Prepared by: University of York

On Behalf of:

Software Systems Engineering Initiative

Interim Standard of Best Practice on Software in the Context of DS 00-56 Issue 4

Document Number: SSEI-BP-000001
Issue: 1
Date: August 2009

The investigations which are the subject of this report were initiated by SSEI under the auspices of the United Kingdom Ministry of Defence for the Software Systems Engineering Initiative.

© Crown Copyright 2009

PREPARED BY: Catherine Menon, Richard Hawkins, John McDermid

APPROVED BY: Professor John McDermid
SSEI Technical Director
EXECUTIVE SUMMARY

This document comprises the Standard of Best Practice for software in the context of Defence Standard 00-56 Issue 4. This is presented in two “strands”: guidance on a managerial level, and guidance on a technical level. This SoBP will be available in both .pdf format (this document) and web format.

Issues 3 and 4 of Defence Standard 00-56 present a goal-based approach to software safety. One of the fundamental requirements of Def Stan 00-56 is the presentation of a compelling argument supported by rigorous evidence in order to demonstrate satisfaction of identified safety properties. This approach to safety contrasts with the prescriptive recommendations of previous UK MOD safety standards (such as Def Stan 00-56 Issue 2). Where prescriptive standards postulate a correlation between objectives completed during software development and the safety of the completed software, goal-based standards require an explicit demonstration of software safety properties.

Because goal-based safety standards describe what must be achieved rather than how to achieve safety, there may be multiple ways in which compliance could be demonstrated. While advantageous in that such standards do not constrain the method of development, this flexibility also means that there is a need for guidance on the possible “best practice” methods of compliance. This (interim) Standard of Best Practice comprises such guidance for compliance with Def Stan 00-56 Issue 4.

In this document, we will refer to ensurance and assurance activities. Ensurance activities are those which are concerned with the development of software to the specifications provided, while assurance activities are those which are concerned with the production of a compelling safety argument. This guidance focuses on the management-level decisions which are necessary throughout the ensurance and assurance of software to ensure compliance with Def Stan 00-56 Issue 4. These decisions allow management personnel to influence the software development processes and to monitor ongoing safety management. Secondly, the guidance then discusses some more technically-focused concerns. This part of the guidance is primarily intended to examine the issues associated with the construction of a compelling software safety argument. In addition to this, we discuss software safety assurance from the perspective of Def Stan 00-56 for two specific scenarios: the use of COTS software and the use of software developed to non-UK MOD standards. Finally, we present the findings of two case studies designed to provide partial validation for this document in the area of COTS, and in the area of supplier selection.

As the findings of this document have not yet been extensively validated, we recommend that this be considered interim best practice. We have obtained partial validation from a number of case studies discussed in this document. This document therefore represents findings which should be assessed further, and
which will evolve with future validation to be considered best practice for compliance with Def Stan 00-56 Issue 4.

This .pdf version of the SoBP will be neither updated nor maintained. The latest version of the SoBP is available from the SSEI website, and is to be considered definitive.
# INITIAL DISTRIBUTION

<table>
<thead>
<tr>
<th>Name</th>
<th>Role / Organisation</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>John McDermid</td>
<td>SSEI Technical Director, (University of York)</td>
<td>York</td>
</tr>
<tr>
<td>Sqn Ldr Mike Place</td>
<td>SSEI Customer Interface Director, (MOD, DE&amp;S)</td>
<td>Abbey Wood</td>
</tr>
<tr>
<td>Rob Brockie</td>
<td>SSEI Contract Holder, (BAE Systems, MAS)</td>
<td>Warton</td>
</tr>
<tr>
<td>Jane Fenn</td>
<td>SSEI Programme Manager, (BAE Systems, MAS)</td>
<td>Brough</td>
</tr>
<tr>
<td>Tim Kelly</td>
<td>SSEI Academic Theme Lead (University of York)</td>
<td>York</td>
</tr>
<tr>
<td>Colin McDonald</td>
<td>SSEI Industrial Theme Lead (MBDA)</td>
<td>MBDA</td>
</tr>
</tbody>
</table>
CHANGE RECORD

<table>
<thead>
<tr>
<th>Change No.</th>
<th>Incorporated by:</th>
<th>Date</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue 1 Draft A</td>
<td>Catherine Menon</td>
<td>6&lt;sup&gt;th&lt;/sup&gt; March</td>
<td>First Issue</td>
</tr>
<tr>
<td>Issue 1 Draft B</td>
<td>Catherine Menon</td>
<td>18&lt;sup&gt;th&lt;/sup&gt; March</td>
<td>External Review</td>
</tr>
<tr>
<td>Issue 1 Draft C</td>
<td>Catherine Menon</td>
<td>31&lt;sup&gt;st&lt;/sup&gt; March</td>
<td>Comments Incorporated</td>
</tr>
<tr>
<td>Issue 1 Draft D</td>
<td>Catherine Menon</td>
<td>6&lt;sup&gt;th&lt;/sup&gt; May</td>
<td>Draft Interim to Interim,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24&lt;sup&gt;th&lt;/sup&gt; August</td>
<td>Annexes Incorporated</td>
</tr>
<tr>
<td>Issue 1</td>
<td>Catherine Menon</td>
<td></td>
<td>Final Issue</td>
</tr>
</tbody>
</table>

Note to users:

1. This document will be amended as and when necessary.
TABLE OF CONTENTS

Executive Summary ........................................................................................................2
Initial Distribution ..............................................................................................................4
Change Record ..................................................................................................................5
Table of Contents ..............................................................................................................6

1 Introduction .....................................................................................................................8

2 Managerial SoBP ..........................................................................................................10
  2.1 Introduction ................................................................................................................10
  2.2 Management in the Context of DS 00-56 ...............................................................11
  2.3 Safety Requirements and Safety Cases ...............................................................13
  2.4 Software Safety Management Phases ....................................................................14
  2.5 Observations and Conclusions .............................................................................34

3 Technical Guidance .....................................................................................................35
  3.1 Pattern Catalogue ....................................................................................................36
  3.2 Assurance-based Argument Development Method ............................................38
  3.3 Conclusions ..............................................................................................................40

4 Specific Scenarios ........................................................................................................40
  4.1 COTS and Legacy Software ..................................................................................41
  4.2 Non-MOD Standards .............................................................................................43

5 Conclusions ...................................................................................................................46

6 Recommendations .......................................................................................................47

7 Acknowledgements .....................................................................................................47

8 List of References ........................................................................................................47

9 Definitions ....................................................................................................................47

10 List of Abbreviations ..................................................................................................48

11 Annexes .......................................................................................................................49
  ANNEX A Guidance for the SoBP .............................................................................50
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>Software Safety Arguments</td>
<td>51</td>
</tr>
<tr>
<td>A.2</td>
<td>Process Overview</td>
<td>60</td>
</tr>
<tr>
<td>A.3</td>
<td>Project Parameters</td>
<td>67</td>
</tr>
<tr>
<td>A.4</td>
<td>Management Decision Guidance</td>
<td>69</td>
</tr>
<tr>
<td>A.5</td>
<td>Table for Assurance based Argument Construction</td>
<td>93</td>
</tr>
<tr>
<td>A.6</td>
<td>Thales Validation of SoBP</td>
<td>96</td>
</tr>
<tr>
<td>ANNEX B</td>
<td>Technical SoBP</td>
<td>101</td>
</tr>
<tr>
<td>ANNEX C</td>
<td>Non-MOD Safety Standards</td>
<td>98</td>
</tr>
<tr>
<td>ANNEX D</td>
<td>COTS and Legacy</td>
<td>99</td>
</tr>
<tr>
<td>ANNEX D.E</td>
<td>IBM Validation of COTS and Legacy</td>
<td>181</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

Defence Standard 00-56 Issues 3 and 4 present a goal-based, or evidential, approach to ensuring and assuring safety. One of the major principles of Def Stan 00-56 is the need to demonstrate system safety by means of a compelling safety argument, supported by rigorous evidence. This represents a departure from earlier prescriptive UK MOD safety standards (such as Def Stan 00-56 Issue 2), in that Def Stan 00-56 Issue 4 states what is required, but not how this is to be achieved. Prescriptive standards, by contrast, correlate the level of safety which is said to have been achieved with specific objectives completed during development.

A major principle of MOD procurement is the need to be “as civil as possible, and only as military as necessary”. This means that it should be possible to comply with the requirements of Def Stan 00-56 via the use of civilian standards, supplemented where necessary. Furthermore, where existing organisation-level software development processes can be supplemented to produce evidence which is adequate for the purposes of Def Stan 00-56, these software development processes (and supplements) can also be used as a means of compliance. Consequently, this guidance has allowed for the use of alternative and novel techniques in complying with Def Stan 00-56.

The primary purpose of this guidance is to aid in the ensurance and assurance (for the purpose of this document, these terms are defined fully in Section 2.2) of software for MOD projects. As much of this work is undertaken in the supply chain, the emphasis of the complete SoBP will be on providing guidance to industry. However, certain issues such as safety management cannot be categorised as occurring solely on one side of the customer-supplier boundary. For example, a trade between cost and operational safety mitigation requires both supply side and customer side agreement. Also, in practice, the prime will often need to make management decisions. Consequently, some of the “management” decisions identified in this document could be carried out by the prime, some by MOD, or some (more likely) by the two working together. Further, management decisions may apply at several levels in the supply chain. Consequently, this guidance aims to be applicable independent of the particular stakeholders in any situation.

Section 2 contains the “Management” SoBP, describing the high-level managerial decisions which will need to be taken in order to produce software which can be shown to be safe with sufficient confidence. We provide a swim-lane diagram showing how safety-related communication should be managed throughout the project. This diagram identifies three major interested parties; the “Management” strand corresponding to the activities of managerial personnel and those responsible for project management of the customer-supplier boundary, the “Ensurance” strand corresponding to the activities of those personnel responsible for developing the software, and the “Assurance” strand corresponding to the
activities of those personnel responsible for demonstrating the safety of the software. In practice, activities are likely to overlap between these strands, although it is important to distinguish the fundamental differences.

Section 2 identifies the input which management personnel may expect to receive from the Ensurance and Assurance activities, and describes how this information can be assessed against the safety criteria of the project. Annex A supports Section 2 by providing reference material and guidance for making the managerial decisions. This Annex also contains background material on software safety arguments and important project parameters, as well as a case study providing partial validation of the SoBP. Throughout all sections, the relevant clauses in 00-56 will be noted where appropriate; for example 00-56 Part 2 Clause 17 will be denoted by (56-2 17).

Section 3 then contains an overview of the Technical Guidance, which consists of an overview of technical issues. This guidance is fully presented in Annex B, which comprises the technical “strand” of the SoBP. This technical material will be developed in detail as part of further SSEI work, and is intended to represent guidance for personnel undertaking an Ensurance or Assurance role. The overview in Section 3 contains sufficient detail to complement the managerial guidance in Section 2.

Section 4 also contains an overview of further applicable technical guidance, demonstrating how software safety can be assured in a number of different scenarios; these being the use of COTS products and the use of software developed to non-MOD standards. For further reading, the full details of this research are presented in Annexes C and D respectively; the overview in Section 4 is nevertheless intended to be sufficiently detailed to serve as an introduction to this research for management personnel.

This interim SoBP applies to the activities of contract assessment, software development, assurance, verification and validation and initial acceptance. It does not consider the in-service phase, nor does it consider in detail the concept and assessment phases. Nevertheless, many of the in-service issues are similar in scope to those presented here, especially considerations such as upgrading systems and the use of COTS products. It is the intent that further work will be performed to provide guidance on through-life safety considerations, including operational safety.

The SoBP will be available in web format, and the web version is to be considered definitive. This release represents an “interim” SoBP, as in some cases we have obtained only partial validation for the guidance. We anticipate updating this SoBP regularly to account for emerging technology, changing standards and further validation.
This .pdf version of the SoBP will be neither updated nor maintained. The latest version of the SoBP is available from the SSEI website, and is to be considered definitive.

2 Managerial Standard of Best practice

2.1 Introduction

This Standard of Best Practice (SoBP) is intended to support the development of safety-critical and safety-related software intended to comply with Defence Standard 00-56 Issue 4 Part 1 [2]. It expands on the guidance provided in Defence Standard (Def Stan) 00-56 Issue 4 Part 2 by providing information on best practice in complying with the requirements of the standard. This section (Section 2) is intended to have a managerial focus, being structured to emphasise safety-related project management decisions. A technically-focused SoBP has also been provided in Section 3 and the material in these two sections is intended to be complementary. This section describes the inputs that may reasonably be expected from the Ensurance and Assurance activities, and provides guidance on how this information should be assessed. The Technical SoBP introduced in Section 3 then provides further guidance for personnel undertaking Ensurance and Assurance activities on acceptable methods and ways in which to provide this information.

Safety-related decisions are not made in isolation, but are affected by all other development considerations. Consequently, this SoBP focuses on these safety-related decisions and seeks to reflect them in a wider context. This document assumes that there are identifiable management roles which can be distinguished, in principle, from the roles requiring the undertaking of engineering activities. The constitution of personnel performing these roles is not considered – that is, no distinctions are made in this document based on the balance between industry and MOD personnel. Def Stan 00-56 Issue 4 does not mandate the appointment of an Independent Safety Auditor (ISA), but places requirements on a contractor if an ISA is appointed. In some areas, e.g. the air sector, it is now standard practice for the MOD to appoint an ISA. For the purpose of this document this role is not distinguished and it is assumed that the decisions discussed here will be supported and informed by the ISA.

This SoBP contains references to the relevant clauses of Def Stan 00-56 for all guidance on managerial decisions. This guidance is to be read in conjunction with Def Stan 00-56 Part 2 Clauses 17 and onwards (i.e. 56-2 17) which addresses complex electronic elements, including software. Section 2.2 discusses the role of the Management, Ensurance and Assurance activities while Section 2.3 provides a brief characterisation of software safety cases. Section 2.4 then introduces the decisions and development activities on which this guidance is based. We include a swim-lane diagram showing the decisions which must be made during the development of systems containing software which is relevant to safety. Accompanying documentation describes the necessary information for making
these decisions, while Section 2.5 contains some final observations and conclusions. Annex A supports all the material of Section 2, and provides sufficient detail for the Managerial SoBP to be read alone, without recourse to the more detailed technical SoBP.

2.2 Management in the Context of DS 00-56 Issue 4

Developing and assessing a system according to Def Stan 00-56 requires the interaction of three main strands of activities: Management, Assurance and Assurance. Management activities are typically concerned with overseeing safety management, facilitating customer-supplier interaction and formally assessing relevant deliverables for acceptance. Assurance activities are typically concerned with the development of software to the specifications provided. Finally, assurance activities are typically concerned with the production of a compelling safety argument, supported by rigorous evidence. In practice, the remit of these activities may overlap. For example, performing hazard analysis and deriving safety requirements will require interactions between activities in all three strands.

A key requirement of Def Stan 00-56 is the production of a safety case (SC) and safety case reports (SCRs) (56-1 0.3g, h; 9) to justify the safety of the system. In the context of software, the SC and SCR are intended to justify the safety of software in its system context. Safety is required to be considered throughout the development (56-1 0.3 c), and consequently interim safety case reports should be delivered to confirm that safety is being managed correctly. Part of the role of Management personnel is to ensure that appropriate activities and processes are undertaken in order to produce these interim reports. Furthermore, personnel undertaking both management and assurance activities may assess these interim reports and influence the development of the software to ensure that the final product can be shown to be acceptably safe.

2.2.1 Management

The Management strand is concerned with the key decisions on a project level, which include issues of supplier selection (where relevant) and acceptance of the SC. The set of decisions are discussed in section 2.4 below, and illustrative guidance provided for these in Annex A.4. For each decision we identify:

- Inputs to the decision making activity from Assurance, Assurance or external activities (for example safety case reports or development plans). This list of inputs is intended to be indicative of the minimum information which will be needed, and should not be considered exhaustive.
- Comparator data to allow assessment of the inputs. This data may be available from a wide range of sources.
- Criteria for making the decision (for example, the acceptability of a safety case, or the extent to which risks are shown to be reduced to an acceptable level). It is likely that for each decision, the criteria should be weighted according to their importance. A mechanism for doing this is not provided in
this SoBP (in many cases, the weighting will be project-specific), but should be considered by managerial personnel.

- Possible outcomes, which in each case will be one of:
  - Proceed without change to plans
  - Proceed with further safety risk management
  - Iterate selected process steps with remedial action
  - Terminate the process and end the project development

In all cases, the guidance is framed so that all reasonable ways of proceeding will have been evaluated before reaching a decision to terminate the development.

Each of these decisions may involve input from the ISA, Safety Committee or external domain experts. These roles are not distinguished and are assumed to support the Management decision-making activities. In each case, the identity of personnel involved with this decision must be recorded in relevant project documentation. Where appropriate, the documents supporting their decision must also be preserved in order to provide both traceability and accountability.

Annex A.3 provides some general guidance on the project parameters which management may use as “levers” throughout the development. In all cases, it is expected that the personnel undertaking Ensurance and Assurance activities will provide input to these management decisions. Furthermore, relevant domain and safety experts may be consulted where appropriate.

### 2.2.2 Ensurance

The Ensurance strand is concerned with developing a system which satisfies all requirements, including safety requirements. Personnel undertaking Ensurance roles are responsible for performing the actual development of a system – requirements gathering, design, coding, testing etc. – and can therefore provide information on a number of issues, including development progress and the feasibility of satisfying particular safety requirements. Ensurance personnel may contribute to Management decisions, for example by providing information on costs of development options in order to help management personnel assess the potential for further risk reduction. Note that, even where the system contains considerable elements of COTS or legacy software – software items which are not being developed as part of the project – the Ensurance role still needs to be undertaken. More detailed guidance for personnel undertaking an Ensurance role is provided in both Annex A and the Technical SoBP introduced in Section 3.

### 2.2.3 Assurance

Finally, the Assurance strand is concerned with providing evidence that the software meets its safety requirements. For example, Assurance activities include the provision of safety case reports, as well as the final SC and SCR. Where Ensurance activities focus on the satisfactory completion of an acceptably safe
system, Assurance activities focus on the production of a compelling argument and supporting evidence to show that the system is acceptably safe. In some cases, the same personnel will be responsible for Assurance and Ensurance – for example, testing of the software throughout development is important both for the assurance and assurance of this software. This guidance retains the distinction between production of a safe system and the demonstration that the delivered system is safe in order to emphasise the need for a safety argument as required by Def Stan 00-56 (56–1 0.3). As before, more detailed guidance for personnel undertaking an Assurance role is provided in both Annex A and the Technical SoBP introduced in Section 3.

2.3 Safety Requirements and Safety Cases

2.3.1 Safety Requirements

If software is safety critical or safety related it will need to meet relevant safety requirements. If the software development is being outsourced, then some of these (high-level) requirements will be known at the time of placing a contract for the software (or system containing the software) (56-1 6.2, 6.3). Additionally, software safety requirements will emerge or be derived as the project proceeds, typically in order to mitigate hazards (56-1 10.3.3, 11.1). These are often referred to as derived safety requirements and are obtained as safety information emerges throughout the development. The Ensurance and Assurance activities include the establishment of the derived safety requirements, as well as the production of software which demonstrably satisfies all contractual and derived requirements.

2.3.2 Safety Cases and Safety Case Reports

From Annex A of 00-56, a safety case is “a structured argument, supported by a body of evidence that provides a compelling, comprehensible and valid case that a system is safe for a given application in a given operating environment” [2]. A safety case will evolve throughout a project, and the current state of safety should be reflected via regular safety case reports. The personnel undertaking ensurance and assurance roles are responsible for producing these reports, as well as a final safety case report. The acceptability of these reports may be dependent upon input from the ISA or Safety Committee. Each software safety case report must consider all relevant aspects associated with software safety, including the following:

- **Requirements validity** – the argument must demonstrate that all software safety requirements are complete and accurate for the purposes of mitigating the software contribution to system-level hazards.

- **Requirements satisfaction** – the argument must comprehensively demonstrate satisfaction of all the identified software safety requirements. (56-1, 6.3)
Requirements traceability – the argument must demonstrate that the high-level software safety requirements are traceable to system hazards, and also down through all levels of development (derived software safety requirements, design, code, evidence etc.) to supporting evidence. (56-1 10.7.4)

Software quality – the argument must demonstrate that the software and the development processes exhibit the basic qualities necessary to place trust in the evidence presented. For example, the software must be free from intrinsic errors (e.g. buffer overflows and divide-by-zero errors), and adequate configuration consistency and version control must be demonstrated.

The purpose of a software safety case is to provide a compelling, comprehensive and valid argument that the software in question is safe for a particular purpose in a particular environment. Annex A.1 describes the fundamentals of safety arguments, and notes that these are inductive, meaning that the conclusions do not necessarily follow with absolute certainty from the premises. Rather, the argument is intended to promote justifiable confidence in the reader that the software is safe.

The justifiable confidence in the truth of a safety claim is referred to as the assurance of that claim. A related concept is that of assurance deficit: an uncertainty or lack of information which affects assurance. Assurance deficits are almost inevitable; the question is whether such deficits are justified. An unjustified assurance deficit is one for which the cost of addressing this deficit (e.g. by providing additional evidence) is not grossly disproportionate to the benefit that would be gained. The Technical Guidance introduced in Section 3 contains further information on this, as does Annex A.1.

2.4 Software Safety Management Phases

Software development processes can depend on the organisation, context of the project, scope of the project, and so on. However this guidance document identifies four major phases which we believe can usefully be mapped to all software development projects. These phases are Initial (pre-development managerial safety activities relevant to Def Stan 00-56), Development (development of acceptably safe software as defined by Def Stan 00-56), Acceptance (assessing the completed software to determine whether the risk posed is acceptable) and Containment (judging whether any safety-related problems which occur during software development can be contained to ensure compliance with Def Stan 00-56). Note that, as discussed previously, the operational phase (post-development) is not explicitly considered. These identified phases are not intended as a software development lifecycle, but rather to identify the key decision points relevant to Def Stan 00-56 which occur during contractual interactions.

In this section we will provide guidance regarding the decisions which occur in each phase. This guidance will be in the format outlined in Section 2.2.1 i.e.
inputs, criteria and possible outcomes. First we summarise the process in a “swim-lane” diagram which relates the Management, Ensurance and Assurance activities (note that these boundaries are blurred, in practice) and places each decision in a development context. This diagram should be examined in conjunction with Annex A.2, which explains the process flow and discusses the background for each phase.
2.4.1 Swim-lane Diagram
**Figure 1: Swim-Lane Diagram**

In Figure 1, blue requests (lines from the Management lane) indicate communication from Management to Ensurance, Assurance, or to external domain experts. Red lines (lines into the Management lane) indicate communication from these strands to Management. These reflect the inputs needed in order for management to make the decisions identified in the Management swim-lane.

This swim-lane diagram identifies seven types of decision in the Management lane, and shows how these Management decisions interact with the activities undertaken in the Ensurance and Assurance lanes. The diagram shows a simple view of Ensurance and Assurance; this can be thought of as what Management needs to know in order to interface to these activities in order to discharge its responsibilities. To enhance readability, the diagram reflects only the major input(s) for each decision. In practice, there may be many such inputs, and these are described in the text below for each decision. Similarly, there are a number of iterations and ongoing activities in each lane which are not explicitly shown in the diagram for reasons of clarity. While major iterations are shown (e.g. the main iteration of the development phase reflecting ongoing monitoring of safety management and safety case reports), there will be iteration and ongoing activities within each lane. A single activity – as represented in the diagram – may correspond to a number of iterations of that activity, informed by safety dialogues and checkpoints.

We emphasise that this diagram should be read in conjunction with Annex A.2. This annex provides a textual accompaniment explaining the ongoing activities, iterations and multiple inputs referred to above. To enhance readability, the relevant portions of the swim-lane diagram have been enlarged in each section below.

Def Stan 00-56 Issue 4 is intended to constrain development as little as possible. Nevertheless we provide guidance for all development phases. This guidance is not based on any particular process or project content. Rather, it describes some of the activities and decisions which may be encountered during development, and which can help the project arrive at a satisfactory outcome (in this case, successful acceptance). The guidance does in part address the mandated documents in Def Stan 00-56, such as the hazard log and the safety case reports.

**2.4.2 Initial Phase**

There are a number of activities and decisions which take place prior to establishing contractual arrangements. These include gathering requirements, establishing a project budget, writing an Invitation to Tender, assessing bids against project criteria, and negotiating with selected suppliers. As Def Stan 00-56 is a contractual standard, it does not strictly apply to these activities and decisions which take place at a pre-contractual stage.
However, the Initial phase is important from a safety perspective because of the importance of supplier selection. The capabilities of potential suppliers – including in-house developers, if the software development is not to be contracted out – must be assessed in order to determine the likelihood of eventual delivery of acceptably safe software. Although these decisions relating to supplier selection are not strictly constrained by Def Stan 00-56, this SoBP provides high-level guidance on the topic. Issues identified during the assessment of suppliers may usefully form the subject of contractual negotiation, or additional contractual criteria. In this way, adequate assessment of suppliers and bids can enhance the likelihood of eventual delivery of software which is compliant with the requirements of Def Stan 00-56.

This SoBP is not intended to provide full guidance on supplier selection. Rather, the guidance provided here is intended to be used in conjunction with any other selection and assessment procedures which apply, such as POSMS, TickIT or AQAP. Consequently, we have not provided guidance on issues such as writing an Invitation to Tender, as such activities are suitably described in existing procedures and policies.

Supplier selection as described in this SoBP is based on the assessment of documentation intended to support the bid. We emphasise that where software is developed in-house – and an Invitation to Tender is therefore not published – the relevant documentation should also be available to justify this action from a safety perspective.

**Decision: Selecting a supplier**

As indicated above, this decision does not directly relate to safety of the software in question. However, it does influence the likelihood of reaching the acceptance point and being able to accept the system into service. Where software is being developed in-house, a formal tender assessment will not be performed. However, as stated earlier, many of the inputs listed below for this decision must still be provided, and many of the recommendations of this section still apply, although
some – such as the need for evidence supporting supplier capability – may not be required. This process ensures that the in-house team are capable of carrying out the work. Annex A.1 and Annex A.4 contain further information on this decision.

Part of the selection process involves the performance of a criticality analysis to determine the importance to safety of this software. This activity may be undertaken by the acquirer prior to accepting tenders, or may be requested as a supplier activity.

**Inputs**

Tender Documents should ideally include:

- Safety Management Plan (SMP) (56-1 8).
- Proposed safety case strategy or preliminary safety case, possibly as part of the SMP (56-1 9).
- Criticality analysis of the proposed software, where the tender requires this activity to be undertaken by the supplier.
- Software sizing, costing and project timing data.
- Evidence supporting supplier capability.

**Comparator Data**

- SMP for similar projects
- Safety cases for similar projects and software safety argument pattern libraries
- Where criticality analysis was undertaken by the supplier, suitable comparator data includes the results of criticality analyses using similar methods for similar projects.
- Industry performance norms for similar projects.
- Public domain maturity models, e.g. the Capability Maturity Model Integrated (CMMI).

**Criteria**

The SMP content should include the following:

- A well-defined process for the proposed software development, with a justification of the suitability of this process against the requirements of Def Stan 00-56. (56-2 17.3). The process may be based on the standards being used, or internal processes used effectively on similar projects. Standards with the widest acceptance (i.e. international standards) are to be preferred.
- Identification of the evidence which will be obtained by the above process for the four elements of the safety case (validity, satisfaction, traceability, software quality). Where this requires additional processes beyond those defined in relevant international standards, these additional processes should be identified and described.
• Identification of safety legislation, regulations and standards judged relevant to the safety of this system (56-1 6.1), where these are known.

The proposed safety case strategy or preliminary safety case should demonstrate the following properties:

• It should address all four argument elements (validity, satisfaction, traceability and quality) and identify evidence and argument strategies for each.
• It should describe how adequate assurance will be achieved with respect to all safety requirements, if the development proceeds as planned.
• It should be sufficient to account for evolving safety requirements, or the emergence of new safety-related information.

Where undertaken by the supplier, the criticality should satisfy the following properties:

• It should be appropriate and realistic for software in this role and system.
• The initial hazard assessment should consider all hazards which could reasonably have been identified given the information provided.
• The contribution of this software towards these hazards should be adequately justified, given the information available.

The estimated sizing, costing and timing data should be satisfactory as below:

• The size (LOC) should be realistic for comparable software, given industry norms and expected growth rate.
• The costing is realistic for comparable software (including consideration of criticality), under comparable circumstances. The cost should include consideration of technical risks, and the work required to remedy these.
• The estimated time to complete should be realistic, given the size and costing estimates, and the estimate of this software’s criticality.

The evidence of supplier capability should consider the following points:

• The processes used to demonstrate supplier capability – e.g. development processes used on previous projects – should offer evidence of repeatability (this ensures that previous experience of these processes is a good indicator for future use).
• The extent to which the safety processes used on this project will be transparent to external examination.
• Demonstration of supplier experience developing comparable software, including a consideration of criticality.

Possible Outputs

The possible outputs for this decision are as listed below. In each case, the identity of personnel involved with this decision must be recorded in relevant project documentation. Where appropriate, the documents supporting their
decision must also be preserved in order to provide both traceability and accountability.

- Accept this supplier and proceed. This outcome reflects that all the above criteria are met.
- Accept this supplier and proceed with further risk management. This outcome reflects that any of the above criteria are not met, and that means to remedy the deficiencies have been identified. These may include risk mitigation activities on the part of the acquirer.
- Iterate (repeat) process steps, with remedial action. For this decision, this outcome reflects that there has been no satisfactory tender, and one or more bidders are asked to retender; in this case feedback to the bidders will identify those areas where the bid fell short of expectations. This outcome may also encompass acceptance of this supplier conditional upon a satisfactory contract negotiation procedure to remedy identified deficiencies.
- Terminate the process. For this decision, this outcome reflects that there are no suitable bids, and retendering is not an option.

2.4.3 Development Phase

The development phase is concerned more directly with achieving software safety than the Initial phase. Specifically, the managerial decisions in the development phase are intended to confirm that the software is being developed in an acceptably safe manner according to the requirements of Def Stan 00-56.

These decisions also identify the need for corrective action where necessary to enhance the likelihood of successful acceptance. There are three decisions in the phase, although the last one: Development Complete? has no direct safety implications and so is not further discussed here. The development phase is iterative as shown in the swim-lane diagram, and consequently the decisions in this phase may be taken multiple times. This iteration is not considered explicitly, but is implicitly considered to take place throughout software development.
**Decision: Safety Management Acceptable**

The safety management issues are largely concerned with Ensurance, or with the extent to which the software under development satisfies safety requirements. However, the identification of derived safety requirements, hazards and mitigation strategies encompasses both Ensurance and Assurance activities. Reports on safety management may be expected throughout software development, and the level of detail contained therein will necessarily be dependent upon the extent to which development is complete.

Significant Management activities associated with this decision (although not explicitly shown) here may involve entering into a dialogue with multiple suppliers and organisations in order to adequately manage interfaces (56-1 12).

**Inputs**

Reports or updates (e.g. included in an interim safety case report) on safety management progress including:

- Progress in identifying hazards (e.g. hazard logs), risks, and associated derived safety requirements.
- Progress in risk reduction, including proposed strategies for further risk reduction.
- Configuration management system used to demonstrate consistency.
- Identification of assumptions relevant to risk analysis.
- Identification of interfaces relevant to the software or the wider system safety case.
Comparator Data

- Hazard and risk data for similar projects.
- Hazard logs and safety requirements for the system in which this software will operate (this may serve to identify gaps in the hazard analysis).
- Information on successful risk reduction strategies for similar projects.
- Interfaces between this software and the wider system which have previously been identified by external developers or other suppliers.

Criteria

The management of hazards and risks should demonstrate the following criteria:

- The timeliness of progress in identifying hazards and safety requirements is according to plan and typical or better.
- All identified hazards have identified controls and associated derived safety requirements which are sufficient and credible, given the software contribution to system hazards.
- All software-specific derived safety requirements can be traced to the associated hazards/risk mitigation actions and vice versa.

The risk reduction activities should demonstrate the following:

- All software-specific derived safety requirements are covered in the design documentation, and include production of sufficient evidence to support the safety case (this relates to project risk reduction).
- The project is on track to deliver the implementation of the software to meet the derived safety requirements, and to deliver the evidence needed in support of the safety case (this relates to project risk reduction).
- Where identified risks have not been reduced to a level where a system-level ALARP argument could be made, risk reduction strategies should show how this will be adequately remedied.
- The level of visibility and detail in the risk reduction activities should be commensurate with that indicated in the tender.

The configuration management system should satisfy the following:

- Adequate software development documentation is supplied, and all safety evidence provided is explicitly traceable to the software state and conditions under which it was derived.

Assumptions should be justified as below:

- Where there are uncertainties relating to hazards beyond the scope of this project, these hazards are explicitly noted in the Hazard Log (56-1 10.4.4).
• Assumptions are made explicit and where appropriate justified according to evidence presented in the safety argument (see Section 2.4.4)

Interfaces should be managed as described below:

• Processes and procedures for managing interfaces should be adequately documented, which may be in the SMP or updates to this (56-1 12).
• All information needed by the supplier to manage interfaces should be clearly identified.

Possible Outputs

The possible outputs for this decision are as listed below. In each case, the identity of personnel involved with this decision must be recorded in relevant project documentation. Where appropriate, the documents supporting their decision must also be preserved in order to provide both traceability and accountability.

• Proceed with development. This is represented as iteration in the development phase of the swim-lane diagram, and occurs when all the above criteria are met.
• Proceed with development, where this includes specified further risk management. This outcome reflects that many of the above criteria are met, and that means to remedy the deficiency in risk management have been identified and will inform future development.
• Iterate (repeat) process steps, with remedial action. For this decision, this outcome reflects that a significant deficiency has been identified, and could render nugatory any further development unless rectified. The next development steps should be intended to rectify this deficiency.
• Terminate the process. For this decision, this represents an exit to the containment phase. This occurs where there is no credible strategy to remedy the observed deficiencies.

