: the mechanism by which more specific elements incorporate structure and behaviour of more general elements related by behaviour.

Class dependency: a relationship between classes.

Association class link: a link that joins an association class to a relationship node, used to create associative relationships between the association class and two or more other classes.
C.14 Statechart diagram (SCD)

A statechart shows a state machine; statechart diagrams address the dynamic view of a system. [Jacobson, Booch & Rumbaugh, 1999]

An SCD depicts a finite-state-machine that describes how the class responds to different external stimuli. These stimuli are the receipt of messages by instances of the class, in the form of calls of the class' operations. Dynamic behaviour is described in terms of a set of potentially nested states, the transitions between these states, the events (calls of the class' operations) that trigger these transitions, and actions that are performed, both while transitioning between states, and on entry to, whilst within, and on exit from a state.

Figure C-14 shows an example of SCD whose elements are defined in the following [3SL, 1998]:

Comment: an explanatory note for the diagram.
Initial state: a condition at the beginning of the life of an object or an interaction during which it satisfies some condition, performs some action, or waits for some event.
Final state: a condition at the end of the life of an object or an interaction during which it satisfies some condition, performs some action, or waits for some event.
State: a condition during the life of an object or an interaction during which it satisfies some condition, performs some action, or waits for some event.
Superstate: a state that contains other states.
History indicator: when a transition to a superstate occurs, a history indicator shows control resumes at the state within the superstate that was current when the superstate was interrupted.
Transition: A relationship between two states, indicating that an object in the first state will enter the second state and perform certain specified actions when a specified event occurs, if specified conditions are satisfied.

C.15 Activity diagram (ACD)

An activity diagram (ACD) is a diagram that shows the flow from activity to activity; activity diagrams address the dynamic view of a system. A special case of a state diagram in which all or most of the states are action states and in which all or most of the transitions are triggered by completion of actions in the source states. [Jacobson, Booch & Rumbaugh, 1999]

In an UML model, the static representation of the system is a set of CD's. Each class may have an associated ACD to show the internal behaviour and/or algorithm for the class. The class ACD shows one or more parallel sequences of possible behaviour, distributed along a time line, which in turn may have branches and optional synchronisation. The ACD can show the sending and receipt of messages, which are the means by which classes communicate, by calling each others' operations, as shown on the associated CD.[3SL, 1998]

Figure C-15 shows an example of ACD whose elements are defined in the following [3SL, 1998]:

Comment: an explanatory note for the diagram.
Initial state: a condition at the beginning of the life of an object or an interaction during which it satisfies some condition, performs some action, or waits for some event.
Final state: a condition at the end of the life of an object or an interaction during which it satisfies some condition, performs some action, or waits for some event.
Activity: a component part of the behaviour of a class.
Decision activity: an activity that selects between transitions, on the basis of guard conditions.
Synchronisation bar: a synchronisation point for activities.
Send signal: sending a message to an instance of another class, calling an operation of another class.
Receive signal: receiving a message from another class, a call of one of this class’ operations by another class.
Trigger: a transition that links activities.

C.16 Component diagram (CPD)
A component diagram shows a set of component and their relationships; component diagrams address the static component view of a system. [Jacobson, Booch & Rumbaugh, 1999]

In the component view, the CPDs show the dependencies between compilation units, typically including source files, and the runtime components of the final system. In the process view, the CPDs show the dependencies between processes (schedulable units that may be selected for execution on the host machine’s processor) in terms of runtime components. Runtime components typically include: executables, dynamically linked libraries, overlays, shell scripts or batch files. These diagrams are intended to show the allocation of classes to compilation units and processes, respectively. [3SL, 1998]

Figure C-16 shows an example of CPD whose elements are defined in the following [3SL, 1998]:

![Component diagram example](source: [3SL, 1998])

Comment: an explanatory note for the diagram.
Component: a software component, which may or may not be executable.
Task specification: an executable component, typically known as a process, a thread, or a task.
Component dependency: a dependency between components.

C.17 Deployment diagram (DPD)
A deployment diagram shows a set of nodes and their relationships; a deployment diagram addresses the static deployment view of a system. [Jacobson, Booch & Rumbaugh, 1999]
DPDs are UML's attempt to recognise that there is a world outside software. These diagrams attempt to show the physical processors in the system (called nodes in DPDs), the processes that are allocated to them, and, to a limited extent, how these processors are interconnected. [3SL, 1998]

Figure C-17 shows an example of DPD whose elements are defined in the following [3SL, 1998]:

![Diagram of DPD showing its elements](image)

**Comment**: an explanatory note for the diagram.

**Node**: a runtime physical object that represents a processing resource, generally having at least a memory and often processing capability as well.

**Node dependency**: a communication association.

**References**


Appendix D
Details of diesel PCS project in Cradle

This appendix aims to present detailed information about the diesel PCS (Powertrain Control System) project as implemented in Cradle.

D.1 Project setup

This section contains details of Cradle’s project setup for the car and diesel PCS projects. The following is a text report produced by Cradle. The project setup consists of the definition of graphic parameters, categories of items of information, frames, types and groups of cross-references. Specific definitions of terms in the following report can be found in [3SL, 1998].

---

This section contains details of Cradle’s project setup for the car and diesel PCS projects. The following is a text report produced by Cradle. The project setup consists of the definition of graphic parameters, categories of items of information, frames, types and groups of cross-references. Specific definitions of terms in the following report can be found in [3SL, 1998].
This section contains a requirements list report provided by Cradle. The list below includes requirements for a car, focusing on those requirements to be allocated to the PCS. For each requirement, the list contains the name/number (e.g. C000), the category values associated to that requirements (e.g. driveability, mandatory), the descriptive text of the requirement, frame descriptions (e.g. verifiability). 'Key:' value is generated by Cradle and correspond to a search criteria.
VERIFIABILITYVerifiable by test CETPO04

C000
Key:
1. Driveability
2. Mandatory
3. Condition
4. Technical

TEXTWhen starting the vehicle

VERIFIABILITYVerifiable by test CETPO01

C001
Key:
1. Driveability
2. Mandatory
3. Condition
4. Technical

TEXTEngine temperature

VERIFIABILITYVerifiable by test CETPO01

C001.001
Key:
1. Driveability
2. Mandatory
3. Condition
4. Technical

TEXTEngine off for 2 hours

VERIFIABILITYVerifiable by test CETPO01

C001.002
Key:
1. Driveability
2. Mandatory
3. Condition
4. Technical

TEXTEngine just turn off

VERIFIABILITYVerifiable by test CETPO01

C002
Key:
1. Driveability
2. Mandatory
3. Condition
4. Technical

TEXTA/C status

VERIFIABILITYVerifiable by test CETPO 013

C002.001
Key:
1. Driveability
2. Mandatory
3. Condition
4. Technical

TEXTA/C on

VERIFIABILITYVerifiable by test CETP 013

C002.002
Key:
1. Driveability
2. Mandatory
3. Condition
4. Technical

TEXTA/C off

VERIFIABILITYVerifiable by test CETPO 013

C003
Key:
1. Driveability
2. Mandatory
3. Condition
4. Technical

TEXTElectrical load

VERIFIABILITYVerifiable by test CETP 014

C003.001
Key:
1. Driveability
2. Mandatory
3. Condition
4. Technical

TEXTFull electrical load

VERIFIABILITYVerifiable by test CETP 014

C003.002
Key:
1. Driveability
2. Mandatory
3. Condition
4. Technical

TEXTNo electrical load

VERIFIABILITYVerifiable by test CETP 014

C004
Key:
1. Driveability
2. Mandatory
3. Condition
4. Technical

TEXTEnvironment temperature

VERIFIABILITYVerifiable by test CETPO-01

C004.001
Key:
1. Driveability
2. Mandatory
3. Condition
4. Technical

TEXT(41±1) degrees Celsius

VERIFIABILITYVerifiable by test CETP-001

C004.002
Key:
1. Driveability
2. Mandatory
3. Condition
4. Technical

TEXT(35±1) degrees Celsius

VERIFIABILITYVerifiable by test CETP-001

C005
Key:
1. Driveability
2. Mandatory
3. Condition
4. Technical

TEXTAltitude

VERIFIABILITYVerifiable by test CETP004

C005.001
Key:
1. Driveability
2. Mandatory
3. Condition
4. Technical

TEXTSea level
5: To be defined

TEXT Individually controlled climate control outlets

VERIFIABILITY Verifiable by demonstration
S003.003.005
Key: 2.5
1: Package/Ergonomi 2: Mandatory
3: Performance 4: Stakeholder
5: To be defined 6:

TEXT Memory for individual adjustments

VERIFIABILITY Verifiable by demonstration
S003.002.006
Key: 2.6
1: Package/Ergonomi 2: Mandatory
3: Performance 4: Stakeholder
5: To be defined 6:

TEXT Personal lamps

VERIFIABILITY Verifiable by demonstration
S003.002.007
Key: 2.7
1: Package/Ergonomi 2: Mandatory
3: Performance 4: Stakeholder
5: To be defined 6:

TEXT Driver has ability to override all rear HVAC functions

VERIFIABILITY Verifiable by demonstration
S003.003
Key: 3
1: Package/Ergonomi 2: Mandatory
3: Performance 4: Stakeholder
5: To be defined 6:

TEXT Comfortable interior environment under all weather conditions

VERIFIABILITY Verifiable by demonstration
S003.004
Key: 4
1: Comfort 2: Mandatory
3: Performance 4: Stakeholder
5: To be defined 6:

VERIFIABILITY Verifiable by demonstration
S003.005
Key: 5
1: Package/Ergonomi 2: Mandatory
3: Performance 4: Stakeholder
5: To be defined 6:

TEXT Easy entry/exit for driver and front seat passengers

VERIFIABILITY Verifiable by demonstration
S003.006
Key: 5
1: Package/Ergonomi 2: Mandatory
3: Performance 4: Stakeholder
5: To be defined 6:

TEXT Easy entry/exit for rear seat passengers

VERIFIABILITY Verifiable by demonstration
S003.007
Key: 6
1: Package/Ergonomi 2: Mandatory
3: Performance 4: Stakeholder
5: To be defined 6:

TEXT Comfortable rear seats

VERIFIABILITY Verifiable by demonstration
S003.008
Key: 8
1: Comfort 2: Mandatory
3: Function 4: Stakeholder
5: To be defined 6:

VERIFIABILITY Verifiable by demonstration
S003.009
Key: 9
1: Comfort 2: Mandatory
3: Function 4: Stakeholder
5: To be defined 6:

VERIFIABILITY Verifiable by demonstration
S003.010
Key: 10
1: Comfort 2: Mandatory
3: Function 4: Stakeholder
5: To be defined 6:

VERIFIABILITY Verifiable by demonstration
S003.011
Key: 11
1: Comfort 2: Mandatory
3: Function 4: Stakeholder
5: To be defined 6:

VERIFIABILITY Verifiable by demonstration
S003.012
Key: 12
1: Comfort 2: Mandatory
3: Function 4: Stakeholder
5: To be defined 6:

VERIFIABILITY Verifiable by demonstration
S003.013
Key: 13
1: Comfort 2: Mandatory
3: Function 4: Stakeholder
5: To be defined 6:

VERIFIABILITY Verifiable by demonstration
S003.014
Key: 14
1: Comfort 2: Mandatory
3: Function 4: Stakeholder
5: To be defined 6:

VERIFIABILITY Verifiable by demonstration
S003.015
Key: 15
1: Package/Ergonomi 2: Mandatory
3: Performance 4: Stakeholder
5: To be defined 6:

VERIFIABILITY Verifiable by demonstration
S003.016
Key: 16
1: Package/Ergonomi 2: Mandatory
3: Performance 4: Stakeholder
5: To be defined 6:

VERIFIABILITY Verifiable by demonstration
S003.017
Key: 17
1: Comfort 2: Mandatory
3: Performance 4: Stakeholder
5: To be defined 6:

VERIFIABILITY Verifiable by demonstration
S003.018
Key: 18
1: Comfort 2: Mandatory
3: Function 4: Stakeholder
5: To be defined 6:

VERIFIABILITY Verifiable by demonstration
S003.019
Key: 19
1: Comfort 2: Mandatory
3: Function 4: Stakeholder
5: To be defined 6:

VERIFIABILITY Verifiable by demonstration
S003.020
Key: 20
1: Package/Ergonomi 2: Mandatory
3: Performance 4: Stakeholder
5: To be defined 6:

VERIFIABILITY Verifiable by demonstration
S003.021
Key:
S004
Key:
1: Package/Ergonomi 2: Mandatory
3: Performance 4: Stakeholder
5: To be defined 6:

TEXT Comfortable and easy to install seat belt

VERIFIABILITY Verifiable by demonstration
S005
Key:
1: Package/Ergonomi 2: Mandatory
3: Performance 4: Stakeholder
5: To be defined 6:

TEXT Excellent visibility all around

VERIFIABILITY Verifiable by demonstration
S006
Key:
1: Vehicle dynamics 2: Mandatory
3: Performance 4: Stakeholder
5: To be defined 6:

TEXT Responsive handling and steering

VERIFIABILITY Verifiable by subjective attribute evaluation of vehicles
CETP R202 applied
S007
Key:
1: Vehicle dynamics 2: Mandatory
3: Performance 4: Stakeholder
5: To be defined 6:

TEXT Excellent ride

VERIFIABILITY Verifiable by CETP R202
S008
Key:
1: Vehicle dynamics 2: Mandatory
3: Performance 4: Stakeholder
5: To be defined 6:

TEXT Excellent brakes

VERIFIABILITY Verifiable by test
S009
Key:
1: Performance 2: Mandatory
3: Performance 4: Stakeholder
5: To be defined 6:

TEXT Has personal convenience items
Excellent powertrain

**VERIFIABILITY** Verifiable by CETP R202

**S009.001**

Key: 1

1: Performance
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Excellent powertrain

**VERIFIABILITY** Verifiable by CETP R202

**S009.002**

Key: 2

1: Performance
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Good power and pickup

**VERIFIABILITY** Verifiable by CETP R202

**S009.003**

Key: 3

1: Performance
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Good sound engine and transmission

**VERIFIABILITY** Verifiable by CETP R202

**S009.004**

Key: 4

1: Performance
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Good automatic transmission shift quality

**VERIFIABILITY** Verifiable by CETP R202

**S009.005**

Key: 5

1: Driveability
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Easy manual transmission gear and clutch operation

**VERIFIABILITY** Verifiable by CETP R202

**S009.006**

Key: 6

1: Driveability
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Engine helps to provide good braking when slowing down

**VERIFIABILITY** Verifiable by CETP R202

**S009.007**

Key: 7

1: Driveability
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Dependable engine and transmission

**VERIFIABILITY** Verifiable by CETP R202

**S009.007.001**

Key: 7.1

1: Driveability
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Starts quickly every time whether hot, cold or wet

**VERIFIABILITY** Verifiable by CETP R202

**S009.007.002**

Key: 7.2

1: Driveability
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Never stalls when cold or hot

**VERIFIABILITY** Verifiable by CETP R202

**S009.007.003**

Key: 7.3

1: Driveability
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Quick to idle and settle

**VERIFIABILITY** Verifiable by CETP R202

**S009.007.004**

Key: 7.4

1: Driveability
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Constant start times

**VERIFIABILITY** Verifiable by CETP R202

**S009.007.005**

Key: 7.5

1: Driveability
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** No overheating or coolant loss

**VERIFIABILITY** Verifiable by CETP R202

**S009.007.006**

Key: 7.6

1: Driveability
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** No unpleasant noise from engine

**VERIFIABILITY** Verifiable by CETP R202

**S009.007.007**

Key: 7.7

1: Electrical/Infect
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Battery should not discharge when vehicle is parked with lights on

**VERIFIABILITY** Verifiable by CETP R202

**S009.007.008**

Key: 7.8

1: Driveability
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** No axle leaks or failures

**VERIFIABILITY** Verifiable by CETP R202

**S009.007.009**

Key: 7.9

1: Life_cycle
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** No major problems for 100,000 miles

**VERIFIABILITY** Verifiable by CETP R202

**S009.007.010**

Key: 7.10

1: Life_cycle
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Transmission does not break down

**VERIFIABILITY** Verifiable by CETP R202

**S009.007.011**

Key: 7.11

1: Life_cycle
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Maintains good power and pickup

**VERIFIABILITY** Verifiable by CETP R202

**S009.007.012**

Key: 7.12

1: Life_cycle
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Transmission maintenance free transmission operation

**VERIFIABILITY** Verifiable by CETP R202

**S009.007.013**

Key: 7.13

1: Life_cycle
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Engine does not overheat

**VERIFIABILITY** Verifiable by CETP R202

**S009.007.014**

Key: 7.14

1: Life_cycle
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Adjustment maintenance free transmission operation

**VERIFIABILITY** Verifiable by CETP R202

**S009.007.015**

Key: 7.15

1: Life_cycle
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Engine and transmission maintain good sound over time

**VERIFIABILITY** Verifiable by CETP R202

**S009.007.016**

Key: 7.16

1: Life_cycle
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Engine and transmission will be trouble free and have a long life

**VERIFIABILITY** Verifiable by CETP R202

**S009.007.017**

Key: 7.17

1: Driveability
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Vehicle starts easily and never stalls

**VERIFIABILITY** Verifiable by CETP R202

**S009.007.018**

Key: 7.18

1: Driveability
2: Mandatory
3: Performance
4: Stakeholder
5: To_be_defined
6: 

**TEXT** Vehicle always starts easily and never stalls

**VERIFIABILITY** Verifiable by CETP R202
| Key: | 7.17 | 1: Driveability 2: Mandatory 3: Performance 4: Stakeholder 5: To be defined 6: |
| TEXT | An engine that always starts easily |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 7.19 |
| TEXT | Vehicle is reliable after clearing all areas under the hood |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 8 |
| TEXT | Engine idles smoothly |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 9 |
| TEXT | Excellent powertrain |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 10 |
| TEXT | Excellent powertrain |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 11 |
| TEXT | Excellent powertrain |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 12 |
| TEXT | Excellent powertrain |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 13 |
| TEXT | Excellent powertrain |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 14 |
| TEXT | Excellent powertrain |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 15 |
| TEXT | Excellent powertrain |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 16 |
| TEXT | Excellent powertrain |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 17 |
| TEXT | Excellent powertrain |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 18 |
| TEXT | D- 6 |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 19 |
| TEXT | Excellent powertrain |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 20 |
| TEXT | Excellent powertrain |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 21 |
| TEXT | Excellent powertrain |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 22 |
| TEXT | Smooth engine response |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 22.1 |
| TEXT | No surge at cruise |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 22.2 |
| TEXT | Steady and predictable acceleration |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 22.3 |
| TEXT | No surge at a stop in drive gear (automatic transmission) |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 22.4 |
| TEXT | Smooth engine and transmission response |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | 22.5 |
| TEXT | Quiet vehicle |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | M.A.A. |
| TEXT | Desirable |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | Styling/Appearance 2: Mandatory 3: Performance 4: Stakeholder 5: To be defined 6: |
| TEXT | Excellent powertrain |
| VERIFIABILITY Verifiable by CETIP R202 |

| Key: | Driveability 2: Mandatory 3: Performance 4: Stakeholder 5: To be defined 6: |
| TEXT | Smooth engine and transmission response |
| VERIFIABILITY Verifiable by CETIP R202 |
3.4 g/mile; 