**Decision: Safety Argument Acceptable**

The managerial decisions relating to the safety argument are likely to be focused on the sufficiency of the argument as presented. Domain and safety experts (including the ISA) can be utilised to judge the technical sufficiency of the evidence, further guidance on which is provided in the Technical Guidance. Where uncertainty or a lack of information means the safety argument does not deliver sufficient confidence in the safety of the software for this to be accepted into service, this lack of information is referred to as an assurance deficit. The assurance deficit should be justified using the techniques described in Annex B. (Routine interim safety case reports will be presented to Management for approval
of the safety argument development. These will be essentially the same
documents used as input for the previous decision, but in this case these
documents are being considered explicitly to determine the development of the
safety argument and assess any existing assurance deficits.

The routine interim safety case reports will be presented to Management for
approval (this decision) only where the personnel undertaking an assurance role
consider that all assurance deficits identified thus far are justified as far as is able
at this stage of development, or that these deficits are likely to be justified by
planned future processes. If personnel undertaking Assurance activities identify
an assurance deficit which they consider will not be justified given future
development, the Containment phase will be entered instead.

**Inputs**

Report on safety management progress including:
- Development of the safety argument.
- Production of evidence to support the safety argument.

**Comparator data**
- Safety arguments and evidence for similar projects.
- Software safety argument patterns which illustrate typical successful
  patterns of argumentation.

**Criteria**

The safety argument should satisfy the following criteria:
- It should address all four argument elements (validity, satisfaction,
  traceability and quality) with respect to all safety requirements.
- It should be sufficient to provide adequate assurance with respect to all
  safety requirements, or indicate how this assurance will be obtained.
- It should identify and justify all assurance deficits.
- All assumptions should be identified and where appropriate justified,
  with references to supporting documentation where relevant.
- Evidence of a search for counter-evidence should be presented, and
  the effect of relevant counter-evidence upon the argument should be
  assessed.

The evidence provided to support the safety argument should satisfy the
following:
- The evidence should adequately support the relevant safety
  requirements.
- The integrity of the evidence chain should be evident, meaning that
  sufficient visibility into evidence-gathering procedures is provided.
• The trustworthiness and applicability of the evidence should be justified and it should be sufficiently diverse (56-1 11)

Possible Outputs

The possible outputs for this decision are as listed below. In each case, the identity of personnel involved with this decision must be recorded in relevant project documentation. Where appropriate, the documents supporting their decision must also be preserved in order to provide both traceability and accountability.

• Proceed with development. This is represented as iteration in the development phase of the swim-lane diagram, and occurs when all the above criteria are met. There is no unjustified assurance deficit.

• Proceed with development where this includes specified further safety management. This outcome reflects that there is currently an unjustified assurance deficit, but this can be justified by means which have been identified and which will inform future safety argument development.

• Iterate (repeat) process steps, with remedial action. For this decision, this outcome reflects that an unjustified assurance deficit is present and can be addressed only by modifying or repeating activities in the development of the safety argument.

• Terminate the process. For this decision, this represents an exit to the containment phase. This occurs when there is an unjustified assurance deficit and no identified strategy for sufficiently reducing this deficit.

In practice management of information flows across contractual or organisational boundaries can be problematic. It may be the case that shortfalls in the (demonstrated) safety of the system are related to such boundaries and interfaces. Consequently, it is desirable that management explicitly consider this flow of information. It may also be the case that the flow of information down from the system level to the software is inadequate. In order to assess the potential for the software to contribute to system hazards, a degree of information is needed about the system context. In some cases, this may mean providing information to the supplier about the wider system in order to ensure that safety requirements are satisfied (56-1 12).

It should be noted that as shown in the swim-lane diagram, the only way to proceed to the acceptance phase is by judging the safety argument to be acceptable. This is in keeping with the requirement for an adequate safety case (56-1 9).

2.4.4 Acceptance Phase

As with the Initial phase, there are a number of decisions and activities which must be undertaken when assessing completed software against the acceptance
criteria. Not all of these decisions and activities are related to safety; in general it is also unlikely that Def Stan 00-56 will be the sole applicable contractual standard. Additionally, there may be existing organisational-level or MOD-specific policies and procedures in place for assessing completed software against the acceptance criteria.

This SoBP provides guidance for assessing the completed software against the requirements of Def Stan 00-56 – in practice, this may be said to equate to guidance for assessing the completed safety case to determine whether this provides sufficient confidence in the software. This is not intended to imply that assessment of the safety case is the only activity necessary for acceptance of the software. Rather, the guidance presented here as to the assessment of the safety case against the requirements of Def Stan 00-56 is intended to be considered in conjunction with all existing policy for assessing software prior to acceptance.

It is expected that management will be supported by technical staff in judging the adequacy of the safety case. Nevertheless, Management personnel are also required to contribute to the discussions and trade-offs involved in the decisions in this and the next phase. Further guidance on this is provided in Annex A.1. It is important to note that the acceptance phase is entered only if the personnel involved with Assurance activities believe the safety case is adequate. That is, if there are known deficiencies with the safety case, then software development will not conclude with an assessment of the software against the acceptance criteria; rather, the Containment phase is entered.

If the safety case provides sufficient confidence that the risk posed by the software is acceptable, software development concludes. If not, the containment phase is entered to attempt to remedy this problem.

**Decision: Accept At Risk**

**Inputs**

Safety case including:
• The safety argument
• Evidence to support the safety argument

Comparator data
• Safety arguments and evidence for similar projects
• Software safety argument patterns which illustrate typical successful patterns of argumentation

Criteria
The completed safety argument should satisfy the following criteria:
• It should address all four argument elements (validity, satisfaction, traceability and quality) with respect to all safety requirements.
• It should be sufficient to provide adequate assurance with respect to all safety requirements.
• It should adequately demonstrate that all risks have been reduced to a level which would permit a system-level ALARP argument to be made.
• It should identify and justify all assurance deficits
• All assumptions should be identified and justified, with references to supporting documentation where relevant.
• It should demonstrate that a sufficient search for counter-evidence has been undertaken, and the effect of any identified counter-evidence assessed.

The safety evidence should satisfy the following criteria:
• The evidence should adequately support the relevant safety requirements
• The integrity of the evidence chain should be evident, meaning that sufficient visibility into evidence-gathering procedures is provided.
• The trustworthiness and applicability of the evidence should be justified, and it should be sufficiently diverse (56-1 11)

Possible Outputs
The possible outputs for this decision are as listed below. In each case, the identity of personnel involved with this decision must be recorded in relevant project documentation. Where appropriate, the documents supporting their decision must also be preserved in order to provide both traceability and accountability.
• Proceed to acceptance of software, subject to the satisfaction of all relevant criteria besides those relating to safety.
• Proceed with further risk management – not applicable
Iterate (repeat) process steps, with remedial action. For this decision, this outcome reflects that an unjustified assurance deficit has been identified at this (comparatively late) stage. Since further development of the completed software is not possible, this prompts a transition to the containment phase, with the possibility of remedial action.

Terminate the process. For this decision, this outcome reflects that an unjustified assurance deficit or significant contractual conflict has been identified at this (comparatively late) stage. There is no immediately obvious remedial action, prompting a transition to the containment phase.

The last two outputs are very similar, although the former suggests that the problem is more likely to be contained.

2.4.5 Containment Phase

The containment phase is entered only on encountering a significant problem during development which cannot be remedied. Entry can be triggered in one of two ways. Firstly, personnel undertaking assurance activities may note that they are unable to adequately justify all assurance deficits, and that future development is unlikely to provide information which will justify these deficits. Secondly, management personnel may consider that significant problems are exhibited by ongoing safety management, by ongoing safety case development or by the final safety case. These problems may result in a lack of information which has the potential to affect assurance – an assurance deficit.
**Decision: Acceptable Mitigation for Assurance Deficits**

The decision is encountered when there is – or is likely to be – an unjustified assurance deficit, which is unlikely to be remedied within the bounds of the original safety management plan and proposed safety case structure. The (potential) presence of this assurance deficit should be communicated to Management in a timely manner. In addition to this communication, activities should be undertaken to identify possible methods of addressing the assurance deficit. These activities are undertaken across all three strands of Ensurance, Assurance and Management, and in some cases external personnel may also be involved to identify methods to address this deficit. Once these methods have been identified, Management must determine whether they represent an acceptable solution to reduce or justify the presence of this assurance deficit.

**Inputs**

Safety case report including:
- The safety argument
- Evidence to support the safety argument
- A report on the unjustified assurance deficit

Report on proposed methods for addressing the assurance deficit including:
- Input from Ensurance / Assurance / external personnel as relevant

**Comparator data**

- Safety arguments and evidence for similar projects
- Software safety argument patterns which illustrate typical successful patterns of argumentation
- Information on techniques for resolving assurance deficits (Annex A contains further details on this)

**Criteria**

The supplied safety argument should satisfy the following criteria, with any discrepancies addressed by the proposed methods for resolving the assurance deficit.

- It should address all four argument elements (validity, satisfaction, traceability and quality) with respect to all safety requirements.
- It should be sufficient to provide adequate assurance with respect to all safety requirements, or indicate how this assurance will be obtained.
- It should identify and justify all assurance deficits
- All assumptions should be identified and justified, with references to supporting documentation where relevant.
Evidence of a search for counter-evidence should be presented, and the effect of relevant counter-evidence upon the argument should be assessed.

The evidence provided to support the safety argument should satisfy the following, with any discrepancies addressed by the proposed methods for resolving the assurance deficit:

- The evidence should adequately support the relevant safety requirements
- The integrity of the evidence chain should be evident, meaning that sufficient visibility into evidence-gathering procedures is provided.
- The trustworthiness and applicability of the evidence should be justified and it should be sufficiently diverse (56-1 11)

The assurance deficit report should provide the following information:

- An assessment of the local and system effects of this deficit, where known.

The report on methods for addressing the assurance deficit should include the following:

- Identification where possible of techniques to address this deficit, with consideration of how these may fit into the safety management plan.
- A comparison of these techniques to demonstrate how they will provide additional assurance.

Possible Outputs

The possible outputs for this decision are as listed below. In each case, the identity of personnel involved with this decision must be recorded in relevant project documentation. Where appropriate, the documents supporting their decision must also be preserved in order to provide both traceability and accountability.

- Proceed with no change – not applicable
- Proceed with further risk management. For this decision, this outcome is applicable in two cases. Where the identified assurance deficit can possibly be remedied with further software safety management (there are no scheduled future assurance tasks which could address this deficit, but some may be added), the identified remedial actions should inform the future development of the safety argument. Where the deficit cannot be remedied (project constraints mean that it is not feasible to add further assurance tasks to address this deficit) development may proceed provided that system-level risk management techniques are identified to justify this deficit. This latter choice will require the
cooperation of external developers and approval across the entire system.

- Iterate (repeat) process steps, with remedial action. For this decision, this outcome reflects that further development cannot proceed until this assurance deficit is reduced. Alternative verification processes must be undertaken, as this assurance deficit could render nugatory all further development activities.

- Terminate the process. This outcome reflects that there is no identified strategy to reduce this assurance deficit, and the next step is to consider a possible change to the circumstances and environment of this software.

Annex A contains some further guidance on addressing assurance deficits. It is unlikely that it is practical to give more detailed guidance than that supplied in the Annex, as the details will depend on the project.

**Decision: Circumstances Can Be Changed Acceptably**

This decision is encountered when there is an unjustified assurance deficit for which all credible remedies are considered unacceptable. This may be prompted by significant problems with ongoing safety management, safety case development, or the final safety case. This decision is designed to identify whether there is a radical change which may cause the software – which is currently deficient in some way – to reach successful acceptance. Radical changes of this type may feasibly involve a change to one or more of the management “levers” of scope, quality, resources and timing (Annex A.3).

**Inputs**

There may be multiple inputs for this decision, depending upon the reasons why this software is considered deficient. Relevant inputs include the following:

Safety case report, including:

- The safety argument
- Evidence to support the safety argument.
- A report on all unjustified assurance deficits.

Report detailing why the software has been deemed deficient:

- For example, the presence of unjustified assurance deficits such as those caused by inadequate analyses of hazards as well as unmet acceptance criteria or contractual conflicts. This report may originate from the supplier, or may be the result of discussions between the supplier, acquirer and relevant external personnel.

Report on possible role or circumstance changes, including:
- Data about the “project parameters” – scope, quality, resources and timing – and how these might be acceptably changed to address the above deficiency.
- Information about the proposed environment for the software and how it might be acceptably changed to address the above deficiency.

**Comparator data**
- Safety arguments and evidence for similar projects.
- Software safety argument patterns which illustrate typical successful patterns of argumentation.
- Information on similar software operating under different circumstances or in different environments (e.g. with reduced criticality).

**Criteria**

The supplied safety argument should satisfy the following criteria, with the caveat that these may not all apply depending on the precise nature of the deficiency:

- It should address all four argument elements (validity, satisfaction, traceability and quality) with respect to all safety requirements.
- It should be sufficient to provide adequate assurance with respect to all safety requirements, or indicate how this assurance will be obtained.
- It should identify and justify all assurance deficits.
- All assumptions should be identified and justified, with references to supporting documentation where relevant.
- Evidence of a search for counter-evidence should be presented, and the effect of relevant counter-evidence upon the argument should be assessed.

The evidence provided to support the safety argument should satisfy the following, with the above caveat:

- The evidence should adequately support the relevant safety requirements.
- The integrity of the evidence chain should be evident, meaning that sufficient visibility into evidence-gathering procedures is provided.
- The trustworthiness and applicability of the evidence should be justified and it should be sufficiently diverse (56-1 11).

The report detailing reasons why the software has been considered deficient should include the following information:

- The identified hazards to which this software contributes and the extent to which this deficiency affects the assurance of the software.
- The interfaces – and therefore other system components – which are affected by this deficiency, where known.
• An assessment of the local and system effects of this deficiency, where known.
• Any contractual issues which have contributed to this deficiency.

The report on possible role and circumstance changes should include the following:

• The extent to which the deficiency can be addressed by a change to one of the following: scope, quality, resources, timing.
• The effect of this change on the interfaces and components identified above.
• The extent to which these changes will ameliorate any unjustified assurance deficits.

It is not possible to provide more detailed information about inputs, as the issues which have caused software deficiencies may be system-specific. Further guidance is given in Annex A.4 on this topic.

Possible Outputs

The possible outputs for this decision are as listed below. In each case, the identity of personnel involved with this decision must be recorded in relevant project documentation. Where appropriate, the documents supporting their decision must also be preserved in order to provide both traceability and accountability.

• Proceed with no change – not applicable
• Proceed with further risk management – not applicable
• Iterate (repeat) process steps, with remedial action. For this decision, this outcome reflects that there is an unjustified assurance deficit which cannot be addressed without alteration to the original “project parameters”. Any further development steps must be focused on these alterations, as development cannot proceed until these are complete.
• Terminate the process. This outcome reflects that there is no identified role or circumstance change which is acceptable in terms of reducing this assurance deficit. Software development will cease, and the software will not be accepted.

2.5 Observations and Conclusions

The aim of this SoBP has been to provide managerial level guidance in such a way that managers can make informed decisions while being aware of both the nature of those decisions and the information needed to support the decisions. The companion technical SoBP introduced in Section 3 provides more detail on how to analyse safety-related software development and to provide appropriate
information to management. However, it is intended that this managerial SoBP should be sufficient to inform activities undertaken by management personnel.

Nevertheless, a certain level of technical awareness will help managers in their decision making. The annexes to this SoBP are intended to provide enough contextual information to enable managers to make informed decisions. Additionally, we also provide a case study (Annex A.6) to complement this SoBP. It is anticipated that the number and variety of these case studies will be updated at regular intervals.

Finally, it is recognised that an SoBP in this area is unlikely to be static, as the notion of best practice evolves over time. Thus the intention is that the SoBP will be updated periodically reflecting the experience of using the document and changes in what is accepted as best practice in this area.

### 3 TECHNICAL GUIDANCE

This section contains a summary of the material for an interim Technical SoBP, intended to support the managerial guidance discussed above. This Technical SoBP provides guidance of relevance to anyone involved in the development or assessment of safety arguments for software in the context of Def Stan 00-56. Full details are provided in Annex B.

Def Stan 00-56 requires the production of a safety argument which is commensurate with system risk (56-1 9.5):

> “The Safety Case shall contain a structured argument demonstrating that the evidence contained therein is sufficient to show that the system is safe. The argument shall be commensurate with the potential risk posed by the system,…”

[2]

This document provides guidance on how to comply with this requirement when considering the software components of systems. In particular, the focus of this technical guidance is on methods to construct compelling software safety arguments and justify their sufficiency.

A software safety argument must demonstrate that the software under consideration is acceptably safe to operate as part of the embedding system. This requires that it be demonstrated that the potential contribution made by the software to the identified system hazards is acceptable. To be compelling, the software safety argument must provide sufficient confidence in claims relevant to this objective. It is inevitable for the software aspects of a system that there will exist inherent uncertainties that affect the assurance with which it is possible to demonstrate the safety of the software. The reason for this is that the amount of information potentially relevant to demonstrating the safety of the system is vast. This may be information relating to the software itself, or to the system within which the software operates. There will also be information relating to the
environment and operation of the system, all of which potentially has a role in demonstrating that the software is acceptably safe. It is simply not possible therefore to have complete knowledge about the safety of the software. This leads to uncertainty, for example through having to make assumptions or accept known limitations in the integrity of the evidence generated, or the strength of support that evidence provides. For this reason it is not normally possible to demonstrate with absolute certainty that the claims made in a software safety argument are true. For a software safety argument to be compelling it must instead establish *sufficient confidence* in the truth of the claims that are made. It is worth noting at this point that such uncertainties in demonstrating the safety of the software are always present, but are often left implicit. Adopting a safety argument-based approach, as is required by Def Stan 00-56, facilitates the explicit identification of such uncertainties, which makes them easier to reason about, and therefore justify. Reasoning explicitly about the extent and impact of the uncertainties in a safety argument aids in the successful acceptance of the argument as part of a safety case.

Annex A.1 briefly considers how to demonstrate sufficient confidence in a software safety argument from a managerial perspective, but from an Ensurance / Assurance position, more detail is needed in order to construct sufficiently compelling safety arguments. The material contained in this section is intended to provide this detailed guidance.

The assurance of a claim is the justifiable confidence in the truth of that claim. A useful approach to ensure that a software safety argument is sufficiently compelling is to consider assurance throughout the development of that argument. Full guidance on an “assurance-centric” approach to argument construction is provided in a separate SSEI document [6], available in Annex B. The guidance is split into two main parts: a software safety argument pattern catalogue and an assurance based argument development method.

### 3.1. Pattern Catalogue

The software safety argument patterns (the details of which are provided in Annex B) are methods of arguing about particular claims, or “sample arguments” which can be instantiated with specific claims and evidence supporting those claims. They are intended to capture current good practice for compelling software safety arguments. To that end, they have been constructed based on existing patterns and an understanding of current practice for software safety arguments. A primary consideration during the development of these patterns has been flexibility and the elimination of system-specific concerns and terminology. Consequently, these patterns can be instantiated for a wide range of systems and under a variety of circumstances. To be compelling it is necessary to be able to justify that the instantiation decisions taken in constructing the argument result in a sufficiently compelling argument for the system under consideration (such as why particular
claims are chosen whilst others are not required). Guidance for justifying such decisions is provided in Section 3.2.

The software safety argument pattern catalogue contains a number of patterns which may be used together in order to construct a software safety argument for the system under consideration. The following argument patterns are currently provided in [6] and Annex B:

1. High-level software safety argument pattern – This pattern provides the high-level structure for a generic software safety argument. The pattern can either be used to create the high level structure of a software safety argument either as a stand alone argument or as part of a system safety argument.

2. Software contribution safety argument pattern - This pattern provides the generic structure for an argument that the contributions made by software to system hazards are acceptably managed. This pattern is based upon a generic ‘tiered’ development model in order to make it generally applicable to a broad range of development processes.

3. Software Safety Requirements identification pattern - This pattern provides the generic structure for an argument that software safety requirements (SSRs) are adequately captured at all levels of software development.

4. Hazardous contribution software safety argument pattern – This pattern provides the generic structure for an argument that the identified SSRs at each level of software safety development adequately address all identified potential hazardous failures.

5. Argument justification software safety argument pattern - This pattern provides the generic structure for an argument that software safety argument presented is sufficient.

It is intended that the software safety argument pattern catalogue will be updated and expanded over time to ensure that it reflects current understanding of good practice.

3.2. Assurance-based Argument Development Method

As discussed earlier, there exist many potential sources of uncertainty in demonstrating the safety of the software. Any such residual uncertainty can be considered to be an assurance deficit. Assurance deficits can reduce the assurance which is achieved. It is possible to identify how assurance deficits may arise by explicitly considering how information may be lost at each step in the construction of the argument. As an argument is constructed, decisions are continually being made about the best way in which to proceed. Decisions are made about how goals are stated, the strategies that are going to be adopted, the
context and assumptions that are going to be required, and the evidence it is necessary to provide. Each of these decisions has an influence on what is, and is not, addressed by the safety case. The things that are not sufficiently addressed are referred to as assurance deficits.

Annex A.5 contains a summary of the assurance-based argument development method. This provides an approach for systematically considering how assurance deficits may be introduced at each step of software safety argument development. By identifying where potential assurance deficits may arise, this approach can be used to inform the decisions that are made on how to construct the argument. In order to produce a sufficiently compelling software safety argument, all identified assurance deficits must be satisfactorily addressed, or justification must be provided that the impact of the assurance deficit on the claimed safety of the system is acceptable. Section 3.2.1 discusses how such justifications may be made.

Def Stan 00-56 states, “Throughout the life of the system, the evidence and arguments in the Safety Case should be challenged in an attempt to refute them. Evidence that is discovered with the potential to undermine a previously accepted argument is referred to as counter-evidence.” (56-2 9.5.6) Since an assurance deficit corresponds to a lack of relevant information, an identified assurance deficit reveals the potential for counter-evidence. That is, there is the possibility that in addressing the assurance deficit (i.e. gaining the relevant information) the information gained would reveal previously unidentified counter evidence. Reasoning about assurance deficits can therefore be helpful in identifying areas in which counter evidence may exist. Conversely, where there is knowledge of existing counter evidence, this can be used to help determine the potential impact of assurance deficits. For example, if other similar projects have identified counter evidence which relates to a particular identified assurance deficit, then the observed impact of this counter evidence on the safety of the other project can be used to indicate the expected impact that such an assurance deficit may imply.

### 3.2.1 Addressing Assurance Deficits

The discussion above illustrated how assurance deficits may be systematically identified throughout the construction of a software safety argument. The existence of identified assurance deficits raises questions concerning the sufficiency of the argument. Therefore where an assurance deficit is identified, it is necessary to demonstrate that the deficit is either acceptable, or addressed such that it becomes acceptable (for example through the generation of additional relevant information). There will typically be a cost associated with obtaining the information to address an assurance deficit. In theory it would be possible to spend large sums of money generating sufficient information to address all assurance deficits. However in practice the benefit gained from addressing each assurance deficit does not necessarily justify the cost involved in generating the additional information. In order to assess if the required level of expenditure is warranted, the impact of that assurance deficit on the sufficiency of the argument...
must be determined. Firstly we therefore discuss how the impact of an assurance deficit can be assessed.

The software safety argument establishes a claimed position on the hazard identification, risk estimation, and risk management of the software contribution to system hazards. Since assurance deficits have the potential to undermine the sufficiency of the argument, the impact of any assurance deficit should be assessed in terms of the impact it may have on this claimed position. Is the assurance deficit significant enough that that position can no longer be supported? For example, an assurance deficit may be sufficient to challenge the completeness of hazard identification, or may be sufficient to challenge the estimated residual risk. It may also be possible, for example, that an assurance deficit challenges a claim that the software contribution to system hazards are acceptably managed.

Def Stan 00-56 requires that the safety case demonstrate that risks be reduced to a level that is ALARP. However with respect to systems containing complex electronic elements, there is an acknowledgement that “…much of the effort only improves confidence that requirements have been met. In applying ALARP, the confidence achieved should be proportionate to the risk.” (56-2, Annex B3.2). In establishing the overall claimed position of the software safety argument, some of the argument claims can be recognised as being more important than others. For example claims regarding the behaviour of an architectural component (such as a voter), which carries a greater responsibility for risk reduction than other components, are more important to the overall software safety argument. Therefore claims relating to those components would require a greater degree of assurance (more confidence must be established). Def Stan 00-56 describes how, “An example of a way of defining the variation of the degree of rigour with potential risk is the specification of a safety integrity requirement for the system” (56-2, 9.5.2), it goes on to state, “In setting safety integrity requirements, it is therefore important to consider how much confidence is needed.” (56-2, Annex C1.2). Where safety integrity requirements have been defined, they can be used as a way of determining the importance of the software safety argument claim to which they relate.

The impact of an assurance deficit should first be determined by considering the importance of the truth of the related claim or claims in establishing the claimed risk position of the safety case (when considered in the overall context of the system). Secondly the relative importance of the assurance deficit to establishing the truth of that claim must also be considered. It then becomes possible to determine the overall impact of the assurance deficit in terms of its potential impact on the claimed risk position. Further guidance on determining the impact of assurance deficits is provided in a separate SSEI document [6], available in Annex B.

In a similar manner to risks in the ALARP approach, the impact of the identified assurance deficits may be usefully classified into three categories. An “intolerable”
deficit could be one whose potential impact on the claimed risk position is too high to be justified under any circumstances. In contrast, assurance deficits may be categorised as "broadly acceptable" if the impact of the assurance deficit on the claimed risk position is considered to be negligible i.e. the “missing information” has a negligible impact on the overall confidence in the system safety argument. In such cases no additional effort to address the assurance deficit need be sought. Finally, a potentially “tolerable” assurance deficit is one whose impact is determined to be too high to be considered negligible, but which is also not necessarily considered to be intolerable. For a potentially “tolerable” assurance deficit it may be considered acceptable only if the cost of taking measures to address that assurance deficit are out of proportion to the impact of not doing so. The greater the impact of the assurance deficit is, the more system developers may be expected to spend in addressing that deficit.

Making decisions relating to the acceptability of residual assurance deficits should, where necessary, involve personnel undertaking management and assurance activities as well as those involved in assurance. If unable to form a judgement on the acceptability of an assurance deficit, then expert assistance should be sought. Further guidance is also provided in Annex B.

Note that the impact of an assurance deficit can only be determined on a case-by-case basis for a specific argument relating to a particular system. The same type of assurance deficit (such as a particular assumption) whose impact is categorised as broadly acceptable when present in the software safety argument for one system, may be considered intolerable when present in the argument for a different system. This is because the impact of an assurance deficit considers it’s impact in terms of the overall safety of the system. It is for this reason that particular argument approaches (such as the patterns discussed in section 3.1) cannot be stated as sufficient for particular claims, but must be adapted on each use to be appropriate for the particular application. This is discussed in more detail in a separate SSEI document [6], available in Annex B.

Addressing an assurance deficit requires 'buying' more information or knowledge about the system relevant to the safety claims being made. There will typically be a cost associated with obtaining this information. For those assurance deficits categorised as tolerable, the value of the information in building confidence in the safety case must be considered when deciding whether to spend that money. In theory it is possible to do a formal cost-benefit analysis based on a quantitative assessment of the costs associated with the available options for addressing the assurance deficit, and the costs associated with the potential impact on the claimed risk position (such as the necessity to provide additional system level mitigations). In many cases however, there will simply be more value to be gained through a qualitative consideration of these issues. In all cases an explicit justification should be provided as to why the residual assurance deficit is acceptable and, wherever appropriate, an argument should be used to provide this justification. The software safety argument pattern catalogue discussed in section 3.1 (provided in a separate SSEI document [6] and available in Annex B)
contains an argument pattern for constructing an argument to justify that the residual assurance deficits are appropriate.

The approach described above, although similar to ALARP, rather than considering the necessity of adopting measures to directly decrease risk, instead considers measures intended to increase the confidence that is achieved. As such the framework could be considered to help establish a claimed risk position in the software safety case that is ACARP (As Confident As Reasonably Practicable).

3.3. Conclusions

The discussions above summarise the material in [6] and Annex B (comprising the current technical guidance). The aim of this Technical SoBP is to provide guidance for the developers of software safety arguments (Ensurance / Assurance) on how to construct arguments which are compelling, and how to justify the sufficiency of those arguments. In addition, the guidance should help those involved in assessing software safety arguments (Management) to determine whether or not the arguments provided are sufficiently compelling.

It is our recommendation that the findings from this work be considered as an interim Standard of Best Practice. We recommend that these findings be assessed further and validated against industry and military experience. Validation of the guidance by application to a number of case studies is planned. Upon successful validation, the interim SoBP may be considered as best practice. Finally, it is recognised that this SoBP is unlikely to be static, as the notion of best practice evolves over time. Thus the intention is that the SoBP will be updated periodically reflecting the experience of using the document and changes in what is accepted as best practice in this area.

The material discussed above may be found in the paper, “A Systematic Approach to Software Safety Argument Construction” [6], and also in Annex B.

4. SPECIFIC SCENARIOS

The Management Guidance (Section 2) and Technical Guidance (Section 3) are intended to apply to all systems in which safety-critical or safety-related software is used. However, there are specific development scenarios in which assuring the safety of software requires some additional consideration. In this section we present an introduction to two of these situations: the use of COTS software and the use of software developed to non-MOD standards. The full results of the research into these scenarios are available in Annexes C and D.

In these two scenarios, the software in question typically has not been developed to the specific requirements of Def Stan 00-56. Consequently, the focus of both discussions is on how to assess the evidence relating to the safety of the software, and how to determine where there may be “gaps” between what is
demonstrated by this evidence and what is required by Def Stan 00-56. This approach is in accordance with the underlying principle that Def Stan 00-56 be “as civil as possible, and only as military as necessary”. In other words, compliance with Def Stan 00-56 should be dependent only upon the production of a compelling safety argument supported by evidence, not upon the particular processes and standards which were applied during development.

Pragmatically, however, there are some extra considerations when attempting to show compliance with Def Stan 00-56 when this has not been a stated aim of the development process. These extra considerations relate to possible deficiencies in the evidence demonstrating safety of the software, relative to what is needed for compliance with Def Stan 00-56. The evidence relating to the safety of COTS software, for example, may not sufficiently consider the integration of the software into the proposed environment (56-1 12). Similarly, a safety case for software developed to a non-MOD standard such as IEC 61508 need not contain a safety management plan as required by Def Stan 00-56 (56-1 9), nor need all risks be reduced to a level which is ALARP (56-1 10). The purpose of the following discussions is to identify where such “gaps” as these may commonly exist, and suggest methods of addressing these deficiencies.

### 4.1. COTS and Legacy Software

The first specific scenario we will consider in this guidance is the use of COTS software in systems intended to be compliant with Def Stan 00-56. While there are a range of perceived benefits associated with the use of COTS products (described in more detail in Annex C), there are also a number of potential safety-related issues associated with COTS software. These issues arise out of the difficulty in obtaining sufficient (both quantity and quality) evidence to adequately assure the safety of the COTS software in a given system.

One of the major differences between COTS and bespoke solutions is the potential lack of process evidence available for COTS software. In many cases design and process documentation may be considered by COTS suppliers as proprietary information; access to it will then consequently be restricted. If this is the case, it may be difficult for the acquirer to construct a compelling argument about the safety of the software, as the rigour of the development processes cannot be assessed. Def Stan 00-56 acknowledges this difficulty in assessing developmental rigour, stating that for COTS products the required rigour “may have to be provided at the evaluation stage” [2] (56-2 17.3). This emphasis on rigour during the evaluation stage highlights the importance of sufficiently detailed acceptance criteria, as discussed in Section 2.4.4 and Annex A.4.

As well as a lack of process evidence, COTS products may also suffer from a lack of sufficiently rigorous product evidence. It can be problematic for the acquirer to generate product evidence through examination of the software; intensive examination is generally not possible prior to purchase. Consequently, selection
of the COTS product must rely on publicly-available product information, supplemented by information about the supplier. The process of selecting a supplier (discussed in Section 2.4.2 and Annex A.4) is therefore of crucial importance.

Available product evidence for COTS software may take a number of forms, from certification packs to user testimonials to in-service (historical) evidence. Even if there is a large quantity of product evidence available, the quality may be such that this evidence cannot provide sole support to a compelling safety argument. One of the major reasons for this is that the circumstances under which the software was examined (by the supplier or a third party) to obtain this evidence may not be sufficiently similar to the circumstances under which the acquirer proposes to use the software. The evidence may therefore demonstrate the satisfaction of certain safety properties which will not be satisfied in the acquirer’s proposed operational environment. Another reason why existing evidence might not be sufficient to support a safety argument relates to the possibly doubtful provenance of such evidence. Because – as discussed earlier – process evidence may not be available, the integrity of the evidence chain may be unknown. Configuration consistency, for example, could therefore be an issue, especially given the frequent upgrading which is sometimes associated with COTS products. The trustworthiness of evidence such as anonymous user testimonials may also be called into question.

The material presented in Annex D fully discusses the use of COTS components, and is drawn from previous SSEI research [4]. This Annex addresses the use of COTS and legacy software from the perspective of the evidential approach. The annex discusses the points above, describing how existing evidence can be used to assess a product prior to purchase, and then combined with evidence generated by the acquirer. In this way, it may be possible to obtain sufficiently rigorous evidence to support a compelling safety argument about the safety of the COTS software in the environment in which it is to be used. One of the most important discussions in Annex D concerns a method of gap analysis for evidence; this gap analysis consists of an assessment of the evidence available for an item of COTS software to determine the extent to which it is relevant to the circumstances in which the acquirer will use the software. The deficiencies in the relevance of this existing evidence then represent assurance deficits; these deficits may be addressed by the generation of new evidence which takes into account the proposed operational environment. Finally, in this annex we describe a possible method for evaluating COTS products during an assessment and acceptance phase.

The solutions proposed in Annex D have been partially validated by case studies from both Thales and IBM, as shown in Annex D.1 This describes a more in-depth case study undertaken by IBM in order to validate the conclusions of this research. Further validation is recommended before the principles described in this annex can be considered Best Practice. However, preliminary results have confirmed that the discussions of available evidence and suitable evidence-
gathering processes are accurate and (provisionally) sufficient. As such, we recommend the findings of this research be used to inform current MOD COTS selection and evaluation processes. In particular, we recommend that the gap analysis method proposed in Annex D and validated in Annex D.1 be used in the initial phase of assessment, in order to determine the relevance of existing evidence. We also recommend that validation of existing evidence be performed as described.

In terms of the swim-lane diagram and associated decisions of Section 2, the majority of COTS assessment takes place in the Initial and Acceptance phases. Consequently, the guidance in Annex A.4 and Sections 2.4.2 and 2.4.5 are of importance when assessing the safety of COTS products as required by Def Stan 00-56.

### 4.2. Non-MOD Standards

The second specific scenario we will consider is the use of non-MOD standards to show compliance with Def Stan 00-56. This scenario is fully discussed in Annex C, containing material drawn from previous SSEI research [5]. Annex C examines two international standards, DO-178B [3] and IEC 61508 [1], and draws conclusions about the differences between those safety properties which are required by Def Stan 00-56 and those which are required by these standards. The purpose of this work is to determine to what extent compliance with Def Stan 00-56 can be achieved via other standards, and where further work would be necessary to achieve this compliance.

We reiterate the basic objectives of Def Stan 00-56 as introduced in Section 2.3.2: requirements validity, requirements satisfaction, requirements traceability and software quality. Differences between the standards examined and Def Stan 00-56 can therefore be related back to potential deficiencies in one of these four areas.

**DO-178B**

The first standard examined in Annex C is DO-178B. Similarly to Def Stan 00-56, DO-178B emphasises the need to provide evidence which demonstrates the satisfaction of identified safety requirements. However, some important aspects are omitted from DO-178B; these include hazard identification and analysis, and system safety management. Similarly, risk reduction is not explicitly discussed, nor is the ALARP principle for risk reduction required.

In addition to this, DO-178B requires independence in evidence for the Development Assurance Levels A and B, where Def Stan 00-56 requires diversity in evidence (56-1 11). Additionally, DO-178B explicitly excludes operational aspects of software, such as dynamic configuration and user-modifiable data. By
contrast, Def Stan 00-56 does not exclude these aspects, requiring as it does that the safety management process address the full life of the system (56-1 10).

These issues are discussed in more detail in Annex C, which also notes the similarities between the two standards. These similarities and differences can also, as mentioned earlier, be related back to the four major objectives of Def Stan 00-56. This process will allow developers to identify where possible further work will have to be done in order to use DO-178B to comply with Def Stan 00-56. Of these two objectives, requirements satisfaction and software quality are perhaps the ones most likely to be demonstrably met during development as described by DO-178B. Table A-6 in DO-178B considers requirements satisfaction, while Table A-8 considers software quality. A software Accomplishment Summary (AS) is required by DO-178B, and this AS should show that these objectives have been met. Exclusions and limitations detailed in the AS must also be assessed, as well as the extent to which risk reduction is demonstrated.

When considering requirements validity, however, the omission of hazard analysis and system safety management in DO-178B must be assessed. The AS should show traceability from high-level requirements to low-level requirements (Table A-3). However, because DO-178B considers hazard identification and analysis to take place prior to development, the AS may legitimately omit a description of the traceability from the system safety assessment (e.g. identification of hazards) to software safety requirements. This then represents a potential “gap” which will need to be addressed if the software is to comply with Def Stan 00-56. Likewise, the final objective – requirements traceability – may be affected by this “gap”, as well as by the omission of software safety lifecycle activities. Such activities can identify emergent hazards and aid in the elicitation of safety information, thereby providing additional traceability.

**IEC 61508**

The second standard assessed is IEC 61508. Again, the requirements of IEC 61508 are in some respects similar to those of Def Stan 00-56. However, there are three major differences relating to safety management and the presentation of a safety case. The first of these differences is that IEC 61508 does not explicitly require a safety management system. The second major difference between Def Stan 00-56 and IEC 61508 is that the latter does not specify that the safety case include a safety argument which explicitly relates evidence of safety to safety requirements. Finally, the ALARP principle is not mandated in IEC 61508.

While these differences between the two standards may initially appear significant, compliance with all requirements of IEC 61508 can effectively negate such differences. Application of 61508 in itself can be said to constitute a safety management system, while the standard's position on traceability and satisfaction of requirements provides the fundamentals for the creation of a safety argument as required for Def Stan 00-56. (However, as for DO-178B, it is important to
assess the evidence generated as a result of compliance with IEC 61508 against the requirements of Def Stan 00-56. That is, the development processes required for different SILs may not necessarily correlate with the evidence as required by Def Stan 00-56.) Finally, although not mandated, the ALARP principle is discussed in a normative index. Since ALARP assessments might be considered only at the system level, the omission of this principle does not automatically constitute a significant difference between the use of IEC 61508 and Def Stan 00-56 for software development.

**Future Work**

Despite the differences summarised above, these two standards are more similar to Def Stan 00-56 than otherwise. We recommend that these findings should be used to inform developers and management personnel about the types of work which are likely to be necessary when using other standards to comply with Def Stan 00-56. The research presented in Annex C has been partially validated by industry experience. However, both DO-178B and IEC 61508 are scheduled for revisions in the immediate future. Consequently, we recommend that the findings of this paper be validated as appropriate, and that this section of the SoBP be updated to reflect the revisions to the standards in question.
5. CONCLUSIONS

This document contains the findings from SSEI Tasks intended to establish best practice for software development in the context of Def Stan 00-56. Section 2 contains the management-focused guidance, which describes activities and decisions which influence the development of acceptably safe software. Inputs, comparator data, criteria and possible outputs are provided for each decision, and further guidance for these given in Annex A.4. Annexes A.1 – A.3 provide background information on software safety arguments as well as further guidance to supplement the swim-lane diagram of Section 2.

Section 3 contains an introduction to the technical guidance, which is fully presented in Annex B. This technical guidance introduces the concept of assurance-based argument development. It also provides a discussion on the ways in which a compelling safety argument may be constructed, and the properties which such an argument may exhibit. Section 3 contains sufficient technical detail to support the managerial guidance of Section 2, but readers are referred to Annex B for the full technical guidance.

Section 4 contains an introduction to software compliance with Def Stan 00-56 in the case of two specific scenarios: the use of COTS software and the use of software developed to non-MOD standards. We discuss the need for gap analysis to determine where the existing safety cases will need to be augmented to satisfy the requirements of Def Stan 00-56. This work is fully detailed in Annexes C and D.

The findings of this SoBP have not yet been extensively validated, and therefore should be regarded as interim only. The recommendations should not be represented or used as MOD policy at this stage. This document will be validated with input from industry, and after sufficient validation may be considered as representing best practice for software compliance with Def Stan 00-56. It is the intent that this SoBP will be regularly updated to reflect current best practice for emerging technologies and standards.
6. RECOMMENDATIONS

It is our recommendation that the findings from this work be considered as an interim Standard of Best Practice. We recommend that these findings be assessed further and validated against industry and military experience. Upon successful validation, the interim SoBP may be considered as best practice for software compliance with Def Stan 00-56.

7. ACKNOWLEDGEMENTS

We would like to thank IBM, Thales and QinetiQ for their assistance in providing case studies to validate this work. We would also like to thank the UK MOD for their support and funding for this task.

8. LIST OF REFERENCES


9. DEFINITIONS

Acceptance Phase: the phase of development in which completed software is assessed to determine whether it meets the acceptance criteria.
Assurance (activities): The activities involved in demonstrating that the software meets its safety requirements

Assurance (of a claim): the justifiable confidence in that claim.

Assurance Deficit: an uncertainty or lack of information which affects the assurance of a claim.

Containment Phase: the phase of development in which significant problems which have been identified are assessed and solutions proposed.

Development Phase: the phase of development in which software is developed, and a safety case constructed.

Ensurance (activities): The activities involved in developing the software to meet its functional and safety requirements.

Initial Phase: the phase of development prior to establishment of a contract, during which managerial pre-contractual activities occur

Requirements satisfaction: (of a safety argument) the property that the argument comprehensively demonstrates satisfaction of all the safety requirements.

Requirements traceability: (of a safety argument) the property that the argument demonstrates that all derived software safety requirements are traceable to high-level safety requirements and vice versa.

Requirements validity: (of a safety argument) the property that all safety requirements and appropriate derived safety requirements are complete and accurate

Software safety argument patterns: methods of arguing about particular claims, or “sample arguments”.

10. LIST OF ABBREVIATIONS

ALARP: As Low As Reasonably Practicable

AS: Accomplishment Summary

COTS: Commercial Off-The-Shelf Software

DSSR: Derived Software Safety Requirement

ISA: Independent Safety Auditor

SC: Safety Case
SCR: Safety Case Report
SIL: Safety Integrity Level
SMP: Safety Management Plan

11. ANNEXES

There are four annexes which serve as additional guidance for the SoBP. Annex A contains an introduction to software safety arguments, additional annotation for the swim-lane diagram, an introduction to “levers” that can be used to address project management issues, additional background and guidance on making Management decisions, and a brief summary of aspects of the Technical Guidance.

Annex B contains the Technical Guidance, expanding on the material presented in Section 3. This will be regularly updated with further work. Annex C contains a discussion of the evidential approach, focusing on the specific scenario of the use of non-MOD standards. The standards in question, DO-178B and IEC 61508, are compared and contrasted with Def Stan 00-56. Finally, Annex D contains a discussion of the evidential approach, focusing on the use of COTS and legacy software.
ANNEX A  Guidance for the SoBP

This annex contains guidance on how to interpret the Management SoBP of Section 2.

Annex A.1 contains a discussion of the basic principles of software safety arguments and how the evidence supporting these arguments affects the assurance of a safety claim. Further detail is provided in the Technical Guidance introduced in Section 3, as well as the more detailed guidance in Annex B.

Annex A.2 then provides annotation and guidance for the swim-lane diagram of Section 2. We explain the relationship between the different decisions in this diagram, as well as discussing how iteration can occur in each lane. Annex A.2 also contains detailed information on each phase (Initial, Development, Acceptance, Containment), and the activities of each swim-lane during that phase. **Annex A.2 should be treated as a necessary text for understanding and utilising the Management SoBP of Section 2.**

Annex A.3 provides a brief introduction to four major high-level concerns in the successful development of software: scope, quality, time and resources. These represent common managerial “levers” which can be used to influence the progression of software development.

Annex A.4 contains detailed guidance for making the decisions outlined in the Management SoBP of Section 2. We describe the possible sources of input for each decision, the factors which might affect it, and the information available at each stage. Annex A.4 also discusses how expectations and criteria change over time, and how these can be best represented across contractual boundaries. **Annex A.4 should be treated as a necessary text for understanding and utilising the Management SoBP of Section 2.**

Annex A.5 contains a summary of the techniques introduced in Section 3 for constructing assurance-based software safety arguments. This Annex, in conjunction with Section 3, contains sufficient information from the Technical SoBP to inform managerial decisions. Nevertheless, should more detail be required, this is contained in Annex B.

Annex A.6 is a summary of a Thales case study carried out to validate the SoBP. The information provided from this case study was used to inform subsequent drafts of the guidance, and this is indicated where appropriate.
A.1 Software Safety Arguments

In this section we briefly discuss why software safety is important, what makes it different from more general system safety and what the UK MOD requires in terms of software safety. This discussion summarises the Technical Guidance of Section 3, but emphasises managerial concerns over technical details. Where more detail is required, the reader is referred to the relevant sections of the Technical Guidance.

The applicable UK MOD standard for system safety is Defence Standard 00-56 Issue 4. Herein, a system which is safe is defined to be one for which, in a given application and a given operating environment:

1. The associated risks have been demonstrated to be reduced to a level which is ALARP and broadly acceptable or tolerable
2. The relevant prescriptive safety requirements have been met

The safety properties of interest for software will be broadly dependent upon the potential for the software to contribute to system hazards. Clearly more confidence will be required in software which is safety-critical than in software which is neither safety-critical nor safety-related. Historically this has been achieved through the definition of “levels” of safety (for example, the Development Assurance Levels of DO-178B or the Safety Integrity Levels of IEC 61508). To achieve a given safety level, certain processes throughout the software development had to be completed – for example, particular types of testing, provision of a software development plan, demonstration of sufficiency of software requirements. These safety levels are then in some cases mapped to failure rates, leading to the conclusion that following the given development processes will result in producing software which satisfies a certain target failure rate.

These are two major problems with this approach. Firstly, there is no evidence that a good process will result in a good product (although there is a correlation between bad processes and bad products!). That is, there is no indication that completion of a given set of lifecycle activities really will produce a system with a target failure rate as claimed. Secondly, while the notion of “failure rate” has some credence in terms of hardware, software failures are systematic rather than random in nature. That is, given particular inputs, the software will react in the same way every time. The failure rate of the software is then determined solely by the rate at which the software will be presented with input which causes it to fail. This rate is hard to predict, requiring significant knowledge of residual faults.

Instead, Def Stan 00-56 introduces an evidential approach to software. Briefly, this requires that any claim about the safety of the software must be supported with rigorous evidence, and with a compelling safety argument demonstrating how this evidence supports the claim. The extent and rigour of the evidence supporting the claim must be commensurate with the risk. That is, the safety
requirements for software which can contribute to a catastrophic hazard must be supported with more rigorous evidence than the safety requirements for software which can contribute only to minor hazards.

A.1.1 Problems Associated with Software Safety

There are some factors involved in software development projects which affect safety, making the delivery and management of an acceptably safe item of software a complex proposition. Firstly, unlike hardware, software is “pure design” and there is no manufacture. When developing a tangible product, delays and budget overruns in the design can in some cases be compensated for during manufacture. When developing software, by contrast, there is no phase during which design delays can be absorbed. That is, any change to the four major properties (scope, quality, time, resources) in the software design is likely to impact on the project.

Additionally, safety can be hard to manage in software development. There is often little management visibility into the technical issues which develop – errors in code are often inaccessible to management personnel and the quality of the software is hard to judge. Consequently, it can be difficult to determine whether safety is being managed correctly. One solution to this problem is to make use of domain experts or ISAs where possible, in order to obtain an independent assessment of the progression of the development.

Safety management throughout the development cycle can also be an important and somewhat problematic issue. At the time of establishing contracts for a software development project, detailed information about the safety requirements for that software may not be known. In many cases, these will only emerge later, as further work on the project provides more information about the properties which must be present in the software for it to be deemed acceptably safe. Consequently, safety management is an important issue throughout the entire project, as emerging requirements can affect any of the four considerations of scope, quality, time and resources.

A.1.2 Aspects of Software Safety

The overall aims of safety management are to ensure that safety requirements are met with sufficient confidence (assurance). There are three major factors in obtaining the required assurance, as described below. These can broadly be considered as providing the following benefits:

1. Capability Assessments – these aid in selecting a supplier who is likely to satisfy the acceptance criteria and wider managerial concerns.
2. Visibility into Safety Processes – this provides process evidence, as well as increased confidence in the assurance aspects of software development.
3. Safety Argument – a safety argument should be supported by rigorous evidence and should be sufficiently compelling to justify confidence in the safety of a system.
**Capability Assessment**

Capability assessments (such as Capability Maturity Model Integration appraisals) can provide a certain level of confidence in the capabilities and processes of the supplier. In particular, they can be used to distinguish between suppliers in terms of the maturity of their processes and their conformance with “best practice” (note that there is no guarantee that CMMI level 5 is equivalent to safety-critical software best practice!).

Capability assessments, however, cannot be used as a significant source of support for safety arguments. The reason for this is that these assessments do not provide explicit evidence that any safety properties of the system in question (e.g. “Function X satisfies a target failure rate of $10^{-3}$”) are or will be satisfied. In addition, capability assessments rarely provide much reliable information about the safety of software being developed (that is, they do not contribute significantly in terms of knowledge of assurance). Although some aspects of development processes are repeatable, they cannot be relied upon to guarantee specific properties of the end product.

However, capability assessments still have value. Like SILs, they can be used as tools to address managerial concerns such as selecting a supplier, estimating a budget, setting acceptance criteria and so on. In particular, capability assessments can act as a “forecasting” tool when selecting a supplier. In this capacity they can predict whether a compelling safety case is likely to be produced, whether sufficient visibility into the development process is likely to be provided, whether the supplier shows an understanding of UK safety management and policy, whether functional requirements are likely to be met and whether budget and quality concerns will be satisfied. It is important to understand that a satisfactory capability assessment does not guarantee that a system which is demonstrably acceptably safe will be developed, but in some cases it may indicate that this is more likely than in others.

In addition to this, capability assessments may be used to judge whether problems are “containable” as shown in the swim-lane diagram of Section 2. It is not unknown for there to be little or no visibility into the safety and development processes of software (for example, this situation may arise when purchasing COTS software from the US). In this case the Visibility of Safety Processes – described below – is lacking, even though a safety argument and associated evidence may be provided. In this case, a capability assessment can go some way towards judging whether the vendor is sufficiently trustworthy to accept their claims. Additionally, it may provide confidence that there are no major faults, such as a lack of traceability or satisfaction of requirements. Other existing policies and procedures may also be used for this, where necessary being supplemented by the capability assessments.
Visibility of Safety Processes

Sufficient visibility into the safety processes can increase confidence in the development and safety management on a project. Specifically, visibility into the safety processes can increase confidence in the competence of the personnel, as well as provide process evidence to help support a safety argument. However, visibility into safety processes does not itself constitute product evidence. That is, while visibility of safety processes can provide confidence that safety is being managed acceptably, it does not necessarily constitute sufficient evidence to support a safety argument. However, the process evidence obtained from this visibility can increase confidence in the integrity of the evidence chain, thereby supporting the trustworthiness of any product evidence which is generated.

Safety Argument

Def Stan 00-56 requires a compelling safety argument supported by rigorous evidence to be provided in order to show software is acceptably safe (56-1 9). In general, a software safety argument should be supported by both product and process evidence (respectively, evidence relating directly to safety properties of the software, and evidence relating to the sufficiency of the development processes). Sufficient visibility of safety management processes can provide both the required process evidence, as well as confidence in the trustworthiness of the product evidence. Where this visibility is not provided, capability assessments may be used instead to increase confidence in the trustworthiness of product evidence, and in the suitability of development processes.

A.1.3 Principles of Safety Arguments

The evidential approach requires that specific safety requirements be identified and supported with rigorous evidence. The purpose of the safety argument is then to convince the reader that the requirement is satisfied, by relating the evidence to the requirement. For example, a safety requirement stating that “Safety-related function X is correctly implemented” may be supported by evidence consisting of test results demonstrating the execution of Function X. In this case the safety argument would then be designed to convince the reader that these test results really do demonstrate that Function X is correctly implemented.

The software safety requirements are – in the first case – obtained from system safety requirements. That is, in order for the entire system to function safely, this software component must meet certain safety requirements. Safety requirements obtained in this way are often very high-level and abstract, requiring that the software should not contribute unacceptably to system hazards. In order to obtain these high-level software safety requirements, it is therefore necessary to examine the system and software boundaries, and the system consequences of software failure. High-level software safety requirements are refined to obtain derived safety requirements, for example that the software should not fail in a given way under given conditions, where this failure can contribute to system
hazards. The task of obtaining these derived safety requirements can only be undertaken once a certain level of knowledge of the software and its operating conditions has been obtained.

The Technical Guidance of Section 3 contains more detailed analyses of safety arguments. Nevertheless, as management personnel may be responsible for assessing the suitability of a completed safety argument, a briefer summary of important considerations is provided here.

**A.1.3.1 Safety Argument Terminology**

Safety arguments will be defined in more detail in the Technical Guidance, but in this section we will briefly summarise their main characteristics.

Within a software safety case, a *claim* is a statement made about the software which may or may not be true. For example ‘All omission failures are detected and handled acceptably’ is a claim about a system.

A *software safety argument* is a connected sequence of statements intended to establish a claim. For example, ‘All omission failures are detected and handled acceptably, because Components A and B are present in the system, and tests show that they detect and handle all possible omission failures’ is (part of) an argument.

Typically the top-level claims in a software safety argument are high-level software safety requirements obtained from system safety requirements. The safety argument is intended to demonstrate the truth of these by breaking them down into *derived software safety requirements*.

There are four high-level objectives (the last of which contains three facets which are more fully explained in the Technical Guidance) which are required in a software safety argument:

- **Requirements validity** – the argument must demonstrate that all software safety requirements are complete and accurate for the purposes of mitigating the software contribution to system-level hazards.

- **Requirements satisfaction** – the argument must comprehensively demonstrate satisfaction of all the identified software safety requirements.

- **Requirements traceability** – the argument must demonstrate that the high-level software safety requirements are traceable to system hazards, and also down through all levels of development (derived software safety requirements, design, code, evidence etc.)

- **Software quality** – the argument must demonstrate that the software (and the development processes) exhibits the basic qualities necessary to place trust in the evidence presented.
A.1.3.2 Assurance of Safety Arguments

Safety arguments are typically inductive, meaning that the supporting claims do not imply the higher-level claim with absolute certainty. That is, the premises of the argument support the conclusion, but do not entail it. The intent of a safety argument is to provide sufficient confidence in the truth of the top level safety claim. To do this, the argument may make deductive, inductive and judgemental conclusions (56-2 9.5.5). Because software safety arguments are rarely completely deductive, it is generally impossible to achieve 100% confidence in the truth of a safety claim.

The assurance of a claim is the justifiable confidence we have in the truth of that claim. The assurance of a claim is determined by whether the safety argument is sufficiently compelling, and whether it is supported with sufficiently rigorous evidence. However, determining what constitutes “sufficient confidence” in a claim can be both subtle and complex. The confidence which is required in a claim should reflect the contribution of that claim to the safety of the system. For example, more confidence is required in a piece of software which could cause a catastrophic hazard on failure than a piece of software which could cause only a minor hazard.

Determining the confidence needed in a particular safety requirement – and how to obtain that confidence – can require significant domain and technical knowledge. Further details are available in the Technical Guidance, but from a managerial perspective, it is sufficient to know that the assurance required in the safety of software should be commensurate with the potential risk posed by that software.

An assurance deficit associated with a claim is a lack of information which affects the assurance of that claim. When constructing a safety argument, the intent is to justify all assurance deficits. This requires assessing the overall effect of this assurance deficit in terms of its contribution to system risk. A form of gap analysis can be performed to determine whether the assurance deficit is intolerable, tolerable, or broadly acceptable. If an assurance deficit is categorised as tolerable, its presence must be justified by demonstrating that the cost of addressing this deficit is grossly disproportionate to the benefit gained. Section 3 and Annex B contain further details on this.

A.1.3.3 Determining Assurance of Safety Claims

Broadly speaking, the assurance of a higher-level software safety requirement is dependent upon two major factors. The first of these is the assurance of the derived software safety requirements (DSSRs) which support this. That is, if we
have little confidence in the premise of an argument, we will have little confidence in its conclusion. The second factor is the extent to which these supporting DSSRs entail the entirety of the higher-level requirement (sometimes referred to as the “inductive gap”). That is, the extent to which the premises of the argument entail the conclusion. Consequently, if there is an assurance deficit it must be due to one of two factors:

1. The supporting claims are themselves not adequately assured
2. The supporting claims do not provide sufficient information about the truth of the higher-level claims

This means that to reduce an assurance deficit at the top level we must either increase the assurance of one or more supporting claims, or change the argument structure so that the inductive gap is narrowed.

When considering the first approach, however, it is important to note that not all supporting claims are equally important to the assurance of the higher-level claim. Consequently, increasing the assurance of one supporting claim may have a larger effect than increasing the assurance of a different supporting claim. From a project management perspective, it is important to ascertain the most cost-effective way of reducing the assurance deficit, whether this be choosing the best supporting claim to assure, or altering supporting claims to narrow the inductive gap. In terms of the four “project parameters” (scope, quality, resources and timing), this amounts to addressing the deficit and minimising the impact on time and resources.

In general terms, the contribution of a supporting claim to the higher-level claim can be assessed by considering the following questions:

- How much confidence do we have in the truth of this supporting claim?
- How much information relevant to the truth of the higher-level claim is provided by this supporting claim?
- How important is this information?
- How “justified” is our confidence in the assurance of this supporting claim?
  - Is this information also supplied by a different supporting claim?
  - Is the information vulnerable to any common-cause failures?

We briefly discuss some of these questions below; further detail can be found in the Technical Guidance.

**Scope**

The scope of a supporting claim is defined as the extent of the information about the higher-level claim which is provided by that supporting claim.

The scope of a supporting claim addresses the question “what information does this supporting claim provide about the higher-level claim?” The scope is important because it tells us what information will be missing if we do not provide a particular supporting claim. Scope can also be useful for what it tells us about the inductive gap. By considering the scope of each supporting claim in turn, we
can deduce if there is some information about the higher-level claim which is not provided by any of the supporting claims. The more information which is “missing” in this way, the greater the inductive gap will be.

However, scope by itself only tells us what information is provided by the supporting claim. In order to estimate the importance of this supporting claim to the higher-level claims, we must also consider some other questions.

User-defined Preference

The user-defined preference of a supporting claim is defined as the “importance” placed by the reader on the information about the higher-level claim which is provided by that supporting claim.

The user-defined preference addresses the question of how important the reader considers the information provided by each supporting claim is to the higher-level claim. User-defined preference may be the result of personal bias, or may originate from the use of legislation and standards which identify certain safety principles as carrying more weight than others in an argument. Similarly, user-defined preference may take the form of a preference for using a particular argument strategy, or a preference for a particular evidence type (e.g. formal analysis vs. testing). For example, one assessor may prefer a safety argument which demonstrates the safety of the system by arguing that all applicable safety standards have been satisfied. Another, by comparison, may prefer a safety argument which argues over the absence of all hazards.

This guidance does not recommend or encourage user-defined preference with regards to assessing safety arguments, but recognises that, pragmatically, it may sometimes have an effect on the perceived assurance of a claim. Although safety arguments are always written to be compelling – and therefore user-defined preference plays an implicit part – it should nevertheless be emphasised that a preference for a particular strategy or evidence type does not negate the need for a logically coherent argument. That is, although a particular argument strategy may be preferred, this preference will not affect the risks associated with the software, nor the consequent assurance needed in its safety. Equally, user-defined preferences should not be permitted to emerge – and affect the safety management – throughout the project. Rather, any effect which these may have should be negotiated and included in the contract from the earliest stage of development.

Reinforcement

Reinforcement is defined as the extent to which multiple supporting claims provide compatible information about a higher-level claim.

The degree of reinforcement between supporting claims addresses the question of whether multiple supporting claims may provide compatible information (overlap) about a higher-level claim. The obvious consequence of reinforcement is
that we can omit one of these supporting claims without losing the overlapped information – although we will lose any information supplied only by the omitted supporting claim. However, we may lose some confidence in the overlap information. It is important to note, that the effect of reinforcement is not always clear-cut. Reinforcement from an independent source is likely to increase confidence more (the reason for this is that independent sources are less likely to be vulnerable to common-cause failures). The effect of reinforcement must therefore always be assessed in context.

**Combined Assurance Factors**

The above discussion constitutes guidance for Management personnel when assessing the structure of a safety argument in order to determine if it is sufficiently compelling. However, as well as assessing the structure, it is also important to assess whether the evidence is sufficient to demonstrate satisfaction of safety requirements with the necessary confidence.

There are several factors which affect the extent to which the evidence supports a safety claim in any given argument. These factors include the trustworthiness of the evidence, the extent to which it demonstrates the requisite safety property in the proposed environment and the thoroughness of the evidence-gathering process. Determining the quality of the evidence in many cases requires specialist knowledge of the different types of evidence. Consequently, we will not provide explicit guidance beyond the brief summary in this Annex. Management personnel should seek the input of domain experts in assessing the contribution of individual items of evidence to the safety argument.

In general, the acceptability of an assurance deficit for a safety claim can be assessed only by considering the significance of the missing information, and assessing the cost of providing this information relative to the decrease in risk which may eventuate from its provision. That is, to determine whether an assurance deficit is justified it is necessary to determine the risk which is associated with this assurance deficit, and the risk associated with the reduced assurance deficit consequent on providing additional evidence supporting the truth of this claim. If the cost of providing this evidence is not grossly disproportionate to this decrease in risk, then the original assurance deficit cannot be said to be justified. It is important to note that the effect of assurance deficits must be assessed relative to the risk associated with this deficit. Section 3.2.1 contains some further information on this topic.
A.2 Process Overview

In this section we provide further guidance on interpreting the swim-lane diagram. This diagram represents the key decisions which may influence safety during software development. It is not intended to represent a software lifecycle, but rather information flow throughout the contractual interaction between supplier and acquirer.
A.2.1 Initial Phase

This phase includes those activities which occur prior to the technical development of software. They include publishing an invitation to tender, managing the pre-contractual arrangements, selection of a tender and...
establishing the necessary contracts. These activities are likely to involve significant managerial input.

There are several Managerial activities which relate to the identification of a need for the development or acquisition of new software. The activities involved in creating an Invitation to Tender (ITT) are not explicitly shown in the diagram, but may include a criticality analysis of the software, requirements elicitation, scoping analyses and so on. At this stage an Invitation to Tender is then sent out and responses accepted. Management personnel are then responsible for selecting a supplier (the first decision box, labelled “This tender is acceptable?”) and requesting that work on the project commence. Annex D contains guidance on making this decision, including a discussion of what must be considered during the selection process.

The request is labelled “Request project commencement” in the swim-lane diagram. It initiates the parallel activities entitled “System build and risk management” (Ensurance), “Hazard Analysis and Risk Management” (Assurance) and “Develop assurance argument” (Assurance). It is likely that at this stage extensive software safety requirements will not be known in full. Consequently, part of the work involved in the “Develop assurance argument” activity will be to begin eliciting more detailed safety requirements.

A.2.2 Development Phase

The development phase, as can be seen from the diagram, is iterative. The main iteration consists of ongoing monitoring of safety management and safety case development. However, it should be noted that all swim-lanes will also have their own iterations within the general Development phase iteration. Some of the major considerations for management personnel at this stage include managing contractual requirements, managing change (including emerging safety requirements), formally accepting risk and overseeing safety case reports. The development phase may be curtailed when assessing a COTS product, as the role of Ensurance is likely to be limited [4].

Management Swimlane

The SoBP of Section 2, which makes use of this diagram, has a managerial focus. Consequently the main iteration of the Development phase is shown as being driven by the Management strand. The first safety-related decision for management personnel is prompted by receipt of a “Report on Safety Management” resulting from the Ensurance / Assurance activities. This should first occur relatively early in the project, as such corresponding to preliminary safety management. Further such reports may reasonably be expected throughout the project, possibly as part of interim safety case reports.

This report prompts the managerial decision “Safety Management Acceptable?”, which is analysed further in Section 2.4.2 and Annex A.4. If management
personnel confirm that safety management is proceeding acceptably so far then a request “Continue with Development” will be sent to personnel undertaking the Ensurance and Assurance activities. As described in Section 2.4.2., this request that development continue may also include further instructions about required remedial actions to address any safety management problems which have been identified. Although not explicitly shown, these instructions may be intended for Ensurance, Assurance or external developers. This “Proceed with Development and remedial action as required” request represents the first opportunity for iteration in the Development phase. This iteration reflects ongoing safety monitoring, and it is expected that the software development and safety management will then continue (and further safety management reports will be made and will reflect further progress). If the safety management is not deemed acceptable by management a transition to the Containment phase is made.

The next activity shown for managerial personnel in the Development phase is to assess the safety argument “Safety Argument Acceptable?”. As before, the first report on the safety argument should be received relatively early in the project, and will correspond to preliminary argument development. Development of the safety argument (Assurance) will be performed in parallel with risk management (Ensurance) and consequently safety case reports will be expected throughout the project, facilitated by the iteration described below. Again, these reports would typically be expected with a frequency which reflects each major project milestone.

Receipt of this report prompts the managerial decision “Safety Argument Acceptable?”, which is analysed further in Section 2.4.2 and Annex A.4. If Management confirms that safety case development is proceeding acceptably so far, and if the development is not yet complete, then the request “Continue with Development and remedial action as required” will be sent to Ensurance and Assurance. Once again, this request may also include further instructions about required remedial actions to address any problems identified so far with the development of the safety case. This request represents the second opportunity for iteration in the Development phase. Further development on the safety argument will then proceed, and will be informed by any recommended remedial action. As progress continues, further safety case reports may be expected and should reflect the further development of the safety argument. If the safety argument is not considered acceptable (Management decision “Safety Argument Acceptable?”), a transition to the Containment phase is made.

A transition to the Acceptance phase will occur once the development is complete and the safety argument considered acceptable, in as far as can be judged from safety case reports. Note that this process flow reflects that it is impossible to transition to the Acceptance phase without the presentation of a safety argument which is considered provisionally acceptable by Management. However, it is possible to enter the Acceptance phase even if no “Report on Safety Management Progress” has been made to Management. This reflects the situation in which there has been no visibility into the safety management process.
As can be seen, there are two separate points at which iteration can be initiated within the Management swim-lane. On receiving – and approving – an interim report on safety management, management personnel can request that the development activities continue (with remedial action if required). Similarly, on receiving and approving a safety case report, management personnel can similarly request that development activities continue (with remedial action if required). We emphasise that Management personnel should expect to receive reports of these two types at regular intervals throughout the project.

**Ensurance Swimlane**

The major Ensurance activity in the Development phase is “System build and risk management”, which corresponds to the development of the software (note, this activity will in reality consist of several individual activities and checkpoints, with the likelihood of iteration between these). It should be noted that this activity will be undertaken in parallel with the Assurance activities “Hazard analysis and risk management” and “Develop assurance argument”. A continuous safety analysis dialogue is expected between all three of these activities.

Personnel undertaking Ensurance activities are also responsible for delivering regular reports on safety management progress to Management (56-1 6). These reports will provide visibility into the safety processes, and will constitute reviews of the Safety Management Plan (56-1 8). If these reports are acceptable, Management will request that Ensurance development and safety management activities continue (“Proceed with development and remedial action as required”).