TEXT A competitive price offering good value for the money

VERIFIABILITYVerifiable by demonstration

S013 Key:
1: Safety 2: Mandatory
3: Performance 4: Stakeholder
5: To_be_defined 6:

TEXT Vehicle is safe

VERIFIABILITYVerifiable by demonstration

S013 Key:
1: Cost 2: Desirable
3: Performance 4: Stakeholder
5: To_be_defined 6:

TEXT Good fuel economy

VERIFIABILITYVerifiable by demonstration

S013 Key:
1: Cost 2: Desirable
3: Performance 4: Stakeholder
5: To_be_defined 6:

TEXT Uses non-premium fuel

VERIFIABILITYVerifiable by demonstration

S013 Key:
1: Emissions 2: Desirable
3: Performance 4: Stakeholder
5: To_be_defined 6:

TEXT Emission system does not hurt gas mileage or performance

VERIFIABILITYVerifiable by demonstration

S013 Key:
1: Emissions 2: Desirable
3: Performance 4: Stakeholder
5: To_be_defined 6:

TEXT Maintenance good fuel economy

VERIFIABILITYVerifiable by demonstration

S013 Key:
1: Cost 2: Desirable
3: Performance 4: Stakeholder
5: To_be_defined 6:

TEXT Gauge dependability

VERIFIABILITYVerifiable by demonstration

S013 Key:
1: Emissions 2: Desirable
3: Performance 4: Stakeholder
5: To_be_defined 6:

TEXT Environmentally conscious

VERIFIABILITYVerifiable by demonstration

S013 Key:
1: Cost 2: Desirable
3: Performance 4: Stakeholder
5: To_be_defined 6:

TEXT Good warranty

VERIFIABILITYVerifiable by demonstration

S013 Key:
1: Life_cycle 2: Desirable
3: Performance 4: Stakeholder
5: To_be_defined 6:

TEXT Low operating cost (maintenance, repair and insurance costs)

VERIFIABILITYVerifiable by demonstration

S013 Key:
1: Cost 2: Desirable
3: Performance 4: Stakeholder
5: To_be_defined 6:

TEXT Good resale value

VERIFIABILITYVerifiable by demonstration

S013 Key:
1: Cost 2: Desirable
3: Performance 4: Stakeholder
5: To_be_defined 6:

TEXT Competitive fuel economy

VERIFIABILITYVerifiable by demonstration

S013 Key:
1: Cost 2: Desirable
3: Performance 4: Stakeholder
5: To_be_defined 6:

TEXT Uses non-premium fuel

VERIFIABILITYVerifiable by demonstration

S013 Key:
1: Emissions 2: Desirable
3: Performance 4: Stakeholder
5: To_be_defined 6:

TEXT Multi-fuel capability

VERIFIABILITYVerifiable by demonstration

S013 Key:
1: Emissions 2: Desirable
3: Performance 4: Stakeholder
5: To_be_defined 6:

TEXT Emission system does not hurt gas mileage or performance

VERIFIABILITYVerifiable by demonstration

S013 Key:
1: Emissions 2: Desirable
3: Performance 4: Stakeholder
5: To_be_defined 6:

TEXT Multi-fuel capability

VERIFIABILITYVerifiable by demonstration

S013 Key:
1: Emissions 2: Desirable
3: Performance 4: Stakeholder
5: To_be_defined 6:

TEXT Emission system does not hurt gas mileage or performance

VERIFIABILITYVerifiable by demonstration

S013 Key:
1: Emissions 2: Desirable
3: Performance 4: Stakeholder
5: To_be_defined 6:
The vehicle must comply with CO emission limits no greater than 0.3 g/km verified according to the FTP-75 standard.

Verified by testing following the FTP-75 test procedure.

Key:
1: Emissions
2: Mandatory
3: Performance
4: Technical
5: To_be_approved
6: Verifiable by test procedure CETP002

The vehicle must comply with NOx emission limits no greater than 0.1 g/km, verified according to the FTP-75 standard.

Verified by testing following the FTP-75 test procedure.

Key:
1: Fuel_Economy
2: Mandatory
3: Performance
4: Technical
5: To_be_approved
6: Verifiable by test

The vehicle must comply with NOx emission limits no greater than 0.1 g/km, verified according to the FTP-75 standard.

Verified by testing following the FTP-75 test procedure.

Key:
1: Driveability
2: Desirable
3: Performance
4: Technical

Starting

Verified by test CETP01

Key:
1: Driveability
2: Desirable
3: Performance
4: Technical

After releasing the throttle with a negative square pulse of force of (10+/-1)N and duration (500+/-1) ms, the engine must return from (2000+/-100)RPM to the nominal idle speed in 1 RPM/100ms.

Verified by testing following the test procedure CETP002

Key:
1: Driveability
2: Desirable
3: Performance
4: Technical

Engine must not cut-out following start-up with A/C on or off, full or no electrical load, at sea level or at (2000+/-100) metres of altitude, at temperatures of (4+/-1) degrees Celsius or (35+/-1) degrees Celsius, after 2 hours of vehicle with engine stopped or just after turning engine off.

Verified by testing following the test procedure CETP002

Key:
1: Driveability
2: Desirable
3: Performance
4: Technical

Stalling

Verified by test procedure CETP002

Key:
1: Driveability
2: Desirable
3: Performance
4: Technical

Engine must not cut-out following start-up with A/C on or off, full or no electrical load, at sea level or at (2000+/-100) metres of altitude, at temperatures of (4+/-1) degrees Celsius or (35+/-1) degrees Celsius, after 2 hours of vehicle with engine stopped or just after turning engine off.

Verified by testing following the test procedure CETP002

Key:
1: Driveability
2: Desirable
3: Performance
4: Technical

Stumbling

Verified by test procedure CETP004

Key:
1: Driveability
2: Desirable
3: Performance
4: Technical

When starting the engine, the RPM shall not fluctuate with amplitude more than 5% idle RPM without stalling with A/C on or off, full or no electrical load, after 2 hours engine being off or just after turning engine off and on again, at (4+/-1) degrees Celsius or (35+/-1) degrees Celsius, at sea level or (2000+/-100) metres of altitude.

Verified by testing following the test procedure CETP004

Key:
1: Driveability
2: Desirable
3: Performance
4: Technical

Engine does not pause during stabilised drive.

Verified by testing following the test procedure CETP004

Key:
1: Driveability
2: Desirable
3: Performance
4: Technical

Stabilised drive

Verified by testing following the test procedure CETP004

Key:
1: Driveability
2: Desirable
3: Performance
4: Technical

Hastenings

Verified by testing following the test procedure CETP004

Key:
1: Driveability
2: Desirable
3: Performance
4: Technical

Engine does not pause during stabilised drive.

Verified by testing following the test procedure CETP004

Key:
1: Driveability
2: Desirable
3: Performance
4: Technical

Stabilised drive
TEXT Engine RPM shall not fluctuate more than 5% causing vehicle speed to vary more than 5% during operation at vehicle speed no greater than 30 miles per hour.

TEXT Engine RPM shall not fluctuate more than 5% causing vehicle speed to vary more than 5% during operation at vehicle speed no greater than 30 miles per hour.

TEXT Engine shall not drop off during more than 250ms during acceleration rate greater than 10ms^-2.

TEXT Engine shall not drop off during more than 250ms during acceleration rate greater than 10ms^-2.

TEXT Longitudinal deceleration shall not vary more than 5% of its nominal value.

TEXT Longitudinal deceleration shall not vary more than 5% of its nominal value.

TEXT Engine shall not pause during operations at speeds higher than 70 miles per hour.

TEXT Engine shall not pause during operations at speeds higher than 70 miles per hour.

TEXT Engine shall fire on all cylinders.

TEXT Engine shall fire on all cylinders.

TEXT Engine shall not pause during operation with speed lower than 30 miles per hour.

TEXT Engine shall not pause during operation with speed lower than 30 miles per hour.

TEXT Engine shall not drop off during more than 250ms during acceleration rate greater than 10ms^-2.

TEXT Engine shall not drop off during more than 250ms during acceleration rate greater than 10ms^-2.

TEXT Engine shall not pause during operations at speeds higher than 70 miles per hour.

TEXT Engine shall not pause during operations at speeds higher than 70 miles per hour.

TEXT Engine shall fire on all cylinders.

TEXT Engine shall fire on all cylinders.

TEXT Engine shall not drop off during more than 250ms during acceleration rate greater than 10ms^-2.

TEXT Engine shall not drop off during more than 250ms during acceleration rate greater than 10ms^-2.

TEXT Engine shall not pause during operations at speeds higher than 70 miles per hour.

TEXT Engine shall not pause during operations at speeds higher than 70 miles per hour.

TEXT Engine shall fire on all cylinders.

TEXT Engine shall fire on all cylinders.

TEXT Engine shall not drop off during more than 250ms during acceleration rate greater than 10ms^-2.

TEXT Engine shall not drop off during more than 250ms during acceleration rate greater than 10ms^-2.

TEXT Engine shall not pause during operations at speeds higher than 70 miles per hour.

TEXT Engine shall not pause during operations at speeds higher than 70 miles per hour.

TEXT Engine shall fire on all cylinders.

TEXT Engine shall fire on all cylinders.

TEXT Engine shall not drop off during more than 250ms during acceleration rate greater than 10ms^-2.

TEXT Engine shall not drop off during more than 250ms during acceleration rate greater than 10ms^-2.

TEXT Engine shall not pause during operations at speeds higher than 70 miles per hour.
D.3 Cross-references list

This section presents an example of cross-references list report generated by Cradle. Each cross-reference can be defined by its ‘from’ item, its ‘to’ item, a link type, reason and rationale.

<table>
<thead>
<tr>
<th>From Requirements</th>
<th>To System Notes</th>
<th>Number: A1</th>
<th>Baseline:</th>
<th>Last modifier: MANAGER Date:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Link type:</td>
<td>Creator: MANAGER Date:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5:</td>
<td>_to _target</td>
<td>Reason:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: 1: \_Parameter 2: \_Target 3: \_Function 4: Technical 5: To be defined 6: Verifiability

**VERIFIABILITY** Verifiable by CETP-013

TC00.005.004

1: Driveability 2: Mandatory 3: Function 4: Technical 5: To be defined

**TEXT** Longitudinal deceleration shall not vary more than 5% from its nominal value.

**VERIFIABILITY** Verifiable by CETP-013

TC00.005.004.001

1: Driveability 2: Mandatory 3: Function 4: Technical 5: To be defined

**TEXT** Engine shall not drop off during more than (250-1-10) m/sec in acceleration rate more than 10n/s/3.
From: Requirements
To: Requirements

Number: 0004.001
Type: U
Link type: HIERARCHY
Creator: MANAGER
Date: 15/05/99
Baseline: 15/05/99
Reference: 11-202
Rationale: requirement hierarchy
Note:

From: Requirements
To: Requirements

Number: 0004.001
Type: U
Link type: TRACABILITY
Creator: MANAGER
Date: 20/05/99
Baseline: 20/05/99
Reason: link_tracable_to
Reference: 11-202
Rationale: tracability_links
Note:

From: Requirements
To: Requirements

Number: 0004.001
Type: U
Link type: HIERARCHY
Creator: MANAGER
Date: 15/05/99
Baseline: 15/05/99
Reference: 11-202
Rationale: requirement hierarchy
Note:

From: Requirements
To: Requirements

Number: 0004.001
Type: U
Link type: TRACABILITY
Creator: MANAGER
Date: 20/05/99
Baseline: 20/05/99
Reason: link_tracable_to
Reference: 11-202
Rationale: tracability_links
Note:

From: Requirements
To: Requirements

Number: 0004.001
Type: U
Link type: HIERARCHY
Creator: MANAGER
Date: 15/05/99
Baseline: 15/05/99
Reference: 11-202
Rationale: requirement hierarchy
Note:

From: Requirements
To: Requirements

Number: 0004.001
Type: U
Link type: TRACABILITY
Creator: MANAGER
Date: 20/05/99
Baseline: 20/05/99
Reason: link_tracable_to
Reference: 11-202
Rationale: tracability_links
Note:

From: Requirements
To: Requirements

Number: 0004.001
Type: U
Link type: HIERARCHY
Creator: MANAGER
Date: 15/05/99
Baseline: 15/05/99
Reference: 11-202
Rationale: requirement hierarchy
Note:

From: Requirements
To: Requirements

Number: 0004.001
Type: U
Link type: TRACABILITY
Creator: MANAGER
Date: 20/05/99
Baseline: 20/05/99
Reason: link_tracable_to
Reference: 11-202
Rationale: tracability_links
Note:

From: Requirements
To: Requirements

Number: 0004.001
Type: U
Link type: HIERARCHY
Creator: MANAGER
Date: 15/05/99
Baseline: 15/05/99
Reference: 11-202
Rationale: requirement hierarchy
Note:

From: Requirements
To: Requirements

Number: 0004.001
Type: U
Link type: TRACABILITY
Creator: MANAGER
Date: 20/05/99
Baseline: 20/05/99
Reason: link_tracable_to
Reference: 11-202
Rationale: tracability_links
Note:
Better understanding of a requirement

Reference: National: Requirement hierarchy

Note:

From: Requirements
To: Requirements

Number: 0909.007
Type: U
Link type: Creator: MANAGER
Date: 04/09/98
Last modifier: MANAGER

Reason: Better understanding of a requirement

Reference: National: Requirement hierarchy

Note:

From: Requirements
To: Requirements

Number: 0909.007
Type: U
Link type: Creator: MANAGER
Date: 04/09/98
Last modifier: MANAGER

Reason: Better understanding of a requirement

Reference: National: Requirement hierarchy

Note:

From: Requirements
To: Requirements

Number: 0909.007
Type: U
Link type: Creator: MANAGER
Date: 04/09/98
Last modifier: MANAGER

Reason: Better understanding of a requirement

Reference: National: Requirement hierarchy

Note:

From: Requirements
To: Requirements

Number: 0909.007
Type: U
Link type: Creator: MANAGER
Date: 04/09/98
Last modifier: MANAGER

Reason: Better understanding of a requirement

Reference: National: Requirement hierarchy

Note:

From: Requirements
To: Requirements

Number: 0909.007
Type: U
Link type: Creator: MANAGER
Date: 04/09/98
Last modifier: MANAGER

Reason: Better understanding of a requirement

Reference: National: Requirement hierarchy

Note:

From: Requirements
To: Requirements

Number: 0909.007
Type: U
Link type: Creator: MANAGER
Date: 04/09/98
Last modifier: MANAGER

Reason: Better understanding of a requirement

Reference: National: Requirement hierarchy

Note:

From: Requirements
To: Requirements

Number: 0909.007
Type: U
Link type: Creator: MANAGER
Date: 04/09/98
Last modifier: MANAGER

Reason: Better understanding of a requirement

Reference: National: Requirement hierarchy

Note:

From: Requirements
To: Requirements

Number: 0909.007
Type: U
Link type: Creator: MANAGER
Date: 04/09/98
Last modifier: MANAGER

Reason: Better understanding of a requirement

Reference: National: Requirement hierarchy

Note:

From: Requirements
To: Requirements

Number: 0909.007
Type: U
Link type: Creator: MANAGER
Date: 04/09/98
Last modifier: MANAGER

Reason: Better understanding of a requirement

Reference: National: Requirement hierarchy

Note:

From: Requirements
To: Requirements

Number: 0909.007
Type: U
Link type: Creator: MANAGER
Date: 04/09/98
Last modifier: MANAGER

Reason: Better understanding of a requirement

Reference: National: Requirement hierarchy

Note:

From: Requirements
To: Requirements

Number: 0909.007
Type: U
Link type: Creator: MANAGER
Date: 04/09/98
Last modifier: MANAGER

Reason: Better understanding of a requirement

Reference: National: Requirement hierarchy

Note:

From: Requirements
To: Requirements

Number: 0909.007
Type: U
Link type: Creator: MANAGER
Date: 04/09/98
Last modifier: MANAGER

Reason: Better understanding of a requirement

Reference: National: Requirement hierarchy

Note:

From: Requirements
To: Requirements

Number: 0909.007
Type: U
Link type: Creator: MANAGER
Date: 04/09/98
Last modifier: MANAGER

Reason: Better understanding of a requirement

Reference: National: Requirement hierarchy

Note:

From: Requirements
To: Requirements

Number: 0909.007
Type: U
Link type: Creator: MANAGER
Date: 04/09/98
Last modifier: MANAGER