**Assurance Swimlane**

There are two major Assurance activities in the Development phase; these are “Hazard analysis and risk management” and “Develop assurance argument”. These will be undertaken in parallel with the Ensurance “System build and risk management” activity, and a continuous safety analysis dialogue is expected between all three of these activities. We emphasise that the development of the assurance argument and the analysis of hazards, in particular, are inter-related tasks. Although not explicitly shown on the swim-lane diagram, work on these tasks is by its nature iterative. Consequently, interim results from any of these three activities will inform and direct the progress of the others.

An important Assurance activity is to identify any unjustified assurance deficits (“Unjustified assurance deficit?”). It is important that any unjustified assurance deficits are identified as soon as possible and remedial action taken to address the problems. This can be best achieved by ensuring that personnel undertaking Assurance activities periodically assess all assurance deficits throughout the development of the safety case. If no deficits of this type are identified, the Development phase activities of Assurance and Ensurance may continue. This represents a third point of iteration in the development phase. If
unjustified assurance deficits are identified, however, this prompts a transition to
the Containment phase.

In particular, this decision ("Unjustified assurance deficit?") must be undertaken
prior to any safety case report being made to Management ("Communicate
assurance argument as required for SCR"). The process flow as shown implies
that any safety case report presented to Management as part of the development
phase should not contain any assurance deficits which are unlikely to be justified
according to the current SMP.

A.2.3 Acceptance Phase

It is important to note that the Acceptance phase can only be entered if
Management considers the software development to be complete and the safety
argument acceptable. The Acceptance phase consists of a single Management
decision as to whether the software poses an acceptable risk; Section 2.4.3 and
Annex A.4 provide further guidance on this. It should be noted that if the project is
deemed not to be acceptable, a transition to the Containment phase is
automatically made.

A.2.4 Containment Phase

The Containment phase is entered only if a significant safety-related problem is
encountered. There are four ways in which the Containment phase may be
entered:

**Unacceptable Safety Management:** In this situation, the Containment phase is
entered because a problem has been identified with the way in which safety is
being managed. This is represented by the Management decision box labelled
"Safety Management Acceptable?" and will typically indicate an issue arising from
the way in which the proposed Safety Management Plan is being implemented.
The Containment phase is entered only if there are no credible remedial actions
for the identified safety management issue. Therefore, if not addressed by the
activities in the Containment phase, this issue will result in an unacceptable
assurance deficit.

**Unacceptable Assurance Development:** This transition occurs when
Management identifies a problem with the development of the safety argument,
where this problem was not previously identified as a result of Assurance
activities. The Containment phase is entered only if there are no credible remedial
actions for the identified issue. If not addressed by the activities in the
Containment phase, this problem with the safety argument will therefore result in
an unacceptable assurance deficit.

**Unjustified Assurance Deficit:** This transition occurs when the Assurance
personnel identify a problem with the development of the safety argument. This
typically indicates that the current SMP and safety case structure are insufficient to justify an identified assurance deficits.

**Unacceptable Risk Posed:** The Containment phase may also be entered from the Acceptance phase. This will take place if the safety case does not provide sufficient assurance that the risk posed by the software is acceptable (that is, if there are unjustified assurance deficits).

**Management Swimlane**

The “first” activity undertaken in the Containment phase involves all swim-lanes, and is intended to identify methods to address the relevant assurance deficit. These methods may include performing further analysis of the software, performing remedial hazard analysis, or eliciting further information required to satisfy currently un-met safety requirements. As shown on the swim-lane diagram, external information about the system may also be sought at this point, as well as further input from additional personnel such as the ISA.

Once these possible solutions – if any are found – are identified, Management personnel must then decide whether the solutions and their associated costs are acceptable (decision box labelled “Acceptable mitigation?”). Section 4.5.1 and Annex A.4 provide further guidance for this decision. If Management deems the solutions appropriate, this is communicated to Ensurance and Assurance (“Request implementation of changes”). This prompts a transition to the development phase; development of the proposed remedial actions can now commence. Where information must be communicated to external developers (e.g. if changes in the project Hazard Log are necessary), this communication is also undertaken.

However, if the solutions and their costs are not acceptable, this prompts a transition to the Management activity “Identify possible role and environment changes to negate safety issue”. This activity is intended as a last resort to identify any possible alterations to the project which will permit the software to be accepted. That is, at this stage there is no credible way of addressing the unjustified assurance deficits which are present. As can be seen, this is again a cross-lane activity which involves Management, Assurance, Ensurance and relevant external personnel.

Management personnel must then decide whether the solutions proposed are acceptable (the Management decision “Circumstances can be changed acceptably?”, for which guidance is provided in Annex A.4). If there are no acceptable role or circumstance changes then the software must be rejected – as the development has already reached the containment phase the software is, by definition, unsatisfactory for its proposed purpose. If, however, there is a change in circumstances which would address the safety problems, then Management will request that this change be implemented. This will typically have an effect on the development and safety management. Consequently, this decision is
communicated to the Ensurance and Assurance personnel (“Request implementation of changes”). This prompts a transition back to the development phase as further development can proceed, given these changes. Where relevant, the necessary changes are also communicated to external developers and managerial personnel.

**Ensurance Swimlane**

In the Containment phase, Ensurance and Assurance activities are similar; these are “Identify methods to address assurance deficit” and “Identify possible role and environment changes to negate safety issue”. The first of these is triggered by a communication from Assurance, prompting a transition to the Containment phase. The second is triggered by a communication from Management.

**Assurance Swimlane**

Assurance personnel can initiate a transition to the Containment phase by identifying an assurance deficit which is unlikely to be justified given the current SMP. On identifying this, the Assurance personnel will inform both the Management and Ensurance strands (“Communicate assurance deficit to Management and Ensurance”).

This communication triggers the cross-lane activity “Identify methods to address assurance deficit”. This activity is designed to address the assurance deficit, usually by suggesting alterations to the safety case structure or SMP, such as further testing and analysis to be performed on the software. Once completed, any solutions identified are communicated to Management (“Report on methods to address assurance deficit”).

The other activity in the Containment phase with which Assurance may be involved is “Identify possible role and environment changes to negate safety issue”. This is triggered by a request from Management (“Dialogue about changes to circumstances”), and indicates that the only possible way in which the software can now be accepted is to alter the role or environment in which it will be used. This alteration is intended to provide circumstances under which the difficulties in adequately assuring the software will not necessarily prevent its use. For example, it may be possible to deploy the software in a less safety-critical role, or to omit functionality which cannot be demonstrated to be safe. We emphasise that this activity is intended to be undertaken in conjunction with the Ensurance and Management swim-lanes, as well as external developers.
A.3 Project Parameters

In this section we discuss the four major high-level concerns in the successful development of software: scope, quality, time and resources. We discuss each of these below, with an emphasis on military development of safety-critical or safety-related software.

**Scope**

The scope of a project is the intended purpose of the software being developed. This can be further refined to describe both the intended role of the software and its proposed environment. The scope of a project may also describe properties such as the operational modes of the software, the proposed operational lifetime and the parameters under which it is expected to operate. In some cases safety requirements may be included as part of the scope of a software development project.

The scope of defence projects can be somewhat constrained, in that they are largely determined by military capability. Because of this, it may be impractical to alter or compromise on the scope of a project once a contract has been established. Despite this, changes in scope are not uncommon. If this occurs, the interactions of the changed software will need to be assessed to determine whether these alterations will result in the development of a product which no longer satisfies the identified need. User requirements may be prioritised to facilitate trade-offs which involve a change of scope.

**Quality**

The quality of a software development project refers to the level of achievement of the completed software as assessed against certain specific factors. For military projects some common factors involved in the assessment of the quality of software include safety, security and certifiability. It should be noted that, used in this sense, quality is not necessarily restricted to reflect whether the software is “good”. Some software development projects may not constrain these factors particularly stringently, meaning that the quality requirements are lower. This does not, however, necessarily imply any inferiority of the software in terms of the number of errors present, or the complexity of the algorithms used.

Military requirements also constrain the quality of projects in several ways. Most notably, UK Defence projects are required to comply with relevant military standards (e.g. compliance with Def Stan 00-56 as regards safety) as well as applicable civil standards. This compliance is usually necessary — although, particularly with Def Stan 00-56 – a required method of compliance may not be specified. Similarly, there are also minimum levels of security, maintainability and so on which are required to be maintained. If necessary, it may be possible to compromise on the required quality of a project after it has begun. However, this
must be discussed with all personnel, and may require a role (scope) change for the software.

**Time**

The time allocated for a project is a factor which may be subject to change. Although hard deadlines are usually set as part of a contract (e.g. for completion of the project, or for certain milestones within that completion), other timing requirements may be more flexible. The timescales set for projects may be calculated with reference to other deadlines – for example, if there is a deadline for the completion of system development, each component of the system must be completed within that time also. The timing of the project stages are also linked directly to the cost, covered below.

**Resources**

The resources of a software development project include the budget (as well as subsidiary budgets for each part of the project), as well as the personnel, the tools necessary to develop and test the software, the domain experts needed to analyse problems and so on. There is a direct relationship between the time allocated for a project and the resources – all other considerations being equal, a project which takes longer is also likely to cost more.
Section A.4 Management Decision Guidance

As seen in the swim-lane diagram in Section 2.4, there are several major decisions which Management personnel must take, and for which guidance is necessary. These decisions are:

Selecting a supplier – Section 2.4.2

Approving safety management – Section 2.4.3

Approving safety case reports – Section 2.4.3

Accept at risk – Section 2.4.4.

Accepting mitigation for assurance deficits – Section 2.4.5

Accepting role or circumstance changes – Section 2.4.5

Section 2.4 considers the required inputs, criteria, comparator data and possible outcomes for each decision. In this annex, we supplement those details by providing additional guidance on making these decisions. In the following discussions we will assume that Management has access to input from safety experts (perhaps ISAs). In general, these domain experts should be distinct from the Ensurance and Assurance personnel.

In the guidance for each decision we have included a section headed “Results”. This section is intended as a pragmatic description of the information and other problems which are relevant to this decision. These sections will demonstrate how information and identified problems are typically “pushed” down the Management swim-lane. For the sake of clarity, we will refer in these sections to Information, Expectations and Criteria. Information will include anything known or learned about the supplier and the development process as a result of decisions taken so far. For example, a capability assessment will provide Information, as will visibility into the development process. Expectations refer to the expectations which Management personnel might have concerning the supplier. For example, Management might expect to be provided with a certain visibility into the development process, or might have expectations relating to the way in which safety issues should be communicated. Finally, the Criteria category includes contractual terms, and supplier / acquirer agreements which have been clearly and explicitly conveyed in official communications.

Ideally, all Expectations will be informed by Information obtained and will all be reflected as Criteria. Pragmatically, however, it is not unknown for there to be supplier / acquirer expectations which are not explicitly expressed as contractual
criteria. In some cases these expectations may be mistakenly informed by information which is no longer current, or which is not reliable. By explicitly considering these possibilities in this guidance, we illustrate some common problems with the way in which safety-related decisions are made.

A.4.1 Selecting a Supplier

Selecting a supplier is one of the most important managerial decisions. The intent of this decision is to select a supplier who will meet the functional requirements of the software, deliver acceptably safe software, keep to budget and resource constraints, deliver an acceptably compelling software argument and meet contractual requirements for all other considerations besides safety.

Section 2.4.2 contains information on some of the issues which are faced by Management personnel when making this decision. When assessing a tender, there are some major safety-related considerations to make before acceptance.

Sufficiency of Proposal

The first consideration is whether the proposal is sufficient from a safety perspective. Proposals which do not include all information requested in the ITT are unlikely to be accepted, as are proposals which provide only minimal information for each category. Tender documents should include those listed in Section 2.4.2 as essential inputs.

Credibility

Safety-related and functional claims should both be assessed for credibility, if necessary by domain experts. For example, a claim that a large item of software can be produced with zero errors may lack credibility if included in a tender with no justification. Similarly, where claims are otherwise outside the scope of accepted wisdom, the tender should justify this claim by reference to independent reviews or confirmation.

The capability of the supplier must also be assessed where possible. A supplier who is more familiar with the domain might be regarded as more likely to deliver acceptably safe, as might a supplier who has provided similar software to the MOD before. In this case, although the claims may be credible, a lack of familiarity with the domain or with 00-56 might mean that additional justification is required to support each claim about supplier capability. Where a supplier is unfamiliar to the MOD, or where they propose a novel approach to safety assurance, some form of external assessment may help provide confidence in the capability of the supplier. In some cases it is unfeasible to assess the capability of suppliers (e.g. prohibitive cost to conduct detailed assessments for all tenders, difficulty identifying sub-contractors etc). This does not imply that there are no suppliers competent to undertake the work, simply that the potential increase in confidence obtained from a capability assessment of suppliers is not provided.
Potential Issues

When considering a particular tender, a third major consideration is whether any safety-related issues (e.g. insufficient visibility into safety processes is provided) are likely to arise if this bid is selected. Omission of important tender documents, unacceptably ambiguous process descriptions and a history of contractual disputes are all indications that potential problems may arise after selection.

Two important potential issues are an unforeseen lack of visibility into the safety argument, or an unexpected lack of suitably rigorous evidence to support a safety argument. The proposed Safety Management Plan (56-1 8) included as an input for this decision should discuss both the extent to which visibility into safety processes will be provided, and the nature of the evidence which will be provided to support the safety argument. The expectation of both an adequate safety argument and the necessary visibility into development processes should be explicitly expressed in the contract, especially where the proposed SMP does not adequately address these issues. Where the project requirements are significantly beyond the supplier’s previous experience, the tender may also reasonably propose extra monitoring (i.e. additional visibility) to compensate for this.

Another potential issue is the proposed method of compliance to Def Stan 00-56. The underlying principle of 00-56 is to be “as civil as possible and only as military as necessary”. Consequently, it is theoretically possible to comply with Def Stan 00-56 via other mutually agreed-upon standards, supplemented where necessary. Where the supplier proposes this, the standard in question should be well-recognised and accepted (e.g. DO-178B or IEC 61508). Use of a proprietary standard may indicate a potential issue, especially where such a standard is domain-specific. If this solution is chosen, the tender should also indicate how the other standards will be used to show compliance with D 00-56 (as is described in Section 2, this may form part of the safety case strategy). Additionally, the tender should indicate where supplementary processes may be necessary to address the gap between the proposed standard and Def Stan 00-56.

Finally, it is important to note that this is not an exhaustive list of the potential issues which may be discerned from a tender. Other major issues include questionable integrity analysis (e.g. the assigning of a SIL which cannot be justified), the choice of software language, and the potential for the integration of a software safety argument into a broader system safety argument.

Resources and Estimations

Other important considerations include the proposed use of resources, budget and the time by which each supplier proposes to complete the project. These can have an impact on wider safety concerns, in that extra resources used in ensuring the safety of this software may mean that safety concerns in the wider system are overlooked due to a lack of money or time. The selection of a supplier must
always be justified, and the major project factors of budget and timing are important to consider in this justification.

It is also important to consider aspects besides safety when selecting a supplier. Domain experts may be able to provide input when it comes to accepting “trade-offs” between bids (e.g. where one offers a higher degree of maintainability, but another has the advantage when it comes to safety). However, it is the responsibility of Management to factor in the cost and timing of the proposed projects when selecting a bid.

A.4.1.1 Decision Background

There are three major sources of information when assessing bids. Firstly, capability assessments can be used to determine the readiness of the supplier to undertake the work and the likelihood that adequate processes will be used. Secondly, the record of the supplier in question – such as their previous involvement in software development for the MOD – may provide information about their ability to deliver software complying with Def Stan 00-56. Finally, the input of domain experts is valuable when determining whether claims are credible. The advantage of capability assessments is that they provide well-defined criteria on which to judge a wide variety of organisations. However, it is important to judge a capability assessment in context. As described earlier, the MOD definition of good practice may not be equivalent to the definition used in the capability assessment. Additionally, in order for it to have any value, the organisation and processes being assessed must be those who are to be involved in this software development. If these conditions are satisfied, capability assessments may provide important information about the ability of the supplier to perform the work requested.

In addition to capability assessments, examining the track record or previous history of a supplier can also provide valuable information. This can indicate potential safety issues by noting if any such issues arose during previous development projects. The previous history can also be useful to determine the extent to which each supplier has been involved with developing comparable software. Importantly, the previous history can be used to judge the extent to which the supplier may be expected to be familiar with the requirements of Def Stan 00-56 Issue 4.

Finally, domain experts should be used to assess technical questions. Management personnel may not have sufficient technical knowledge to identify claims which are not credible, or missing dependencies in a bid. The domain experts chosen to provide input should not be associated with any supplier being considered. Additionally, they should be sufficiently familiar with MOD requirements to be able to assess whether the proposed processes and work packages are technically appropriate.
In addition to these sources of information, direct interaction with the supplier may also be helpful both in selection and in identifying contractual criteria. This will help to determine the operational and systems boundaries of the software – and the required safety properties relevant to those boundaries – as well as to determine relevant legislation and standards.

A.4.1.2 COTS Software

When purchasing COTS, the evidential rigour may have to be provided at the evaluation stage [2]. Because of this, the selection process for COTS software may often need to be a far more involved process, and any considerations of potential issues must be explicitly detailed. To this end, we have provided additional guidance for the assessment and selection of COTS software [4] in Annex C. This guidance is intended for both technical and managerial personnel, and discusses how to select COTS software with a view to obtaining sufficient evidence to demonstrate compliance with Def Stan 00-56.

A.4.1.3 Results

The results associated with this decision should be relatively simple. The Information about the selected bid is supplied by the capability assessments, the track record of the supplier (56-1 0.3), the information contained in the bid itself, any input from domain and safety experts and finally all discussion with the supplier themselves.

Management Expectations at this stage might relate to the ways in which the supplier is expected to fulfil the contract. Typical expectations might include the frequency of safety case or safety management reports, the expected extent to which integration of this software will be considered, the methods of communication between teams and so forth. For example, deadlines for major milestones may be contractually included (as Criteria, see below), but in addition to this Management may also have an expectation of regular update reports.

Where Management Expectations include details of visibility into safety processes, these expectations should be expressed as Criteria. This is because a lack of the expected visibility can result in unjustified assurance deficits further into the project. Because of the importance of this visibility, we also recommend that the Expectations which relate to visibility originate from identifiable sources in Information. In other words, any expectation of visibility into the safety process should be justified by a capability assessment, an explicit discussion in the bid, an inclusion into the contract etc.

In terms of the Criteria, it is important to note that Def Stan 00-56 does not constrain the ways in which safety is demonstrated. Consequently, the Criteria which relate to Assurance may not be particularly detailed. For example, a contract need not include a clause specifying that formal methods be used, when the same confidence in the software could be achieved by other means.
A.4.2 Approving Safety Management

Another major Management decision is to determine whether ongoing safety management is acceptable. This decision, as part of the Development phase, is likely to occur multiple times throughout the project. Management will typically expect input from independent safety experts as well as from the Ensurance and Assurance activities to make this decision.

As discussed in Section 2.4.2, there are three situations in which safety management will be (provisionally) approved – where no change in safety management is required, where further risk management is required and where remedial action on existing safety management is required. In some cases, however, there is no possible remedial action that can be taken, and the containment phase is entered. As a general rule, this means that for safety management to be approved by management, any identified problems must be addressable within the initial project parameters (scope, quality, cost, resources).

In general, the only reason for safety management to be judged unacceptable (no remedial action can be identified) should be because important Criteria are not (likely to be) met. Unsatisfied Expectations, by contrast, should typically not prompt a transition to the Containment phase. The reason for this is that all crucial factors related to software safety are intended to be explicitly recorded as contractual criteria, or at the very least as requirements which are officially communicated to the supplier. A failure on the part of the supplier to meet Management Expectations which were not made explicit does not, in itself, constitute unacceptable safety management.

The exception to this is where an un-met Expectation is likely to lead to further problems related to the issue. For example, it is difficult to quantify the necessity for hazard analysis to proceed in a “timely manner”. Consequently, this may be a Management Expectation rather than specified Criteria. However, if safety management reports show that hazards are not being identified in a sufficiently timely manner, this could impact negatively on the project’s quality, cost and timing. Consequently, the anticipation of these related problems may be sufficient for Management to consider that safety is not being managed acceptably, and that the Containment phase should be entered before more problems accrue.

Technical Concerns

When making this decision, it is important to adequately consider all technical issues related to safety management. The following section is intended to constitute guidance on some of the input which Management may reasonably request from independent safety experts, or from Ensurance personnel in support of the SMP.
The first major concern is whether hazard identification and risk analysis is proceeding acceptably. Independent safety experts should be able to determine whether hazards are being identified correctly, and whether the contribution of this software is being assessed at the correct level. Additionally, personnel from both sides of the contractual boundary will be involved in hazard analysis, as this is a cross-lane activity. Where hazard analysis is insufficient, remedial action should be taken. However, as discussed above, in some cases significant issues with the hazard analysis may prompt a transition to the Containment phase; this should only occur where no remedial action can be identified.

When considering the sufficiency of hazard analysis, it is also important to consider the traceability of hazards to the safety requirements which mitigate these hazards. Where derived requirements cannot be traced back to any hazards, remedial action must be taken. This may include the development of detailed documentation showing how derived safety requirements were obtained. However, if the project is too far advanced, insufficient hazard analysis may prompt a transition to the Containment phase.

Further to this, Ensurance should be able to inform Management personnel whether the risks and mitigation strategies identified so far are sufficient to reduce the risk to an acceptable level, given the contribution of this software to system hazards. This can be a complex question, so it is important that independent safety experts provide input which explicitly addresses this topic. Risks which are not acceptable should be addressed with remedial action where possible.

Finally, it is important to assess the degree of visibility into safety processes as provided by the safety management reports. Insufficient visibility can lead to assurance deficits, due to the difficulty in assessing the integrity of the evidence chain. Domain experts should determine whether the visibility as provided is likely to produce evidence which can support a safety argument. If not – even if the degree of visibility was a Management Expectation only, rather than contractual Criteria – this may reasonably prompt a transition to the Containment phase.

**Management Concerns**

In addition to the technical concerns discussed above, safety management might not be approved where certain safety-related contractual Criteria are (likely to be) met. These criteria may be technical as above (e.g. that all risks should be reduced to an acceptable level), or more managerial in focus (e.g. that particular hazard identification techniques should be used, that hazard logs should be presented in an agreed format, that provision of SMP updates should occur at specified intervals).

Identifying contractual issues such as these is the responsibility of Management personnel, although input from independent safety experts should be sought as described above for technical Criteria. Similarly, discussions between Management, Ensurance and independent safety experts will help identify
possible remedial actions. In some cases the remedial action may be to renegotiate the contract, particularly where the Criteria in question are not crucial to demonstrating the safety of the software. This renegotiation should take place immediately, as it may affect the basis on which any further development work is performed.

Finally, it is the responsibility of Management personnel to engage in discussions with the supplier to guide the project in the right direction. Examples of this include the setting of Tolerability Criteria (56-1-10), the managing of information related to wider systems issues, and determining the extent to which future processes will be transparent to external examination. In some cases, information obtained from the safety management on this project will need to be shared with other developers, and vice versa (56-1-12). In this case, it is the responsibility of Management personnel to obtain this information and supply it to the relevant parties.

A.4.2.1 Results

The processes involved in making this decision will supply additional Information to that already provided at this stage. In particular, regular safety management reports will provide a significant amount of information about the (proposed) visibility of safety management processes. This may contradict previous Information (e.g. guarantees in the bid), with the consequence that some Expectations or Criteria based on the initial Information are now likely to be unfulfilled. Naturally, the Information category will also be expanded at this stage to include emerging safety information about the system. This emerging information may includes Tolerability Criteria for risks – which then sets up the Expectation that emerging risks will satisfy these criteria.

If any safety management issues identified, these may also lead to additional Expectations. In particular, where remedial action has been identified, this will give rise to the Expectation that the software development and safety management will proceed along these lines (unless the new safety management plans are written into an altered contract, in which case these are new Criteria). Additionally, there may also be the Expectation of further related problems which are a consequence of the safety management issue identified. Finally, any renegotiation of the contract, or provision of mutually agreed-upon information (e.g. Tolerability Criteria) should now be regarded as new Criteria.

A.4.3 Approving Safety Case Reports

This decision is similar to the decision to approve ongoing safety management (Annex A.4.2), but relates specifically to safety arguments. Once again, as part of the Development phase, this decision is likely to occur multiple times throughout the project as the safety argument is developed.
In general, the primary reason why Management may consider a safety argument unacceptable is due to the presence of unjustified assurance deficits. Because every safety argument contains different claims, we have used the four major objectives discussed in Section 2.3 to illustrate ways to identify unjustified assurance deficits. It is important to note that any given safety argument may not contain specific claims which correspond exactly to these objectives. However, unjustified assurance deficits in any safety claim can be traced back to their effect on one of the major objectives below.

Furthermore, it is important to note that Assurance personnel will deliver a safety case report only if they consider all assurance deficits justifiable (else a transition to the Containment phase will occur). If Management considers—in contrast to the verdict of this safety case report—that these objectives are not met given the content of the safety case report, a transition to the Containment phase will occur.

A.4.3.1 Requirements Validity

Requirements validity for a safety argument means that the safety requirements identified are sufficient to ensure safe behaviour of the software if satisfied. In the case of an incomplete argument (for example, a preliminary safety case report delivered relatively early in the project), the safety case report should provide sufficient confidence that the eventual set of requirements will be valid.

Ensurance Swimlane

There will be a requirements validity problem related to Ensurance if the safety argument and development processes omit an essential safety requirement or contain one which is incorrect. The consequence of this is that the software may be unacceptably unsafe, as the correct and necessary software requirements have not been met. This may also be identified via the safety management reports (Section A.4.2).

An Ensurance-related requirements validity problem can occur in one of two ways. The first way is where the safety requirement in question was explicitly present in the contract but has been omitted from the safety argument or relaxed without consent from the acquirer. In this case, remedial action must be taken immediately to address the problem. This action may consist of further development work to ensure the satisfaction of this safety requirement and its inclusion into the safety argument. Another type of remedial action to address this problem may be to renegotiate the contract to relax or omit the problematic requirement. This decision should only be taken after consultation with domain experts, safety experts and Ensurance personnel. If neither of these types of remedial action is possible, a transition to the Containment phase will eventuate.

The second way in which an Ensurance requirements validity problem can occur is related to derived software safety requirements (DSSRs) and emerging software safety requirements. In many projects the supplier is intended to elicit
further safety information during the development process, or to refine software safety requirements into more detailed requirements. Monitoring of safety management should therefore include reviews of the safety information which has been elicited so far, to determine whether all requirements which may reasonably have been expected to be identified at this stage of the development are adequately considered at an appropriate level of detail. Domain and safety experts will most likely be needed in order to identify “missing” safety requirements and to judge whether these should reasonably have emerged by now. Remedial action in this case may involve further hazard and safety analysis to avoid a situation where missing emergent requirements or inadequately detailed DSSRs result in unjustified assurance deficits.

Assurance Swimlane

Assurance-related requirements validity problems occur where the safety argument does not provide the reader with sufficient confidence that the set of identified requirements is sufficient and correct. The safety argument must be sufficiently compelling to convince the reader that satisfaction of the identified safety requirements will result in acceptably safe behaviour. However, the confidence with which this can be stated is necessarily dependent upon the stage of development. It is important to remember that preliminary safety cases are not expected to contain fully-detailed safety arguments; safety information will typically emerge throughout the development phase.

A.4.3.2 Requirements Satisfaction

The primary purpose of a safety argument is also to show that all safety requirements are satisfied with the required confidence. In particular, this necessitates that any assurance deficits are appropriately justified, as discussed in the Technical Guidance. Because this justification requires that assurance be assessed against cost, management personnel are required to provide significant input when assessing safety arguments.

Technical Considerations

In general, judging whether requirements satisfaction has been achieved will require input from domain and safety experts. These experts should be able to provide information to Management on a number of issues, including – but not limited to – the potential for this software to contribute to system hazards, the level of mitigation provided by the environment and the level of confidence which has been previously required by the MOD in software presenting a similar risk profile. Most importantly, safety experts should be able to provide Management with information about whether the (proposed) evidence and argument structure provides (or is likely to provide) sufficient confidence in the safety of the system. Domain and safety experts should also consider whether the safety argument justifies any assumptions which are made. These assumptions may relate to the operational conditions for the software, or to interactions with the wider system. In
this case, Management is responsible for providing the necessary information to allow the experts to make this decision.

It is important to note that “typical” safety case reports should be made only where the Assurance activities have not (yet) explicitly identified any unjustified assurance deficits which are considered unlikely to be addressed under the current SMP (this does not guarantee that such assurance deficits are not or will not be present; there may simply be insufficient information to identify them as yet). The identification by the Assurance activities of assurance deficits which are not (likely to be) justified under the current SMP should prompt immediate action in the form of a transition to the Containment phase. This is because early assessment of these deficits is important, and should not be postponed until a regularly scheduled safety case report is issued.

Remedial action for inadequate requirements satisfaction will generally involve the production of more rigorous evidence, or a more compelling argument structure. However, the receipt of a regularly scheduled safety case report indicates that the supplier does not consider remedial action necessary (i.e. the supplier considers that all identified assurance deficits are (likely to be) justified under the current SMP). If Management does not accord with this view, then a difference of opinion already exists between the supplier and acquirer. In this case, it may not be possible to identify remedial action which is mutually agreeable. A transition to the Containment phase will then occur, if no satisfactory remedial action can be identified.

**Management Considerations**

It is important to note that Def Stan 00-56 does not constrain the way in which satisfaction is demonstrated. Thus, although Management or domain experts providing input may hold certain *Expectations* relating to the methods which will be used to demonstrate satisfaction of safety requirements – e.g. via the use of formal methods – these will not always be represented explicitly as *Criteria*. Where such preferences as to the methods used to demonstrate satisfaction are *Expectations* only (not *Criteria*) they may be termed user-defined preference, as discussed in Annex A. A safety argument which provides sufficient confidence in the satisfaction of all software safety requirements should not prompt any remedial action, regardless of the methods used (note that this implies that user-defined preference, if not stated as part of contractual criteria, cannot be permitted to influence the perceived acceptability of the safety argument).

An important role for Management in determining whether requirements satisfaction has been achieved is determining whether sufficient assurance in the safety properties is (likely to be) provided. In making this decision, domain and safety experts may be required to provide input; this input may include a discussion of the information which can be obtained from each (proposed) item of evidence supporting the safety argument, the nature of any possible alternative items of evidence, and of the costs and subsequent effect on confidence
associated with the supply of these additional or alternative evidence items. Based on this information, Management will be able to make a decision about whether the (proposed) evidence and safety argument justify the presence of any residual assurance deficits. If not, remedial action must be taken. This remedial action may be to request that the supplier provide the alternative evidence profiles suggested by domain and safety experts.

One important reason why the (proposed) evidence may not provide sufficient confidence in the safety of the software is due to insufficient visibility into the safety processes. A lack of visibility will reduce confidence in the integrity of the evidence chain. At very early stages of the development, capability assessments may be of use to determine whether the processes used are likely to produce sufficiently rigorous evidence. However, the utility of capability assessments is doubtful once the preliminary stage of development is over. Remedial action for a lack of visibility may include a negotiation for increased development transparency.

Finally, Management are additionally responsible for ensuring that Assurance personnel and domain experts are provided with sufficient information. Where the argument relies on assumptions made about the operating environment, Management must be able to provide sufficient information to determine whether these assumptions are justified. Additionally, any identified hazards which affect the wider system should be communicated to external developers as required.

A.4.3.3 Requirements Traceability

Requirements traceability is the objective concerned with showing that high-level requirements can be explicitly traced down to low-level requirements (and vice versa). Traceability also requires that all software safety requirements should be traceable to the hazards they mitigate.

Causes of Traceability Issues

One reason for a lack of traceability from high-level (design) safety requirements to low-level requirements is a lack of transparency in code. Software is not always developed in a stratified manner, so the evolution of a software requirement throughout many modules may result in a number of subsequent related requirements which are eventually implemented into the code. In this case a small number of high-level requirements are decomposed into a large number of lower-level requirements, all of which depend on the presence of the others. Other reasons for a lack of traceability include limited visibility of code and design documents – a situation which may arise when making use of COTS products – as well as a lack of adequate care when preserving the integrity of the evidence chain.

As a general rule, more complex systems are more likely to suffer from a lack of traceability. Where traceability between high-level and derived requirements is lost
after a certain point in the development due to the complexity of the software, it may in some cases be possible to recover this, either via audits or further examination of the software. Where there is sufficient visibility and sufficient confidence in the safety processes, additional testing and analysis (also remedial actions) at all levels of design and code may be sufficient to recover traceability to a – possibly limited – degree. If satisfaction of the high-level requirements can be demonstrated via this testing and analysis, or via audits, then the assurance deficit caused by inadequate traceability may be acceptable.

**Domain Experts**

In this situation, it may be necessary for domain and safety experts to attempt to re-establish traceability as described above. To do this they must consider how to demonstrate satisfaction of the high-level requirements, given the lack of traceability, and how to show that these high-level requirements are translated into design and code. Many high-level requirements cannot be directly shown via testing, such as a claim that “The software mitigates all hazards”. In this case, it is important to re-establish traceability, as without this it is impossible to gauge the extent to which the software will operate safely. If this cannot be achieved, a transition to the Containment phase will occur.

**Management**

Management personnel are required to approve any remedial action relating to a loss of traceability. One important point to note is that contractual Criteria are likely to include issues relating to traceability (56-1 10). Consequently, if full traceability cannot be recovered, this may preclude acceptance of the software. Where it is possible to recover an acceptable level of traceability, the contract may be negotiated to recognise that the assurance deficit associated with the lack of traceability is acceptable.

Furthermore, a lack of traceability may lead to related problems, especially in the area of maintainability. If traceability from high-level requirements to derived requirements is lost, then it may be harder in future to determine the effects of an upgrade to the software. This represents an Expectation of future related issues, which may be sufficient for Management to decide to transition to the Containment phase.

**A.4.3.4 Software Quality**

Software quality refers to the requirement that the safety argument adequately address issues of interference by other components, issues of configuration consistency and issues related to the basic integrity of this software. To assess these, Management must rely to a certain extent on input from domain and safety experts.
Basic Integrity

Basic integrity refers to the presence of particular types of bugs in the code (e.g. buffer-overflow errors, divide-by-zero errors, type errors), and consequently can often be shown by testing or static analysis. This can be supported by using well-specified coding standards on the project. Where the use of a particular coding language, technique or standard is claimed to increase basic integrity, this should be justified. In addition to this, all testing performed to demonstrate basic integrity should be documented. In each case, the contribution to basic integrity of the tests performed should be justified, as should any assumptions made during the tests. In general, Management will rely on input from domain and safety experts to determine the extent to which basic integrity is demonstrated by the safety argument.

Configuration Consistency

Configuration consistency requires that the evidence used to support the safety argument is obtained from the version of the software which is being provided, and is not compromised in any way. Typically, process evidence is required to demonstrate this, and may in some cases not be available (e.g. for COTS products). In the absence of adequate process evidence (caused by a lack of visibility into the safety processes), confidence in this property may be augmented by reference to official documentation relating to the record management and version control structures which were used on the project.

Non-Interference

Non-interference requires that this software should not be vulnerable to interference by other components (e.g. data overwrites, scheduling conflicts), nor should it interfere with the rest of the system. In some cases the supplier will not have sufficient knowledge of the rest of the system to demonstrate this. Consequently, Management are responsible for ensuring that any information “gaps” are made explicit in the SMP (56-1 12). These should then be addressed in consultation with the supplier.

A.4.3.5 Results

As for Annex A.4.2, the processes involved in making this decision will supply additional Information to that which has already been provided by this stage. Safety case reports will supply information on the proposed safety case structure, as well as the nature of the evidence which is intended to support the arguments. This information may contradict previous Expectations (for example, an expectation or user-defined preference for a particular evidence type) or Criteria which were based on the initial Information.

Additional Expectations are likely to relate to remedial actions which have been requested to address problems. As before, there may now be additional
**Expectations** that the safety case structure will develop along these lines (unless the remedial actions have been explicitly written into the contract as **Criteria**). Management may also have additional **Expectations** relating to future problems which could arise. These will be informed by the additional **Information** relating to the extent to which supplier and acquirer have so far been willing to compromise. Management personnel are also likely to now have gained increased **Information** about the effects of safety problems on the wider system. This may again create **Expectations** of future related problems, especially where safety problems relate to interfaces (56-1 12).

### A.4.4 Accept At Risk

This decision requires Management to determine whether the completed software poses an acceptable risk. In more detail, this requires an assessment of the safety case to determine whether it provides sufficient assurance that the risk posed by the software is acceptable. It is important to note that at this stage, the **Expectations** are no longer relevant – Management should instead be concerned only that those **Criteria** which relate to acceptance are satisfied. Ill-defined or inconsistent acceptance criteria can make this decision more difficult.

It will be necessary to judge the completed safety argument, and consequently the guidance in Section A.4.3 also applies here. In particular, we emphasise that to be accepted, the software must be shown (Assurance) to be acceptably safe (Ensurance). An important consideration here is whether all identified safety risks are reduced to an acceptable level (given the contribution of this software to system hazards), and whether this is shown with sufficient confidence. It is important to note that ALARP arguments may be made at the system level only, and will therefore depend upon the risk of software failing in a given way under given circumstances, where this can contribute to a system-level hazard. The Acceptance phase should ideally introduce very few unknown problems. That is, the safety management process should have identified any major issues during development.

As well as input from domain experts, Management will be required to consider some wider issues. Firstly, all legal requirements affecting this software development should be satisfied. Secondly, all criteria should be met as stated in the contract if the software is to be accepted. Finally, the final safety case should be in a form which will permit its successful integration into the system safety case. Although this guidance relates specifically to safety, acceptance of software is also dependent upon adequate demonstration of all other contractual requirements.

#### A.4.4.1 Results

At this stage very little new **Information** should become available. The extent to which development processes are visible should have been identified earlier as a result of safety management reports (note that if none of these are produced, the
visibility is effectively zero). Similarly, safety case reports should already have conveyed information about the types of evidence which will be used to support the safety argument. COTS products may be an exception to this, where additional Information in the form of documentation about upgrades and support might become available at this stage.

As discussed earlier, Expectations should not be taken into account when making this decision. As this phase is entered only upon completion of a project there can be no Expectations of further problems (although they may indicate “lessons learnt” for future projects). Similarly, by this stage of the project any expectations about the way in which the supplier will fulfil the contract should have been translated into Criteria or have been accepted as preferences, but not necessities.

A.4.5 Mitigation for Assurance Deficits

This decision is part of the Containment phase, and consequently may not be encountered in all projects. This decision is made when an unjustified assurance deficit is identified as a result of Assurance activities, and furthermore where Assurance personnel consider that the current SMP will not allow this deficit to be justified without significant alteration. In this situation, Ensurance and Assurance will jointly propose mitigation strategies, with input as required from Management and external personnel.

Where a potential assurance deficit is identified early – that is, where it is identified before it eventuates – mitigation strategies are easier to implement. These will generally take the form of proposing alternative safety management techniques, to address the area in which the deficit is likely to occur. Although such changes will typically require alterations to the SMP and to the safety case structure, they do not need to be applied retroactively – that is, development up to this point can be considered satisfactory. Rather, these mitigation strategies must be implemented as the “next development steps” if the deficit is not to eventuate.

It can be more complicated to identify mitigation strategies for an unjustified assurance deficit which has already eventuated. The following table describes mitigation strategies for assurance deficits in each of the four argument elements (validity, satisfaction, traceability and software quality). Because visibility into the development and safety processes is an important aspect of assurance, we also include mitigation strategies to address a lack of visibility.

<table>
<thead>
<tr>
<th>Deficiency</th>
<th>Mitigation Strategy</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validity</td>
<td>Repetition of parts of the safety process and requirements process in order to re-establish safety requirements</td>
<td>Likely always to leave residual doubts (assurance deficits)</td>
</tr>
</tbody>
</table>
Satisfaction

Undertake focused testing or analysis to reduce the deficiency

With COTS and legacy software likely always to leave residual doubts (assurance deficits)

Traceability

Examination of records from the development process to see if it is possible to reconstruct traceability

Often very difficult and costly to resolve; consideration should be give to restarting the process as an option

Quality

Examination of records from the development process to see if it is possible to remedy the deficiencies

Often very difficult and costly to resolve; consideration should be give to restarting the process as an option

Process

Examination of records from the development process to see if it is possible to construct an adequate process argument

Often very difficult and costly to resolve; consideration should be give to restarting the process as an option

It should be noted that it is unlikely that these mitigation strategies will completely eliminate the assurance deficit. In considering whether these solutions are acceptable, Management should obtain input from domain and safety experts on technical issues, including an estimate of the extent to which each strategy will reduce the assurance deficit. Additionally, the technical risks associated with each of these strategies should be assessed. Similarly, the strategies above should also be assessed to determine whether they will be sufficient to address any global (system) effects of this deficit. Should any of these mitigation strategies be accepted, the SMP and safety case structure will need to be reassessed to allow for process steps to be repeated in a manner that is informed by the selected mitigation strategy.

**Management Considerations**

As well as input from domain and safety experts, there are some Management concerns when considering whether the proposed mitigation strategies are acceptable. Firstly, the mitigation strategies must be acceptable in terms of the original project parameters of scope, quality, resources and timing. This can be a complex calculation, as in some cases the short-term cost of these mitigation strategies may be sizeable. However, it is important to consider how the cost may be amortized over the entire development of the system which will include this software. To assess this, Management will need to consult with external developers and project managers.

Secondly, there may be project risks associated with the adoption of these mitigation strategies. While the adopted mitigation strategy should not affect the
overall timing for the project, it is possible that internal deadlines will be missed as a result of the associated process iteration. This may have an impact on the deadlines and deliverable dates for the rest of the system. As before, Management should consult with external developers and project managers to ensure that this effect is acceptable.

A.4.5.1  Example

The purpose of this example is to illustrate the types of circumstances which might lead to a transition to the Containment phase. This example is based on a real scenario, but with identifying details removed or modified.

Scenario

The MOD is purchasing a twin-engined transport aircraft, which is a variant of a civil aircraft. To meet the new role the engines are being replaced with engines of much higher power. It is planned to conduct flight tests of the new engines by replacing one engine of the standard commercial aircraft with the new engine. The original aircraft is ETOPS rated so is certified to fly on a single engine. Thus, although in flight shut down (IFSD) is undesirable – as the aircraft will have to fly on one engine – it is only classified as a minor hazard.

However due to the higher power rating of the new engine there is an aircraft hazard (thrust asymmetry) if the new engine runs at maximum power, and power cannot be brought down to the same level as the other engine. Thus engine overspeed is classified as a catastrophic hazard in the safety assessment for the flight trials. This hazard is an issue in all phases of flight including cruise, as the engines would normally be operating at low thrust. However, it is it is of perhaps the most concern during take-off climb, due to the limited time available to control the problem.

The engine is managed by a Full Authority Digital Engine Controller (FADEC) with software control. The software is developed by the FADEC manufacturer and the development plans are based on adherence to DO-178B.

Inputs

The IPT receive the following inputs:

Safety case report including:

- Outline safety case report for the engine and the FADEC
- Results of ground engine testing to support (parts of the safety case)
- A letter from the prime saying that the software supplier has lost traceability in the software development process. The letter proposes that this may not be a problem as low-level software testing has been carried out (note that
the loss of traceability gives rise to an assurance deficit) as stated in the original SMP.

Comparator data:

- Examples of DO-178B Software Accomplishment Summary (SAS) and other guidance, e.g. this SoBP
- There is no recommendation on resolving the assurance deficit, as the original SMP does not consider loss of traceability, and the prime asserts that low-level testing is sufficient.

The letter from the prime is received very shortly before the scheduled first test flight, and the IPT is concerned both about the impact of the assurance deficit on the aircraft programme and the safety of the test flight. Note that because the prime has identified this assurance deficit which will not be addressed (the original SMP does not cover regaining traceability), the Containment phase is entered.

**Satisfaction of Criteria**

The situation can be summarised against the criteria set out in section 2.4.5 of the main body of the SoBP:

- It should address all four argument elements (validity, satisfaction, traceability and quality) with respect to all safety requirements – failed, traceability lost
- It should be sufficient to provide adequate assurance with respect to all safety requirements, or indicate how this assurance will be obtained – failed, the proposal that low level testing is sufficient is not valid, as this does not show that high level safety requirements have been met
- It should identify and justify all assurance deficits – failed (there is no acceptable justification proposed in the prime’s letter for this assurance deficit)
- All assumptions should be identified and justified, with references to supporting documentation where relevant – not applicable
- Evidence of a search for counter-evidence should be presented, and the effect of relevant counter-evidence upon the argument should be assessed – not applicable

The evidence provided to support the safety argument should satisfy the following:

- The evidence should adequately support the relevant safety requirements – failed, the low level testing is not sufficient to show that high level safety requirements are met
- The integrity of the evidence chain should be evident, meaning that sufficient visibility into evidence-gathering procedures is provided – failed – if DO178B had been followed rigorously this would be no problem, but the loss of traceability undermines the argument
The trustworthiness and applicability of the evidence should be justified and it should be sufficiently diverse (56-1 11) – failed – if DO178B had been followed rigorously this would be no problem, but the loss of traceability undermines the argument.

The assurance deficit report should provide the following information:

- An assessment of the local and system effects of this deficit
- Identification where possible of techniques to address this deficit, with consideration of how these may fit into the safety management plan
- A comparison of these techniques to demonstrate how they will provide additional assurance.

The assurance deficit report was not provided, so the IPT asked for an expert assessment of what can be done both for the purpose of the test flights and for the programme in the longer term. This activity — “Identify methods for addressing assurance deficit” — involves both the Ensurance and Assurance strands, as shown in the swim-lane diagram. However, in this situation the methods to address the deficit are wider ranging and include some aircraft level activity. Consequently, external personnel are involved.

**Investigation**

A brief discussion with the Assurance personnel confirms that there is no credible and quick way of regaining traceability. Documents have been changed outside configuration control, and attempts to reconstruct the traceability will produce results which cannot be fully trusted. Consequently, the only way to address this deficit is via the use of system-level solutions for the test flight.

As the aircraft is being used for test flights there in an additional flight engineer station in the back of the cockpit (normally the aircraft would be operated with just a two-man crew). The flight engineer has a panel which provides extensive engine data including commanded thrust level, actual thrust level, and key parameters, e.g. turbine gas temperature. The engineer also has a fuel shut off valve which enables them to stop the engine by starving it of fuel. The engine overspeed hazard is discussed with the engine performance engineers and the test pilots. It is agreed that it will take 5 seconds from start of an overspeed excursion for the engine to reach an unsafe power level (this is confirmed to be in take-off climb).

One concern is whether or not the flight engineer can detect the situation fast enough to control the hazards; a combination of simulation and ground test runs is carried out and it is confirmed that the situation can be managed successfully (typically within 3 seconds). Another concern is whether or not the engine monitoring and control is separate from the FADEC. An engineering analysis confirms that this is the case. The flight engineer’s panel is fed from special purpose instrumentation which uses the normal engine sensors, but via separate signal processing electronics and software. The fuel shut off involves a high pressure shut off valve (HPSOV) in the fuel system.
A brief safety case report is produced showing the credibility of managing the hazard, should it arise. The essence of the argument is:

- There is a lack of evidence about the possibility of software causing the engine overspeed hazard (but no evidence from tests that the software does cause the hazard);
- The flight engineer is continuously monitoring the engine conditions and can detect the problem and shut the engine down before the overspeed situation becomes uncontrolled;
- Engine condition monitoring is independent of the FADEC;
- Engine shutdown is independent of the FADEC;
- Procedures have been put in place to ensure that there is proper communication between the flight engineer and pilots to ensure that any power excursions are managed cooperatively, e.g. to avoid a “surprise” to the pilots if the test engine is shut off;
- The aircraft can safely be flown on a single engine in any phase of flight;
- The reliability of the standard engine is such that the overall risk is acceptably low for the duration of the test flight.

This safety case report is presented to the IPT.

Outputs

The IPT receive the safety case report and decide to proceed on its recommendations.

The four possibilities identified in section 2.4.5 are as follows; note that one applies to the current situation (flight testing) and one applies to the longer term programme:

- Proceed with no change – *not applicable*
- Proceed with further risk management. For this decision, this outcome is applicable in two cases. Where the identified assurance deficit can possibly be remedied with further software safety management (there are no scheduled future assurance tasks which could address this deficit, but some may be added), the identified remedial actions should inform the future development of the safety argument. Where the deficit cannot be remedied (project constraints mean that it is not feasible to add further assurance tasks to address this deficit) development may proceed provided that system-level risk management techniques are identified to justify this deficit - *(The second case applies to this scenario, in that the flight trials will take place as identified in the safety*
case report. However, system-level risk management techniques (addition of human factors to mitigate risk) will be implemented.

- Iterate (repeat) process steps, with remedial action. For this decision, this outcome reflects that there is an unjustified assurance deficit. Alternative verification processes must be undertaken, as this assurance deficit could render nugatory all further development activities – for the main programme, the IPT require a reworking of the software so that the process is fully compliant with DO-178B; whilst the IPT do not wish to insist on a complete new development (that is an issue for the suppliers) they make clear to the prime that this option needs to be considered.

- Terminate the process. This outcome reflects that there is no identified strategy to reduce this assurance deficit, and the next step is to consider a possible change to the circumstances and environment of this software – not selected, but this would have been the next option if the mitigation for the flight trials could not have been established.

This decision has not resolved the underlying software engineering problem but it has found a pragmatic solution to the immediate problem which also “buys time” for the root cause of the problem with the software to be addressed. Whilst a specific example has been given this is not atypical of the sort of system-level work-around which might prove necessary.

A.4.5.2 Results

For this and the following section (A.4.6) we do not provide guidance on the results (Information, Expectations and Criteria) which may be obtained. This is because both this and the following decision occur in the Containment phase. This phase represents “abnormal development” and may not eventuate in all software development projects. As such, there are no typical or preferred results from this phase.

A.4.6 Accepting Role or Circumstance Changes

This decision is part of the Containment phase, and is in some sense the “last resort” to address safety problems before the development has to be abandoned. This decision is intended to assess methods to negate one of the following situations: unacceptable safety management, unsatisfied acceptance criteria, and unjustified assurance deficits. It is important to note that the proposed methods for negating these situations are not intended to “fix the problem” (at this stage, all potential remedial options have been rejected), but rather to place the software in a context in which these problems are no longer as significant. This change of context can typically be done by permitting a change to the original project parameters of scope, quality resources or timing beyond the flexibility which was initially agreed.
Because this decision is dependent on many project-specific factors, we will not give general guidance on how to assess role and circumstance changes to determine if they are acceptable. Instead, we present some examples which illustrate how these changes can work to negate safety-related problems. We emphasise that any proposed role or circumstance changes must be assessed in context, with input from all stakeholders. Depending on the circumstances under which this decision was reached, there may also be no possible role or circumstance changes which are acceptable. In this case, the software development will terminate.

**Project Parameters**

Although it is not practicable to provide general guidance for this decision, we describe here the four major management “levers” (these being the project parameters) which are available to address the problems. Annex A.3 contains more detail on these.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Reduction in the scope of supply (removing requirements) by changing the intended use of the system may mean that the current assurance is sufficient to permit deployment.</td>
</tr>
<tr>
<td>Time</td>
<td>Delaying the in service date (ISD) for the system may give time for remedial work to be undertaken to obtain sufficient assurance.</td>
</tr>
<tr>
<td>Resource</td>
<td>Deploying additional resources, at additional cost, may enable remedial work to be undertaken to obtain sufficient assurance.</td>
</tr>
<tr>
<td>Quality</td>
<td>A change in the operational mode of usage of the system so that the evidence now available is sufficient (or the assurance deficit can be bridged cost-effectively).</td>
</tr>
</tbody>
</table>

The time and resource “levers” are likely to be used together and will usually require additional funds to be made available. Changes in scope and in quality have an operational impact, with the underlying question being whether or not the system would be acceptable in a different role. The following examples show how changes to the quality, scope and resources / time can address safety-related problems.

An unjustified assurance deficit may be addressed by using “quality” as a lever; it may be possible to use the software in a less safety-critical role. For example, it might be possible to provide a back-up for the functionality provided by this software (the backup may be hardware, software or cross-platform including human mitigation) in the proposed system. The back-up functionality provides an extra degree of protection, meaning that both this software and the back-up will
need to fail in order to cause a system hazard. Because of this extra degree of protection, the criticality of this software may be lessened, and thus the assurance deficit may now be acceptable. Again, we emphasise that in order to determine whether this strategy is acceptable, it is important to assess the software and wider system in context.

Another potential role / circumstance change is to remove functionality which cannot be adequately assured (thereby using “scope” as a lever). For example, where an incorrect result from a function will have more serious consequences than a missing result, and where the ability of the software to correctly perform that function cannot be assured, removal of the function entirely may be a suitable option. This solution can only be used where the assurance deficit relates to a well-defined and isolated part of the software. It is important to remember, as well, that the removal of this functionality may present its own problems in terms of assurance, and the need to show that this action has not affected the remaining functionality. Again, it is important to assess this in context, as the removal of functionality is likely to have an impact upon the wider system.

A third potential change relates to the cost and time taken to develop the software and assure its safety. In some circumstances, given extra resources and time, it may be possible to reduce an assurance deficit until the software can be shown to be acceptably safe. One possible combination of “levers” relates to scope, cost and time. It may be possible to justify spending more on the software if its role is actually extended (for example, if it performs monitoring functions for other components, as well as its initially proposed functionality). Because of the extended role, it may be possible to allocate additional budget to the software development which will allow – as well as the implementation of this extended functionality – further safety management to address the unjustified assurance deficit. The budget allocation will have to be examined carefully, however, to ensure that the extra budget will allow for remedial safety management actions as well as further development.
# Section A.5  Table for Assurance Based Argument Construction

This table provides a summary of the techniques for assurance based argument construction. Further guidance on the use of these techniques is provided in Annex B.

<table>
<thead>
<tr>
<th>Step</th>
<th>Purpose</th>
<th>Assurance impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Identify goals to be supported</td>
<td>To clearly and unambiguously state the goals to be supported.</td>
<td><strong>More</strong> - If in stating the goal, an attempt is made to claim more than it is actually possible to support with the available evidence, then the assurance that can be achieved in that goal will inevitably be low.</td>
</tr>
<tr>
<td>2. Define basis on which goals are stated</td>
<td>To clarify the scope of the claim, to provide definitions of terms used, to interpret the meaning of concepts.</td>
<td><strong>None</strong> - Any claim is only true or false over a particular scope. If the scope of the claim is unclear, due to lack of context, then the level of truth or falsity of the claim becomes more difficult to determine. This increases the uncertainty associated with the assurance in that claim, and therefore makes it more difficult to determine the assurance.</td>
</tr>
</tbody>
</table>
### 3. Identify strategy to support goals

| To identify how a goal can be adequately supported. | **More, Less, Other Than** - This step of the safety argument process is the most crucial for the assurance achieved since it is at this step that the decisions are made about which strategy should be adopted to support each claim. Assurance is lost at this step if the proposed strategy does not provide sufficient support to the goal. This could happen for two reasons.  
- Firstly the inductive gap may be too large. If this is the case, then even if the premises are believed, it doesn't provide sufficient confidence in the truth of the conclusion.  
- Secondly the fundamental beliefs upon which the strategy is based may be open to question. In such a case the premises may not provide confidence in the conclusion. |

### 4. Define basis on which strategy is stated

| To identify any assumptions upon which the sufficiency of the strategy depends. | **No, Less** - It is inevitable that some assumptions will be made during the development of any safety argument, however these assumptions may not always be explicitly captured. Any assumptions that are left implicit introduce uncertainty, and reduce assurance.  
| **More** - All assumptions are, by definition, unsupported. The argument holds only on the basis that the assumptions are true. If there is a lack of confidence in the truth of the assumptions, then this will also result in a lack of confidence in the truth of the claim. It is therefore recommended, for any assumptions that may be open to any significant doubt, that an argument is presented, rather than an assumption.  
| **Other Than** - Assumptions may be stated which are not actually true. Any false assumptions undermine the whole basis upon which the argument is made.  

| To provide justification for why a particular strategy is being used. | **No, Less** - No justification is provided as to why the adopted strategy is sufficient. This can result in a loss of assurance, since there may be a lack of confidence in the sufficiency of that strategy. It is important, if it's likely that the justification may be unclear, not to leave it implicit, but to explicitly record the justification in the argument.  
| **More** - Although not leading to a loss of assurance, it is important to note that providing an argument to justify the strategy chosen in each decomposition in the argument is not necessary. For many strategies, the justification will be obvious to the reader and may be left implicit.  
|
5. Elaborate strategy

<table>
<thead>
<tr>
<th>Specify the goals that implement the chosen strategy.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Less, As Well As, Part Of</strong> - The strategy that is actually implemented does not fully and accurately reflect the one that was chosen. Assurance may be lost at this step, even though a chosen strategy may be considered acceptable.</td>
</tr>
</tbody>
</table>

6. Identify basic solution

<table>
<thead>
<tr>
<th>Identify the solutions which provide adequate support to the goal.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Less</strong> - The solution provides less confidence in the goal being supported than is required. Assurance is lost at this step if it is unclear why the evidence gives confidence in the goal being supported. It may be unclear because:</td>
</tr>
<tr>
<td>1. there may be an inductive gap between the claim and the evidence (the nature of the evidence does not provide a compelling reason to believe the claim is true)</td>
</tr>
<tr>
<td>2. there is uncertainty about the trustworthiness of the evidence itself. Note that evidence which is untrustworthy will undermine assurance even in the situation where there is a deductive relationship between the claim and the evidence.</td>
</tr>
<tr>
<td><strong>Other than</strong> - Counter evidence is any evidence which undermines the confidence in the claim being made. The presence of counterevidence does not necessarily mean that the argument is inadequate. It simply means that the confidence in the claim may now be lower than it was before the counter evidence was identified. It is necessary to determine the impact of the counter evidence on the claim’s assurance. In many cases it may still be possible to make a sufficiently compelling argument despite the identification of counter evidence, particularly where there are mitigations which limit the uncertainty caused by the counter evidence. If counter-evidence isn’t correctly identified then the potential effect on the assurance of the argument cannot be determined. It is important to justify that there is sufficient confidence that relevant counter evidence has been identified.</td>
</tr>
</tbody>
</table>
A.6 Thales Validation of SoBP

The SSEI is currently undertaking a task to provide guidance for designing and implementing software for use in systems complying with Defence Standard 00-56 Issue 4. One of the major principles of this standard is an emphasis on the evidential approach to safety. This approach requires the provision of a compelling safety argument, supported by rigorous evidence, to justify the inclusion of all software.

Thales have provided a case study (the CIPHER project) to assist in validation of the Initial stage of software development (as shown in the swim-lane diagram of Section 2). The relevant decisions which the Thales personnel were asked to consider include selecting a supplier, and setting acceptance criteria. In addition, where possible, they have provided information on the preliminary activities involved with structuring a safety case to meet the requirements of Def Stan 00-56.

A.6.1 CIPHER Introduction

The CIPHER project, run by a consortium of companies led by Thales UK, CIPHER is intended to deliver the future MOD crypto capability and sustain current capabilities until the out of service date (OSD) of current devices is met or a replacement capability is available. CIPHER brings together a number of existing projects within the Defence Cryptographic Authority DCA and looks to manage them to produce a coherent capability for future crypto equipment and support.

The Interoperable Electronic Key Distribution (IEKD) project and the Future Crypto Programme (FCP) separately passed Initial Gate (IG) in August 2007, but have been combined within CIPHER due to their interconnected nature, the synergies in taking a single capability approach and the possibility that future crypto and its management could be provided under common service arrangements. In addition, the CIPHER project is intended to address the requirements of the Security Management Infrastructure (SMI) project.

Currently, the Thales consortium is in the assessment phase for the CIPHER project. Down-selection and Main Gate approval for CIPHER are planned for 31 December 2010 with an Initial Operational Capability by 2012 and a Full Operational Capability by 2015.

A.6.2 Suppliers and Standards

The CIPHER project extends across land, sea and air domains, and consequently there are in excess of 60 platform types which require crypto capability. The intention is to rationalise the present number of crypto devices, but because there are domain specific issues, it is highly likely that a large number of sub-contractors
will be required to both provide the necessary capability in terms of equipment, bearer infrastructure and associated expertise. The extent of the software requirement will depend upon the amount of COTS is employed, the extent to which it needs to be modified and the degree of functionality and performance is required. At this stage of the project, all of these facets are under consideration, so the extent to which new software is required and its degree of integrity have yet to be determined. However, it is not unreasonable to expect that some software will be required to enable functionality of specific crypto devices, the management of crypto keys and the overall control and administration of CIPHER. The system integrity requirements will depend upon meeting both safety and security requirements.

The primary safety standard applicable on the CIPHER project is Def Stan 00-56 Issue 4, which will be applied to sub-contractors. However, it is expected that other safety standards will have to be assessed for their ability to comply with Def Stan 00-56, particularly where COTS is procured from overseas. The Safety and Environmental Plan contains a complete list of anticipated standards to cover safety, environmental, and where necessary international requirements of the project. This question of using other standards to comply with Def Stan 00-56 is addressed in Section 4.2 and has also informed some of the criteria in Section 2.4.2.

The safety requirements for CIPHER are likely to be unique because of the complexity of a system, which needs security and safety to be harmonised across multiple platforms and multiple operating environments. So, whilst the consortium has an existing Safety Management System from Thales, it is proposed to take advantage of a number of ongoing developments in the UK within the system safety domain. The consortium would like to determine – with the Authority – what opportunities these developments may offer to benefit CIPHER.

A.6.2.1 Supplier Assessment

The CIPHER project has not yet entered a formal stage of supplier assessment. A number of potential suppliers have been approached, but there is as yet no formal relationship (i.e. contractual or pre-contractual agreements) with these potential suppliers. The information contained in this section is therefore obtained from the CIPHER Safety and Environmental Plan (SEMP).

When selecting a COTS component for use in a system, the intent is to select the candidate which most nearly satisfies all requirements, including safety requirements. Because performing tests on all candidates can be time-consuming and expensive, existing evidence can be used to estimate the component’s ability to meet safety requirements. Existing evidence is evidence about the safety of the COTS component which is obtained from the vendor or other third parties. Existing evidence therefore encompasses the following types:

- Sub-Contractor questionnaires
- Certification packs provided by the vendor
- Test results, lifecycle data, development information or product analyses provided by the vendor (not necessarily in the form of a certification pack)
- Third-party testimonials
- Historical (in-service) evidence.

The CIPHER project anticipates the use of some or all of these forms of existing evidence. Further analysis of this topic is provided in Section 4.1 and previous SSEI work [4], in which we propose a gap analysis to determine the sufficiency of such existing data.

In terms of assessing supplier capability, the CIPHER project will depend heavily upon their previous track record together with any previous experience of working together. This is reflected in the Comparator Data of section 2.4.2, as well as the discussion of supplier track record in Annex A.4.1. Third party testimonials may also be utilised, however, CIPHER is not sufficiently advanced as a project to provide specific instances or examples.

Capability assessments are another method of determining the ability of suppliers to carry out the proposed work. Thales is dedicated to progressing through CMMI accreditation with various business units having achieved differing levels of CMMI accreditation. This reliance on CMMI has informed the guidance of Annex A.4.1. In dealing with external suppliers, Thales recognises that CMMI can apply on both a capability and maturity axis. That is, CMMI assess the ability of the supplier to carry out the work, as well as the maturity of the processes which are proposed. However, the CIPHER project has not yet determined what levels should apply to both axes, and this will almost certainly depend upon the service being supplied.

Whilst CMMI is the Thales preferred capability assessment, this is not the case across all countries and projects. In particular, personnel involved with the CIPHER project have stated that latitude will be given for accepting alternative methodologies. To reflect this experience, this guidance does not recommend the use of a particular capability assessment.

Another form of assessment, mentioned briefly above, is the use of domain and safety experts. Because of the complexity of the CIPHER system, this project requires significant domain expertise across all aspects of secure data transfer provision in order to assess suppliers and tenders. This guidance addresses the input which may reasonably be expected from domain and safety experts in Section 2.4.2 and Annex A.4.1. In particular, domain experts may provide invaluable input when considering the credibility of supplier claims.

A.6.2.2 Safety Considerations and Visibility

Thales personnel have indicated that traditionally, for projects which are not safety-critical, safety has tended to be considered at a later stage of the project.
than might be ideal. However, CIPHER has been determined from the outset to fully involve safety expertise in all aspects of the project and develop a greater emphasis on safety culture within the project. This emphasis on safety has highlighted some areas of conflict which might otherwise have gone undetected. As a result of this experience, the need for safety assessment throughout project development has been stressed in this guidance. The swim-lane diagram of Section 2.4.2 is intended to enforce safety considerations, beginning in the Initial phase.

Thales personnel were also asked about the importance – and credibility – of an indication by suppliers of the level of visibility into their development processes. The judgement - from experience in previous projects, as the CIPHER project is not yet at this stage – is that this greatly depends upon the complexity and integrity of the component or system being procured. Generally, the higher the integrity or complexity, the greater the need for visibility of development and safety management processes, including invitations to participate in internal safety reviews and witness testing, etc. Most reputable suppliers are keen to demonstrate these facets to their customers, although experience has also shown that where COTS is concerned, this level of visibility is more difficult to obtain.

This guidance discusses the need for determining the level of visibility proposed at an early stage. It is assumed from this response that – although the level of visibility provided may not be ideal – in general, it is expected that the supplier will indicate the transparency of their processes. Section A.4.1 contains more information on this.

### A.6.2.3 Assessment of Tenders

This section contains information obtained from Thales personnel about the methods which will be used on the CIPHER project to assess tenders. Where CIPHER-specific information is not available (for reasons of security, or because of the current stage of the project), recommendations have been given based on previous development experience at Thales.

To date no Invitations To Tender have been published on the CIPHER project. However, it is anticipated that the content of the requested information within the ITT would be specific to the system or component in question. Smaller components would be expected to provide safety data sheets, whilst larger components and systems would need to provide safety cases with relevant supporting safety documentation. In this guidance we have indicated some of the relevant inputs which may be expected for a tender (Section 2.4.2), but stress that this is not intended to be an exhaustive list.

Certain aspects of a tender must be able to show conformity or an intention to conform to relevant statutory requirements. In addition, for the CIPHER project there may well be additional contractual requirement place upon Thales by its...
customers, which in turn have to be met by sub-contractors. As a result, sub-contract management on the CIPHER project may be undertaken by specialist staff dedicated to ensuring that a consistent approach is taken across all sub-contractors.

In terms of the additional information which might be expected in a tender, Thales personnel have indicated this is again completely dependant upon the service being requested. As discussed in Annex A.3, it is important to realise that an ITT will be seeking a lot more information other than purely safety information.

In Section 2.4.2 we discuss some relevant comparator data for assessing tenders. From a safety perspective, the primary means of assessing a tender on CIPHER is intended to be via a questionnaire. However, Thales personnel stress that it must be understood that safety is but one criterion under assessment. There will be many other factors which determine whether a tender is accepted such as ISO 27001 accreditation and the ability for the supplier to provide fully trained, and qualified support through the life of the project. As a consequence, this guidance emphasises that comparator data may originate from a wide range of sources (Section 2.1).

A.6.3 CIPHER recommendations

One of the major points which came out of the Thales response is the need to consider issues which are not necessarily related to safety when selecting suppliers. That is, there are many criteria which determine the suitability of a potential supplier and whilst safety is an important consideration, it is not the sole consideration, and is not always the primary consideration.