Reason: Better understanding of a requirement

Reference: National: Requirement hierarchy

Note:
<table>
<thead>
<tr>
<th>From: Requirements</th>
<th>To: Requirements</th>
<th>Number: TO01.001</th>
<th>Type: U</th>
<th>Link type: TRACEABILITY</th>
<th>Creator: MANAGER</th>
<th>Date: 20/05/99</th>
<th>Baseline: MANAGER</th>
<th>Data: 20/05/99</th>
<th>Reason: Better understanding of the requirement</th>
<th>Reference: requirements_hierarchy</th>
<th>Note:</th>
</tr>
</thead>
<tbody>
<tr>
<td>From: Requirements</td>
<td>To: Requirements</td>
<td>Number: TO01.002</td>
<td>Type: U</td>
<td>Link type: TRACEABILITY</td>
<td>Creator: MANAGER</td>
<td>Date: 20/05/99</td>
<td>Baseline: MANAGER</td>
<td>Data: 20/05/99</td>
<td>Reason: Better understanding of the requirement</td>
<td>Reference: requirements_hierarchy</td>
<td>Note:</td>
</tr>
<tr>
<td>From: Requirements</td>
<td>To: Requirements</td>
<td>Number: TO01.003</td>
<td>Type: U</td>
<td>Link type: TRACEABILITY</td>
<td>Creator: MANAGER</td>
<td>Date: 20/05/99</td>
<td>Baseline: MANAGER</td>
<td>Data: 20/05/99</td>
<td>Reason: Better understanding of the requirement</td>
<td>Reference: requirements_hierarchy</td>
<td>Note:</td>
</tr>
<tr>
<td>From: Requirements</td>
<td>To: Requirements</td>
<td>Number: TO01.004</td>
<td>Type: U</td>
<td>Link type: TRACEABILITY</td>
<td>Creator: MANAGER</td>
<td>Date: 20/05/99</td>
<td>Baseline: MANAGER</td>
<td>Data: 20/05/99</td>
<td>Reason: Better understanding of the requirement</td>
<td>Reference: requirements_hierarchy</td>
<td>Note:</td>
</tr>
<tr>
<td>From: Requirements</td>
<td>To: Requirements</td>
<td>Number: TO01.005</td>
<td>Type: U</td>
<td>Link type: TRACEABILITY</td>
<td>Creator: MANAGER</td>
<td>Date: 20/05/99</td>
<td>Baseline: MANAGER</td>
<td>Data: 20/05/99</td>
<td>Reason: Better understanding of the requirement</td>
<td>Reference: requirements_hierarchy</td>
<td>Note:</td>
</tr>
<tr>
<td>From: Requirements</td>
<td>To: Requirements</td>
<td>Number: TO01.006</td>
<td>Type: U</td>
<td>Link type: TRACEABILITY</td>
<td>Creator: MANAGER</td>
<td>Date: 20/05/99</td>
<td>Baseline: MANAGER</td>
<td>Data: 20/05/99</td>
<td>Reason: Better understanding of the requirement</td>
<td>Reference: requirements_hierarchy</td>
<td>Note:</td>
</tr>
<tr>
<td>From: Requirements</td>
<td>To: Requirements</td>
<td>Number: TO01.007</td>
<td>Type: U</td>
<td>Link type: TRACEABILITY</td>
<td>Creator: MANAGER</td>
<td>Date: 20/05/99</td>
<td>Baseline: MANAGER</td>
<td>Data: 20/05/99</td>
<td>Reason: Better understanding of the requirement</td>
<td>Reference: requirements_hierarchy</td>
<td>Note:</td>
</tr>
<tr>
<td>From: Requirements</td>
<td>To: Requirements</td>
<td>Number: TO01.008</td>
<td>Type: U</td>
<td>Link type: TRACEABILITY</td>
<td>Creator: MANAGER</td>
<td>Date: 20/05/99</td>
<td>Baseline: MANAGER</td>
<td>Data: 20/05/99</td>
<td>Reason: Better understanding of the requirement</td>
<td>Reference: requirements_hierarchy</td>
<td>Note:</td>
</tr>
<tr>
<td>From Requirements</td>
<td>To: Requirements</td>
<td>Type</td>
<td>U</td>
<td>Link Type: Traceability</td>
<td>Number: T003.001.001</td>
<td>Creator: MANAGER</td>
<td>Date: 21/05/99</td>
<td>Baseline: 20/05/99</td>
<td>Reason: ia_truable_to</td>
<td>Reference:</td>
<td>Rational:</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------</td>
<td>------</td>
<td>---</td>
<td>--------------------------</td>
<td>---------------------</td>
<td>------------------</td>
<td>------------------</td>
<td>------------------</td>
<td>------------------</td>
<td>----------</td>
<td>------------------</td>
</tr>
<tr>
<td>From Requirements</td>
<td>To: Requirements</td>
<td>Type</td>
<td>U</td>
<td>Link Type: Traceability</td>
<td>Number: T003.001.002</td>
<td>Creator: MANAGER</td>
<td>Date: 21/05/99</td>
<td>Baseline: 20/05/99</td>
<td>Reason: ia_truable_to</td>
<td>Reference:</td>
<td>Rational:</td>
</tr>
<tr>
<td>From Requirements</td>
<td>To: Requirements</td>
<td>Type</td>
<td>U</td>
<td>Link Type: Traceability</td>
<td>Number: T003.001.001</td>
<td>Creator: MANAGER</td>
<td>Date: 21/05/99</td>
<td>Baseline: 20/05/99</td>
<td>Reason: ia_truable_to</td>
<td>Reference:</td>
<td>Rational:</td>
</tr>
<tr>
<td>From Requirements</td>
<td>To: Requirements</td>
<td>Type</td>
<td>U</td>
<td>Link Type: Traceability</td>
<td>Number: T003.001.001</td>
<td>Creator: MANAGER</td>
<td>Date: 21/05/99</td>
<td>Baseline: 20/05/99</td>
<td>Reason: ia_truable_to</td>
<td>Reference:</td>
<td>Rational:</td>
</tr>
<tr>
<td>From Requirements</td>
<td>To: Requirements</td>
<td>Type</td>
<td>U</td>
<td>Link Type: Traceability</td>
<td>Number: T003.001.001</td>
<td>Creator: MANAGER</td>
<td>Date: 21/05/99</td>
<td>Baseline: 20/05/99</td>
<td>Reason: ia_truable_to</td>
<td>Reference:</td>
<td>Rational:</td>
</tr>
<tr>
<td>From Requirements</td>
<td>To: Requirements</td>
<td>Type</td>
<td>U</td>
<td>Link Type: Traceability</td>
<td>Number: T003.001.001</td>
<td>Creator: MANAGER</td>
<td>Date: 21/05/99</td>
<td>Baseline: 20/05/99</td>
<td>Reason: ia_truable_to</td>
<td>Reference:</td>
<td>Rational:</td>
</tr>
</tbody>
</table>
D.4 Diesel PCS functional analysis

Figures D-1 to D-8 presents the functional decomposition of the New Diesel diesel PCS. The diagrams were based on the TeamWork (a commercial CASE tool) [Cadre, 1995] models produced by the New Diesel PCS development team. Process Activation Tables (PATs) and Decision Tables (DT) from TeamWork were translated into State Transition Diagrams (STDs) in Cradle. The Data Flow Diagrams (DFDs) reproduced in Cradle are very similar to the ones produced in TeamWork.

![New Diesel Control System](image)

*Figure D-1: DFD 01.1.5.1 – Puma Lynx Control System context diagram*
Figure D-3: DFD 01.1.5.1.1.1 Detect Engine State

Figure D-4: DFD 01.1.5.1.1.1.3 DT_Determine_Engine_State
Figure D-5: DFD 01.1.5.1.1.5 – Engine_Cranking

Figure D-6: DFD 01.1.5.1.1.7 Engine_Running
Figure D-7: STD 01.1.5.1.1.23 STD_Master_Mode_Control

Figure D-8: STD 01.1.5.1.1.24 PAT_Setup_Modes
D.5 Diesel PCS life cycle processes functional analysis

Figure D-9 shows the PCS life cycle processes functional model depicted in a Behaviour Diagram (BD) in Cradle. Figures D-10 to D-12 focus on the development and evolution processes of the PCS.

Figure D-9: BD 01.05.01.1 PCS_life_cycle_process_func_model

Figure D-10: BD 01.05.01.1.1 PCS_development_cycle
Figure D-11: BD 01.05.01.1.1.6 Develop_PCS

Figure D-12: BD 01.05.01.1.6.18 Improve feature
D.6 Diesel PCS physical analysis

Figures D-13 to D-16 are Data Flow Diagrams (DFDs) representing the physical decomposition of the New Diesel diesel PCS up to the point the software subsystems to be programmed in the CPU (Central Processing Unit) are represented (Figure D-16). The New Diesel PCS development team was responsible for developing the software and specifying the hardware.

Figure D-13 shows the PCS components: the module hardware in the central bubble (EEC module) surrounded by sensors and actuators. Figures D-14 and D-15 expand the EEC module and the microcontroller, respectively into its components. Figure D-16 expands the CPU into the software subsystems to be developed.

---

*Figure D-13: DFD 01.01.04.01 Powertrain_Control_System*
Figure D-14: DFD 01.01.04.01.1 EEC_Module

Figure D-15: DFD 01.01.04.01.1.1 83C196_Microcontroller
D.7 EGR_control feature functional analysis

Figures D-17 to D-26 shows a set of DFDs and STDs developed in Cradle to represent the functional decomposition of the EGR_control feature shown in Figure D-16. The New Diesel development team, however, developed the feature functional models in TeamWork's implementation model as a further deployment of the PCS physical model.

Figure D-17: DFD 01.01.04.01.1.1.1.6 EGR_Control
Figure D-18: STD 01.1.4.1.1.1.1.6.1 PAT_EGR_Control

Figure D-19: DFD 01.01.04.01.1.1.6.9 Engine_Running_EGR
Figure D-20: DFD 01.01.04.01.1.1.6.9.1 Determine_EGR_Valve_Set_Point

Figure D-21: DFD 01.01.04.01.1.1.6.9.2 EGR_Valve_Control
Figure D-22: DFD 01.01.04.01.1.1.6.9.3 Determine_EGR_MAF_Set_Point

Figure D-23: DFD 01.01.04.01.1.1.6.9.4 MAF_Valve_Control
Figure D-24: DFD 01.01.04.01.1.1.6.9.6 Determine_EGR_Throttle_Set_Point

Figure D-25: DFD 01.01.04.01.1.1.6.9.7 EGR_Throttle_Control
D.8 EGR_control feature physical analysis

Figures D-27 to D-29 shows the physical analysis models of the EGR_control feature represented by Structure Charts (STCs) in Cradle’s implementation model. By comparing these models to the ones shown in Section D.7, one may conclude that although there is a correspondence between the functional models and the physical models there is no one-to-one mapping. The STCs in Figure D-28 and D-29 are in fact continuation of the one presented in Figure D-27.
Figure D-27: STC 1 Calculate EGR valve and throttle position

Figure D-28: STC 2 EGR Control Throttle Position
References


Appendix E
Details of the seat height adjuster example
This appendix aims to present with more detail the documents, models, requirements and attributes used for developing the seat height adjuster (SHM) example.

E.1 Source documents
Source documents include the following:

- Johnson Controls’ SHM desired characteristics: contains a list of geometric constraints required by Johnson Controls’ of the SHM. Johnson Controls’ is the seat manufacturer. (see Figure E-1)
- Johnson Controls’ FMEA (Failure Mode Effects Analysis) cause and effect diagram: showing a ‘fish-bone’ diagram to identify the failure modes and causes of raise/lower cushion total failure. (see Figure E-2)
- 4 Way FMEA: identification of potential issues to be observed on the SHM that may cause problems for effort, driveability, freplay, feel, noise, weight and cost. (see Figure E-3)
- List of customer requirements from the Ford document 00.00QFD-D02-1 Complete Vehicle, Quality Function Deployment [Ford Motor Company, 1995] (see Figure E-4)
- process sheet: containing a diagram that includes all assembly and manufacturing operations and a list of the SHM’s component parts. (see Figure E-5)

Johnson Controls Automotive U.K Ltd

Manual Height Adjuster Leadscrew Assembly Supplied by Adcock’s.

For CPF supply 13.5 Turns + Spring Assist.
J.C.A Ref - 119271
Adcock Ref - C052A

Significant characteristics of the component:

<table>
<thead>
<tr>
<th>REF</th>
<th>DESCRIPTION</th>
<th>MEASUREMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Front Trunnion bush diameters</td>
<td>13.8/13.9mm</td>
</tr>
<tr>
<td>2.</td>
<td>Front Trunnion width</td>
<td>26.9±0.1mm</td>
</tr>
<tr>
<td>3.</td>
<td>Rear fixing hole diameter</td>
<td>Dia 8.0±0.15/0</td>
</tr>
<tr>
<td>4.</td>
<td>Rear fixing slot width</td>
<td>3.1mm Min</td>
</tr>
<tr>
<td>5.</td>
<td>Open centre dim over front &amp; rear location</td>
<td>240.5mm</td>
</tr>
<tr>
<td>6.</td>
<td>Stop - stop to stop</td>
<td>54.0mm</td>
</tr>
<tr>
<td>7.</td>
<td>Torque with pre-load (T.B.A)</td>
<td>?NM min/max</td>
</tr>
<tr>
<td>8.</td>
<td>Handle interface - Knurl</td>
<td>Ref Adcock No T3010</td>
</tr>
<tr>
<td>9.</td>
<td>M5 Thread</td>
<td>M5</td>
</tr>
<tr>
<td>10.</td>
<td>M5 Depth</td>
<td>?mm</td>
</tr>
</tbody>
</table>

NOTE!!! This is for discussion

Figure E-1: Johnson Controls requirements
Figure E-2: Johnson Controls' SHM DFMEA

**Customer Specifications / Best in Class Performance**

**Effort**
- Thread pitch & tolerance
- Foreign body
- Noise
- Idealistic customer spec.
- Spring rating
- Lead screw straightness
- Co-linearity of thrust
- Thrust washer position/angle
- Number of balls
- Trim to shaft tolerance
- Surface finish turning surface

**Durability/freeplay/feel/noise**
- Material grades/specifications
- Thrust/wear life/calculated
- Wear interface tolerances/material
- Number of balls
- Thrust washer grade
- Gear
- Environmental effort
- Surface finish spec.
- Customer specifications!
- Gap to pass/step selected
- Insufficient pitch diameter

**Weight / Cost**
- Over-engineered
- Choice of materials
- Customer specifications
- Process selection
- Supplier selection
- Test requirements

Figure E-3: SHM DFMEA regarding effort, driveability, freeplay, feel, noise, weight and cost aspects
Figure E-4(a): Customer requirements captured by Ford Motor Company

Figure E-4(b): Customer requirements captured by Ford Motor Company
4.1.50  Driver’s window switch is easily distinguished from the other window switches

4.2.0  Controls Provide Quality Feel and Sound When Operated (For Example, Effort, Smoothness, Precision)

4.2.1  Functions understandable by touch

4.2.2  Low efforts

4.2.3  Smooth, precise operation

4.2.4  Good tactile, audible or visual feedback

4.2.5  Power/automatic assist

5.0.0  EXCELLENT VISIBILITY ALL AROUND

5.1.0  Good Forward Visibility (For Example, Clear View to Front and Side)

5.1.1  Large glass area

5.1.2  Ability to judge vehicle size (clearance for parking)

5.1.3  Command seating position

5.1.4  Underfloor view for all models

Figure E-4 (c): Customer requirements captured by Ford Motor Company

E.2 Product functional model

This section aims to present the SHM’s functional model developed in Cradle. Figures E-6 to E-18 illustrate the SHM functional analysis model at various levels of functional decomposition. They show 12 DFDs (Data Flow Diagrams) and 1 STD (State Transition Diagram). The numbering of the DFDs...
and STDs follow the functional decomposition hierarchy. For example, DFD 16 is the decomposition of the function 16, 'Provide_seat_height_adjustment', in DFD 0 (see Figure E-6). The functions in DFD 16 will be decomposed in the diagrams DFD 16.41, 16.42, and so on.

Figure E-6: DFD 0 – The SHM functional context diagram

Figure E-7: DFD 16 – The 'Provide_Seat_Height_Adjust_Function'
Figure E-8: DFD 16.41 - The 'Handle_Interface' function

Figure E-9: DFD 16.42 - The 'Holding_Interface_To_Frame' function

Figure E-10: DFD 16.43 - The 'Handle_at_a_constant_position_from_user' function
Figure E-11: DFD 16.44.41 - The 'Provide_frame_displacement' function

Figure E-12: DFD 16.44.41 - The 'Transmit_handle_rotation' function
Figure E-13: DFD 16.44.42 – The 'Transform_rotation_from_handle_into_translation' function

Figure E-14: DFD 16.45 – The 'Moving_interface_to_frame' function
Figure E-15: DFD 16.47 - The 'Provide a constant handling effort' function

Figure E-16: DFD 16.47.41 - The 'Provide pre_load' function
E.3 Process functional model

This section aims to present the detailed process functional model elaborated for the manufacturing and assembly processes of the SHM. The top level diagram contains all life cycle processes of the SHM, but only the production process is further decomposed. The models were developed in Cradle and consist of a series of Behaviour Diagrams at various levels of decomposition. Figures E-19 to E-25 present the Behaviour Diagrams. Similarly to the DFDs and STDs, the BD numbering follows the functional decomposition of the life cycle processes in BD-16, for example BD 16.2.
Figure E-19: BD - 16: SHM life cycle processes

- N number of parts to be manufactured
- M number of subassemblies
- Depends on levels of subassemblies

Figure E-20: BD - 16.2: SHM production
Figure E-21: BD - 16.2.1: Manufacture N parts

Figure E-22: BD - 16.2.1.1: Manufacture part 1
**Figure E-23: BD – 16.2.3: Assemble SHM**

**Figure E-24: BD – 16.2.4: Assemble M sub-assemblies**
E.4 Process physical model

This section aims to present the process physical model developed for the SHM. Focus is given to the manufacturing and assembly processes. The top level process physical model consists of the process sheet shown in Figure E-5 written in a Behaviour Diagram format, following the IDEF0 notation. It is presented in Figure E-26. The process physical model consists of a series of lower level Behaviour Diagram decomposing the processes identified in Figure E-26. The models were also developed using Cradle. Figures E-26 to E-42 shows the Behaviour Diagrams developed. The numbering corresponding to the physical decomposition starts with 16 as a reference to the SHM and is followed by the numbers of the life cycle processes being decomposed. '16' is in reality a short reference to the SHM. The SHM's number in the vehicle breakdown structure is in reality 11.13.17.12.16: 11 corresponds to the vehicle, 13 to the interior, 17 to the seat and 12 to mechanisms.
Figure E-26: BD – 16.22: SHM_manufacturing_and_assembly
Figure E-27: BD 16.22.21 - Manufacture_spindle

Figure E-28: BD 16.22.21.21 - Produce_blank_D069_P01
Figure E-29: BD 16.22.21.22 – M5_thread_D069_P02

Figure E-30: BD 16.22.23 – Manufacture_hobbin
Figure E-33: BD 16.22.24.21 - Cut_to_length_D096_P01

Figure E-34: BD 16.22.25 - Manufacture_ball_race_assembly_D103_P01
Figure E-35: BD 16.22.26 - Manufacture_ferrule

Figure E-36: BD 16.22.26.21 - Produce_blank_D104_P01
E.5 Requirements

This section aims to present the requirements captured for the SHM and documented in Cradle. The requirements were derived from the source documents presented in Section E.1 and from the [Boothroyd, Dewhurst & Knight, 1994] and [Bedworth et al., 1991] books on design for assembly and concurrent engineering, respectively. The following is a Cradle report called: Requirements List. Types of requirements are identified by the numbering used: JC refers to requirements from the documents provided by Johnson Controls (please see Figures E-1 to E-3), S refers to requirements provided by Ford (please see Figures E-4), MA refers to requirements provided by the books just mentioned above.