Other SSEI tasks (e.g. Task 2 and Task 3) are intended to consider the importance of the supply chain, and the difficulties involved in managing the flow of information over contractual boundaries. These difficulties can sometimes give rise to the need to “trade-off” safety for other properties, such as security. As a consequence of the Thales response, it is anticipated that the SoBP – although a document which has a primary focus on software safety – will be informed by the consideration of these issues. In particular, there is a need for further guidance and case studies on the ways in which the trade-off between safety and other parameters can be adequately addressed at the stage of tender assessment.
The following Annexes are taken from previous SSEI work as listed in the references. They may be subject to updates.
ANNEX B    Technical SoBP

This Annex contains the full version of the Technical SoBP, intended to support the summary provided in Section 3. The Technical SoBP provides guidance of relevance to anyone involved in the development or assessment of safety arguments for software in the context of Def Stan 00-56. The material is drawn from the SSEI deliverable “A Systematic Approach to Software Safety Argument Construction” [6]. All references in this Annex are to internal sections and citations are to the list of references in Annex B.6.
EXECUTIVE SUMMARY

Creating safety arguments for the software aspects of safety related or safety critical systems has the potential to bring many benefits over a more traditional prescriptive approach. Constructing compelling software safety arguments can however be challenging, and there is considered to be currently a lack of guidance available.

This report helps to address this need by describing a systematic approach to the construction of software safety arguments. This approach is based upon an extended argument construction method which explicitly considers assurance throughout the development of the software safety argument. This methodology is supported by a software safety argument pattern catalogue which proposes clear and coherent structures for software safety arguments through which it is easier to demonstrate that the resulting argument is sufficiently compelling.
## INITIAL DISTRIBUTION

<table>
<thead>
<tr>
<th>Name</th>
<th>Role/Organisation</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>John McDermid</td>
<td>SSEI Technical Director, (University of York)</td>
<td>York</td>
</tr>
<tr>
<td>Sqd Ldr Mike Place</td>
<td>SSEI Customer Interface Director, (MOD, DE&amp;S)</td>
<td>Abbey Wood</td>
</tr>
<tr>
<td>Rob Brockie</td>
<td>SSEI Contract Holder, (BAE Systems, MAS)</td>
<td>Warton</td>
</tr>
<tr>
<td>Jane Fenn</td>
<td>SSEI Programme Manager, (BAE Systems, MAS)</td>
<td>Brough</td>
</tr>
</tbody>
</table>
VERSION HISTORY

<table>
<thead>
<tr>
<th>Change No.</th>
<th>Incorporated By:</th>
<th>Date</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue 1</td>
<td>Richard Hawkins</td>
<td>December 2008</td>
<td>First Release</td>
</tr>
<tr>
<td>Draft A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Issue 1</td>
<td>Richard Hawkins</td>
<td>December 2008</td>
<td>Updated in response to comments</td>
</tr>
<tr>
<td>Draft B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Issue 1</td>
<td>Richard Hawkins</td>
<td>January 2009</td>
<td>First issue</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Issue 2</td>
<td>Richard Hawkins</td>
<td>May 2009</td>
<td>New version created to reflect comments received in response to SoBP</td>
</tr>
<tr>
<td>Draft A</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note to users:
1. This document will be amended as and when necessary
Contents

1 INTRODUCTION 110

2 SOFTWARE SAFETY ARGUMENTS 111
   2.1 The challenge of software safety argument construction . . 113
   2.2 The nature of safety arguments . . . . . . . . . . . . . . . . 114
   2.3 An assurance-centric view of software safety arguments . . 118

3 CONSTRUCTING SOFTWARE SAFETY ARGUMENTS 120
   3.1 Extending the six-step method - An assurance-based ap-
       proach . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 121
       3.1.1 Step 1 - Identify goals . . . . . . . . . . . . . . . . 123
       3.1.2 Step 2 - Define basis for goals . . . . . . . . . . . 125
       3.1.3 Step 3 - Identify strategy . . . . . . . . . . . . . . 126
       3.1.4 Step 4 - Define basis for strategy . . . . . . . . . . 128
       3.1.5 Step 5 - Elaborate strategy . . . . . . . . . . . . . . 131
       3.1.6 Step 6 - identify solutions . . . . . . . . . . . . . . 131
   3.2 Addressing Assurance Deficits . . . . . . . . . . . . . . . . 139

4 SOFTWARE SAFETY ARGUMENT PATTERNS 143
   4.1 An introduction to argument patterns . . . . . . . . . . . . . 143
   4.2 Modular extensions to GSN . . . . . . . . . . . . . . . . . . 147
       4.2.1 The use of modular GSN in software safety arguments147
   4.3 A software safety argument pattern catalogue . . . . . . . . 150
       4.3.1 High-level software safety argument pattern . . . . . 151
       4.3.2 Software contribution safety argument pattern . . . . 156
       4.3.3 SSR identification software safety argument pattern 164
       4.3.4 Hazardous contribution software safety argument pat-
           tern . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 169
       4.3.5 Argument justification software safety argument pat-
           tern . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 173
       4.3.6 Evidence selection . . . . . . . . . . . . . . . . . . . 178

5 CONCLUSIONS AND FURTHER WORK 179

6 LIST OF REFERENCES 180

7 LIST OF ABBREVIATIONS 183
A SOFTWARE SAFETY ARGUMENT STAKEHOLDER CONSULTATION 184
A.1 Introduction ........................................... 184
A.2 The consultation process .............................. 184
   A.2.1 Consulted stakeholders .......................... 184
   A.2.2 Stakeholder projects ............................. 184
A.3 Consultation results .................................. 185
   A.3.1 The software safety case product ............... 185
   A.3.2 The software safety case process ............... 190
   A.3.3 Specific issues .................................. 191
   A.3.4 Guidance requirements ........................... 192
A.4 Conclusions .......................................... 194

B EXAMPLE SOFTWARE SAFETY ARGUMENT DEVELOPED USING AN EXISTING METHOD 196
1 INTRODUCTION

Creating safety arguments for the software aspects of safety related or safety critical systems has the potential to bring many benefits, as described in this report. Constructing compelling software safety arguments can however be challenging, and there is considered to be currently a lack of guidance available. These positions are supported by the results of a set of interviews conducted with a cross-section of software safety case stakeholders (see Annex A).

This report helps to address this need by providing guidance on the construction of compelling software safety arguments. The document begins by discussing the nature of software safety arguments and the challenges they raise. In Appendix B an example software safety argument is developed using an existing approach. This illustrates some of the typical features of a software safety argument, as well as highlighting many of the potential weaknesses of existing approaches. Section 3.1 then goes on to describe a systematic approach to developing software safety arguments. This approach extends the existing six-step method to explicitly consider the assurance of the software safety argument throughout its creation. Section 4 then provides a set of software safety argument patterns. These patterns can be used within the framework of the extended six-step method in order to ensure that a sufficiently compelling software safety argument is generated.
2 SOFTWARE SAFETY ARGUMENTS

For any safety related, or safety critical system containing software, it is necessary to gain sufficient confidence that the potential contributions that the software may make to system hazards are sufficiently controlled. There are two main approaches that can be adopted to demonstrate the safety of the software aspects of a system. These are a prescriptive approach (often referred to as a process-based approach) and a goal-based approach (often referred to as a product, or evidence based approach). With a prescriptive approach, the developer of the software demonstrates the safety of the system by demonstrating compliance with requirements set out by the appropriate regulatory authority, generally in the form of a prescribed process in a standard. This has traditionally been the most common approach, and is the basis of the standards most commonly used for developing software used in safety related applications, such as [26] and [10]. With a prescriptive approach, the process to be followed will normally be varied according to the risk associated with the software, or the criticality of the outcome of software failure. The greater the risk, or more critical the outcome, the more onerous the prescribed process is. For software with which a lower risk is associated, many of the requirements will not be required to be met.

With a goal-based approach, the regulatory authority does not prescribe a method for demonstrating that the software is acceptably safe. Instead it is the responsibility of the developer of the software to demonstrate to the regulatory authority that the software is acceptably safe to operate. A goal-based approach has a number of advantages over a prescriptive approach. The philosophy of a prescriptive approach is heavily focussed on controlling the processes that are used to develop the software. Generally the processes that are specified in prescriptive standards are very sensible, and the software evidence produced may be of a high level of integrity. The approach relies, however, on the assertion that the processes used, and the evidence generated as a result of following those processes, are adequate to sufficiently control the contribution of the software to the system hazards. This relationship between the prescribed processes and the system hazards is generally tenuous and always implicit. It is generally fairly easy with all prescriptive approaches to conceive of a situation where the prescribed processes have been followed, but there remain software contributions to hazards which are not sufficiently controlled. This is discussed in more detail in [17].

In contrast to this, a goal-based approach does not rely solely on con-
trolling the processes that are used, but instead focusses directly on con-
trolling the contribution of the software to system hazards. This is achieved
by producing a safety argument which demonstrates that the software is
sufficiently safe. The software safety argument demonstrates how the
available evidence can be reasonably interpreted as indicating acceptable
safety. When using a prescriptive approach there exists an implicit ar-
gument (that following the prescribed process will result in an acceptably
safe system). By generating an explicit software safety argument, the way
in which the evidence supports the objectives of the safety case for the
particular system under consideration becomes clear.

The development of a software safety argument relies upon the devel-
oper of the software to determine the most appropriate way to demonstrate
the safety of the software they are developing. A prescriptive approach re-
lies upon the regulatory authority determining the most appropriate way
to demonstrate the safety of any system within their domain. Clearly the
developer of the software itself is normally the most appropriate person to
determine what should be done for their system. The most appropriate role
for the regulatory authority is to assess whether what the developer has
done is sufficient for their system. A software safety argument approach
supports these roles. One further advantage of adopting a goal-based ap-
proach is that, because it is not prescriptive about the methods and tech-
niques that should be adopted, it facilitates the use of new approaches
and technologies that could bring increased capability and efficiency.

Software safety arguments are most clearly represented using a struc-
tured graphical argument notation. The most commonly used notations
are Claims, Arguments, Evidence (CAE) [1] and Goal Structuring Nota-
tion (GSN) [12]. Throughout this report GSN is used to represent safety
arguments. The basic GSN symbols are shown in figure 1.

![Figure 1: Main elements of the GSN notation](image-url)
These symbols can be used to construct an argument by showing how claims (goals) are broken down into sub-claims, until eventually they can be supported by evidence (solutions). The strategies adopted, and the rationale (assumptions and justifications) can be captured, along with the context in which the goals are stated. An example GSN argument is shown in figure 2.

![Figure 2: An example GSN goal structure](image)

2.1 The challenge of software safety argument construction

Constructing compelling safety arguments for software is challenging. As discussed earlier, there are a number of weaknesses with a prescriptive approach, but there is one advantage, which is that the developer of the software knows clearly from the outset the processes that must be followed, and the techniques that must be used. This helps with the planning and management of the development project. In contrast to this, when adopting a safety argument based approach, the necessary activities and processes are not specified up front. Instead the high level objectives are specified, and the developer must determine what techniques and ev-
idence are necessary and sufficient to construct a compelling safety argument. Identifying what evidence will be sufficient to demonstrate that the contribution of the software to the safety of the system is acceptable is a major challenge. In many ways this challenge exists for all safety arguments, however this has been seen to be especially so when considering the software aspects of a system (see Annex A).

The aim of producing a safety argument is to demonstrate that the safety risks of a system have been adequately controlled. Clearly then, the techniques used, and the evidence generated must be appropriate for that risk. Defence standard 00-56 issue 4 [21] requires the production of a safety argument which is commensurate with the system risk:

9.5 The Safety Case shall contain a structured argument demonstrating that the evidence contained therein is sufficient to show that the system is safe. The argument shall be commensurate with the potential risk posed by the system....

It also requires that the risk is sufficiently reduced:

10.7.1...the Contractor shall carry out Risk Reduction by identifying and implementing a combination of mitigation strategies until the Tolerability Criteria are met, enabling Risk Acceptance to take place...

As we shall see, a goal based approach explicitly addresses risk reduction in the safety argument, however there still remains the problem of knowing if the mitigations put in place are sufficient for a given risk.

There have been a number of attempts to define the types of evidence that are required for different levels of risk, for example as part of the guidance within 00-56 [21] itself, and also in other guidance documents such as [7]. However, in all existing guidance the relationship between the evidence generated and the risk reduction achieved is unclear. This means that it is very difficult to justify the sufficiency of the software safety argument.

2.2 The nature of safety arguments

As discussed in [3], all arguments can be split into two types, deductive and inductive. Baggini and Fosl characterise deductive arguments as those where, if the premises are true, then the conclusion must also be true. An example of a deductive argument is shown in figure 3.
Premise 1
There are 32 books on the top shelf

Premise 2
There are 12 books on the bottom shelf

Premise 3
There are no other books on the bookcase

Conclusion
There are 44 books on the bookcase

Figure 3: An example of a deductive argument

In this example if we know premises 1, 2 and 3 to be true, then we will also believe the conclusion with certainty (there can be no doubt). In contrast, an inductive argument is characterised as one where the conclusion follows from the premises not with necessity, but only with probability. An example of an inductive argument is shown in figure 4.

Premise 1
This month is December

Premise 2
It has rained in Manchester every December in recorded history

Conclusion
It will rain in Manchester this month

Figure 4: An example of an inductive argument

In this example, even if we know the premises to be true, we cannot say with certainty that the conclusion is also true. The premises give us some level of confidence in the conclusion, but since the conclusion is based on prediction from past experience, there is also a chance that the conclusion may not be true. It is more common to see safety arguments which are inductive in nature.

In order for a safety argument to be compelling, it is necessary to provide and demonstrate a sufficient level of confidence in the conclusion. It
is possible to increase the confidence in the truth of a claim by providing additional support. For example consider figures 5 to 7. In figure 5, a certain amount of confidence can be gained in Function X being provided by supporting this conclusion with premise 1. The amount of confidence provided by premise 1 alone may not be sufficient however. By also supporting premise 2, the confidence in the conclusion increases, and increases even further with the addition of premise 3.

![Figure 5: Increasing confidence in a safety argument 1](image1)

The question still remains as to whether the confidence achieved in the conclusion through the support of premises 1, 2 and 3 is sufficient. It is necessary to be able to determine when enough confidence has been
Figure 7: Increasing confidence in a safety argument

Achieved in a safety argument. When considering safety arguments the term *assurance* is often used. Assurance is a term that is used in Defence Standard 00-56 [21], where it is defined as “Adequate confidence and evidence, through due process, that safety requirements have been met.” It is believed that this definition of assurance is inadequate. In particular assurance can relate to any claim made in a safety argument, not just to meeting safety requirements. It is also felt that due process relates to a method of achieving assurance, rather than the property itself. Instead we propose another definition:

**Assurance of a claim** is justifiable confidence in the truth of that claim\(^1\)

The assurance of a conclusion is related to:

- The assurance of the premises
- The extent to which the premises give reason to believe the conclusion is true

It is crucial in developing a compelling safety argument to demonstrate that sufficient assurance has been achieved. Due to the inductive nature of most safety arguments, determination of assurance is always going to be subjective. What is important is that agreement can be reached between the safety argument provider, and the safety argument reader, that the subjective position is acceptable. This can be achieved by arguing that the argument is sufficient. It is important to note that even with a prescriptive

\(^1\)Author’s definition
approach, the subjectivities discussed above are still present, it is just that they are left implicit. By reasoning explicitly about the subjectiveness of a safety argument, it is easier to reach mutual agreement.

2.3 An assurance-centric view of software safety arguments

Many existing approaches to the assurance of safety arguments, such as [5] and [29], have been predominantly focused on retrospective assessment of the assurance achieved once the argument is in place. Instead, we propose that the best way to ensure that a software safety argument is sufficiently compelling is to consider assurance throughout the development of the software safety argument. Considering assurance from the outset of argument construction makes it possible to identify early on in the lifecycle what arguments and evidence will be required to achieve sufficient assurance. This makes it easier to manage the software development activities.

Continuous consideration of assurance should also reduce the risk that a safety argument will be produced that is not sufficiently compelling. As assurance is being considered throughout argument development, it will be identified early on if there is an assurance deficiency which must be addressed. Addressing a deficiency early (through, for example the identification of a requirement for additional evidence) is generally cheaper than identifying an additional requirement late on in the development process.

With a prescriptive approach, all of the process activities recommended for a particular integrity level must be assumed to carry equal weight. By considering assurance in a safety argument, the relevant importance of the various aspects of the argument can be determined which enables effort to be allocated to supporting the aspects of the argument that are most important (i.e. those that will have most impact on the assurance). This will mean that effort isn’t wasted on undertaking activities that don’t increase assurance.

One further advantage of an explicit consideration of assurance throughout, is that it helps to facilitate an assurance-by-design approach, in that decisions that are made during the design of the software can be made with knowledge of the effects of such decisions on the assurance that can be achieved based on a particular design choice. As we shall see, decisions that are made throughout the software development (including requirements, design and implementation decisions) have the potential to affect the assurance achieved in the final product. Making such decisions with an understanding of their assurance impact should again reduce the
risk of producing a safety argument that is not sufficiently compelling.
3 CONSTRUCTING SOFTWARE SAFETY ARGUMENTS

In this section we show how a sufficiently compelling software safety argument may be constructed. There exists a general method for constructing and defining goal structures which was developed by Kelly [13]. This approach is often referred to as the ‘six-step method’, figure 8 shows diagrammatically the steps of the method.

![Diagram of the six-step GSN construction method]

It is possible to use this existing method to develop a safety argument for a software system. An example of this is provided in Annex B. This example illustrates the types of claims, strategies and evidence that may be put forward for the software aspects of a safety related system. The example provided in Annex B is however unacceptable, in particular it would be very difficult to justify that the argument presented is sufficiently compelling. In this section, and in section 4 we propose a method for constructing compelling software safety arguments, and justifying their sufficiency.
3.1 Extending the six-step method - An assurance-based approach

To be acceptable, the software safety argument produced must be sufficiently compelling. The argument must provide sufficient confidence in the truth of the claims made. The example discussed in Annex B, illustrates how an existing software safety argument approach could be used to construct a safety argument for an example software system. The problem still remains however of judging whether such an argument would be good enough if it was presented as the software safety argument for the example LRAAM system. Is the argument presented sufficiently compelling? In fact, what this example argument demonstrates is that many activities have been undertaken, and lots of evidence has been generated in support of the safety of the software. However, is what was done appropriate? Could, and should, more have been done in order to make the argument more compelling? To know if a safety argument is sufficiently compelling it is necessary to be able to answer these questions, and this requires an explicit consideration of assurance.

It is inevitable for the software aspects of a system that there will exist inherent uncertainties which affect the assurance with which it is possible to demonstrate the safety of the software. The reason for this is that the amount of information potentially relevant to demonstrating the safety of the system is vast. This may be information relating to the software itself, or to the system within which the software operates. There will also be information relating to the environment and operation of the system, all of which potentially has a role in demonstrating that the software is acceptably safe. It is simply not possible therefore to have complete knowledge about the safety of the software. This leads to uncertainty, for example through having to make assumptions or accept known limitations in the integrity of the evidence generated, or the strength of support that evidence provides. Such considerations are the reason why, as discussed earlier, software safety arguments are generally inductive in nature. It is worth reiterating at this point that such uncertainties in demonstrating the safety of the software are always present. The uncertainties are present in a prescriptive approach, however they are often left implicit. Adopting a goal-based approach facilitates an explicit identification of such uncertainties, which makes them easier to reason about, and therefore justify. We shall see how the assurance-based approach described here makes such justification possible.

Any identified residual uncertainty in demonstrating the safety of the software (such as those discussed above) can be considered to be an as-
surance deficit. They can reduce the assurance which is achieved. Such assurance deficits, although always present, are often left implicit, and therefore their impact on the safety of the system is overlooked. In this section we consider how assurance deficits may be explicitly identified during the development of a software safety argument.

We propose that it is possible to identify how assurance deficits may arise by explicitly considering how information may be lost at each step in the construction of the argument. As an argument is constructed, decisions are continually being made about the best way in which to proceed. Decisions are made about how goals are stated, the strategies that are going to be adopted, the context and assumptions that are going to be required, and the evidence it is necessary to provide. Each of these decisions has an influence on what is, and is not, addressed by the safety case. The things that are not sufficiently addressed are referred to as assurance deficits.

In order to produce a sufficiently compelling software safety argument, all identified assurance deficits must be satisfactorily addressed, or justification must be provided that the impact of the assurance deficit on the claimed safety of the system is acceptable. Section 3.2 discusses how such justifications may be made.

To extend the existing six-step method to explicitly identify assurance deficits, the potential ways in which assurance may be lost at each of the steps in the method must be determined. In order to achieve this a deviation-style analysis of each of the six steps was performed. This considered the purpose of each of the steps, and then considered the ways in which uncertainty may be introduced into the argument at that step due to missing information. This deviation analysis is based on the widely-used HAZOP technique, which was originally developed as a way of analysing process plants [4] but has since been developed for used in other applications including the analysis of software [25]. HAZOP uses a set of guidewords to prompt the identification of deviations from normal behaviour. The standard HAZOP guidewords are:

- No or None
- More
- Less
- As well as
- Part of
• Other than
• Reverse

Below we apply and interpret the HAZOP guidewords for each step to consider how assurance deficits may be introduced. Only those guidewords with a meaningful interpretation are considered for a particular step.²

An overview of the argument construction guidance can be found in Appendix C which provides a summary for each of the six steps.

3.1.1 Step 1 - Identify goals

Purpose of step To clearly and unambiguously state the goals to be supported.

More - If in stating the goal, an attempt is made to claim more than it is actually possible to support with the available evidence, then the assurance that can be achieved in that goal will inevitably be low. By making a more realistic claim, it may in fact be possible to achieve the required assurance. For example, a goal could be stated “All failure modes of function X are correctly identified”. In practice it may be very difficult, given the techniques available to achieve a high level of confidence in such an exhaustive claim. Instead, claims may be made about identifying all failure modes for function X of a particular type, in such cases the confidence achievable may be higher. In making such weaker claims however, it is important to consider the opposing effects detailed below.

Less - The stated goal may claim less than is actually required to support the argument. Although in this case it may be easier to achieve higher confidence in the stated goal, this confidence will not result in the expected assurance in the parent goal, since the claim is insufficient to support the conclusion.

It is important at this point to consider the distinction between making a claim about the probability of something, and reasoning about the uncertainty in that claim. This is the difference between aleatoric uncertainty, characterised by randomness and predictions of future behaviour based on the past, and epistemic uncertainty, which is characterised by imperfect knowledge. As McDermid points out in [18], epistemic uncertainty is generally the normal case when dealing with software.

²Note that no meaningful interpretation of the reverse guideword was found in this context, so this does not appear for any of the steps.
It is important that when considering aleatoric uncertainty, this is reflected in the nature of the claims themselves, either, where appropriate, through inclusion of qualitative probabilities, or through the use of qualifier statements in goals, such as sufficiently or acceptably. This should not be confused with making a definitive statement in the claim, and then associating confidence, or assurance, with that claim. Although there will be an epistemic element of uncertainty to all claims, where aleatoric uncertainty exists (such as is the case for hardware related failures) the claim should reflect this. Since software, due to the non-random nature of its failure, deals mainly in epistemic uncertainty, it would here be appropriate to make a definitive statement in the claim. The epistemic nature of the claim is then captured in the associated confidence.

That is not to say that probabilistic claims cannot be made about the reliability of software. Such claims are possible, however there are limitations to the level of aleatoric certainty that can be demonstrated. It has been suggested that failure rate claims for software better than \(10^{-4}\) per hour, the lowest rate which can realistically be shown by statistical testing, should not be accepted unless and until the software is proven in service [16]. Such reliability claims for software will also generally rely on a large number of caveats about the representativeness of the verification and the continuity of the software behaviour. All of these assumptions reduce confidence in the reliability claim, and thus increase the associated epistemic uncertainty.

This is a subtle but important distinction. The uncertainty in an aleatoric claim in a safety argument captures a consideration of the level of risk. Once we start to consider epistemic uncertainty, we are moving from a consideration of risk, into a consideration of assurance. In producing a software safety argument it is normal to transition from claims relating primarily to risk, to claims relating solely to assurance. Section 3.2 considers the relationship between risk and assurance more closely.

**As Well As** - A strategy or solution may be erroneously included in the claim. This can inadvertently limit the assurance that can be achieved in the goal. If we consider the claim “All sources of hazardous failure are mitigated through use of design-by-contract”, we could be preventing the opportunity to provide additional assurance because of the way this goal is stated. If for example a goal was instead stated as “All sources of hazardous failure are mitigated in the design”, there is an opportunity to make a decision on different strategies to support this goal, if that is required to achieve the necessary assurance, other than just a reliance on design-by-contract.

**Other Than** - the claim made may not actually be that in which as-
Assurance is required. Assurance may be lost through failing to correctly capture the true intent of the claim. If an incorrect goal is stated, even if this goal is then adequately supported, the reader of the safety argument will not be satisfied because the wrong point has been demonstrated.

As an example of how assurance may be lost at this step, consider the claim “Software process is appropriate for system X”. It may well be possible to get high assurance in this particular claim, however if what the safety argument reader really wants is assurance about system X hazards being mitigated, this claim does not provide them with that. In fact the confidence in system X hazards being mitigated would actually be quite low based on the argument produced, because the objective of the claim was incorrectly captured.

The key point at this step is to consider precisely what it is that confidence is required in, and ensuring that the goal correctly captures this. It is important in doing this to always consider the intended readership, and what it is that they want the argument to give them confidence in.

3.1.2 Step 2 - Define basis for goals

**Purpose of step** To clarify the scope of the claim, to provide definitions of terms used, to interpret the meaning of concepts.

**None** - Context is not provided for the claim. Any claim is only true or false over a particular scope. If the scope of the claim is unclear, then the level of truth or falsity of the claim becomes more difficult to determine. This increases the uncertainty associated with the assurance in that claim, and therefore makes it more difficult to determine the assurance. Context helps to clarify the scope of the claim such that it is unambiguous. This reduces the uncertainty associated with the assurance, and thus makes it easier to determine its sufficiency.

Context also provides a mechanism for clarifying the meaning of terms used in stating the claim. Describing precisely what is meant by the claim helps to explain to the reader why they should have confidence that the claim is true. Missing out required contextual information increases the uncertainty in the assurance. It is important at this step to consider what the reader of the argument may need to know in order to have confidence in the claim, and to provide that information clearly and precisely.

Consider the following claim, “Software system is acceptably safe to operate”. The assurance consideration for such a claim is whether it is sufficiently clear which system is being referred to, and whether the boundary for what is considered as part of the system has been clearly defined.
Without this context in place it is not possible for the reader to have confidence that the claim is sufficiently supported by the argument put forward. As another example, we can consider the claim “Software component Y has no credible failure modes”. To understand what assurance is achieved in this claim, it must be clear what is meant by a credible failure mode.

**More** - The scope of the claim as defined by the context may be too narrow. If the context were not specified so tightly, the scope covered by the claim would be much wider. The result of this is that although a certain level of assurance may be achieved over the scope defined by the context, the narrowness of the scope is actually limiting that in which you have confidence. For example, context may be used to specify the operating modes over which a safety property holds. The context may overly constrain this claim by stating a very small number of the operating modes. Although assurance may be achieved in the claim over those modes, due to this context there is no assurance over any other operating modes. In fact it may be possible that the claim holds more generally. Widening the context may allow the assurance to be achieved in a larger number of operating modes, which may result in more assurance in higher level claims.

**Less** - The scope of the claim is too loosely defined. The effect of this would be similar to having no context at all, in that it leads to uncertainty, and a corresponding reduction in assurance.

### 3.1.3 Step 3 - Identify strategy

**Purpose of step** To identify how a goal can be adequately supported.

**More, Less, Other Than** - This step of the safety argument process is the most crucial for the assurance achieved since it is at this step that the decisions are made about which strategy should be adopted to support each claim. Assurance is lost at this step if the proposed strategy does not provide sufficient support to the goal. This could happen for two reasons. Firstly the inductive gap may be too large. If this is the case, then even if the premises are believed, it doesn’t provide sufficient confidence in the truth of the conclusion. Secondly the fundamental beliefs upon which the strategy is based may be open to question. In such a case the premises may not provide confidence in the conclusion.

If we firstly consider the inductive gap, the objective at this step is to choose the strategy (the argument approach) that is sufficient to support the claim. This involves firstly identifying the potential sub-claims that may provide support to the claim and then justifying the decision on which of
the potential strategies to adopt in the argument. In order to provide guidance on the potential compelling strategies or argument structures which might be used in a software safety argument, a software safety argument pattern catalogue is provided in section 4. As is discussed in section 4 these software safety argument patterns capture good practice for software safety arguments, and therefore can be used to identify the potential strategies that may be adopted.

Even if using the software safety argument pattern catalogue to identify the argument structure, this does not guarantee that the argument will be sufficiently compelling. As discussed further in section 4, there are many decision which must be made when instantiating the general patterns for specific systems. To be compelling it is necessary to be able to justify that the instantiation decisions taken in constructing the argument result in a sufficiently compelling argument for the system under consideration (such as why particular claims are chosen whilst others are not required). Guidance for justifying such decisions is provided in section 3.2.

**Fallacies and anti-patterns**

The discussion above has considered how assurance deficits may arise due to the inductive gap between the premises and the conclusion in the software safety argument. If the premises of the chosen strategy do not provide sufficient confidence that the conclusion is true then the argument will not be sufficiently compelling. The assurance of the conclusion is also undermined if the fundamental beliefs upon which the chosen strategy is based are unsound. These fundamental beliefs have been called *warrants* by Toulmin [28], who describes them as the reasons why it should be believed that the grounds (premises) lead to the claim (conclusion). If the warrants upon which the argument is made are not valid, then it is *unwarranted* to conclude that the claim follows from the stated grounds. The reader will have no confidence in an unwarranted argument.

In [6] the authors refer to such faulty reasoning as fallacies. They have performed analysis of a number of system safety cases to identify the presence of fallacious reasoning, as defined in classical philosophical literature, in the arguments. A large number of fallacies were identified in each of the safety cases considered. Based on this, the authors have developed a taxonomy of safety argument fallacies, which can be used to assist in the identification of fallacies in safety arguments. A total of 33 fallacies were identified, split into eight categories. The fallacy categories are listed below, the full safety argument fallacy taxonomy can be seen in [6].
• Circular reasoning
• Diversionary arguments
• Fallacious appeals
• Mathematical fallacies
• Unsupported assertions
• Anecdotal arguments
• Omission of key evidence
• Linguistic fallacies

The taxonomy is general to safety arguments and is therefore equally applicable to software safety arguments. It is also possible however to identify more specific fallacies for software arguments. We have decided that an effective way to capture examples of fallacious software safety arguments would be in the form of software safety argument anti-patterns, which would provide examples of software safety argument structures which, although at first sight may appear to be compelling, are in fact fallacious. These patterns could be used when constructing a software safety argument as a way of ensuring that known ‘pitfalls’ are avoided when deciding upon an appropriate strategy to support a claim. The development of antipatterns will be the subject of future work.

3.1.4 Step 4 - Define basis for strategy

Purpose of step To identify any assumptions upon which the sufficiency of the strategy depends and to provide justification for why a particular strategy is being used.

Firstly we shall consider assumptions.

No, Less - It is inevitable that some assumptions will be made during the development of any safety argument, however these assumptions may not always be explicitly captured. Any assumptions that are made in the argument which are left implicit introduce uncertainty, since it is not clear to the reader whether the assumptions have been considered, or whether they impact upon the truth of the claim being made. For this reason it is important that all assumptions made are identified and captured explicitly within the argument.
More - Assumptions may be made whose truth cannot be reasonably assumed. All assumptions are, by definition, unsupported. Assumptions are taken to be true, and the argument holds on the basis that the assumptions are true. Consequently, if there is a lack of confidence in the truth of the assumptions, then this will result in a lack of confidence in the truth of the claim as well. The presence of assumptions will only not impact the assurance of the claim if the truth of that assumption is beyond question (*res ipsa loquitur*)\(^3\). It is therefore recommended, for any assumptions that may be open to any significant doubt, that an argument is presented, rather than an assumption. Making an argument means that the reader is not obliged to simply accept the truth of the assumption, instead the truth can be demonstrated using an argument.

As an example, consider the argument shown in figure 9. In this example the argument is based on two assumptions: that the necessary resources are available, and that failures in other functions do not interfere with the function of concern. The presence of these assumptions would significantly reduce the assurance of goal ‘DesignFailures’, because it is likely that, without any further justification, most readers would have little confidence that these assumptions are true. Figure 10 shows how justification may be provided by making these assumptions as claims, and supporting those claims with an argument as to why they are true. It then becomes possible to demonstrate that there is sufficient assurance in goal ‘DesignFailures’.

---

\(^3\)A legal term from the Latin meaning ‘the thing speaks for itself’.

---

Figure 9: Using assumptions in an argument
Other Than - Assumptions may be stated which are not actually true. Any false assumptions undermine the whole basis upon which the argument is made.

We must also consider justifications.

No - No justification is provided as to why the adopted strategy is sufficient. This can result in a loss of assurance, since there may be a lack of confidence in the sufficiency of that strategy. As with assumptions it is important, if it’s likely that the justification may be unclear, not to leave it implicit, but to explicitly record the justification in the argument. The justification can be captured using the GSN justification node, however it may in some cases be necessary make an argument about why the chosen strategy is justified. Such a justification may be made based on the ACARP assessment described at step 3. A pattern for an argument that might be presented to justify the use of a strategy based on an ACARP consideration in presented in section 4.

Less - The justification provided may be insufficient. The justification may be provided as a simple GSN justification node where actually, to convince the reader that the chosen strategy is ACARP, a justification argument is actually required.
More - Although not leading to a loss of assurance, it is important to note that providing an argument to justify the strategy chosen in each decomposition in the argument is not necessary. For many strategies, the justification will be obvious to the reader and may be left implicit. The effort involved in a full ACARP assessment and justification argument is only warranted where the assurance impact of the strategy decision is likely to be high.

3.1.5 Step 5 - Elaborate strategy

Purpose of step Specify the goals that implement the chosen strategy.

Less, As Well As, Part Of - The strategy that is actually implemented is not the one that was chosen. Assurance may be lost at this step, even though a chosen strategy may be considered acceptable (i.e. it is determined that such a strategy could provide sufficient confidence in the claim being supported). If the goals to implement that strategy are stated incorrectly, then the strategy may not be correctly implemented. The result of this is that achieving confidence in the stated goals, does not necessarily provide the confidence in the parent that was expected. If we consider the example in figure 32, to show that DSSRs are correctly identified by identifying HSFMs and deriving DSSRs to mitigate them may be an acceptable strategy. If we consider goal ‘RqtSHSFMIdent’ however, it is possible to identify a potential weakness in the way this strategy has been implemented. This goal does not make any claim that the HSFMs are complete, or correct, and therefore having confidence in the truth of this claim does not necessarily provide the expected confidence in the parent claim. It may in fact be that the argument and evidence supporting ‘RqtSHSFMIdent’ is sufficient to support a claim that the HSFMs identified are complete and correct, but since the claim does not say this, it introduces uncertainty, and therefore reduces assurance.

3.1.6 Step 6 - identify solutions

Purpose of step identify the solutions which provide adequate support to the goal.

Less - The solution provides less confidence in the goal being supported than is required. Assurance is lost at this step if it is unclear why the evidence gives confidence in the goal being supported. It may be unclear because there may be an inductive gap between the claim and
the evidence (the nature of the evidence does not provide a compelling reason to believe the claim is true). It may also be unclear because of uncertainty about the trustworthiness of the evidence itself. If the evidence is not trustworthy, then this introduces uncertainty and a corresponding reduction in assurance. Note that evidence which is untrustworthy will undermine confidence even in the situation where there is a deductive relationship between the claim and the evidence.

The issue of the inductive gap is addressed later in this section, firstly we consider the trustworthiness of the evidence. Trustworthiness of evidence is a measure of how likely it is that the evidence is free from errors, or the quality of that evidence. In [8] the authors describe how demonstrating that the evidence used to support an argument is sufficiently trustworthy can be demonstrated by providing a ‘trustworthiness argument’ for the evidence. They propose that such a trustworthiness argument should be based upon a consideration of the processes used to generate the evidence. The types of factors that could influence trustworthiness include:

- Competency of personnel
- Suitability and reliability of the methods used
- Qualification of any tools used, e.g. development or verification tools
- Suitability and clarity of notations

---

4Other terms such as rigour, integrity and provenance have been used to refer to the same property. The author feels trustworthiness is the most appropriate term since it captures the idea of having confidence in the evidence.
- Independence of those involved in producing the evidence

Figure 12 shows an example trustworthiness argument taken from [8].

![Diagram of a trustworthiness argument example]

Figure 12: An argument that black box testing is trustworthy

In selecting the evidence that is used to support a claim, the confidence that the type of evidence being used provides in the truth of the conclusion also needs to be considered. For different claims, the confidence gained from different types of evidence will be different, depending upon the nature of the claim being supported. In many cases, one type of evidence will be insufficient to provide the necessary confidence in a claim. We propose that to identify the type of evidence necessary to support a claim, it is first identified what information would need to be known in order to
have confidence that the claim is true. For a claim regarding the fulfilment of a specific safety requirement for example, information about specific properties of a function may be required. The information needed will be determined by the nature of the requirement itself, for example information may be required about the response time of the function or perhaps information regarding the accuracy of a calculation performed. These claims would require very different types of information from the evidence in order to have confidence.

It will often be the case that, particularly for higher assurance requirements, the required confidence in the claim cannot be achieved through a single piece of evidence. This could be the case because the information provided by that evidence is insufficient to gain the necessary confidence in the truth of the conclusion. In this case the additional information may be provided by a different type of evidence. This diversity can be used in order to provide confidence in those areas for which there may be weaknesses, or necessary assumptions in one type of evidence. In other cases diversity may be required because the trustworthiness of the evidence may be insufficient to be relied upon without additional corroboration.

Some guidance has been provided on the types of evidence that may be required to support claims of varying levels of confidence, particularly in [21] where Annex C of Part 2 provides tables such as that in figure 13 for different confidence levels.

In principle guidance such as this gives an indication of what evidence might be considered sufficient, however it is important that the evidence requirements of each claim are considered on a case by case basis. It is also necessary to justify the sufficiency of the evidence provided. Diversity should not be seen as a guarantee of increased confidence. Unless the diversity in the evidence is focussed on providing additional information relevant to the requirements of the supported claim, the diversity may not increase the confidence in the claim as expected. This theory is also supported by other ongoing research [15], investigating the efficacy of diverse arguments.

In addition to considering the general trustworthiness of the evidence, it is therefore also necessary to consider if the evidence provides everything that it is required to provide in order to support the parent claim. As an example, if we consider the argument in figure 32, it can be seen that FHA results are used as evidence to support goal ‘FHA’. To understand if this evidence provides sufficient assurance to the goal, as well as the general trustworthiness of the FHA results, it is also necessary to consider what the evidence must provide in order to give confidence that the HSFMs of the premature launch prevention function are correctly identified. In this
Table 3 Gaining Confidence in Evidence

<table>
<thead>
<tr>
<th>Required level of confidence</th>
<th>Suggested evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>High — The highest level of confidence possible given the state of the art. The range of uncertainty (confidence) in the quantitative evidence claims should err on the side of pessimism.</td>
<td>Diverse forms of evidence, each providing high confidence, should be combined. Argument based on integrating evidence from testing (including accelerated life testing, environmental testing, etc.) to a high degree of statistical confidence, experience of use (again giving a high statistical confidence) and analytical arguments, non-destructive examination, etc., together with qualitative arguments of good practice (may require best practice), and other confidence building measures such as process metrics or quantitative modelling. Evidence should be rigorous and comprehensive (e.g. 100% coverage in testing). The functionality, performance and behaviour of the system should be understood in its entirety in its operating context; there should be confidence that there are no hidden or misunderstood properties. For COTS components, the cooperation of the supplier is essential, as design and manufacturing specifications and quality assurance data is typically required. Evidence should be subjected to rigorous, detailed independent scrutiny.</td>
</tr>
</tbody>
</table>

Figure 13: Def Stan 00-56 guidance on evidence requirements for different levels of confidence

e.g. It would be necessary to consider the following:

- Does the FHA consider the correct function?
- Are the failure conditions correctly identified as part of the FHA?
- Are the HSFMs identified as part of the FHA results?
- How exhaustive is the FHA?
- Is the information used in the analysis correct and current?

To demonstrate that the evidence provides sufficient assurance to the claim it is necessary to provide an argument addressing each of these issues, in addition to the general trustworthiness of the evidence. If confidence cannot be gained in any one of these aspects
then the impact upon the assurance of not having that information must be assessed, and if necessary an ACARP justification provided for why the evidence is good enough. The argument justification software safety argument pattern presented in section 4 provides a way of making an argument regarding the acceptability of the chosen solution with respect to the claim.

Other than - It is important that the solution provided is relevant to the claim being supported. It is important in this to consider both the nature of the claim itself, and also any contextual information which might affect the relevance of the evidence provided. For example, it is important to ensure that the evidence provided to support a software claim is evidence about the correct version of the software. If the context within which the claim is stated is not compatible with the scope of the evidence provided, then the assurance of the claim will be reduced.

COUNTER EVIDENCE

Counter evidence is any evidence which undermines the confidence in the claim being made. As Defence Standard 00-56 [21] states:

“Throughout the life of the system, the evidence and arguments in the Safety Case should be challenged in an attempt to refute them. Evidence that is discovered with the potential to undermine a previously accepted argument is referred to as counter-evidence” (56-2 9.5.6)

An example of counter evidence is shown in figure 14.

![Figure 14: An example of counter evidence](image-url)
The evidence ‘test failure’ undermines the belief that the DSSRs are implemented in the code. It is important to note that the presence of counter evidence does not necessarily mean that the argument is inadequate. It simply means that the confidence in the claim may now be lower than it was before the counter evidence was identified. It is necessary to determine the impact of any identified counter evidence. Whenever counter evidence is identified it is necessary to consider what particular information relating to the claim is now in doubt and what additional information is now known.

In many cases it may still be possible to make a sufficiently compelling argument despite the identification of counter evidence, particularly where there are mitigations which limit the uncertainty caused by the counter evidence. What is important is that the argument reflects the fact that the counter evidence exists, and demonstrates that there is sufficient assurance. Figure 15 shows an example of how the argument from figure 14 may be restructured to demonstrate how the counter evidence is dealt with.

Figure 15: An argument considering the effect of counter evidence

As discussed above, the presence of counter evidence may reduce the confidence in a claim. If we are not confident that all counter evidence relating to a claim has been identified this will also increase uncertainty. If the counter evidence is correctly identified then it is possible to deal with that counter-evidence or justify that the argument is still sufficient. If
the counter-evidence isn’t even identified then the potential effect on the assurance of the argument cannot be determined. For this reason it is important to gain confidence that counter evidence which could affect the assurance of the claim has been identified. Defence standard 00-56 Issue 4 [21] explicitly requires that:

“The process of searching for potential counter-evidence as well as the processes of recording, analysing and acting upon counter-evidence are an important part of a robust Safety Management System and should be documented in the Safety Case.”

To ensure that claims are sufficiently assured, it is necessary to justify that there is sufficient confidence that relevant counter evidence has been identified. The most effective way to provide this justification is using an argument about the counter evidence identification. This argument must consider the methods used to identify counter evidence, and also the rigour with which the search was undertaken. In searching for counter evidence relating to the argument, the potential search space is very large, since for any claim there are likely to be many potential sources of counter evidence. Knowledge about the existence of counter evidence can be used however to help focus a counter evidence search.

Since an assurance deficit corresponds to a lack of relevant information, an identified assurance deficit reveals the potential for counter-evidence. That is there is the possibility that in addressing the assurance deficit (i.e. gaining the relevant information) the information gained would reveal previously unidentified counter evidence. Reasoning about assurance deficits can therefore be helpful in identifying areas in which counter evidence may exist. Conversely, where there is knowledge of existing counter evidence, this can be used to help determine the potential impact of assurance deficits. For example, if other similar projects have identified counter evidence which relates to a particular identified assurance deficit, then the observed impact of this counter evidence on the safety of the other project can be used to indicate the expected impact that such an assurance deficit may imply. The impact of assurance deficits is discussed in more detail in section 3.2.

In a similar approach to that described for assurance deficits in section 3.2, it is necessary to consider whether the cost of searching for additional counter evidence is disproportionate to the potential impact that the identified counter evidence may have. In order to determine if it necessary to search for counter evidence in a particular area, it is necessary to consider the type of counter evidence that such a search may generate. It is then necessary to determine what information that type of counter evidence
could provide relevant to claims in the argument. The potential impact on the confidence in the claim of identifying that type of counter evidence can then be determined.

**More than** - It may seem strange to suggest that providing more evidence than is actually required can impact upon the assurance achieved. If however the solution provided is not well ‘focussed’ upon the claim which it is supporting, it can be hard for the reader to understand how that solution supports the claim, and therefore can reduce the confidence in that claim. For example, a claim relating to a specific property of a particular software function may be strongly supported by the results of some analysis activities. If however the solution provided is a huge set of documentation relating to a wide range of properties of the entire software, it may be hard to pick out the relevant part for that claim. It is important that the argument clearly identifies the solution relating to the claim.

### 3.2 Addressing Assurance Deficits

The discussion in section 3.1 above illustrated how assurance deficits may be systematically identified throughout the construction of a software safety argument. The existence of identified assurance deficits raises questions concerning the sufficiency of the argument. Therefore where an assurance deficit is identified, it is necessary to demonstrate that the deficit is either acceptable, or addressed such that it becomes acceptable (for example through the generation of additional relevant information). There will typically be a cost associated with obtaining the information to address an assurance deficit. In theory it would be possible to spend large sums of money generating sufficient information to address all assurance deficits. However, in practice the benefit gained from addressing each assurance deficit does not necessarily justify the cost involved in generating the additional information. In order to assess if the required level of expenditure is warranted, the impact of that assurance deficit on the sufficiency of the argument must be determined. Firstly we therefore discuss how the impact of an assurance deficit can be assessed.

The software safety argument establishes a claimed position on the hazard identification, risk estimation, and risk management of the software contribution to system hazards. Since assurance deficits have the potential to undermine the sufficiency of the argument, the impact of any assurance deficit should be assessed in terms of the impact it may have on this claimed position. Is the assurance deficit significant enough that that position can no longer be supported? For example, an assurance deficit
may be sufficient to challenge the completeness of hazard identification, or may be sufficient to challenge the estimated residual risk. It may also be possible, for example, that an assurance deficit challenges a claim that the software contribution to system hazards are acceptably managed.

One of the challenges of determining the impact of an assurance deficit is that the activities undertaken to address an assurance deficit (such as generating additional evidence from testing) can only increase confidence in a safety claim, and do not directly reduce risk. In establishing the overall claimed position of the software safety argument however, some of the argument claims can be recognised as being more important than others. For example claims regarding the behaviour of an architectural component (such as a voter), which carries a greater responsibility for risk reduction than other components, are more important to the overall software safety argument. Therefore claims relating to those components would require a greater degree of assurance (more confidence must be established). Safety standards such as [21], and [10] describe how safety integrity requirements may be defined for software functions or components. These safety integrity requirements define the integrity or reliability required in order to support the safety of the system. Where safety integrity requirements have been defined, they can be used as a way of determining the importance of the software safety argument claim to which they relate.

We therefore propose that the impact of an assurance deficit should be determined in two stages. Firstly, considering the claim to which the identified assurance deficit relates, the importance of the truth of that claim to establishing the claimed risk position of the system must be considered. Secondly, it is necessary to determine the extent to which the identified assurance deficit affects the confidence achieved in that safety claim. The question to be asked here is, how strongly does the assurance deficit undermine the confidence achieved in the claim? Not all information relevant to a claim leads to the same increase in confidence in that claim. It is necessary to assess the extent to which any information provided to address the assurance deficit might increase confidence in the truth of the claim. One way to approach this is to consider which aspects of the claim are still assured in the presence of the assurance deficit (due to other evidence or information), and which are not.

Knowing the importance of the truth of the claim or claims in establishing the claimed risk position, and the relative importance of the assurance deficit to establishing the truth of that claim, it then becomes possible to determine the overall impact of the assurance deficit. In a similar manner to risks in the ALARP approach, the impact of the identified assurance deficits may be usefully classified into three regions. An “intolerable” as-
surance deficit could be one whose potential impact on the claimed risk position is too high to be justified under any circumstances. Such assurance deficits must be addressed. In contrast, assurance deficits may be categorised as broadly acceptable if the impact of the assurance deficit on the claimed risk position is considered to be negligible i.e. the “missing information” has a negligible impact on the overall confidence in the system safety argument. In such cases no additional effort to address the assurance deficit need be sought. Finally, a potentially “tolerable” assurance deficit is one whose impact is determined to be too high to be considered negligible, but which is also not necessarily considered to be intolerable. For a potentially “tolerable” assurance deficit it may be considered acceptable only if the cost of taking measures to address that assurance deficit are out of proportion to the impact of not doing so. The greater the impact of the assurance deficit, the more, system developers may be expected to spend in addressing that deficit.

Note that the impact of an assurance deficit can only be determined on a case-by-case basis for a specific argument relating to a particular system. The same type of assurance deficit (such as a particular assumption) whose impact is categorised as broadly acceptable when present in the software safety argument for one system, may be considered intolerable when present in the argument for a different system. This is because the impact of an assurance deficit considers it’s impact in terms of the overall safety of the system. It is for this reason that particular argument approaches cannot be stated as sufficient for particular claims, but must be adapted on each use to be appropriate for the particular application. This is discussed in more detail in section 4.

As has already been discussed, addressing an assurance deficit requires ‘buying’ more information or knowledge about the system relevant to the safety claims being made. There will typically be a cost associated with obtaining this information. For those assurance deficits categorised as tolerable, in a manner similar to that adopted for an ALARP assessment process (such as that described in [24]), the value of the information in building confidence in the safety case must be considered when deciding whether to spend that money. In theory it is possible to do a formal cost-benefit analysis based on a quantitative assessment of the costs associated with the available options for addressing the assurance deficit, and the costs associated with the potential impact on the claimed risk position (such as the necessity to provide additional system level mitigations). In many cases however, there will simply be value to be gained through a qualitative consideration of these issues. It should be noted that even for ALARP assessments of conventional systems, a comparison with good
practice, and qualitative arguments will often be claimed before turning to a qualitative first principles argument [9]. In all cases an explicit justification should be provided as to why the residual assurance deficit is acceptable and, wherever appropriate, an argument should be used to provide this justification. Section 4 provides an argument pattern for constructing an argument to justify that the residual assurance deficits are appropriate.

The approach described above, although similar to ALARP, rather than considering the necessity of adopting measures to directly decrease risk, instead considers the necessity of increase the confidence in safety claims by addressing assurance deficits. As such the framework could be considered to help establish a claimed risk position in the software safety case that is ACARP (As Confident As Reasonably Practicable).
4 SOFTWARE SAFETY ARGUMENT PATTERNS

4.1 An introduction to argument patterns

Software safety argument patterns provide a way of capturing good practice in software safety arguments. Patterns are widely used within software engineering as a way of abstracting the fundamental design strategies from the details of particular designs. The use of patterns as a way of documenting and reusing successful safety argument structures was pioneered by Kelly in [13]. Kelly proposes extensions to GSN that can be used to support the abstractions necessary to capture patterns of argument. These patterns can be instantiated to create an argument for the specific system under consideration. Kelly also proposes an approach to documenting the patterns.

To create patterns, GSN is extended to support multiplicity, optionality and abstraction. The multiplicity extensions shown in figure 16 are used to describe how many instances of one entity relate to another entity. They are annotations on existing GSN relational arrows.

<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>A solid ball is the symbol for many (meaning zero or more). The label next to the ball indicates the cardinality of the relationship.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A hollow ball indicates “optional” (meaning zero or one).</td>
</tr>
<tr>
<td></td>
<td>A line without multiplicity symbols indicates a one to one relationship (as in conventional GSN).</td>
</tr>
</tbody>
</table>

Figure 16: GSN multiplicity extensions

The optionality extension shown in figure 17 is used to denote possible alternative support. It can represent a 1-of-n or an m-of-n choice.

The abstraction extensions shown in figure 18 allow GSN elements to be generalised for future instantiation.

Kelly also details a method of documenting patterns, such that the information necessary for their successful instantiation is captured. He suggests that for each pattern, information is documented under the following headings:
Figure 17: GSN optionality extensions

**Pattern name** The name of the pattern should communicate the central argument being presented in the pattern.

**Intent** Should state what the pattern is trying to achieve.

**Motivation** Communicates why the pattern was constructed.

**Structure** Here GSN is used to present the structure of the argument pattern.

**Participants** Provides additional information on each of the elements of the GSN argument.

**Applicability** Records under what circumstances the pattern can and should be applied.

**Consequences** What remains to be done after having applied the argument pattern.

**Implementation** This should describe how to implement the pattern, in particular providing hints and techniques for the successful application of the pattern, describing ways in which it is possible to get it wrong, and recording any common misinterpretations of the terms or concepts used.

**Related patterns** Identify any related safety argument patterns.

Figure 19 shows part of an example safety argument pattern taken from [13].
As with software design, software safety argument patterns can be used to abstract the fundamental argument strategies from the details of a particular argument. Software safety argument patterns can therefore be very useful as a tool for determining the options for strategies that might be adopted to support particular claims made in the argument. This helps to identify argument strategies which might be sufficiently compelling, and provides a method for identifying assurance deficits since the patterns provide a guide as to the nature of the information that may be missing from a chosen strategy.

There exist a number of examples of safety argument patterns. Kelly himself developed an example safety case pattern catalogue which provided a number of generic solutions identified from existing safety cases, such as an ALARP argument pattern (see figure 19), and a Control system architecture breakdown pattern. Although providing a number of useful
Figure 19: Extract from the Kelly ALARP pattern

generic argument strategies, Kelly acknowledges that this catalogue does not provide a complete set of patterns for developing a safety argument, it merely represents a cross-section of useful solutions for unconnected parts of arguments. Kelly’s pattern catalogue does not deal specifically with any software aspects of the system.

The safety argument pattern approach was further developed by Weaver [29], who specifically developed a safety pattern catalogue for software. The crucial differences with this catalogue were firstly that the set of patterns in the catalogue were specifically designed to connect together in order to form a coherent argument. Secondly the argument patterns were developed specifically to deal with the software aspects of the system. There are a number of weaknesses that have been identified with Weaver’s pattern catalogue. Firstly, the argument patterns take a fairly narrow view, focussing on the mitigation of failure modes in the design. Secondly, the patterns present an essentially "one size fits all" approach, with little guidance on alternative strategies, or how the most appropriate option is determined.

A software safety pattern catalogue has also been developed by Ye [30], specifically to consider arguments about the safety of systems in-
Including COTS software products. Ye’s patterns provide some interesting developments to Weaver’s, including patterns for arguing that the evidence is adequate for the assurance level of the claim it is supporting. Although we do not necessarily advocate the use of discrete *levels* of assurance, the patterns are useful as they support the approach of arguing over both the trustworthiness of the evidence and the extent to which that evidence supports the truth of the claim.

### 4.2 Modular extensions to GSN

As originally proposed by Kelly in [14], modular safety cases provide a means of organising large or complex safety cases into separate but inter-related component modules of argument and evidence. When splitting an argument into modules it becomes necessary to be able to refer to goals which exist within other modules. To refer to goals in other modules, the GSN element “Away Goal” is used. Each away goal contains a module identifier, which is a reference to the module where the goal can be found (see figure 20). Away goals can only be used to reference goals that have explicitly been declared as *public* in another module (see figure 20). Away goals can be used as a way of providing support for a goal in one module, with a goal in another module (see figure 20). Away goals can also be used to provide contextual backing for goals, strategies and solutions (see figure 20).

Using modular GSN can be useful in helping to more easily understand and reason about large, complex safety arguments such as those encountered for software. In particular modular GSN can have a useful role in more clearly understanding argument assurance.

#### 4.2.1 The use of modular GSN in software safety arguments

Toulmin [28] represents arguments as containing the following basic elements:

**Claims** What position are you asking the reader to agree to as the outcome of your argument?

**Grounds** What kind of underlying foundation is required if a claim of this particular kind is to be accepted as solid and reliable?

**Warrants** How do you justify the move from these grounds to that claim?

Are the grounds just a lot of irrelevant information which have nothing to do with the claim in question?
Figure 20: Some modular extensions to GSN

**Backing** What information do you have to back up your trust in a particular warrant?

The relationship between these elements is shown in figure 21.

It is fairly easy to see the direct relationship between claims in Toulmin’s description and goals in GSN. Similarly, the grounds can directly relate to GSN solutions. At any level of decomposition in the GSN argument, the grounds can also be the sub-goals supporting a parent goal. In this way GSN goals can play both the role of a claim (when considering its support from other claims) and grounds (when considering the support it provides to other claims). The role of the warrant is to justify that the grounds are sufficient to support the claim. As such the warrant does not correspond to a GSN strategy (which simply describes the relationship between the grounds and the claim), but to the justification of that strategy. The warrant is key to gaining confidence in the claim based on the grounds put forward.
Figure 21: Toulmin’s representation of an argument

Back ing provides a way of gaining assurance, by providing support for why the warrant is sound (and hence why the claim should be believed).

Modular GSN provides an excellent mechanism for representing these key aspects of the argument (warrant and backing). An away goal in context to the strategy can be used to explicitly capture the warrant for that strategy. An argument can then be provided (in a separate argument module) as to why that warrant is sufficient, i.e. the backing. Lack of confidence in warrants is a key way in which assurance may be lost, modular GSN provides a way of capturing warrants and backing arguments in an explicit manner. Throughout the software safety case patterns in this report, modular GSN is used to capture the argument warrants. By having such backing arguments in a separate module, it more clearly separates the main thread of argument (the grounds and claims) from the warrants and backing which provide assurance to that argument.
4.3 A software safety argument pattern catalogue

In this section a set of software safety argument patterns are proposed. These patterns are intended to capture current good practice for compelling software safety arguments, and have been constructed based on existing patterns, and an understanding of current practice for software safety arguments. They have been developed such that they highlight as clearly as possible where assurance may be gained and lost during the development of the argument.

The instantiation of the patterns for a particular system must always be carried out within the framework of the extended six-step method (section 3.1) in order to ensure that sufficient assurance is achieved from the application of the patterns.
4.3.1 High-level software safety argument pattern

<table>
<thead>
<tr>
<th>High-Level Software Safety Argument Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author</strong></td>
</tr>
<tr>
<td><strong>Created</strong></td>
</tr>
</tbody>
</table>

**INTENT** This pattern provides the high-level structure for a software safety argument. The pattern can either be used to create the high level structure of a ‘stand alone’ software safety argument considering just the software aspects of the system, or alternatively can be used to support claims relating to software aspects within a broader system safety argument.

**STRUCTURE**

Note that there are a number of different possible top goals for this pattern, as indicated by the public goals in the argument structure below.
Goal: SwSystem Safe

(software Y) is acceptably safe to operate within (system Z)

Con: Sw

(Description of software Y)

Ass: hazards

All system hazards have been correctly identified

Goal: swContributionAcc

The contribution made by (software Y) to (system Z) hazards is acceptable

Strat: swContributionAcc

Argument over each hazard to which (software Y) may contribute

Goal: Hazard

Software contribution(s) to (Hazard) is acceptably managed

Con: hazards

(Description of hazards to which (software Y) may contribute)

Con: safetyRqt

Software contribution may in some cases be defined as an explicit safety requirement

Strat: contMit

Argument over each identified software contribution to (Hazard)

Goal: contIdent_contIdent

The ways in which (software Y) may contribute to (Hazard) are completely and correctly identified

Con: contIdent

(Description of the ways in which (software Y) may contribute to (Hazard))

Con: contributions

(number of identified software contributions to (Hazard))

Goal: sw contribution

(software contribution) to (Hazard) is acceptably managed

Con: safetyRqt

Software contribution may in some cases be defined as an explicit safety requirement
**PARTICIPANTS**

**Goal: SwSystem Safe** If a stand alone software safety argument is being produced then this goal should be used as the top goal in the argument since it clearly sets out the overall objective of the software safety argument. It is necessary to provide the three items of context to make the scope of the software safety argument clear to the reader. This goal has been designated as a public goal to indicate that it may be used as the top goal in the argument.

**Goal: swContributionAcc** This goal makes it clear that a hazard directed approach is adopted, by considering the contributions made by the software to the system’s hazards. If the pattern is being used as part of a system safety argument, then this goal may provide the link in to that argument (hence a public goal). This would be the case if the system safety argument considers the contribution of the software all in one place. It is not necessary to include the context to provide descriptions of the system and the software if this is already clear from the system safety argument.

**Ass: hazards** The system hazards can only be identified at the system level. Identification of system hazards is therefore outside of the scope of the software safety argument. It is acceptable therefore to make this assumption as long as the assumption is demonstrated elsewhere at the system level. If an argument to support this assumption exists with a system safety argument then it would be appropriate to link to that argument at this point instead of making an assumption.

**Strat: swContributionAcc** To ensure traceability from the software to system hazards, the strategy adopted is to argue explicitly over each of the hazards identified at the system level.

**Goal: Hazard** For each hazard there may be one or more potential contributions from the software identified at the system level. An instance of this goal is created for each of the system hazards to which the software may contribute. At the system level the software will only be considered from a ‘black-box’ point of view, so the contribution may be identified in the form of high-level functionality, or safety requirements. These contributions would be considered base events at the system level, and would not generally be developed further in a system level argument.
**Goal: contIdent** It is necessary to ensure that all the software contributions are correctly identified at the system level. This is crucial to the assurance of the argument as it provides the warrant for the adopted strategy of arguing over the software contributions. This goal provides context to the strategy contMit and must be supported by an argument contained in a separate module (contIdent). Software contributions are often identified as base events in a fault tree analysis performed at the system level. The argument in module contIdent would, in such a case, reason about the rigour and suitability of that analysis.

**Goal: sw contribution** An instance of this goal is created for each of the identified software contributions to each of the system hazards. The Software contribution safety argument pattern may be used to generate an argument to support this goal.

**APPLICABILITY** This pattern should be applied whenever a software safety argument is required as part of a safety case.

**CONSEQUENCES** Once this pattern has been instantiated, a number of elements will remain undeveloped and requiring support. Firstly ‘Goal: sw contribution’ must be supported. The Software contribution safety argument pattern presented in this catalogue can be used to support this goal. In addition, an argument to support ‘Goal: contIdent’ must also be developed in module contIdent. This argument will be based on analysis performed at the system level, so in some cases a sufficient argument may exist at the system level which can be used to support this claim.

**IMPLEMENTATION** There are a number of different possible top goals for this pattern, as indicated by the public goals. The appropriate top level goal for the argument must be determined through consideration of the structure of any system safety argument which the software safety argument supports. If the pattern is being used to support a system level safety argument, the top goal from this pattern may not actually appear at the top of the overall argument structure. Instead it will appear as a child-goal within the system safety argument. It is important that a stand alone software safety argument begins with the top goal ‘Goal: swSystem Safe’ to capture the overall objective of the argument and all the required contextual information.

**Possible pitfalls** The software contributions may not have been adequately
identified at the system level. This may then necessitate further analysis at the system level. It is therefore clearly advantageous to ensure software is considered as part of the system level safety activities.

**RELATED PATTERNS** This pattern is supported by the **Software contribution safety argument pattern**.
4.3.2 Software contribution safety argument pattern

**Software Contribution Safety Argument Pattern**

<table>
<thead>
<tr>
<th>Author</th>
<th>Richard Hawkins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Created</td>
<td>09/12/08</td>
</tr>
<tr>
<td>Last modified</td>
<td>08/06/09</td>
</tr>
</tbody>
</table>

**INTENT** This pattern provides the structure for arguments that the contributions made by software to system hazards are acceptably managed.

**MOTIVATION** It is necessary to consider all of the ways in which errors may be introduced into the software which could lead to the software contribution. The software development process used will vary between different projects, however in all cases the software development is undertaken through varying levels of design. At each level the design must satisfy requirements of the higher level. These requirements may be explicitly captured as part of a requirements specification, or identified implicitly from the design itself. In [11] Jaffe et al propose an extensible model of development which captures this relationship between components at different tiers\(^5\). Figure 22 illustrates the multi-tiered relationship between successively more detailed requirements and design information. Figure 23 illustrates in more detail the relationship among a tier n component’s requirements, its design representation, and the tier n+1 requirements of the tier n+1 (sub) components identified in the design representation.

\(^5\)The term *tier* is used predominantly to avoid the potential confusion of the term ‘level’.
Figure 22: Illustration of a multi-tiered relationship

Figure 23: More detailed illustration for a tier n component
From a safety perspective, it is necessary to ensure that at each tier, the software safety requirements derived at the previous tier are adequately addressed. This involves making design decisions which mitigate potential failures and adequately allocating and decomposing the software safety requirements (SSRs) through consideration of the design at that tier. At each tier it is also possible to introduce errors into the software which could manifest themselves as hazardous failures. It is therefore important in the software safety argument to also consider additional hazardous contributions that may be introduced at each tier.

This pattern therefore reflects the tier model discussed above in order to make it generally applicable to a broad range of development processes.

STRUCTURE
PARTICIPANTS

**Goal: sw contribution** An instance of this goal is required for each of the identified software contributions to each of the system hazards. For this top claim in the pattern, \{tier n\} will in this case refer to the highest tier in the development process. This highest tier is generally referred to as (high-level) software requirements.

**Strat: sw contribution** The strategy adopted is to argue over all the SSRs which are identified at this tier. These SSRs are either derived from the DSSRs of the previous tier, or through consideration of additional hazardous contributions that may occur at this tier.

**Goal: SSRidentify** The SSRs from the previous tier must be allocated to the tier n design appropriately, having been suitably decomposed where necessary, and correctly apportioned across the design as part of that decomposition. The SSRs may also require interpretation to reflect the tier n design. As part of supporting this goal it is necessary to consider the design decisions that are taken in order to mitigate failures, including mechanisms for failure detection and response. At the highest tier, there are no SSRs from the previous tier, instead, the software contribution itself must be considered. This goal is crucial to the assurance of the argument as it provides the warrant for the adopted strategy of arguing over SSRS identified for tier n. This goal must be supported by an argument contained in a separate module (SSRidentify). The SSR identification software safety argument pattern may be used to generate an argument to support this goal. This goal is optional, since it may not necessarily be required to provide direct traceability at every tier. The decision as to whether this is required at a particular tier must be based on a consideration of assurance. It may be necessary to justify such a decision by providing an argument. The Argument justification software safety argument pattern may be used to provide such an argument. It would be possible, instead of supporting this goal, to simply provide an assumption node stating that SSRs from the previous tier have been adequately allocated, decomposed, apportioned and interpreted. This would however significantly reduce the assurance achieved, so the impact of such a decision must be considered (see section 3.2).

**Goal: hazCont** At any tier in the development there is the possibility of introducing additional contributions to hazards due to errors made
at that tier. This goal claims that such potential hazardous contributions are addressed through the specification of additional SSRs. Supporting this goal requires that the potential hazardous contributions at tier n are adequately identified, and that SSRs sufficient to address those hazardous contributions are specified. This goal is crucial to the assurance of the argument as it provides the warrant for the adopted strategy of arguing over SSRs identified for tier n. This goal must be supported by an argument contained in a separate module (hazCont). The **Hazardous contribution software safety argument pattern** may be used to generate an argument to support this goal. This goal is optional, since it may not necessarily be required to identify hazardous contributions at every tier. The decision as to whether this is required at a particular tier must be based on a consideration of the impact on assurance. It may be necessary to justify such a decision by providing an argument. The **Argument justification software safety argument pattern** may be used to provide such an argument. It would be possible, instead of supporting this goal, to simply provide an assumption node stating that DSSRs have been correctly identified at tier n to address the identified potential additional hazardous contribution. This would however significantly reduce the assurance achieved, so the impact of such a decision must be considered (see section 3.2).