**Key:**
- **TEXT** Seat height adjuster must be durable
- **VERIFICATION** Johnson's Control Durability Cycle Test

**Key:**
- **TEXT** Minimise freeplay
- **VERIFICATION** Johnson Control's operating test
  - Maximum number of turns allowed

**Key:**
- **TEXT** Minimise weight
- **VERIFICATION** Weigh SHM

**Key:**
- **TEXT** Minimise cost
- **VERIFICATION** Minimise coat

**Key:**
- **TEXT** Minimise risk of involuntary lowering
- **VERIFICATION** Johnson's controls cyclic test

**Key:**
- **TEXT** Minimise risk of seat not going up
- **VERIFICATION** Johnson's controls cyclic test

**Key:**
- **TEXT** Minimise risk of seat not going down
- **VERIFICATION** Johnson's controls cyclic test

**Key:**
- **TEXT** Geometric constraints
- **VERIFICATION** CAD data verification

**Key:**
- **TEXT** Front trunnion bush diameters 13.8/13.9mm
- **VERIFICATION** CAD data verification

**Key:**
- **TEXT** Front trunnion width 26.9 +0/−0.2mm
- **VERIFICATION** CAD data verification

**Key:**
- **TEXT** Minimise risk of raise/lower cushion total failure
- **VERIFICATION** Johnson's controls cyclic test

**Key:**
- **TEXT** Minimise risk of seat not going down
- **VERIFICATION** Johnson's controls cyclic test

**Key:**
- **TEXT** Minimise risk of seat not going up
- **VERIFICATION** Johnson's controls cyclic test

**Key:**
- **TEXT** Minimise risk of seat not going down
- **VERIFICATION** Johnson's controls cyclic test

**Key:**
- **TEXT** Minimise risk of raise/lower cushion total failure
- **VERIFICATION** Johnson's controls cyclic test

**Key:**
- **TEXT** Minimise risk of involuntary lowering
- **VERIFICATION** Johnson's controls cyclic test
Rear fixing hole diameter 8.02 +0.5/-0mm

Key: .4

VERIFICATION CAD data verification

Rear fixing slot width 3.1 mm minimum

Key: .5

VERIFICATION CAD data verification

Open centre dimension over front and rear location 236.5 mm

Key: .6

VERIFICATION CAD data verification

MS thread M5x0.8-6G

VERIFICATION CAD data verification

MS depth 12mm minimum

VERIFICATION CAD data verification

Key: .7

VERIFICATION CAD data verification

Functional constraints

VERIFICATION Operational testing/measurements

Stroke-stop to stop 54.9 mm

Key: .1

VERIFICATION Operational testing/measurements

Torque with pre-load 2.8 Nm maximum

Key: .2

VERIFICATION Operational testing/measurements

Handle interface-knurl ref Adcock SP1401

VERIFICATION Operational testing/measurements

Key: .3

Verifica

Design for a minimum number of parts

VERIFICATION Count number of parts

Key: .4

Develop a modular design

VERIFICATION

Key: .5

Minimise part variations

VERIFICATION

Design parts to be multifunctional

VERIFICATION

Design parts for multi-use

VERIFICATION

Design parts for ease of fabrication

VERIFICATION

Avoid separate fasteners

VERIFICATION

Minimise assembly directions; design for top-down assembly

VERIFICATION

Maximise compliance; design for ease of assembly

VERIFICATION

Minimise handling; design for handling presentation

VERIFICATION

Evaluate assembly methods

VERIFICATION

Eliminates adjustments

VERIFICATION

Avoid flexible components; they are difficult to handle

VERIFICATION

Use parts of known capability

VERIFICATION

Allow for maximum Intolerance of parts

VERIFICATION

Use known and proven vendors and suppliers

VERIFICATION

Use parts at derated values with no marginal overstress

MA18

Key: .1

Minimise subassemblies

MA19

Key: .2

Use new technology only when necessary

MA20

Key: .3

Emphasize standardization

MA21

Key: .4

Use the simplest possible operations

MA22

Key: .5

Use operations of known capability

MA23

Minimise setup and interventions

MA24

Key: .6

Undertake engineering changes in batches

MA25

Key: .7

Provide for personal comfort/convenience

VERIFICATION R-202

Key: .8

Can personalize seat, head rest and arm rest position

VERIFICATION R-202

Key: .9

Easy and comfortable to operate controls and displays

VERIFICATION R-202

Key: .10

Easy to see, understand, reach and operate controls and displays

VERIFICATION R-202

Key: .11

Can personalize seat, head rest and arm rest position

VERIFICATION R-202

Key: .12

Can personalize seat, head rest and arm rest position

VERIFICATION R-202

Key: .13

Can personalize seat, head rest and arm rest position

VERIFICATION R-202

Key: .14

Can personalize seat, head rest and arm rest position

VERIFICATION R-202

Key: .15

Can personalize seat, head rest and arm rest position

VERIFICATION R-202

Key: .16

Can personalize seat, head rest and arm rest position

VERIFICATION R-202
E.6 Attributes capture

This section aims to provide some examples of how attributes are captured from the models. The examples are functional attributes being extracted from functional models of product, process and organisation. The models were also developed using Cradle. Attributes are added manually to the models and are associated to the model elements.

E.6.1 Product attributes

Figures E-43 to E-47 shows some DFDs and the STD part of the product functional model and the functional attributes associated to the elements in the models.

Figure E-43: Top level product functional attributes extracted from the DFD 12 - Seat_Height_Adjust_Mechanism_Context
Figure E-44: Second level product/functional attributes extracted from DFD12.16
Provide/seat_height/adjustment

Figure E-45: Third level product/functional attributes extracted from DFD12.16.44—Provide/frame/displacement
Figure E-46: Third level product functional attributes extracted from DFD 12.16.47 –
Provide a constant handling effort

Candidate functional attributes:
- reliability
- risks of failure
- time to respond

Figure E-47: Third level functional attributes extracted from STD 12.16.46 –
Provide limits for displacement
E.6.2 Process attributes

Figures E-48 to E-53 show Behaviour Diagrams (BDs) developed in Cradle and the process functional attributes associated to the elements in the models. The top level diagram in Figure E-48 is part of the diagram shown in Figure E-19.

Legend:
- candidate functional attributes

*Figure E-48: SHM_production attributes extracted from the SHM life cycle process model BD 16*

*Figure E-49: SHM_production attributes extracted from BD 16.2*
Figure E-50: Process functional attributes extracted from the BD 16.2.1 — Manufacture_N_parts

Legend: - candidate functional attributes

Figure E-51: Process functional attributes extracted from BD 16.2.1.1 Manufacture_part_1

Legend: - candidate functional attributes
Figure E-52: Process functional attributes extracted BD 16.2.4 – Assemble_M_subassemblies

Figure E-53: Process functional attributes extracted from BD 16.2.4.1 – Assemble_subassembly_1
E.6.3 Organisation attributes

Figures E-54 to E-57 shows the UCD (Use Case Diagrams), COD (Collaboration Diagram) and SQD (Sequence Diagram) and the organisation functional attributes associated to elements in those diagrams. The diagrams were developed using Cradle.
Candidate functional attribute: - balance

Figure E-56: Organisation functional attributes extracted from the COD 16.2.1.5 – Manufacture tube

Candidate functional attributes: -risks -work-in-progress-levels -cycle time

Figure E-57: Organisation functional attributes extracted from the SQD 16.2.1.5 – Manufacture tube
References


Ford Motor Company, Complete vehicle – quality function deployment (QFD). 00.00QFD-D02, October 1995, 31 pages
Appendix F
Results along the complexity analysis procedure applied to the gasoline PCS

This appendix aims to present the results obtained during the application of a complexity analysis procedure to the Ford's idle speed control strategy software of the gasoline powertrain control system (PCS). The sections in this appendix contain the example results after each step of the procedure.

F.1 Structure of the software strategy document

Figures F-1 to F-15 depict the structure of the ISC_KBAD0 strategy document that was the object under analysis. The document corresponds to section 12 of an overall software application strategy. The document contains software written in C. The way the document is structured reflects the way Ford partitioned the software into software subsystems (features) and modules. The following figures contain name and numbering of the document sections (12.x) and sub-sections (12.x.x and 12.x.x.x).

Figure F-1: Top-level sections of the ISC_KBAD0 strategy document

Figure F-2: Composition of the inlet air control executive feature-IACEX-V3.0_IACEX- section 12.1
Figure F-3: Composition of the 'generic idle speed control' strategy module
IACEX_OVERVIEW
W_4- sub-section 12.1.1

Figure F-4: Composition of the 'mode select' strategy module-
IACEX MODE_SELECT_2-sub-section 12.1.2

Figure F-5: Composition of the 'lac (idle speed control) SCCD output-
IACEX_DTY_CON V_2-Sub-section 12.1.4

Figure F-6: Composition of the IACDN - NO_VERSION_LEVEL feature - section 12.2

Figure F-7: Composition of the 'desired RPM calculation' strategy module
IACDN_DSDRPM_BASE_1 - sub-section 12.2.2

Figure F-8: Composition of the 'DSDRPM (loads) calculation' strategy module
IACDN_DSDRPM_LOADS_1 - sub-section 12.2.4
Figure F-9: Composition of the 'battery charging RPM calculation' strategy module IACDN_DSRPM_BATV_1 — sub-section 12.2.6

Figure F-10: Composition of the 'IACFB-no_version_level' feature — section 12.3

Figure F-11: Composition of the 'IPSIBR calculation' strategy module IACFB_IPSIBR_2 — sub-section 12.3.1

Figure F-12: Composition of the 'KAM update (ISCKAM_update)' strategy module IACFB_ISCKAM_1 — sub-section 12.3.3

Figure F-13: Composition of the 'IACFF-no_version_level' feature — section 12.4
F.2 Software breakdown structure

Figures F-16 to F-24 present structure charts (STCs) depicting the software breakdown structure, i.e., the relationships in terms of subroutines calls among software modules. Those software modules correspond to the lower level sub-sections of the strategy documents (12.x.x.x). The subroutine calling structure implies that these modules are at different levels in the software breakdown structure.

Figure F-16: Top-level software modules and IAC_EXEC (sub-section 12.1.1.1) structure
Figure F-20: ISCKAM_UPDATE (sub-section 12.3.3.2) structure

Figure F-21: DESMAF_PRE_CALC (sub-section 12.4.1.1) structure
F.3 Translation of structure into graph format

Figures F-25 to F-49 show data flow diagrams (DFDs) that translate the structure represented in the structure charts into a graph format. Figures F-25 to F-49 represent a hierarchical decomposition of the
software. The lowest level DFDs contains the software units (DFD bubbles) representing the vertices of the graph and the data or control flows (DFD arrows) representing the edges between vertices.

Figure F-25: DFD 40 - The top-level DFD diagram showing the idle speed control software and its interfaces

Figure F-26: DFD 40.1 - IAC_EXEC decomposition
Figure F-28: DFD 40.1.1.2 - INTS_UPDATE (subsection 12.1.1.3)

Figure F-29: DFD 40.1.1.3 - N_RATCH_UPDATE (subsection 12.1.1.4)

Figure F-30: DFD 40.1.1.5 - IAC_MODE_SELECT (subsection 12.1.2.1)
Figure F-31: DFD 40.1.1.6 - ISCDTY_CALC (subsection 12.1.3.1)

Figure F-32: DFD 40.1.1.7 - IDLE_DRIVE_MODE (subsection 12.1.8.1)
Figure F-33: DFD 40.1.1.9 - IPSIBR_Calc (subsection 12.3.1.1)

Figure F-34: DFD 4.1.1.9.4 - ISCPSI_CONTROL_Calc (sub-section 12.3.1.4)
Figure F-35: DFD 4.1.1.10 DASHPOT_EXECUTION_PROCESS (sub-section 12.4.9.1)

Figure F-36: DFD 40.1.1.10.1 – BASE_DASHPOT_CALCULATION (sub-section 12.4.9.4)
Figure F-37: DFD 40.1.1.11 - DESMAF_PRE_CALC (sub-section 12.4.2.1)

Figure F-38: DFD 40.1.1.12 - DSDRPM_CALC (sub-section 12.2.2.1)
Figure F-39: DFD 40.1.1.12.1 – DSDRPM_OUTPUT (sub-section 12.2.4.2)
Figure F-40: DFD 40.1.13 ISCKAM_UPDATE (sub-section 12.3.3.2)

Figure F-41: DFD 40.1.2 IN CRANK MODE ENGINE MOVING
Figure F-42: DFD 40.1.2.2.1 DSDRPM _CALC (sub-section 12.2.2.1)
Figure F-44: DFD 41 ISC_FMEM (sub-section 12.1.7.1)

Figure F-45: DFD 42 ISC_PERLOAD_ISC (sub-section 12.1.6.1)
Data Dictionary showing the composition of calibration constants (ENGINE CALIBRABLES) in Figures F-25 to F-49:

CONSTANTS COMPLETE FLOW MEANING ENGINE CALIBRABLES

CONSTANTS 1 COMPLETE FLOW MEANING CALIBRATION CONSTANTS

CONSTANTS 3 COMPLETE FLOW MEANING CALIBRATION CONSTANTS

CONSTANTS 2 COMPLETE FLOW MEANING CALIBRATION CONSTANTS

CONSTANTS 4 COMPLETE FLOW MEANING CALIBRATION CONSTANTS

CONSTANTS 16 COMPLETE FLOW MEANING CALIBRATION CONSTANTS

CONSTANTS 17 COMPLETE FLOW MEANING CALIBRATION CONSTANTS

COMPOSITION [COAST_fracbypassedチャー/CHAMPLAOD)] MEANING CALIBRATION CONSTANTS

COMPOSITION [RUNUP_DIFF] MEANING ENGINE CALIBRABLES

COMPOSITION [ISET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALLBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALLBRABLES

COMPOSITION [ACET] MEANING ENGINE CALLBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES

COMPOSITION [ACET] MEANING ENGINE CALIBRABLES
The processes (bubbles in the DFDs) at the lowest level of the DFD decompositions are the vertices of the graph to be used in the complexity analysis procedure. Figure F-50 contains the list of those vertices. Vertex weight represents the number of lines of C code implementing a corresponding process.

![Figure F-50: List of vertices in the graph represented by the lowest level DFD processes](image-url)
In order to obtain the graph edges it is necessary to obtain a list of the flows among the DFD processes at the lowest level in the DFD hierarchy. The list is as following:

```
49.1.1.1 PROCESS SPEC
  F13, E113, D113
  INPUTS
  OUTPUTS
  h1a

49.1.1.10.2 PROCESS SPEC
  BASE OUTPU ITS
  INPUTS
  OUTPUTS
  h1a

49.1.1.10.1.2 PROCESS SPEC
  BASE OUTPUTS
  INPUTS
  OUTPUTS
  h1a

49.1.1.10.1.1 PROCESS SPEC
  BASE OUTPUTS
  INPUTS
  OUTPUTS
  h1a
```

This is a list of DFD flows and is used to obtain graph edges from the lowest level in the DFD hierarchy.
40.1.3 PROCESS SPEC

40.1.4 PROCESS SPEC

40.1.5 PROCESS SPEC

40.1.6 PROCESS SPEC

40.1.7 PROCESS SPEC

40.1.8 PROCESS SPEC

40.1.9 PROCESS SPEC

41.1 PROCESS SPEC

44.1 PROCESS SPEC

42.1 PROCESS SPEC

43.1 PROCESS SPEC

44.1 PROCESS SPEC

45.1 PROCESS SPEC

46.1 PROCESS SPEC

F.6 List of graph edges and associated information

From the flows listed in Section F-5, the following information is extracted: source vertex information, target vertex information, identification of whether the flow is a data (D) flow or a control (C) flow and derived edge weight (D weights 1 and C weights 2). The information for each flow is gathered in Figure 51.

<table>
<thead>
<tr>
<th>Source vertex information</th>
<th>Edges information</th>
<th>Target vertex information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex number</td>
<td>Flow</td>
<td>Data/Control</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure F-51: List of edges and associated information (continues)
<table>
<thead>
<tr>
<th>Source vertex information</th>
<th>Type</th>
<th>State</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data G</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data H</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data J</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data K</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data L</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data O</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data W</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
F.7 List of vertices, their neighbours and edge weights

Figure F-52 shows the final list of vertices, their neighbours and the compiled edge weights derived from Figure F-51.

shows the Chaco-2.0 input files derived from Figure F-52.
F.9 Chaco-2.0 sequencing output files

Figure F-54 shows the output of the vertex sequencing process implemented by Chaco-2.0.

<table>
<thead>
<tr>
<th>% Sequencing output from Chaco</th>
<th>% Order</th>
<th>Vertex Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 51 -0.07924</td>
<td>17 53 -0.05585</td>
<td></td>
</tr>
<tr>
<td>4 52 -0.06375</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 57 -0.05585</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 67 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 70 -0.05152</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 71 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 72 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 73 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 74 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 75 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 76 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 77 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 78 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 79 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17 80 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18 81 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19 82 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 83 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21 84 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22 85 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23 86 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24 87 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25 88 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26 89 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27 90 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28 91 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29 92 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30 93 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31 94 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32 95 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33 96 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34 97 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35 98 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36 99 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37 100 -0.06196</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38 101 -0.06196</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure F-55: Example of Chaco-2.0 partitioning output file

Figure F-55 shows an example of a partitioning output file from Chaco-2.0. The partitioning algorithms used in this example is the imbalanced multilevel KL for obtaining 32 sets (partitions). Balance here means that the summation of vertex weights in the partitions are made as close as possible to each other. In this example there is no requirement for balance. Re-sequencing of vertices always followed partitioning in this complexity analysis procedure.
F.11 Calculation of a complexity index

A given graph is partitioned by Chaco-2.0 and has its vertices re-sequence using Chaco's sequencing capability. Vertices are then deployed on the rows and columns of a square matrix in an Excel spreadsheet. The cells of the matrices are filled in with edge weights following the format of an N2 chart. Figure F-56 presents a subroutine developed in Visual Basic of Microsoft Excel 5.0 to calculate a complexity index according to algorithm described by [Hitchins, 1992].

Function ARCIND(First_Row, First_Column, Number)
ARCIND = 0
For I = First_Row To (First_Row + Number - 1)
    For J = First_Column To (First_Column + Number - 1)
        K = Abs(I - First_Row - J + First_Column)
        If (K > 0) Then ARCIND = ARCIND + ActiveSheet.Cells(I, J).Value * K
    Next J
Next I
End Function

Figure F-56: Calculation of the complexity index

F.12 Resulting matrices

Figures F-57 to F-62 shows the matrices resulting from the partitioning and sequencing processes. The matrices were obtained with the following algorithms:

- Ford's partitioning with Chaco's sequencing, 5 sets (Figure F-57)
- Imbalanced multi-level KL, 4 sets (Figure F-58)
- Balanced spectral bisection, 4 sets (Figure F-59)
- Ford's partitioning with Chaco's sequencing, 32 sets (Figure F-60)
- Imbalanced multi-level KL, 32 sets (Figure F-61)
- Balanced spectral bisection, 32 sets (Figure F-62)

The rows and columns of the matrices contain:

- the sets or partitions identification (e.g. from 0 to 3 or 0 to 31) obtained from the partitioning algorithms in Chaco-2.0;
- the order of the vertices obtained from the sequencing algorithm in Chaco-2.0;
- the actual vertex numbers.

The top left cell of the matrices contains the complexity index. The diagonal cells contain vertex weights. The non-diagonal cells contain edge weights.
Figure F-57: Ford's partitioning into 5 sets and Chaco's sequencing.
Figure F-58: Imbalanced, multi-level KL partitioning into 4 sets
Figure F-59: Balanced spectral bisection partitioning into 4 sets
Figure F-61: Imbalanced multi-level KL partitioning into 32 sets
Figure F-62: Balanced spectral bisection partitioning into 32 sets
References

3SL, *Cradle v3.2 manuals*, Barrow-in-Furness, 1998


A SYSTEMS ENGINEERING ENVIRONMENT FOR INTEGRATED AUTOMOTIVE POWERTRAIN DEVELOPMENT

G. Loureiro, P.G. Leaney and *M. Hodgson
Dept. of Manufacturing Engineering
Loughborough University
Loughborough, U.K.
*Ford Motor Company
Research & Engineering Center
Laindon-Basildon-Essex, U.K.

ABSTRACT

Automotive powertrain control system (PCS) development faces tightening environmental requirements, shortening development cycle times and growing complexity. To cope with such an environment, PCS development is moving from a traditional evolutionary to a structured approach. The structured approach is supported by computerised structured analysis methods. However, these methods concentrate on the functional and interface requirements of the product.

This paper aims to describe how a systems engineering environment (SEE) can be used for integrated PCS development by addressing not only product functional and interface requirements but also life cycle processes and organisational requirements. The particular SEE used, a commercial software package, provides:

- notations from different modelling paradigms, used for integrated functional, product, process and organisational modelling;
- cross-references, used to link functional, product and life-cycle processes attributes;
- frames to capture further information about each data items (e.g. CAD drawings).

A major benefit of the application of the SEE for an integrated PCS development is the ability to investigate early in the product development/evolution process the interactions between requirements and attributes not only of the product but also of its life cycle processes and development organisation. It is demonstrated how an SEE can be used to provide better product quality, lower life cycle cost and shorter development time.

The application of a systems engineering approach to a product, its life cycle processes and its development organisation in an integrated manner encompasses concurrent engineering. It provides a much broader scope for product development, in this case, the PCS subsystem.

NOMENCLATURE

3SL structure software systems limited
AVT advanced vehicle technology
BD behaviour diagram
CAD computer aided design
CAE computer aided engineering
CAM computer aided manufacturing
CASE computer aided software engineering
CE concurrent engineering
CMS configuration management system
DD data dictionary
dfd data flow diagram
DFMA design for manufacturing and assembly
EEC electronic engine control
EGR exhaust gas recirculation
FBD function block diagram
FMEA failure mode effects analysis
FPDS Ford product development system
GT group technology
IT information technology
NIST national (USA) institute for standardisation
OMT object modelling technique
PCS powertrain control system
PCSE powertrain control system engineering
PDM product data management
PP performance parameter
QFD quality function deployment
SE systems engineering
SEE systems engineering environment
SEP systems engineering process
STD state transition diagram
UCD use case diagram

INTRODUCTION

The traditional design approach in the automotive industry is to optimise components (Gormley and Macisaac, 1989). This component approach is founded on the belief that any design could be broken into independent and self-supporting components. The optimisation and the evolution of the whole is assumed from the optimisation and evolution of every component (Ziebart, 1991).

From the mid-80s, concurrent engineering methods (e.g. DFMA, GT, Taguchi, FMEA, QFD, Value Engineering) and tools (e.g. CAD, CAE, CAM, PDM) have been supporting this component approach. Component design started to take into consideration life cycle process requirements. For example, existing components are evaluated according to DFMA rules and the component is modified, preserving its...
component is modified, preserving its functionality, to improve its ease of manufacturing and assembly. CAD/CAM tools such as feature based technology allow the simultaneous design and process planning of components. More recently, communications and integration technologies have been helping to close the gaps between the organisations responsible for the various component life cycle processes.

As the automobile grew in functionality and became a more multidisciplinary product, more components, component interconnections and life cycle process components were necessary. A non-structured evolutionary development process took place. Better components were substitute for old ones and new components were simply added to the current design version. Examples of the consequences of this approach applied to automotive electronics show that it has not led to the optimisation of the whole product and its life cycle processes (Ziebart, 1991):

- more control units, sensors and actuators need more space. However, space for components get less and less because space is required for the convenience of the passengers;
- the overall system reliability may decrease because of the increasing number of parts;
- the increasing complexity of the system structure may deteriorate the serviceability and the handling of the electronic systems;
- in many cases, the driver has been overtaxed by the number of differently reacting electronic systems;
- solutions for interfaces are often developed in a reactive manner because they must be implemented at that time when the system definition was already finished.

Concurrent engineering (CE) does not provide the complete framework to deal with such an increasing complexity. The level of integration and complexity of automobile functionality requires a better understanding of the contribution of each product and/or process component to the whole and the interactions between these components. CE of components would help component evolution, but only an interdisciplinary, collaborative approach to derive, evolve and verify a life-cycle balanced system can deliver better results that meet customer expectations and public acceptability. This approach is systems engineering (IEEE, 1995).

The systems engineering process (SEP) is a structured and requirements-driven process very much applicable to the current needs of automotive product development for a global and highly segmented market. Requirements driving the SEP include:

- corporate, regulatory and customer requirements;
- life cycle process requirements;
- functional, physical and operational requirements.

Systems engineering (SE) is a traditional discipline used in the engineering of complex products (e.g. aerospace sector). The premise of this paper is that the tools and techniques used in the aerospace provide some lessons for the automotive sector. Various attempts have been made in the past (e.g. GM, Chung, 1990; Arnoult, 1991) but it is proposed here that the time is right for a full scale development of SE in the automotive sector because:

- the CAD/CAM/CAE/PDM and IT communications infrastructure is developed now to support a traceable requirements driven process;
- automobiles are increasingly complex products serving an increasingly sophisticated and segmented market place with complex corporate, legal and customers requirements.

The perceived need is now for a structured requirements driven process to maximise the effectiveness of the product development process for the global market.

This paper aims to describe how a systems engineering environment (SEE) can be used for integrated PCS development by addressing not only product functional and interface requirements but also life cycle process and organisational requirements.

The paper is organised to meet the following objectives:

- identify the needs for an integrated PCS development and how they have been addressed so far;
- propose a total view approach that applies, in an integrated manner, the SEP to product, life cycle processes and development organisation;
- introduce a commercial software package SEE that has the necessary capabilities to implement the total view approach;
- demonstrate how the SEE can be used for the a better, cheaper and faster PCS development;
- draw some conclusions and outline the needs for further research and development.

IDENTIFYING AND ADDRESSING THE NEEDS FOR AN INTEGRATED PCS DEVELOPMENT

(Loureiro, 1996), reporting some results of case studies at Ford-AVT-PCSE, affirmed that despite the technological improvement, Ford's gasoline PCS has not evolved in a structured manner over the past 20 years. The development process has been mainly change driven, based on improving past versions of the system. Changes are directly implemented on the detailed design, at the bottom level of the product breakdown structure without links with changes on higher level requirements. Examples of requirements having been used are 'change this subsystem to another version', 'add that sensor', 'delete that subsystem'. To evolve software, for example,
changes are implemented directly on the software code, with very poor visibility of interactions with other software subsystems. The effects of changes have been only evaluated when the whole vehicle is tested to see whether it meets the emissions regulations.

That traditional non-structured development process has no longer been able to cope with (Loureiro, Leaney and Pickman, 1996):

- increasing PCS functionality. The simple addition of functions in a non-structured way, through the addition of parts may lead to decreased overall system reliability, less space for the convenience of the passengers and decreased maintainability;
- increasing PCS complexity. The addition of functions leads also to the addition of desired or undesired interactions among parts what leads to greater complexity. Without an appropriate systems analysis, this increasing complexity may lead to the need of later changes in the overall system, decreased reliability and maintainability;
- the need to shorten development life cycle, considering that some competitors can deliver vehicles with a shorter development life cycle and that microelectronics systems double their functionality every 18 months (while a car has a development time of about 4 years and an average lifecycle of an individual model of 7 to 9 years). The late implementation of changes, not supported by appropriate analysis, increases the development life cycle;
- a very dynamic product development cycle with changes coming not only because of performance or functional challenges, but also due to changes on competitors' products, upper management feeling, technological advances, etc. Changes have been implemented in a very ad-hoc and time consuming manner, without proper risk analysis and configuration management;
- dynamic organisational structure which tries to constantly adapt to market and technological trends;
- constrained resources what requires 'right-first-time' philosophy without wasting of time and money.

Ford’s gasoline PCS case study also identified changes on other items that affected the PCS development (Loureiro, 1999):

- the number and types of stakeholders increased. For example: the market became more differentiated, number of interacting subsystems increased, new environment agencies were created;
- new requirements appeared and constraints became more tightened. For example: new driveability requirements, more stringent emissions and fuel economy requirements;
- number of functions in the product increased. Examples of new functions are on-board diagnostics II, cruise control;
- number of modes of operations increased.

Initially modes of operations were related only to protection of the system being controlled, now there are different modes of operation depending on throttle position, temperature, altitude, engine state;
- same functions implemented by different product architectures;
- better partitions in terms of cohesion and coupling can be found for current software modules;
- the process of evolving the control software has been split into four different streams depending on the nature of changes to be performed and on the bookshelf status of the software control;
- the product development organisation also changed. Control algorithm developers and software developers were merged into the same group.

One primary lesson from the gasoline PCS case study is the need for visibility of interactions among product subsystems. However, as mentioned above, product subsystems are not only affected by changes on other product subsystems but also by changes on product life cycle process systems and product development organisation systems. As a consequence, it is necessary that a development environment be able to integrate product, life cycle processes and development organisation analysis. Such an environment must:

- provide means to manage requirements, functions, product and processes variety;
- provide means to analyse product, its life cycle processes and product development organisation;
- provide means to analyse the interactions of the items above with the environment;
- provide means to analyse the interactions among these items themselves.

From 1994, a diesel PCS has been developed by Ford-AVT-PCSE, using an SE approach (Loureiro, Leaney and Pickman, 1996) to address the product-related needs listed above: increasing product functionality and complexity, functional and interface requirements management, product functional and architecture analysis, behaviour analysis. No life cycle processes or product development organisation requirements have been formally considered. However, the approach, methods and tools used, provided a starting point for a set of functions and capabilities that need to be further expanded to include life cycle process and product development considerations. Table 1 lists the functions and capabilities of tools used for the diesel PCS development.

**THE TOTAL VIEW APPROACH**

The gasoline PCS case study demonstrated the need to analyse the product, its life cycle processes and the product development organisation and their interactions all together to have an integrated PCS development. The diesel...
PCS case study provided the basic functions and capabilities of a set of tools to implement an SE approach to PCS product development. That was still a partial view approach because it focused only on the product system. For a total view approach, the SEP must be applied, in an integrated manner, to the product, process and people systems. These three systems correspond to the product itself, its life cycle processes and the product development organisation, respectively. This section introduces the concept of a total view approach and its relationship with the SEP.

Modern definitions of the word “system” put together product and its life cycle processes within the “basic building block of a system”. The IEEE-Std-P-1220 (IEEE, 1995) standard on SE defines a “basic building block of a system” as:

the system, its related product(s), the life cycle processes required to support the products and customers for the product(s), and the subsystems that make up the product(s).

Examples of life cycle processes in the IEEE-Std-P-1220 are: product system engineering and integration; product distribution, support and disposal; product training; product test; product manufacturing.

As a consequence of this definition of system, it can be concluded that the result of the SEP is not only the product but also its life cycle processes. Product systems and life cycle processes can be further deployed into subsystems producing a hierarchy of building blocks. Each building block can be considered a system on its own. The SEP is applied recursively on each building block at each development layer. Figure 1 presents the SEP as defined by the IEEE-Std-P-1220.

According to (Martin, 1997) there are three types of elements in a project that need to be integrated: organisational elements, end product elements and, enabling product elements. In line with the IEEE-Std-P-1220 (see Figure 1), these elements correspond to the physical architecture of the product development process system, the product system and the life cycle processes systems, respectively. The people, product and process systems of the total view approach being proposed here also include the functional architecture elements. It is important to notice that the people system, being the product development process system, may be regarded as one of the process systems. However, the people system focuses on who implements or interacts with the process whereas the process system focuses on what the process must do and how it should be done. The people system focuses on information flow whereas the process system may contain material flow. (Clark and Fujimoto, 1991) also stresses the importance of an information system view for a process system.

Therefore, a total view approach for an integrated product development must integrate the following systems:

- the product system: which consist of the product and its subsystems as in (IEEE, 1995) or of the end product and its subsystems as in (Martin, 1997). For example, the PCS is a product system. Its subsystems are, for example: the spark control system, the EGR control system, the fuel injection control system;
- the process systems: which consist of the life cycle processes as defined by (IEEE, 1995) or of the enabling subsystems and enabling products as defined by (Martin, 1997). Examples of process systems corresponding to the PCS product system are the following life cycle processes: engineering, manufacturing, calibration, evolution, operation, service, disposal;
- the people system: which consist of the product development process enabling subsystems (functional aspects) and enabling products (architectural or implementation aspects) as defined by (Martin, 1997), emphasising the information flow between the elements of the system. The PCSE organisation is an example of people system.

Each system type above must not only be integrated with the other systems of its own type, but must also be integrated with all the elements of the other two types as shown in Figure 2. The SEP is applied recursively to all levels of the product, processes and people systems breakdown structure. The optimisation of the total system must consider all three elements compromising attributes such as: product performance, process lead time and people satisfaction.

(Mar, 1997) also uses product, process and people systems terminology to define a total system.

SEE AND THE TOTAL VIEW APPROACH

Despite the tools listed in Table 1 be used mainly for software system development, they set some basic capabilities of an SE: requirements management, functional analysis, behavioural analysis and configuration management. Functional analysis provided by CASE tools can also be used for functional analysis of hardware and life cycle process systems. CASE tools can model software functions and architecture. However, for implementing the total view approach, they must be expanded for hardware and life cycle process analysis. Physical architecture analysis is essential for the synthesis subprocess (see Figure 1) of the SEP.

Examples of essential requirements for hardware architecture modelling are:
- to model physical links between entities;
- to retrieve discipline specific models, such as, mechanical drawings, circuit schemas, etc;
- to attach attributes to each entity in the
model for further analysis.

(Schlenoff et al., 1996) provides a list of requirements for process modelling. Examples of these requirements are:

- to represent time sequencing (sequences, concurrency);
- to represent alternative tasks and conditions;
- to attach attributes to each task in the process model for further analysis.

To model the people element of the total view approach the most used model paradigm is object orientation. Jacobson's use case models (Jacobson et al., 1995) and OMT are some examples of modelling tools used for this purpose.

A tool that implements all the basic capabilities listed above and the required expansions for hardware, process and people systems analysis is Cradle (hereafter referred as "the SEE"). It is a commercial software tool, developed by 3SL. It is defined as a systems and software engineering environment that provides through life support from requirements capture to system implementation with supporting configuration management, project control, and document generation capabilities. It provides requirements management in a controlled environment with full life cycle management in an integrated product support environment. (3SL, 1997).

The SEE is composed of the following modules, as illustrated in Figure 3.

The characteristics of each of these modules relevant for the development of a total view approach are:

**PDM:** This module includes configuration management, cross referrer, text and graphics reporters, workgroup management, project database, system notes. The configuration management system (CMS) provides mechanisms for: flexible project structures, formal review and approval, baselines, version control, formal change, audit. The CMS contains knowledge of the relationships between individual items of information. For example, when submitting a diagram, the CMS creates a package of the diagram, including associated requirements, trace links, specifications, decompositions, and Data Dictionary (DD) definitions. Given that each specification and DD entry has frames containing information such as, for example, design documentation, spreadsheets, CAD drawing, circuit simulations, this packaging capability is essential to maintain the consistency of the project's data.

**Requirements management:**
- Deals with stakeholder or internally generated source documents;
- Assesses impact of source document changes;
- Versions source document when changes occur;
- Edits, names and numbers requirements;
- Supports requirements hierarchies;
- Captures requirements from source documents;
- Navigates through requirements using many different search definitions, enabling the user to check for duplication of requirements, non-compatibility;
- Cross reference requirements to other items in the project;
- Categorises requirements;
- Attaches related information to individual requirements without affecting the original text of the requirement (e.g. database, graphs, spreadsheets, tables, diagrams);
- Navigates directly between requirements and other items in the project database.

**System modelling:** The SEE separates the system modelling into an essential model and an implementation model, following Yourdon's terminology (Yourdon, 1989). The essential model models what the system is supposed to do, that is, its functions whereas the implementation model models how the system is implemented to accomplish those functions, that is, its architecture. Cradle supports the following modelling notations: extended Yourdon notation, Function Block Diagrams (FBD), Behaviour Diagrams (BD), Use Case Model and OMT. For hardware and process functional modelling, the extended Yourdon notation can be used not only to model data flows and data items but also energy, material, resources or cash flows. FBDs show the physical elements of a system and how they are physically interconnected. This feature is very important for hardware architecture modelling. BDs are very important for process modelling as it is used to illustrate time-sequenced items and functions within the system. The BD implements many of the requirements for process modelling as established by (Schlenoff et al., 1996). Process and product systems modelling can be connected via BD, since BD symbols can be expanded to symbols in the Yourdon notation. Use Case Model and OMT for object oriented modelling, in this context applied to organisation, business or information modelling. A Use Case Model allows the visualisation of all the stakeholders interacting with the product development organisation. The system in the Use Case Model can be further expanded to show the flow of information among objects using the object orientation paradigm.

**Performance Modelling:** The SEE works essentially at the pre-specification stage. However it provides a performance modelling capability. Performance modelling allows any part of a design to be assessed in terms of the characteristics that it needs in order to be viable when built. This allows the effects of performance requirements and assumptions on the design to be studied. The performance modelling functionality provided is based on instrumenting symbols on state models with performance data expressed as sets of Performance Parameters (PPs). A state model is a set of one or more Implementation Model
DFDs (from the Yourdon notation set) and/or FBDDs. Representing product, process and people elements on DFDs and/or FBDDs makes it possible to evaluate in an integrated manner the effects of PPs from one system to the other.

**Document generation:** This module allows arbitrarily complex documents to be defined and generated from any or all information in a project database. A clear distinction is made between the information reported in a document and the structure of the document. In theory, documents such as QFDs charts, Value Engineering charts, FMEA charts, provided that all the necessary information is in the project database and the chart template is designed in the document generation tool.

**Software engineering:** This module supports code generation (in C, C++, Ada and Pascal) and reverse engineering (given the code, a structure chart diagram is generated)

### EXAMPLES OF APPLICATION

Figure 4 illustrates how the SEE can be used for the implementation of the total view approach. The implementation of the total view approach starts with models of the people, product and process systems at the highest level of the system breakdown structure. For an evolutionary system, it is supposed to have a product, life cycle processes and well established product development organisation and stakeholders beforehand. The product, process and people systems can then be modelled using the SEE’s embedded modelling paradigms.

The Use Case Model models the people system at the top of the people system breakdown structure. For further deployment of the people system, OMT must be used in this particular SEE. The Use Case Model is used to model the interactions between stakeholders and product development organisation (e.g. the PCSE department). Some of these stakeholders have requirements for the product and its life cycle processes. These requirements will drive the development of new subsystems and/or life cycle processes or modifications on existing ones. Stakeholders and requirements for example may be linked through the use of cross-references.

Requirements are linked via cross-references to the product and process essential (functional) models. Functional attributes can be derived for each element on the essential models. These models and attributes are, by their turn, linked to product and process implementation models. Each element on the implementation models may also have implementation attributes corresponding to PPs in the performance modelling tool. Also, these attributes may be linked to the elements of the implementation models via cross-references. The performance modelling tool may analyse in an integrated manner PPs from the people, product and process systems for trade-offs, risk assessment, implementation alternatives and so on.

Considering that products are the results of processes, elements on process models can be directly expanded to product models.

Figures 5 to 11 exemplify the SEE use according to Figure 4.

Figure 5 presents a simplified diagram of a Use Case Diagram (UCD) modelling the interactions of the PCSE organisation with some stakeholders. The system in this UCD is the PCSE organisation. Manage requirements, develop system and deliver system are use cases. The stakeholders are represented on the smaller boxes outside the system. The arrows represent the items that are exchanged between the system and its stakeholders.

Figure 6 illustrates the use of the requirements management module. A requirements hierarchy is defined by the requirements numbering system. Origins of requirements are some of the stakeholders on the UCD diagram. A requirement can be attached to frames to clarify its meaning. It also can be cross-referenced to elements on the essential and/or implementation models.

Figure 7 illustrates the PCS life cycle processes modelled using a BD. PPs can be assigned to time functions (rectangles) and to time items (rounded corners rectangles). A BD supports the modelling of sequences, conditions, concurrencies and iterations. A time item can be a product that can be further expanded using other notations more appropriate for product modelling (e.g. FBD). The process elements of the total view approach are modelled using this notation. It provides direct integration to product models by clicking on time items boxes or through performance modelling. The input or output to a time function can be a requirement, easily linked to the requirements management module through the use of frames.

Figure 8 is the DFD representing the product model of the PCS_PROTOTYPE, which appears as a time item on the process model BD in Figure 7. Figure 8 shows a context diagram showing the data interactions of the PCS module hardware with all its sensors and actuators. Every diagram symbol in Figure 8 may have a PP assigned to it.

Figure 9 shows a simplified FBD of a vehicle with its physical connections to the environment. PPs may be assigned to every symbol in Figure 9. The FPDS vehicle is further expanded to the diagram in Figure 10 that shows the vehicle architecture and the physical connections between subsystems.

Figure 11 shows a very simple example of the performance model of a car. PPs are associated to flows (arrows), processes (circles) and terminators (rectangles). Figure 11 shows examples of windows with the performance modelling menu (top left), performance analysis report (bottom left), state model (top right) and definition of performance data (bottom right).
DISCUSSION

A major benefit of the application of the SEE for an integrated PCS development is the ability to investigate early in the product development/evolution process the interactions between requirements and attributes not only of the product but also of its life cycle processes and development organisation. This leads to better product quality, lower life cycle cost and shorter development time. In particular, the SEE cuts across the interfaces of individual tools used to develop the diesel PCS (see Table I), providing traceability from requirements through to the engineering work undertaken in the development of a PCS.

Better product quality is a result of the fact that the product is developed and verified against requirements that can be traced to original stakeholders’ requirements. Stakeholders requirements include not only customer requirements but also regulatory and corporate requirements, life cycle process requirements and product development organisation requirements. Attributes corresponding to the requirements are assigned to the product, process and people systems implementation models. Performance analysis, based on such attributes can take place very early in the product development/evolution process.

Lower life cycle cost is a consequence of meeting life cycle process requirements and being able to model the performance of life cycle processes even at the pre-specification stage of the product. Cost is an attribute and as such can be modelled as a PP.

A shorter product development and evolution results from the visibility of interactions with many other stakeholders, requirements, functions, product and processes in the product data management systems. Change propagation can be analysed through the use of cross-references and quantified through the use of performance modelling. Process lead-time is also an attribute that can be analysed through performance modelling.

CONCLUSIONS

This paper identified the needs for an integrated PCS development and how they have been addressed so far. PCS development needs to cope with an environment with:

• increasing number of functions;
• increasing number of modes of operations;
• same functions implemented by different product architectures;
• better partitions in terms of cohesion and coupling can be found for current software modules;
• dynamic process of evolving the control software;
• dynamic product development organisation.

As a consequence PCS development must provide means to analyse in an integrated manner product, life cycle processes and product development organisation and their interactions.

So far, the diesel PCS case study identified that this later PCS development has been addressing some of these issues listed above by applying an SE approach to PCS development. However no life cycle processes or product development organisation requirements have been formally considered. It is concluded that the approach, methods and tools used provide a starting point for a set of functions and capabilities that need to be further expanded to include life cycle process and product development considerations. These basic functions are:

• requirements management;
• functional analysis and allocation and software architecture analysis (CASE);
• behavioural analysis;
• configuration management.

The paper proposed a total view approach that applies, in an integrated manner, the SEP to product, life cycle processes and development organisation. Product, life cycle processes and development organisation correspond to product, process and people systems in the total view approach. Each system type must not only be integrated with the other systems of its own type, but must also be integrated with all the elements of the other two types. The SEP is applied recursively to all levels of the product, processes and people systems breakdown structure. The optimisation of the total system must consider all three elements compromising attributes such as: product performance, process lead time and people satisfaction.

A commercial software package SEE that has the necessary capabilities to implement the total view approach was introduced. The SEE expands the basic functions used for diesel PCS development, allowing also:

• hardware architecture modelling;
• process systems modelling;
• people systems modelling, through the use, for example, of Use Case Models and object orientation.

A major benefit of the application of the SEE for an integrated PCS development is the ability to investigate early in the product development/evolution process the interactions between requirements and attributes not only of the product but also of its life cycle processes and development organisation. This leads to
better product quality, lower life cycle cost and shorter development time.

Additional benefits of using the SEE to implement the total view approach are:

- requirements management overheads dramatically reduced;
- integrated configuration management;
- CE is encouraged not losing sight of the interactions with other product and process subsystems;
- change effects analysis possible on a much bigger project scope;
- high levels of reuse;
- projects easier to maintain;
- an integrated managed environment.

FURTHER RESEARCH

This work is part of a research to develop an SE and CE framework for the integrated development of complex products. The framework is being developed considering the needs of the automotive industry. To investigate these needs, case studies of automotive subsystems development have been analysed. Further work necessary to achieve the objectives of the research is:

- investigate approaches to deal with complexity;
- analyse the SE approach;
- justify the suitability of the SE approach for automotive product development;
- investigate further needs of the automotive product development;
- propose an SE and CE framework for automotive product development;
- demonstrate the validity of the framework against examples of applications.

The framework being proposed and the way it is being implemented are described in this paper. Opportunities for further development are:

- demonstrate through an example of application the full integration of the people, product and process systems models with requirements management and performance modelling tools;
- analyse the performance based on a state model that contains elements from people, product and process models;
- demonstrate the integration of the SEE with CAD/CAM/CAE tools within the context of the total view approach;
- integrate detailed design simulation tools with the performance modelling module;
- generate QFD, Value Analysis charts, FMEA charts, using the document generation module.

ACKNOWLEDGEMENTS

The authors thank CNPq (Brazilian Council for Scientific and Technological Development) for Geilson Loureiro's sponsorship. The authors also thank for the support and attention provided by 3SL and Ford Motor Company Ltd and their personnel.

REFERENCES


**FIGURES**

![Fig. 1 SEP based on the IEEE-Std-P-1220](image1)

![Fig. 2 The total view approach – the 3Ps](image2)

![Fig. 3 Cradle modules](image3)

![Fig. 4 Using the SEE to implement the total view approach](image4)
Fig. 5 Use Case Model for the PCSE organisation

Fig. 6 Examples of requirements management
Fig. 7 Behaviour diagram modelling the PCS life cycle processes

Fig. 8 Data flow diagram modelling the PCS module hardware and interfacing sensors and actuators
Fig. 9 Function block diagram modelling a vehicle and its physical connections with the environment

Fig. 10 Function block diagram modelling a vehicle physical architecture
Figure 11: Examples of performance modelling displays for a very simplified vehicle performance model.

### Table 1: Function and capabilities of tools used for diesel PCS development

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>CAPABILITY ELEMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMEA and hazard analysis</td>
<td>- general spreadsheet facilities</td>
</tr>
<tr>
<td>Requirements management</td>
<td>- hierarchically organised requirements sets; - traceability between structured requirements sets and structured requirements sets and external document; - sorting, selection and processing of requirements and - linkage between a structured requirements set and an external structured tool (e.g. a CASE tool)</td>
</tr>
<tr>
<td>Functional analysis and allocation and software architecture analysis (CASE)</td>
<td>- separation of functional and architecture analysis and separation of process and control models; - modelling facilities with enhanced Yourdon notation (Yourdon, 1989) (Hatley and Pirbhai, 1988); - DFD, STD, entity relationship diagram, structured charts, DD, process specifications and decision tables, process activation tables, control specifications and module specifications; - links between functional and architecture models; - baseline mechanism; - configuration management;</td>
</tr>
<tr>
<td>Behavioural analysis</td>
<td>- iterative control system mathematical modelling, analysis and simulation;</td>
</tr>
<tr>
<td>Configuration management</td>
<td>- structured information; - change control at the point of use; - change records submitted as comments by the designer at the time of checking-in or checking-out a configuration item; - change request control.</td>
</tr>
</tbody>
</table>
A SYSTEMS ENGINEERING ENVIRONMENT FOR INTEGRATED SATELLITE DEVELOPMENT

G. Loureiro, Mr. P. G. Leaney, Dr.*
Brazilian Institute for Space Research (INPE), São José dos Campos, Brazil
*Loughborough University, England, UK

ABSTRACT
Space mission implementation faces a very dynamic environment with fast-paced information technology advancement and shrinking space budgets. A more focused use of decreasing public investments in space requires a cost reduction over their entire life cycle, up to the end of the useful life of a spacecraft. The anticipation of cost, schedule, risk and performance requirements from all over the product life cycle to the early stages of product development is generally recognised as a necessary condition to reduce life cycle cost. In order to cope with the intrinsic functional complexity of space products, such requirements engineering activity must be performed in a structured way within a systems engineering approach. This paper aims to describe how Cradle, a commercial systems engineering environment software package, can be used for integrated satellite development, taking into consideration functional and life cycle process requirements. Cradle has requirements management, system modelling, performance modelling, configuration management and document generation capabilities integrated in the same environment. Also, the paper provides some examples of application and highlights how Cradle can enhance the satellite development related activities performed by the Brazilian Institute for Space Research (INPE).

INTRODUCTION
Space mission development, such as the development and operation of satellites and ground systems with scientific purposes, seeks a balanced solution for performance, risk, schedule and cost requirements. Traditionally, the focus has been on meeting performance requirements and on avoiding risks. This has driven the need for new technology and for complex systems development and operation (e.g. real-time monitoring, high data rates).

Systems engineering is the discipline used in the engineering of complex products, such as those necessary to accomplish space mission performance requirements. Systems engineering has been traditionally viewed as referring to documentation, configuration, and cost & schedule management, under the responsibility of a project manager, while technical design tasks are performed by specialists. Functional analysis has been often used for software design, detailed hardware design but seldom for global system activities. However, the true systems engineering process shall establish a proper balance between performance, risk, cost and schedule, employing a top-down iterative process of requirements analysis, functional analysis and allocation, design synthesis and verification, and system analysis and control, focusing on the optimisation of the whole system instead of local optimisation of subsystems.

Space mission implementation faces a very dynamic environment with shrinking government budgets for science and technology. A more focused use of decreasing public investments in space requires a cost reduction over the entire life cycle, up to the end of the useful life of a satellite. Success requires a reduction of a factor 2 to 3 in life cycle costs (LCC), a reduction from 6 to 3 years in development schedule (preliminary and detailed definition, production/ground qualification & testing), a factor 2 to 3 increase in launch rate, while maintaining the scientific data output. Also, the time between the date on which a design began and the date the space system was no longer produced decreased from 12 to 6 years, between 1970 and 1990. In order to achieve success in terms of a better, faster and cheaper product development, space agencies (e.g. NASA, ESA) are developing additional methods for contracting their work, commercialising their technology, and expanding their partnership and other co-operative efforts with commercial firms, universities, and international partners.

To attract commercial interest, it is necessary to reduce LCC and development time and, at the same time, increase the sophistication of space
science missions. As publicly funded support to science declines, current satellite systems engineering faces some challenges, such as: 1) alternative space segment architectures, e.g., satellite constellations; 2) alternative subsystems architecture with the use of commercial-off-the-shelf (COTS) subsystems; and 3) reviewed functional allocation between onboard, other satellite, and ground support systems.

Also, in order to address the cost and schedule reduction issues, the way a space programme is structured needs to be reviewed. In general terms, a conventional space programme is structured in a high number of sequential phases, and is monitored through a sequence of formal reviews, particularly during the Phases B and C/D (see Figure 1(a)). The systems engineering process carried out on phases B and C/D is traditionally described by the ‘Waterfall Model’. 4 This model is based on a top-down approach for system development and includes steps of initiation, requirements analysis, design, test, and so on. Often, in its implementation, the steps are viewed as being relatively independent from one another and must be executed in strict sequence. Additionally, the required interfaces with the other elements of the system (e.g., human, facilities, processes) are not usually considered. 5

On the other hand, the paradigm of ‘faster, cheaper, better’ space missions, as heralded by NASA, relies on a much closer integration of system design, development and verification, and draws heavily on a robust and comprehensive programme of technology development, which must run in parallel and off-line with respect to flight programmes 6 (see Figure 1(b)). NASA and its contractors operate as a team formed, for example, by product delivery people, development contractor, and operations contractor members with common goals and a common vision. The systems engineering process at NASA has evolved from traditional ‘waterfall’ approach to concurrent engineering. 6 Figure 1 illustrates the conventional and NASA’s approach to space programme structure.

It is proposed here, that the time is right for a full-scale integrated satellite development that incorporates systems engineering and concurrent engineering principles, because of the issues mentioned above and of the current computing and information technology (IT) capability available for computer aided design, manufacturing and engineering (CAD/CAE/CAM), Product Data Management (PDM), analysis, simulation and networking.

This paper aims to describe how a commercial systems engineering environment (SEE) software package can be used for integrated satellite development, taking into consideration product functional, life cycle process and organisational requirements.

The paper is organised to meet the following objectives:

- situate INPE and its needs in the current satellite development environment;
- propose a total view approach that applies, in an integrated manner, the SEP to product, life cycle processes and their associated organisations;
- introduce a commercial software package SEE that has the necessary capabilities to implement the total view approach;
- discuss how the SEE can be used for a better, cheaper and faster satellite development;
- draw some conclusions and outline the needs for further research and development.

**INPE’S EXPERIENCE**

In 1978, the Brazilian Government approved the Brazilian Complete Space Mission (MECB). MECB is a program that aims to develop technological capability covering the whole life cycle of space technology, in particular, satellites. 7
During the development of the Engineering Model of the first satellite SCD1 (Data Collecting Satellite 1), INPE also developed the LIT (Integration and Testing Laboratory). It means that not only the development of the satellite product itself was done by INPE but also the development of satellite life cycle processes and of some organisations responsible to perform them.

INPE itself performed the system and subsystems development, manufacturing of electronic Printed Circuit Boards (PCBs), electronic assembly, assembly of subsystems, testing of subsystems (including thermal, vibration, Electromagnetic Interference and Compatibility (EMI/EMC), functional and performance testing), system integration and testing. The scope of the whole system under development during the initial stages of MECB is illustrated in Figure 2. These processes were performed very much in sequence, without great degree of integration between them.

There was opportunity for more design for assembly, testability or integration considerations during the development stages. Problems with assembly, testing and integration were not systematically anticipated to the development stages. They were dealt with when actually performing these life cycle processes.

For the later satellites, SCD-2 onwards (from 1989 onwards), first, the manufacturing of subsystems, and later, their development were progressively subcontracted from the Brazilian industry, as illustrated in Figure 3. INPE was progressively concentrating on the earlier and later stages of satellite systems life cycle. The earlier stages include: mission analysis and needs identification; feasibility analysis; preliminary definition. The later stages include: subsystem testing, system integration and testing and operation. Therefore, over the years, INPE progressively migrated from a full-in-house development and manufacturing to higher level systems engineering. Today INPE's satellite systems engineering activities have close links with industries and with other countries.

Although industry and international partnership is a necessary element for the development of current space programs, it can complicate the already inherently complex space project. This requires means to integrate all elements involved in the space project. Also the scope of a space project may not involve only the product (e.g. satellite) itself but also its life cycle processes (e.g. testing, integration, operation) and some of their performing organisations (e.g. operations organisation).

Dealing with the earlier and later stages of a satellite life cycle still allows INPE to have enormous influence on life cycle cost and schedule management. It has been demonstrated that 70% of the life cycle cost is determined by decisions made up to the conceptual development stage of product development. Also after these stages the cost to introduce changes is very high. Therefore, in order to obtain a life cycle traded-off and balanced system solution that meets stakeholder requirements and public acceptability, the foreseen ways by which an integrated development environment may enhance INPE's activities are by providing means to:

- multi-site integration;
- manage requirements, developing functional and physical architectures and identifying the attributes of their elements;
- analyse product, its life cycle processes and their performing organisation in an integrated manner;
- analyse the interactions of these elements with the environment;
- analyse the interactions among these elements themselves.
THE TOTAL VIEW APPROACH

In order to accommodate the requirements listed above, it is proposed here a framework for integrated product development called ‘the total view approach’. The total view approach is based on the assumption that the result of the product development effort is not only the product itself but also its life cycle processes and some of their performing organisations, as illustrated in Figure 4. It is widely known that product life cycle processes are highly affected by the product design in such a way that for reducing cost and development time it is worth considering life cycle processes requirements up front during the conceptual development stages _ a concurrent engineering principle.

Figure 4: The proposed integrated development model

Also, technological demands and very dynamic market requirements may enforce changes in the organisations performing life cycle processes in order to them remaining competitive. The total view approach is a structured analysis framework
that enlarges the scope of the system under development to contain not only the product, but also its life cycle processes and some of their performing organisations. The total view approach then mirrors the systems engineering process to analyse the whole-integrated system. The mainstream of the systems engineering process, according to modern standards (e.g. Mil-Std-499B, IEEE-Std-1220-1994, and EIA-632), consists basically of the requirements analysis, functional analysis and synthesis. The total view approach performs the requirements analysis, functional analysis and physical analysis subprocesses. The physical analysis models the physical architecture of the system resulting from the synthesis process. The approach is applied recursively to all levels of the product breakdown structure and can then be represented by the pyramid section illustrated in Figure 5.

Integration takes place in the following ways:
- linking stakeholders and development agencies through a common shared central project database. This includes the relationship between customer and supplier or between prime and subcontractor;
- linking requirements to the elements of the functional architecture and these to the elements of the physical architecture;
- linking product elements, process elements and organisation elements within their respective models;
- linking product, process and organisation elements between their respective models;
- linking product, process and organisation elements by identifying the interactions among their attributes.

**THE SEE AND THE TOTAL VIEW APPROACH**

Current environments for developing software systems, have some basic capabilities: requirements management, functional analysis, behavioural analysis, module structure (software physical architecture) and configuration management. To implement the total view approach, these capabilities need to be expanded in order to model hardware physical architecture, processes and organisations.

Some requirements necessary for hardware architecture modelling are: to model physical connections; to retrieve discipline specific models, e.g. circuit layouts, Computer Aided Design (CAD) drawings; to attach attributes to each element in the model.

A list of requirements for process modelling is provided by Schlenoff et al. (1996). The list includes the representation of: time sequencing (sequences, concurrency); alternative tasks and conditions; attributes related to each task in the process model.

Requirements for organisation modelling are detailed in Vernadat (1996). Organisations can be modelled using object oriented modelling techniques such as Jacobson's use case models and Object Modelling Technique (OMT) notation.

A tool that implements all the basic capabilities listed above and the required expansions for hardware, process and people systems analysis is Cradle (hereafter referred to as "the SEE"). It is a commercial software tool, developed by 3SL. It is defined as a systems and software engineering environment that provides through life support from requirements capture to system implementation with supporting configuration management, project control, and document generation capabilities.

The SEE uses the concept of a central project database that can be accessed through Local Area Network (LAN) or Wide Area Network (WAN). The SEE can, for example, integrate the customer into the project team by either providing the customer with on-line access into the project's central database or providing the customer with a read-only copy of the SEE used to create it. Also, in order to integrate prime and subcontractor, each subcontractor can be provided with a separate project group within the overall project organisational structure. The access rights of this group would then be designed such that it has access to all reference information for the work that it is to perform and can populate the database with its deliverables. These deliverables will form just as much a part of the overall database content as those created by the prime contractor. Indeed, the two will be integrated together, for example, the subcontractor adds lower levels of decomposition to requirements hierarchies; the subcontractor could form part of the review process for (some
of the items produced by the prime contractor and vice-versa;

The SEE is composed of the following modules, as illustrated in Figure 6. The characteristics of each of these modules relevant for the development of the total view approach are:

**Product Data Management (PDM):**
This module includes configuration management, cross referencer, text and graphics reporters, workgroup management, project database, system notes. The configuration management system (CMS) provides mechanisms for: flexible project structures, formal review and approval, baselines, version control, cross reference requirements to other items in the project; categorises requirements; attaches related information to individual requirements without affecting the original text of the requirement (e.g. database, graphs, spreadsheets, tables, diagrams); navigates directly between requirements and other items in the project database.

**System Modelling:**
The SEE separates the system modelling into an essential model and an implementation model, following Yourdon's terminology. The essential model models what the system is supposed to do, its functions and the implementation model models how the system is implemented to accomplish those functions, its architecture. Cradle supports the following modelling notations: extended Yourdon notation, Function Block Diagrams (FBD), Behaviour Diagrams (BD), Use Case Model and OMT. For hardware and process functional modelling, the extended Yourdon notation can be used not only to model data flows and data items but also energy, material, resources or cash flows. FBDs show the physical elements of a system and how they are physically interconnected. This feature is very important for hardware architecture modelling. BDs are very important for process modelling once it is used to illustrate time-sequenced items and functions within the system. The BD implements many of the requirements for process modelling. Process and product systems modelling can be connected via BD, since BD symbols can be expanded to symbols in the Yourdon notation. Use Case Model and OMT for object oriented modelling, in this context applied to organisation, business or information modelling. A Use Case Model allows the visualisation of all the stakeholders interacting with the product development organisation. The system in the Use Case Model can be further expanded to show the flow of information among objects using the object orientation paradigm.

**Performance Modelling:**
The SEE works essentially at the pre-specification stage. However it provides a performance modelling capability. Performance modelling allows any part of a design to be assessed in terms of the characteristics that it needs in order to be viable when built. This allows the effects of performance requirements and assumptions on the design to be studied. The performance modelling functionality provided is based on instrumenting symbols on state models with performance data expressed as sets of Performance Parameters (PPs). A state model is a set of one or more Implementation Model Data Flow Diagrams.
(DFDs) (from the Yourdon notation set) and/or FBDs. Representing product, process and organisation elements on DFDs and/or FBDs makes it possible to evaluate in an integrated manner the effects of performance parameters from one system to the other.

Document generation:
This module allows arbitrarily complex documents to be defined and generated from any or all information in a project database. A clear distinction is made between the information reported in a document and the structure of the document.

Software engineering:
This module supports code generation and reverse engineering (given the code, a structure chart diagram is generated)

EXAMPLES OF APPLICATION
Figure 7 illustrates how the SEE is proposed to be used for the implementation of the total view approach. The proposed approach starts with models of the product, its life cycle processes and some of their performing organisation at the highest level of the product breakdown structure. The product, process and organisation elements of the system can then be modelled using the SEE's embedded modelling paradigms. The Use Case Model is used to model the interactions between stakeholders and the organisation to be modelled, e.g. the satellite system development organisation. For further analysis of the organisation, OMT must be used. Some of these stakeholders have requirements for the product and its life cycle processes. These requirements will drive the development of new subsystems and/or life cycle processes or modifications on existing ones.

Stakeholders and requirements for example may be linked through the use of cross-references.

Requirements are linked via cross-references to the product and process essential (functional) models. Functional attributes can be derived for each element on the essential models. These models and attributes are, by their turn, linked to product and process implementation models. Each element on the implementation models may also have implementation attributes corresponding to performance parameters in the performance modelling tool. Also, these attributes may be linked to the elements of the implementation models via cross-references. The performance modelling tool can analyse in an integrated manner performance parameters from the product, process and organisation models for trade-offs, risk assessment, implementation alternatives and so on.

Considering that products are the results of processes, elements on process models can be directed expanded to product models.

Figures 8 to 13 present examples of the SEE use according to Figure 7.

Figure 8 presents a simplified diagram of a Use Case Diagram (UCD) modelling the interactions of a satellite development organisation with some stakeholders. 'Manage requirements' and 'develop system' are the use cases. The stakeholders are represented on the smaller boxes outside the organisation. The arrows represent the items that are exchanged between the system and its stakeholders.

Figure 9 illustrates the use of the requirements management module. A requirements hierarchy is defined by the requirements numbering system. Origin of requirements are some of the stakeholders on the UCD diagram. A requirement can be attached to frames to clarify its meaning. It also can be cross-referenced to elements on the essential and/or implementation models.
Figure 10 illustrates a conventional model of satellite life cycle modelled using a BD. PPs can be assigned to time functions (rectangles) and to time items (rounded corners rectangles). A BD supports the modelling of sequences, conditions, concurrency and iterations. A time item can be a product that can be further expanded using other notations more appropriate for product modelling (e.g. FBD). The process elements of the total view approach are modelled using this notation. It provides direct integration to product models by clicking on time item boxes or through performance modelling. The input or output to a time function can be a requirement, easily linked to the requirements management module through the use of frames.

Figure 11 is the product model of a space system for scientific application, including satellite and ground station. The satellite was represented in Figure 10 as a time item on the process model, BD in Figure 10. Figure 11 shows a DFD showing the interactions of the satellite, ground station and environment. Every diagram symbol in Figure 11 may have a PP assigned to it.

Figure 12 shows a simplified FBD of a satellite showing the physical connections among its subsystems. PPs may be assigned to every symbol in Figure 12. The satellite is further expanded from the diagram in Figure 11.

Figure 13 shows a very simplified performance model of a space mission. PPs are associated to flows (arrows), processes (circles) and terminators (rectangles). Figure 13 shows examples of windows with the performance modelling menu (top left), definition of performance data (bottom left), state model (top right) and performance analysis report (bottom left).
**DISCUSSION**

A major benefit of the application of the SEE for integrated satellite development is the ability to investigate, early in the product development process, the interactions of requirements and attributes not only of the product itself but also of its life cycle processes and their performing organisation. This leads to a more balanced solution in terms of better product quality, lower life cycle cost and shorter development time at a manageable space mission risk. In particular, the SEE cuts across the interfaces of individual tools for requirements management, functional modelling, behavioural modelling and configuration management, providing traceability from requirements through to the engineering work undertaken in the development of a satellite.

This is achieved due to the SEE's multi-notational approach with strong cross-referencing capabilities.

Better product quality results from the fact that the product is developed and verified against requirements that can be traced to the original mission stakeholders' requirements. In the current space mission implementation environment, stakeholder requirements include not only operational and functional requirements but also specific market requirements, commercial partner requirements, international partner requirements and life cycle process requirements. Attributes corresponding to these requirements are assigned to the product, process and organisation models. Performance analysis based on attributes such as
quality, cost, risk and schedule can take place very early in the product life cycle. To assure, for example, that requirements that most impact quality from a user's perspective are being addressed, the SEE allows the creation of requirements categories, e.g. primary and secondary — primary being those requirements imposed by the users and secondary, by the standard way of proceedings while defining a space mission. Secondary requirements are the ones that will be traded off in order to find a balanced solution in terms of cost, risk and schedule. Primary requirements are the ones that must be met in order to obtain a high quality product.

Lower life cycle cost results from addressing life cycle process requirements and being able to model the performance of life cycle processes even at the pre-specification stage of the product. For example, using the SEE for the early modelling of a satellite and its operation process, identifying their respective attributes and investigating their relationships leads to balanced solutions regarding design margin definitions which will greatly affect the operations cost of the mission.

A shorter product development is a consequence of the visibility of interactions among many stakeholders, requirements, functions, product and processes in the product data management module of the SEE. Change propagation can be analysed through the use of cross-references and quantified through the use of performance modelling. Also the project repository provided by the SEE can be totally or partially re-used for future projects leading to a shorter time to orbit for future space missions.

To manage risks, contrasting with the traditional risk avoidance approach of space mission implementation, the SEE's performance modelling module can evaluate cost savings and the impact in product quality in terms of the amount of risk accepted.

Although INPE does not make use of the SEE at the moment, the SEE may enhance INPE's activities by providing multi-site integration with partners and subcontractors and, means to manage requirements and attributes for satellites, their life cycle processes and performing organisations in an integrated manner. Trade-offs between the resulting attributes, which include quality, cost, risk and schedule, produce a more life cycle balanced solution to meet stakeholder requirements and to have public acceptability.

CONCLUSIONS
This paper identified the needs for a move towards an integrated satellite development approach and how they have been addressed so far. Satellite development needs to cope with an environment with the following characteristics:

- reduced public funding for scientific missions;
- increased commercial interest on satellite applications;
- necessary reduction in time to orbit to capture and keep market interest;
- fast pace advance in information and computing technology.

These characteristics have exposed the following needs:

- shift from a performance requirements and risk avoidance focused approach to a more balanced approach that seeks better performance, lower cost, shorter time to orbit and manageable risk;
- shift of systems engineering focus from document management to the current challenges of alternative satellite system architectures, integration of COTS, functional allocation, launching process alternatives;
- move from a very sequential to a more concurrent and integrated systems engineering process.

INPE's experience demonstrated the needs of:

- a broader scope for the satellite system development, including process and organisation elements;
- integration with users, suppliers and subcontractors.

These needs can be addressed by an integrated development approach that provides:

- multi-site integration;
- means to manage requirements, developing functional and physical architectures and identifying the attributes of their elements;
- means to analyse product, its life cycle processes and their performing organisation in an integrated manner;
- means to analyse the interactions of these elements with the environment;
- means to analyse the interactions among these elements themselves.

The paper proposed a total view approach that applies, in an integrated manner, the systems engineering process to product, life cycle processes and their performing organisations. A commercial software package SEE that has the necessary capabilities to implement the total view approach was introduced. The SEE has the following capabilities: requirements management, functional analysis, behavioural analysis, software architecture, configuration management,
hardware architecture modelling, process modelling, organisation modelling.

A major benefit of the application of the SEE for an integrated satellite development is the ability to investigate early in the product development/evolution process the interactions between requirements and attributes not only of the product but also of its life cycle processes and their performing development organisation. This leads to better product quality, lower life cycle cost, less time to orbit and manageable risk, as discussed in the paper.

Some additional benefits of using the SEE to implement the total view approach are:
- requirements management overheads dramatically reduced;
- change effects analysis possible on a much bigger project scope;
- high levels of reuse;
- projects easier to maintain;

**FURTHER RESEARCH**

This work is part of a research to develop a systems engineering and concurrent engineering framework for the integrated development of complex products. The framework is being developed considering the needs of the automotive and aerospace industries. Further work necessary to achieve the objectives of the research are:
- the formalisation of a method that supports the total view framework, integrating product, process and organisation models through the requirements, functional and physical analysis phases;
- the development of an example of application demonstrating the method, and therefore, the applicability of the framework, through the use of the SEE.

The framework being proposed and the way it is being implemented are described in this paper. Opportunities for further development, beyond but complementary to the scope of this research, are:
- demonstrate the integration of the SEE with CAD/CAM/CAE tools within the context of the total view approach;
- integrate detailed design simulation tools with the performance modelling module;

**ACKNOWLEDGEMENTS**

The authors thank CNPq (Brazilian Council for Scientific and Technological Development) for Gellion Loureiro's sponsorship. The authors also thank for the support and attention provided by 3SL and their personnel.

**REFERENCES**

14. 3SL (ed.). Cradle manuals, Barrow-in-Furness, UK, 1997
A Systems Engineering Framework for Integrated Automotive Development

G. Loureiro and P. G. Leaney
Loughborough University
Manufacturing Engineering Dept.
Loughborough University, UK,
LE11 3TU

M. Hodgson
Ford Motor Company Ltd.
Visteon Technical Centre - Room 28/861
Research and Engineering Centre
Laindon-Basildon-Essex, UK - SS15 6EE

Abstract. This paper proposes a systems engineering framework for integrated automotive development -- the total view approach. It is a modelling framework that integrates the product, its life cycle processes and their associated organisations throughout the requirements, functional and physical analysis processes, at all levels of the product breakdown structure, deriving attributes as emergent properties of a whole integrated system. The paper justifies the framework through a review of traditional and current automotive development and two case studies.

INTRODUCTION

The traditional design approach in the automotive industry is to optimise components (Gormley and MacIsaac 1989). This component approach is founded on the belief that any design could be broken into independent and self-supporting components. The optimisation and the evolution of the whole is assumed from the optimisation and evolution of every component (Ziebart 1991).

From the mid-80s, concurrent engineering methods [e.g. DFMA (Design for Manufacturing and Assembly), Group Technology, Taguchi Methods, FMEA (Failure Mode Effects Analysis), QFD (Quality Function Deployment), Value Engineering] and tools [e.g. CAD/CAE/CAM/PDM (Computer Aided Design/Engineering/Manufacturing/Product Data Management)] have been supporting this component approach. Component design started to take into consideration life cycle process requirements. For example, existing components are evaluated according to DFMA rules and the component is modified, preserving its functionality, to improve its ease of manufacturing and assembly. CAD/CAM tools such as feature based technology allow the simultaneous design and process planning of components. More recently, communications and integration technologies have been helping to close the gaps between the organisations responsible for the various component life cycle processes.

As the automobile grew in functionality and became a more multidisciplinary product, more components, component interconnections and life cycle process components were necessary. A non-structured evolutionary development process took place. Better components were substitute for old ones and new components were simply added to the current design version. Examples of the consequences of this approach applied to automotive electronics show that it has not led to the optimisation of the whole product and its life cycle processes (Ziebart 1991):

- more control units, sensors and actuators need more space. However, space for components get less and less because space is required for the convenience of the passengers;
- the overall system reliability may decrease because of the increasing number of parts;
- the increasing complexity of the system structure may deteriorate the serviceability and the handling of the electronic systems;
- in many cases, the driver has been overtaxed by the number of differently reacting electronic systems;
- solutions for interfaces are often developed in a reactive manner because they must be implemented at that time when the system definition was already finished.

Concurrent engineering (CE) does not provide the complete framework to deal with such an increasing complexity. The level of integration and complexity of automobile functionality requires a better understanding of the contribution of each product and/or process component to the whole and the interactions between these components. CE of components would help component evolution, but only an interdisciplinary, collaborative approach to derive, evolve and verify a life-cycle balanced system can deliver better results that meet...
customer expectations and public acceptability. This approach is systems engineering (IEEE 1994).

For seeking a balanced solution and using a requirements-driven process, systems engineering is very much applicable to the current needs of automotive product development for a global and highly segmented market. Requirements to automotive development include the following:

- corporate, regulatory and customer requirements;
- life cycle process requirements;
- functional, physical and operational requirements.

The system solution very often does not include only the product but also, its life cycle processes and their organisation once these elements are highly coupled in automotive development (Clark & Fujimoto 1991). Therefore the consideration of any of these elements in isolation would not be able to deliver a well-balanced solution. It is proposed here the systems engineering approach to be applied within the context of integrated development. Therefore, the whole system is to be considered as an integration of product, processes and organisation elements.

This paper proposes a systems engineering framework for integrated automotive development -- the total view approach. The total view approach treats the whole system as being the product, its life cycle processes and their associated organisations.

The paper is organised to meet the following objectives:

- present two automotive case studies to illustrate the needs for an integrated automotive development and how they have been addressed so far;
- situate integrated development against the Ford Product Development System objectives;
- propose a total view approach that applies, in an integrated manner, the systems engineering process to the product, its life cycle processes and their associated organisations;
- discuss how the total view framework can be used for a better, cheaper, faster and simpler automotive development;
- draw some conclusions and outline the needs for further research and development.

IDENTIFYING AND ADDRESSING THE NEEDS FOR AN INTEGRATED AUTOMOTIVE DEVELOPMENT

This section presents two case studies on Powertrain Control System (PCS) development:

- the gasoline PCS that illustrates the consequences of the traditional automotive development process and the needs for integrated development;
- the diesel PCS that illustrates how these needs are starting to be addressed.

The gasoline PCS. (Loureiro 1996), reporting some results of case studies at Ford-AVT-PCSE (Advanced Vehicle Technology-Powertrain Control Systems Engineering), affirmed that despite the technological improvement, Ford's gasoline PCS has not evolved in a structured manner over the past 20 years. The development process has been mainly change driven, based on improving past versions of the system. Changes are directly implemented on the detailed design, at the bottom level of the product breakdown structure without links with changes on higher level requirements. Examples of requirements having been used are 'change this subsystem to another version', 'add that sensor', 'delete that subsystem'. To evolve software, for example, changes are implemented directly on the software code, with very poor visibility of interactions with other software subsystems. The effects of changes have been only evaluated when the whole vehicle is tested to see whether it meets the emissions regulations.

That traditional non-structured development process has no longer been able to cope with (Loureiro, Leaney and Pickman 1996):

- increasing PCS functionality. The simple addition of functions in a non-structured way, through the addition of parts may lead to decreased overall system reliability, less space for the convenience of the passengers and decreased maintainability;
- increasing PCS complexity. The addition of functions leads also to the addition of desired or undesired interactions among parts what leads to greater complexity. Without an appropriate systems analysis, this increasing complexity may lead to the need of later changes in the overall system, decreased reliability and maintainability;
- the need to shorten development life cycle, considering that some competitors can deliver vehicles with a shorter development life cycle and that microelectronics systems double their functionality every 18 months (while a car has a development time of about 4 years and an average lifecycle of an individual model of 7 to 9 years). The late implementation of changes, not supported by appropriate analysis, increases the development life cycle;
- a very dynamic product development cycle with changes coming not only because of performance or functional challenges, but also due to changes on competitors' products, upper management feeling, technological advances, etc. Changes have been
implemented in an ad-hoc and time consuming manner, without proper risk analysis and configuration management;

- dynamic organisational structure which tries to constantly adapt to market and technological trends;
- constrained resources what requires 'right-first-time' philosophy without wasting of time and money.

Ford's gasoline PCS case study also identified changes on other items that affected the PCS development (Loureiro, Leaney and Hodgson 1998):

- the number and types of stakeholders increased. For example: the market became more differentiated, number of interacting subsystems increased, new environment agencies were created;
- new requirements appeared and constraints became more tightened. For example: new driveability requirements, more stringent emissions and fuel economy requirements;
- number of functions in the product increased. Examples of new functions are on-board diagnostics II, cruise control.
- number of modes of operations increased. Initially modes of operations were related only to protection of the system being controlled, now there are different modes of operation depending on throttle position, temperature, altitude, engine state;
- same functions implemented by different product architectures;
- better partitions in terms of cohesion and coupling can be found for current software modules;
- the process of evolving the control software has been split into four different streams depending on the nature of changes to be performed and on the bookshelf status of the software control;
- the product development organisation also changed. Control algorithm developers and software developers were merged into the same group.

**Highlighted needs.** One primary lesson from the gasoline PCS case study is the need for visibility of interactions among product subsystems and among their attributes. However, as mentioned above, product subsystems drive and are affected by changes not only on other product subsystems but also on product life cycle processes and their associated organisations. As a consequence, it is necessary that a development framework be able to integrate product, life cycle processes and organisation analysis. Such a framework must:

- provide means to manage requirements, functions, product and processes variety;
- provide means to analyse product, its life cycle processes and their associated organisations;
- provide means to analyse the interactions of these elements with the environment;
- provide means to analyse the interactions among these elements' attributes.

The diesel PCS. From 1994, a diesel PCS has been developed by Ford-AVT-PCSE, using a systems engineering approach (Loureiro, Leaney and Pickman 1996) to address the product-related needs listed above: increasing product functionality and complexity, functional and interface requirements management, product functional and architecture analysis, behaviour analysis. No life cycle processes or organisation requirements have been formally considered. However, the approach, methods and tools used, provided a starting point for a set of functions and capabilities that need to be further expanded to include life cycle process and organisation considerations. Table 1 lists the functions and capabilities of tools used for the diesel PCS development.

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>CAPABILITY ELEMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMIA and hazard analysis</td>
<td>General spreadsheet facilities</td>
</tr>
<tr>
<td>Requirements management</td>
<td>Hierarchically organised requirements sets; traceability between structured requirements sets and structured requirements sets and external document; sorting, selection and processing of requirements and linkage between a structured requirements set and an external structured tool [e.g. a CASE (Computer Aided Software Engineering) tool]</td>
</tr>
<tr>
<td>Functional analysis and allocation and software architecture analysis</td>
<td>Separation of functional and architecture analysis and separation of process and control models; modelling facilities with enhanced Yourdon notation (Yourdon 1989) (Hailey and Pirbhai 1983); data flow diagrams, state transition diagram, entity relationship diagram, structured charts, data dictionary, process specifications and decision tables, process activation tables, control specifications and module specifications; links between functional and architecture models; baseline mechanism and configuration management;</td>
</tr>
<tr>
<td>Behavioural analysis</td>
<td>Iterative control system mathematical modelling, analysis and simulation;</td>
</tr>
<tr>
<td>Configuration management</td>
<td>Structured information; change control at the point of use; change records submitted as comments by the designer at the time of checking-in or checking-out a configuration item; change request control.</td>
</tr>
</tbody>
</table>

Table 1. Function and capabilities of tools used for diesel PCS development
INTEGRATED DEVELOPMENT AND THE FORD PRODUCT DEVELOPMENT SYSTEM

The approach used for the diesel PCS is consistent with the new reengineered Ford Product Development System (FPDS).

Ford Motor Company has 5 core business processes. They are: Management Systems, FPDS, Ford Production System (FPS), Order to Delivery and After Sales Service.

FPDS is the development process system by which Ford plan, design, develop and launch new vehicles. FPDS was designed to achieve the following objectives:

- 90% customer satisfaction at 3 months in service and 85% at 6 years in service;
- 25-45% reduction in time-to-market across a range of program milestones and magnitudes;
- 25-30% reduction in warranty due to the FPDS system approach to total program management;
- 30-40% reduction in resources (hours/program) due to time reduction;
- 25-30% improvement in investment efficiency due to reusability.

Meeting these objectives will increase Ford’s ability to launch more new vehicle programs, improve cash flow and enhance employee pride.

Some of the pillars of FPDS are common to those of IPPD (Integrated Product and Process Development), as defined by the DoD, in the Regulation 5000.2-R (DoD 1996). These are:

- systems thinking
- systems engineering
- teamwork
- concurrent engineering

Other more specific characteristics of FPDS are:

- targets driven design
- reusability
- engineering for reliability
- predictive and analytical engineering
- integrated package, function and appearance
- vehicle attribute focus

Under systems thinking, FPDS proposes that decisions need to be made to optimise performance for, first, the whole Ford Motor Company, then for the vehicle program and then for the individual function or activity. These decisions shall seek a balance of timing, resources and program content (vehicle functionality and features). However, FPDS is very much product focussed. For example, targets driven design refers to the cascading of product targets (e.g. functionality, cost and weight) throughout the vehicle breakdown structure. Although reusability may refer to resources reuse and lessons learned, it also refers to carryover design, carryover platform, carryover software. The gasoline PCS case study demonstrated how reuse can be dangerous in terms of increasing the overall complexity of the whole system - product, process and organisation. Engineering for reliability includes the use of FMEA and hazard analysis to increase product reliability. Predictive and analytical engineering refers basically to the use of CAD, CAE, CAM, PDM and analysis tools upfront, which are essentially product modelling tools. Integrated package, function and appearance capture those product attributes for trade-off. Also, the vehicle attributes include 13 product attribute sets and only 1 process attribute and 1 organisation attribute sets.

The systems engineering process that Ford uses can be summarised by ‘design-to-requirements’ and ‘verify-against-requirements’. It follows the ‘V’ model (Blanchard 1998) and also focuses on the product. The product is cascaded down from vehicle, through powertrain, chassis, exterior, interior and electrical to the basic components. These components are developed, verified and manufactured and then integrated into subsystems. These are then tested and integrated into the vehicle.

THE TOTAL VIEW APPROACH

The gasoline PCS case study demonstrated the need to analyse the product, its life cycle processes and their associated organisations and their interactions all together to have an integrated PCS development. The diesel PCS case study provided the basic functions and capabilities of a set of tools to implement a systems engineering approach to PCS product development. FPDS provides all ingredients for product system engineering. However, that was still a partial view approach because it focused on the product system. For a total view approach, the systems engineering process must be applied, in an integrated manner, to the product, process and organisation elements. These three set of elements correspond to the product itself, its life cycle processes and their associated organisations, respectively, as illustrated in Figure 1. This section introduces the concept of a total view approach and its relationship with the systems engineering process.

The total view approach is based on the assumption that the result of the product development effort is not only the product itself but also its life cycle processes and some or all of their associated organisations, as illustrated in Figure 2. It is widely known that product life cycle processes are highly affected by the product design in such a way that for reducing cost and development time it is worth considering life...
The total view approach is a structured analysis framework that enlarges the scope of the system under development to contain not only the product, but also its life cycle processes and some or all of their associated organisations. The total view approach then mirrors the systems engineering process to analyse the whole-integrated system. The mainstream of the systems engineering process, according to modern standards (e.g. Mil-Std-499B, IEEE-Std-1220-1994, and EIA-632), consists basically of the requirements analysis, functional analysis and synthesis. The total view approach performs the requirements analysis, functional analysis and physical analysis sub-processes. The physical analysis models the physical architecture of the system resulting from the synthesis process. The approach is applied recursively to all levels of the product breakdown structure and can then be represented by the pyramid section illustrated in Figure 3.

Integration takes place in the following ways:
- linking stakeholders and development agencies through a common shared central project database. This includes the relationship between customer and supplier or
between prime and subcontractor;
• linking requirements to the elements of the functional architecture and these to the elements of the physical architecture;
• linking product elements, process elements and organisation elements within their respective models;
• linking product, process and organisation elements between their respective models;
• linking product, process and organisation elements by identifying the interactions among their attributes.

The total view approach in practice. A practical way of using the total view approach is through the use of a systems engineering environment [e.g. Cradle™ (3SL 1998)] that is flexible enough to model product, process and organisation. Product, process and organisation attributes can be extracted from the functional and physical analysis models generated. These attributes can be related to each other and to requirements using a matrix representation provided by a spreadsheet tool [e.g. Excel™ (Microsoft 1998)]. As the number of attributes can be very high (e.g. of the order of a 1000), a software [e.g. Chaco-2.0 (Hendrickson & Leland 1995)] that implements a partitioning algorithm manipulating very high order matrices is recommended. Attributes relationships can be used as criteria for team formation, for example. Figure 4 illustrates the use of these tools.

Figure 4. The total view approach in practice

DISCUSSION

A major benefit of the application of the total view approach for integrated automotive development is the ability to investigate early in the product development/evolution process the interactions between requirements and attributes not only of the product but also of its life cycle processes and their associated organisations. This leads to better product quality, lower life cycle cost, shorter development time and manageable complexity.

Better product quality is a result of the fact that the product is developed and validated against requirements that can be traced to original stakeholders' requirements. Stakeholders requirements include not only customer requirements but also regulatory and corporate requirements, life cycle process requirements and organisation requirements. Attributes corresponding to the requirements are assigned to the product, process and organisation functional and physical model elements. The interaction among these attributes can be identified earlier and trade-off can be pursued.

Lower life cycle cost results mainly from change impact analysis early in the product development life cycle. Knowing beforehand the impact of changes in product, process and organisation attributes avoids the need for late changes in the product life cycle. The later the change, the more it costs.

A shorter product development and evolution results from the dynamics of teamwork also enhanced by the total view approach. Once interacting attributes are identified, people responsible for their development are also identified. These people can work together in order to achieve optimisation. These people do not have to wait until formal review meetings are set up. They can contact each other in a day-by-day basis and go to review meetings with a draft solution at hand, instead of a problem to be solved.

(Suh 1990) and (Fey et al. 1994) demonstrated that, when developing a product, one shall seek functional attributes independence. Independence of functional attributes can be achieved if they are implemented through independent physical attributes. The total view approach provide the framework for capturing physical attributes relationships for the whole system composed by product, process and organisation elements. Once, the relationships between functional and physical attributes are known, the relationships among functional attributes can be identified by transitivity. This is specially useful for reusability. Very often, when pursuing the investment efficiency provided by reusability, it is possible to complicate so much the product itself or its life cycle processes or their associated organisations that the gain with reusability may be overtaken by the loss with increased complexity. In essence, the total view approach aims to be, above all, a complexity management framework.

CONCLUSIONS

This paper identified the needs for an
integrated automotive development and how they have been addressed so far. Two automotive case studies were analysed: the gasoline and the diesel PCS.

PCS development needs to cope with an environment with:
- increasing product functionality;
- increasing product complexity;
- shortening development life cycle;
- very dynamic product development cycle;
- dynamic organisational structure;
- constrained resources.

The gasoline PCS case study demonstrated that the PCS changes are driven by:
- increasing number and types of stakeholders;
- new and more tightened requirements and constraints;
- increasing number of functions;
- increasing number of modes of operations;
- same functions implemented by different product architectures;
- better partitions in terms of cohesion and coupling can be found for current software modules;
- dynamic process of evolving the control software;
- dynamic product development organisation.

As a consequence PCS development must provide means to analyse in an integrated manner product, its life cycle processes and their associated organisations and their interactions.

The diesel PCS case study has been addressing some of these issues listed above by applying a systems engineering approach to PCS development. However no life cycle processes or organisation requirements have been formally considered. It is concluded that the approach, methods and tools used provide a starting point for a set of functions and capabilities that need to be further expanded to include life cycle processes and organisational. These basic functions are:
- requirements management;
- functional analysis and allocation and software architecture analysis;
- behavioural analysis;
- configuration management.

The Ford Product Development System (FPDS) within which the diesel PCS development is included is a step towards vehicle systems engineering. Although FPDS is very much product focussed, it recognises the need to seek optimisation of the overall system rather than of components. It represents a relevant shift from the traditional component approach used by the automotive industry. FPDS sets the basis on which a more integrated automotive development approach can be built.

The paper proposed a total view approach that applies, in an integrated manner, the systems engineering process to product, life cycle processes and their associated organisations.

A major benefit of the application of the total view approach for an integrated automotive development is the ability to investigate early in the product development/evolution process the interactions between requirements and attributes not only of the product, but also of its life cycle processes and their associated organisations. This leads to better product quality, lower life cycle cost, shorter development time and manageable complexity.

FURTHER RESEARCH

This work is part of a research to develop a systems engineering and concurrent engineering framework for the integrated development of complex products. The framework is being developed considering the needs of the automotive and aerospace industries. Further work necessary to achieve the objectives of the research are:
- the formalisation of a method that supports the total view framework, integrating product, process and organisation models through the requirements, functional and physical analysis phases;
- the development of an example of application demonstrating the method, and therefore, the applicability of the framework, through the use of a commercial systems engineering environment (SEE).

The framework being proposed and the way it is being implemented are described in this paper. Opportunities for further development, beyond but complementary to the scope of this paper, are:
- demonstrate the integration of the SEE with CAD/CAM/CAE tools within the context of the total view approach;
- integrate the SEE with detailed design simulation tools for performance modelling.

ACKNOWLEDGEMENTS

The authors thank CNPq (Brazilian Council for Scientific and Technological Development) and INPE (Brazilian Institute for Space Research) for Geilson Loureiro's sponsorship. The authors also thank for the support and attention provided by 3SL and Ford Motor Company Ltd and their personnel.

REFERENCES

Clark, K.B. and Fujimoto, T., Product development performance: strategy,

DoD, Mandatory procedures for major defense acquisition programs (MDAfs) and major automated information system (MAIS) acquisition programs. Department of Defense, Regulation 5000.2-R, 1996.

EIA. EIA-632 - Processes for engineering a system. EIA, 1997.


Loureiro, G., Evolution of EEC at Ford: from evolutionary to a systems engineering approach, IPPS Report 96/1, Loughborough University, Loughborough, 1996.


BIOGRAPHIES

Geilson Loureiro graduated from ITA (Technological Institute of Aeronautics), São José dos Campos, Brazil, in Electronics Engineering, in 1987. In 1990, he obtained a Diploma on Production Management and Industry Operations from the Getúlio Vargas Foundation in São Paulo, Brazil. In 1994, he finished his MSc on Mechatronics and Systems Dynamics at ITA. He is currently writing up his PhD thesis on Systems Engineering at Loughborough University, England. He has been working for the Brazilian Institute for Space Research (INPE) since January 1988, having dealt with many stages of satellite life cycle including: manufacturing, integration and testing.

Dr. Paul G. Leaney graduated from Bath University in 1977 with first class honours in Mechanical Engineering. He went on to earn an MSc (by research) on the topic of hydraulic oil filtration. After a brief spell in industry he came to Loughborough University achieving his PhD in 1986 on the topic of fluid power control. He has since worked as a lecturer in the Dept. of Manufacturing Engineering and continues to work with a wide range of industry on the topic of design to manufacture as a single business process.

Mike Hodgson graduated from the Imperial College, in Mechanical Engineering, in 1984. Since graduation he has been employed by Ford within powertrain engineering, mainly at the Engineering Centre in Essex, England. In 1988-1990 he took an assignment to the North American engineering office working on simulation and modelling of powertrain controls. While in the USA, he completed a Masters Degree in Electronics and Computer Control Systems. Latterly, Mike has been he supervisor of the Diesel control systems group which is now part of Visteon.