**Goal: SSRnAddn** An instance of this goal is created for each SSR identified at tier n (represented as SSRn). There is an option for how this goal is supported. It can be supported by either, or both of goals ‘SSRnSat’ and ‘SSRnAddn+1’. It may be necessary to justify such a decision by providing an argument. The **Argument justification software safety argument pattern** may be used to provide such an argument.

**Goal: SSRnAddn+1** It is possible to demonstrate that the SSRs at tier n are addressed by showing traceability down to the subsequent tier of development. The argument then continues through a further instantiation of ‘Strat: sw contribution’. {tier n+1} then becomes {tier n}.

**Goal: SSRnSat** It is possible at any tier to provide verification evidence of the satisfaction of the SSRs for that tier. This may be, for example, testing or analysis performed at that tier. Not all software is subject to the same number of tiers of development. Also, not all aspects of any particular software are necessarily developed over the same
number of tiers. It is therefore also possible for implementation to occur at any tier. At the tier of implementation it is possible to provide argument and evidence to demonstrate that the SSR is satisfied by the implementation. As discussed in 4.3.6, detailed guidance on the development of the argument to support this goal will be the subject of future work.

**APPLICABILITY** This pattern should be applied as part of any hazard-directed software safety argument.

**CONSEQUENCES** Once this pattern has been instantiated, a number of elements will remain undeveloped and requiring support. ‘Goal: SSRI-identify’ must be supported. The **DSSR identification software safety argument pattern** presented in this catalogue can be used to support this goal. ‘Goal: hazCont’ must be supported. The **Hazardous contribution software safety argument pattern** presented in this catalogue can be used to support this goal. Finally ‘Goal: SSRhSat’ must be supported. As discussed, detailed guidance on the development of the argument to support this goal will be the subject of future work.

**IMPLEMENTATION** This pattern should be instantiated as part of a software safety argument. An instantiation of ‘Goal: SSRIIdentify’ must be created for each identified software contribution to each system hazard. \{tier n\}, and \{tier n+1\} must be instantiated with the names of the relevant tier. Note that as the argument is developed over multiple tiers, \{tier n\} will refer to different tiers. \{SSRn\} is used to refer to a SSR at tier n, and should be instantiated with the SSR itself or a unique identifier for the SSR. Note that in this pattern the looping link represents a repeating pattern of argument, and would *not* appear in such a manner in an instantiated argument.

**Possible pitfalls** Whilst acknowledging that in many cases not all the optional goals may be provided at each tier, it is also important to note the significance of this pattern on the achieved assurance. Assurance deficits introduced in instantiating this pattern can have a potentially large impact. In such cases the additional support may prove necessary. It is therefore important that the assurance impact of decisions taken at each tier of development are fully considered, to avoid additional work at a later date. (See section 3.2).

**RELATED PATTERNS** Consideration should be given to the application of the **Argument justification software safety argument pattern** wherever
significant decisions about how to instantiate the optional aspects of this pattern are made. The Argument justification software safety argument pattern should be instantiated in context to this pattern to justify the acceptability of any residual assurance deficits as a result of the instantiation decisions.

This pattern supports the High-level software safety argument pattern. Support for ‘Goal: SSRIdentify’ and ‘Goal: hazCont’ can be provided using the SSR identification software safety argument pattern and the Hazardous contribution software safety argument pattern respectively.
4.3.3 SSR identification software safety argument pattern

| SSR Identification Software Safety Argument Pattern |
|---------------------------------|---------------------------------|
| **Author**                     | Richard Hawkins                |
| **Created**                    | 09/12/08                       |
| **Last modified**              | 08/06/09                       |

**INTENT** This pattern provides the structure for arguments that software safety requirements (SSRs) from a previous tier of development have been adequately captured at the next tier of development through the allocation, decomposition, apportionment or interpretation of the SSRs from the previous tier. This is achieved either through making design decisions which mitigate the SSR, or through the definition of additional SSRs.

**STRUCTURE**
Goal: SSRidentify
SSRs from (tier n-1) have been adequately allocated, decomposed, apportioned and interpreted at (tier n).

Con: SSRn-1
(SSRs from (tier n-1))

Strat: SSRidentify
Argument over the (tier n) design and the SSRs specified at (tier n).

Goal: designDecisions
Design decisions taken at (tier n) facilitate satisfaction of the SSRs from (tier n-1).

Con: tierNdesign
(tier n) design

Goal: SSRcapture
SSRs at (tier n) adequately capture the SSRs from (tier n-1) for the (tier n) design.

Con: SSRn
(SSRs identified for (tier n))

Strat: SSRcapture
Argument over each SSR from (tier n-1).

Con: tierNdesign
(tier n) design

Goal: SSRn
(SSRn-1) adequately captured by one or more SSRn.

Con: relevantSSRs
SSRs relevant to (design decision) are (SSRs).

Strat: designDecisions
Argument over design decisions taken at (tier n).

Con: designDecisions
Design decisions taken at (tier n) facilitate satisfaction of the SSRs from (tier n-1).

Con: relevantSSRs
SSRs relevant to (design decision) are (SSRs).

Strat: designDecisions
Design decisions taken at (tier n) facilitate satisfaction of the SSRs from (tier n-1).

number of relevant design decisions
PARTICIPANTS

Goal: SSRidentify This is a public goal in a separate argument module which can be referenced from other software safety argument modules using an away goal reference. This claim is applicable wherever an argument is being presented over the tiers of the software development lifecycle. \{tier n\} refers to the current tier being considered in the argument. \{tier n-1\} refers to the previous tier of development. At each tier it is necessary to demonstrate that the SSRs from \{tier n-1\} are adequately captured in the design of \{tier n\}.

Strat: SSRidentify This is achieved either through making design decisions at \{tier n\} which facilitate the satisfaction of the \{tier n-1\} SSR, or through the definition of SSRs for \{tier n\} which consider the \{tier n\} design. In some cases a mixture of appropriate design decision and SSR definition might be required to capture all of the \{tier n\} SSRs. In other cases just one approach may be sufficient, this will depend on a number of factors including the nature of the SSRs, which tier is being considered and the nature of the design of \{tier n\}. The Argument justification software safety argument pattern may be used to justify the adopted strategy.

Goal: SSRcapture This goal claims that the design of \{tier n\} has been considered in order to define SSRs for \{tier n\} which adequately capture the SSRs from \{tier n-1\}.

Goal: designDecisions It may be possible to facilitate the satisfaction of some of the \{tier n-1\} SSRs through decisions taken in the design of \{tier n\}. For example, a decision to have redundant components may be taken in order to help satisfy a SSR relating to the availability of an item of data. This goal allows claims to be made that such decisions reflect the SSRs from \{tier n-1\}.

Con: tierNdesign The design of \{tier n\} will be determined by the design decisions made, some of which may have been influenced by \{tier n-1\} SSRs. The \{tier n\} design will also determine the nature of the SSRs defined at \{tier n\}. This context is therefore common to both ‘Goal: SSRcapture’ and ‘Goal: designDecisions’.

Goal: SSRn An instance of this goal is created for each SSR from \{tier n-1\}. To adequately reflect each \{tier n-1\} SSR, one or more SSRs may be required at \{tier n\}. 
Goal: HSFMdetect One of the key ways in which the design at {tier n} may facilitate satisfaction of the {tier n-1} SSRs is through the detection and handling of failures which may lead to a breach of a SSR. This optional goal makes a claim that such detection and handling mechanisms at {tier n} are sufficient to address SSR breaches. Clearly this goal will not be relevant at all tiers.

Goal: SSRprevent Another key way in which the design at {tier n} may facilitate satisfaction of the {tier n-1} SSRs is through design features which prevent interference between components at {tier n} (for example through physical or logical partitioning). Interference between components has the potential to prevent SSRs from being satisfied. This optional goal makes a claim that interference at {tier n} will not prevent SSRs from being fulfilled.

Goal: designDecision An instance of this goal is created for each design decision taken which is relevant to the satisfaction of a SSR from {tier n-1}. Each instance of this goal requires a supporting argument which demonstrates how the design feature supports the SSR satisfaction.

APPlicability This pattern should be applied as part of any hazard-directed software safety argument to provide a warrant for an argument that SSRs from one development tier are adequately addressed at the next tier.

Consequences Once this pattern has been instantiated, a number of elements will remain undeveloped and requiring support. An instance of ‘Goal: SSRn’ must be supported for each SSR from {tier n-1}. An argument should be provided which demonstrates that one or more SSRs specified at {tier n} adequately capture the {tier n-1} SSR for the design at {tier n}. An instance of ‘Goal: designDecision’ must be supported for each design decision which was made to facilitate the satisfaction of SSRs at {tier n}. ‘Goal: HSFMdetect’ and ‘Goal: SSRprevent’, if created, must also be supported.

Implementation {tier n}, and {tier n-1} must be instantiated with the names of the relevant tier. This could for example be class design and high-level software design respectively.

Possible pitfalls The SSRs defined at {tier n} must adequately reflect the {tier n} design. If that design changes, it is necessary to check that the
SSRs defined at \{tier n\} are still valid, and if necessary update the SSRs to reflect the design changes. For this reason it would be advantageous to have a reasonably stable design for \{tier n\} before defining SSRs for that tier. Since the SSRs from \{tier n-1\} may influence the design, it is important that this is considered early in the design of \{tier n\}, such that any resulting design changes are not required late in the development.

**RELATED PATTERNS** This pattern is used to provide context to the **Software contribution safety argument pattern**. Consideration should be given to the application of the **Argument justification software safety argument pattern** wherever significant decisions about how to instantiate the optional aspects of this pattern are made.
4.3.4 Hazardous contribution software safety argument pattern

### Hazardous Contribution Software Safety Argument Pattern

<table>
<thead>
<tr>
<th>Author</th>
<th>Richard Hawkins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Created</td>
<td>09/12/08</td>
</tr>
<tr>
<td>Last modified</td>
<td>08/06/09</td>
</tr>
</tbody>
</table>

**INTENT** This pattern provides the structure for arguments that potential hazardous failures at {tier n} are addressed through the definition of SSRs.

**MOTIVATION** At each tier of software development it is possible to introduce errors into the software which could manifest themselves as hazardous failures. It is therefore important in the software safety argument to consider additional hazardous contributions at each tier. This goal claims that the SSRs defined at {tier n} are sufficient to address these additional hazardous software contributions.

**STRUCTURE**
Goal: hazCont
SSRs at {tier n} address the potential additional hazardous contribution identified at {tier n}

Con: deviations

Strat: HSFMident
Argument over deviations from intended behaviour at {tier n}

Con: deviations

Goal: HSFMaddress
[HSFM] sufficiently addressed by one or more [SSRn]

Goal: HSFMident
HSFMs correctly identified at {tier n}

Con: HSFMs

[LS (tier n) HSFMs]

Strat: SSRe-derived
SSRs sufficient to address identified HSFMs are defined

Goal: deviations

Potential deviations from intended behaviour are sufficiently identified at [tier n]
PARTICIPANTS

Goal: hazCont This is a public goal in a separate argument module which can be referenced from other software safety argument modules using an away goal reference. This claim is applicable wherever an argument is being presented over the tiers of the software development lifecycle. \{tier n\} refers to the current tier being considered in the argument. This goal claims that SSRs sufficient to address the potential hazardous contributions identified at \{tier n\} are identified. The goal is supported by demonstrating that hazardous software failure modes (HSFMs) (that is failure of the software which could contribute to a hazard at the system level) introduced at \{tier n\} are sufficiently identified, and that each of these HSFMs is addressed through the definition of one or more SSRs.

Strat: HSFMident The HSFMs for \{tier n\} are identified by considering the potential deviations from intended behaviour that may occur at \{tier n\} as a result of an error introduced at that tier. Assurance is gained in this strategy through the provision of a warrant about the thoroughness and rigour of the deviation identification. This warrant is supported through ‘Goal: deviations’.

Goal: deviations This goal must be supported by argument and evidence that the potential deviations from the intended behaviour at \{tier n\} have been sufficiently identified. There are various techniques available for identifying deviations from intended behaviour in software designs (such as software HAZOP [25]). The particular technique which it is most appropriate to use will depend upon the tier being considered, and also upon the nature of the software design itself. Appendix B provides some examples of the types of hazard and failure analysis techniques that may be used at some of the possible tiers. This goal should be supported by an argument contained in a separate module (deviations)

Goal: HSFMaddress An instance of this goal is created for each of the HSFMs identified at \{tier n\}. Each instance of this goal requires a supporting argument which demonstrates that the DSSRs derived are sufficient to mitigate that HSFM.

APPLICABILITY This pattern should be applied as part of any hazard-directed software safety argument to provide a warrant for an argument that SSRs from one development tier are adequately addressed at the
CONSEQUENCES Once this pattern has been instantiated, a number of elements will remain undeveloped and requiring support. ‘Goal: deviations’ must be supported by an argument provided in a ‘deviations’ safety argument module. An instance of ‘Goal: HSFMaddress’ must be supported for each HSFM identified at {tier n}. ‘Goal: HSFMs’ must also be supported.

IMPLEMENTATION The techniques most appropriate to use to identify potential deviations from intended behaviour at each tier will vary. Appendix B provides some examples of the types of hazard and failure analysis techniques that may be used at some of the possible tiers.

RELATED PATTERNS This pattern is used to provide context to the Software contribution safety argument pattern.
4.3.5 Argument justification software safety argument pattern

<table>
<thead>
<tr>
<th>Argument Justification Software Safety Argument Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author</td>
</tr>
<tr>
<td>Created</td>
</tr>
</tbody>
</table>

**INTENT** This pattern provides the structure for arguments that the software safety argument presented is sufficient. This is done by arguing that the impact of residual assurance deficits is acceptable. Section 3.2 considers assurance deficits in more detail. This argument pattern is instantiated in context to the main argument. It should be used at any point in the argument at which it is felt that a reader may require such justification that the presented argument is sufficient.

**ALSO KNOWN AS** ACARP argument pattern.

**MOTIVATION** As discussed in detail in section 3.2, assurance deficits may be identified at any step in the construction of a software safety argument. This pattern provides a way of explicitly providing a justification for the acceptability of the argument based upon a consideration of the impact of those assurance deficits.

**STRUCTURE**
Goal: Justification
Adopted argument is adequate

Start: Justification
Argument over assurance deficits

Goal: AssDefTolerable
All assurance deficits whose impact is potentially tolerable are acceptable or have been sufficiently addressed

m+n+p=total no. of identified assurance deficits

Goal: AssDefIdentify
Relevant assurance deficits have been correctly identified

Goal: AssDefNegligible
The impact of any remaining assurance deficits is negligible

Goal: AssDefIntolerable
There are no assurance deficits whose impact is intolerable

Goal: AssDefAddress
{Assurance deficit} has been sufficiently addressed by the additional measures taken

1-of-2

Goal: Cost
Cost of taking further measure to address {assurance deficit} is out of proportion to the impact of not doing so

Goal: ImpactTolerable
Impact of {assurance deficit} has been shown to be potentially tolerable

Goal: Acceptable
Potentially tolerable {assurance deficit} is acceptable

Goal: Cost
Cost of further measures possible to address {assurance deficit} have been considered

Further measures

Goal: ImpactNegligible
Impact of {assurance deficit} has been shown to be negligible

Goal: AssDefEliminate
{Assurance deficit} has been eliminated

Goal: AssDefAccept
Impact of {assurance deficit} is tolerable and acceptable

Strat: AssDefTolerable
Argument over an assessment of the associated costs and impact

Goal: AssDefImpact
The impact of assurance deficits on the claimed risk position of the argument have been determined

Goal: CostDetermine
The cost of potential measures for addressing {assurance deficit} have been correctly determined

Goal: AssDefTolerable
All assurance deficits whose impact is potentially tolerable are acceptable or have been sufficiently addressed

Goal: Justification
Argument over assurance deficits

Con: Justification
(Scope of argument to which the justification relates)

n=no. of assurance deficits whose impact was initially considered potentially tolerable

p=no. of assurance deficits whose impact was initially considered negligible

m=no. of assurance deficits whose impact was initially considered intolerable
PARTICIPANTS

Goal: Justification This is a public goal in a separate argument module which can be referenced from any other software safety argument modules using an away goal reference. The goal claims that the adopted argument is adequate.

Strat: Justification To argue that the software safety argument is adequate, the assurance deficits are considered. Each assurance deficit is either considered potentially tolerable, intolerable or negligible. Section 3.2 provides more guidance on categorising assurance deficits. To support this strategy, it is necessary to argue that the relevant assurance deficits have been correctly identified, and that the impact of those assurance deficits on the claimed risk position have been determined.

Goal: AssDefIdentify This goal should be supported by an argument contained in a separate module (AssDefIdentify), which demonstrates that the assurance deficits which could challenge the claims at that point in the argument have been identified correctly. Section 3.2 provides more details on the identification of assurance deficits.

Goal: AssDefImpact This goal should be supported by an argument contained in a separate module (AssDefImpact), which considers what the impact of each identified assurance deficit on the relevant claimed risk position of the overall software safety argument. Further details on determining the impact of assurance deficits is provide in Section 3.2.

Goal: AssDefTolerable For all those assurance deficits whose impact is categorised as potentially tolerable, it must be argued that they are either ‘acceptable’, or that they are sufficiently addressed.

Goal: AssDefIntolerable There must remain no residual assurance deficits in the argument whose impact is categorised as intolerable. It must therefore be argued that they are all sufficiently addressed.

Goal: AssDefNegligible No further measures need be taken to address assurance deficits whose impact is categorised as negligible. An argument why the assurance deficit is considered negligible should however be provided.
Goal: AssDefAddress An instance of this goal is created for each assurance deficit for which measures have been taken to address the deficit, an argument should be provided that those measure taken are sufficient. It should be argued either: that the measures taken have reduced the impact of the assurance deficit such that it can be considered tolerable and acceptable, that the measures taken have reduced the impact of the assurance deficit such that it can be considered negligible, or that the measures taken have eliminated the assurance deficit (the deficit no longer exists).

Goal: Cost An instance of this goal is created for each assurance deficit whose impact is categorised as potentially tolerable. For such assurance deficits to be acceptable, it must be argued that the cost of taking further measures to address the assurance deficit is out of proportion to the impact that not addressing the assurance deficit would have on the claimed risk position of the overall software safety argument. Further details on determining the acceptability of residual assurance deficits is provided in Section 3.2.

Goal: costDetermine provides a warrant to the cost benefit analysis, by arguing that the costs of the available options for addressing the assurance deficit have been determined correctly. The goal should be supported by an argument provided in a separate argument module.

Applicability This pattern may be used in support of any software safety argument to justify any strategy for which it is felt such justification may be required to convince a reader that the strategy used provides sufficient confidence in the claim it is supporting. The argument pattern is used in context to the strategy to which it relates.

Consequences Once this pattern has been instantiated, a number of elements will remain undeveloped and requiring support. ‘Goal: AssDefImpact’, ‘Goal: AssDeficitIdentity’ and ‘Goal: costDetermine’ must each be supported by arguments provided in separate safety argument modules. ‘Goal: Cost’, ‘Goal: ImpactNegligible’ and ‘Goal: AssDefEliminate’ must also be supported.

Implementation This pattern should be instantiated in context to a software safety argument.

Possible Pitfalls In a top-down argument development, the impact and cost associated with assurance deficit may need to be estimated based
on a prediction of the types of assurance deficits that are anticipated (such as a necessity to generate additional evidence). Although this may not be straightforward, it means that a number of alternative approaches to addressing assurance deficits (including design decisions) may be considered early in the development of the software.

Accurately estimating the costs associated with addressing assurance deficits may be difficult in reality. It is felt that further work is required in this area.

**RELATED PATTERNS** Although this pattern could be used to provide support to any safety argument, it is felt that it can provide particular benefit for justifying the acceptability of instantiations of the *Software contribution safety argument pattern.*
4.3.6 Evidence selection

It is necessary to provide solutions to support the claims made in the patterns presented in this section. In particular it is necessary to support ‘Goal: SSRnSat’ from the Software contribution safety argument pattern. Section 3.1 of this report has discussed the criteria by which the sufficiency of evidence put forward to support a claim may be judged. The example in Appendix B also discusses some of the types of evidence that may be applicable to support some of the claims put forward. The selection of specific evidence that may be sufficient to support the claims made in the software safety argument patterns is not, however, discussed in detail in this report. This will be the subject of further work scheduled under this SSEI task.

There has been other guidance provided on the selection of evidence to support software safety cases, including [2] and [7] which give useful examples of the kinds of techniques that might be used to support a software safety argument. Often however the justification for the appropriateness of the evidence is lacking or uncompelling. The framework, and patterns described in this report provide the basis for a more systematic way of determining what evidence is required based on the claims which need supporting. In particular we propose in further work for this task to begin to categorise evidence according to its appropriateness against a number of criteria including the type of SSRs, the development tier for which the evidence is required, the time in the lifecycle at which the evidence is provided, and its impact upon assurance.
5 CONCLUSIONS AND FURTHER WORK

This report has described a systematic approach to the construction of software safety arguments. This approach is based upon a catalogue of software safety argument patterns which provide guidance on compelling strategies or argument structures which might be adopted. These patterns are supported by a systematic framework for identifying, and justifying assurance deficits during the construction of an argument. Used together, these two aspects of the approach should help to ensure that the resulting software safety argument provides sufficient assurance.

Future work under this task will address a number of areas. Firstly the guidance needs to be extended to include the selection of sufficient evidence to support the software safety arguments, in particular gaining sufficient assurance in ‘Goal: SSRnSat’ from the strategic software safety argument pattern. This will be the focus of deliverable 2. It is also necessary to undertake a number of realistic case studies in order to validate the approach put forward in the guidance. The results of these case studies are planned to be reported in deliverables 3 and 4. In addition to this, it is intended that the guidance presented should not be seen as a static document, but that it should be updated as and when required to ensure that it continues to reflect current understanding of good practice. In particular, it is hoped that the software safety argument patterns will be updated and expanded over time. It is also intended that the software safety argument pattern catalogue should be extended to include software safety anti-patterns and guidance on their usage.
6 LIST OF REFERENCES


7 LIST OF ABBREVIATIONS

ACARP As Confident as Reasonably Practicable
ALARP As Low as Reasonably Practicable
CAE Claims, Argument and Evidence
DSSR Derived Software Safety Requirement
EPU Electronics and Power Unit
FFA Functional Failure Analysis
FHA Functional Hazard Assessment
GSN Goal Structuring Notation
HAZOP Hazard and Operability Studies
HSFM Hazardous Software Failure Mode
LRAAM Long Range Air-to-Air Missile
SHARD Software Hazard Analysis and Resolution in Design
SSEI Software Systems Engineering Initiative
A SOFTWARE SAFETY ARGUMENT STAKEHOLDER CONSULTATION

A.1 Introduction

This annex reports the findings of a series of interviews held with a range of software safety case stakeholders. These interviews were undertaken as part of the SSEI task to develop guidance on the construction of software safety cases. The aim of the interviews was to ensure that the guidance produced is effective in meeting the needs of both the MOD and industry by establishing a clear understanding of the current challenges facing, and guidance needs of, software safety case stakeholders in current projects. The interviews were conducted over the telephone, and were based, as far as possible on the interviewee’s experiences on projects with which they have been involved. Although a standard set of topics were discussed, the interviews were conducted in an open-ended manner to allow the interviewees to express their own opinions, with as little influence as possible from the interviewer.

A.2 The consultation process

A.2.1 Consulted stakeholders

A total of 11 software safety case stakeholders agreed to be interviewed as part of the consultation. Figure 25 provides more information about those participating. All those interviewed had an interest in software aspects of safety cases, some were specialists in software and software safety issues, others worked primarily at the system level. The main stakeholder roles (assessment, certification and development) were covered by the selected interviewees.

A.2.2 Stakeholder projects

The interviewees are involved in a wide range of projects including aircraft carrier electrical systems, aero engine control systems, UAVs, civil and military aircraft mission and avionics systems. The respondents were asked to base their responses on their experiences on actual projects wherever possible.

Figure 24 shows the software and safety standards that were followed
by each of the projects. In total 5 different standards were mentioned: IEC 61508 [10], Def Stan 00-56 Issue 2 [19] and issue 4 [21], Def Stan 00-55 Issue 2 [20] and DO-178B [26]. The interviewees were asked if they followed the standard because they were contracted to that standard, or for some other reason. This is also recorded in the table.

<table>
<thead>
<tr>
<th>Project</th>
<th>61508</th>
<th>0056-4</th>
<th>0055</th>
<th>DO178B</th>
<th>0056-2</th>
<th>Contracted?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project 1</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>Project 2</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>Contracted to 55/56 but worked to 178B and mapped across</td>
</tr>
<tr>
<td>Project 3</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>Project 4</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>Project 5</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>Project 6</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td>Contracted to 56 iss 4, use 55 and 56 iss 2 for guidance</td>
</tr>
<tr>
<td>Project 7</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Project 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Project 9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Project 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Project 11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

Figure 24: Software standards used

It is interesting to note that the only projects with a contractual obligation to comply with a standard were those contracted to one of the defence standards. Both of the projects that currently use Def Stan 00-56 Issue 4 found it necessary to use another standard (IEC61508 or Def Stan 00-55) as guidance for the software aspects of the system.

A.3 Consultation results

A.3.1 The software safety case product

Firstly the interviewees were asked about their experiences of software safety cases themselves. All except one of the respondents said that software aspects of the system were currently considered in the safety case for the projects with which they were involved. Of these there was a split between those who included the software aspects within the system safety case, and those who produced a separate software safety case which was referenced out from the system level case.

One of the respondents for whom software is considered as an integrated part of the system safety case commented that it is not necessarily...
particularly easy to specifically identify the software aspects of the safety case, even though they are present. Since the safety case is structured from an architectural point of view, the software aspects may be spread across the safety case in different sub-systems.

One of the respondents explained that separate documentation is produced for each of the programmable electronics systems because each of the teams responsible for that system produces its own documentation, which is then integrated by the system safety case.

Although safety cases are generally produced for the software aspects, not all of these take the form of a structured safety argument. Six of the respondents were currently using GSN in order to represent their arguments, all the other respondents said that there was an intention, or aspiration to move to an argument-based safety case in the future.

For most of the interviewees, the principal argument in the safety case was based on compliance with standards. One project relied solely on compliance with DO178B to demonstrate software safety. The rest of the respondents however chose to supplement this with additional evidence related specifically to the software product, however the amount of additional evidence generated varied significantly between the interviewees. The additional evidence generally focussed on specific software failures. Some comments were made that there was a lack of guidance provided on what these additional product-focussed safety activities should be.

ENSURING SUFFICIENCY OF SOFTWARE SAFETY CASES
The interviewees were asked for their experiences of judging the adequacy of the software safety cases generated for their projects. Most seemed fairly confident that it was clear what was expected from the safety case. In general the adequacy of the safety case was assessed through some kind of audit exercise. In most cases this involved verifying that the relevant standard had been complied with. It was noted by a number of respondents that for anything beyond the requirements of the standards, there was far less understanding of what was considered to be sufficient. One respondent noted that, to minimise the residual risk, they had wanted, in addition to following the objectives of DO178B, to carry out some software safety activities. He observed that there was very little guidance available on what those additional product-focused activities should be.

It was also noted that the reviewers would often tend to focus on standards compliance as it was a much easier thing to assess. To quote one respondent, they “tend to be looking for the easy things that they can check, like compliance with the “shall” statements in 00-55. Judging sufficiency of a safety argument is a much harder task, so they tend not to focus on that aspect so much [in the review]”. He added that the reviewers “didn’t seem to have a very clear understanding themselves of, for example, how much detail was needed, or how compelling it needs to be.”

It was generally accepted by all the projects that additional evidence would be required as a result of the assessment process before the software safety case would be accepted. This was very common, and indeed considered to be normal practice. It was noted by one of the interviewees that this can cause problems, particularly if the recommendations resulting from a review are not made until late in the lifecycle, when they can often be very hard to implement. It was observed that it is highly desirable that evidence requirements are defined as early in the project as possible.

**Effectiveness of Software Safety Cases**

The interviewees were questioned as to how effective they felt software safety cases were in demonstrating that software is acceptably safe. All the respondents felt that software safety cases were effective. One interviewee stated that, "before there was a safety case, if you were to ask why it was safe, the answer was generally just that the standards had been followed. With the safety case it becomes clearer why it is being done." Another respondent felt that it was particularly useful for helping non-software specialists to comprehend the software aspects of the safety
of the system.

Some of the respondents saw limitations to the use of structured safety argument techniques such as GSN in the safety case. There was an opinion expressed that software safety arguments were more necessary, and most effective for high criticality systems. This may be related to a stated perception that the cost of producing a software safety case is high. Another opinion was that software safety cases can seem to be far too complex. Although the need to provide sufficient detail was acknowledged, it was felt that there was often a tendency to include every single available piece of information.

It was also stated that the safety case evaluation focussed on the evidence much more than the structure of the argument. There was also a feeling amongst some respondents that the argument itself was fairly easy to construct, and played more of an integrating role. This type of response tended to be from those who focussed on compliance with a particular standard, in which case the importance of the argument will be diminished, and its benefit reduced.

Those interviewees who were experienced at developing software safety arguments shared a number of problems that they had encountered. It was noted that people inexperienced at constructing safety arguments would often end up with an ‘explosion’ of GSN, that would make the safety argument unmanageable. This could be one cause of the perception of the high cost and complexity of safety arguments. One respondent had experienced difficulty in modulating the argument to reflect the fact that they had a fairly low criticality system. This had lead to some criticism that their safety activities were more onerous than was necessary.

It was also felt that a lot of people attempting to produce safety arguments don’t really know what an argument actually is, and a lot of the resulting arguments end up being badly constructed and uncompelling. An example of a common problem was that some ‘arguments’ produced are not arguments at all, but explanations. One respondent felt that it was very easy for someone to put forward an argument as support for an objective without properly questioning its necessity or sufficiency. There was a commonly held view amongst many of the interviewees that, although they could understand the benefits of moving away from a prescriptive standard to a more product focus, those prescriptive standards did provide the type of guidance that people were generally looking for. They felt that with a move to an argument based approach there was a lack of guidance on what people were required to do. One respondent noted that this was particularly the case for software, where even identifying claims to support can be very difficult in practice.
One interviewee had found that a lot of decomposition was often required in the argument to show how a solution satisfied a claim. They had found that it was more effective to capture this textually rather than will lots of additional layers of argument.

**SUMMARY OF FINDINGS FOR SOFTWARE SAFETY CASE PRODUCT**

The key findings from this part of the consultation process were as follows:

- All of those interviewed were either using structured arguments in their safety case or said that there was an intention, or aspiration to move to an argument-based safety case in the future.

- Most of the respondents chose to supplement process based evidence with additional evidence related specifically to the software product, however:
  - the amount of additional evidence generated varied significantly between the interviewees.
  - there was perceived to be a lack of guidance provided on what these additional product-focussed safety activities should be.

- Most respondents felt that there was a clear understanding of what was sufficient, based mainly upon compliance with standards, however:
  - for anything beyond the requirements of the standards, there was far less understanding of what was considered to be sufficient.

- It was felt that safety case reviewers often didn’t themselves have a clear understanding of how to judge the sufficiency of a software safety case.

- The requirements for evidence to support the software safety case should be defined as early in the project as possible. Ideally this should occur before the development process is finalised.

- All the respondents felt that software safety cases were an effective way to demonstrate that the software aspects of the system are safe.

- There was a perception that the cost of producing a software safety case is high.
• Software safety cases often seem to be too complex.

• Evaluation focuses more on the evidence than the argument structure.

• There was a feeling that a lot of people don’t understand how to produce compelling safety arguments, key problems being,
  
  – GSN ‘explosion’
  
  – Making the argument appropriate for the criticality

  This may be an area where specialist train should be provided.

• There was felt to be a lack of guidance on what people were required to do when developing a software safety argument.

A.3.2 The software safety case process

The interviewees were then asked for their experiences of the process of generating a software safety argument. All of the respondents felt that early identification of the software safety activities necessary to support the safety case was crucial to the success of the software safety case development. All of the respondents adopted an incremental approach to the development of the safety case, that is to say that there were a number of interim safety cases produced before the final operational safety case. Initially the safety case would focus on the identification of the requirements and setting out higher level objectives. This would then be developed and refined as the development progressed and evidence became available.

It was noted by some respondents that the biggest challenge here was often identifying the high level requirements for the software, particularly as they straddle the boundary between the software and the system. This will often require the identification of specific software requirements by the system safety team which can be used as top level claims in the software safety case. The challenge here is that these requirements need to be defined at the system level, but in a way that software engineers can understand. The respondent felt that what this requires is for the systems people to have a better understanding of software, and how it can contribute to the safety of the system. It was suggested that some guidance could be provided, aimed at systems engineers, looking at software failures, and how software may contribute to system hazards. This should also provide an understanding of why software engineers do the things
that they do. It was recognised that this may be better achieved through
training rather than guidance.

It was acknowledged however that even if the requirements of the safety
case are identified early, it can still be difficult to meet these in practice.
One example that was given was that ideally, to support the safety case,
the safety engineer should be influencing tool selection. In reality however,
these decisions have already been made, so all that can be provided is an
assessment of the chosen tools.

On all of the projects considered, the aspiration was to undertake the
safety case related activities in parallel with the development activities. In
many cases this proved to be difficult in practice, since it wasn’t always
easy to get access to the necessary information when it was required. It
was noted that problems can sometimes occur because the safety case
baselines do not correspond to the design baselines. Ensuring that the
safety case is updated to reflect changes to the design is also crucial.

A.3.3 Specific issues

In addition to the more general discussions reported above, the intervie-
wees were also asked for their views and experiences relating two specific
issues. The first issue was counter evidence, the identification of which
has been included as a requirement in 00-56 issue 4. The aim here was
to get a feeling for how well this concept was understood, and how widely
used it was within current software safety cases. The second issue was
the use of COTS software, where the aim was to understand the experi-
ences of those who had considered COTS software as part of their soft-
ware safety case.

COUNTER EVIDENCE

Only around half of those interviewed felt that they understood what was
meant by counter-evidence, and a much smaller number claimed to actu-
ally make use of counter-evidence within their safety cases. These gener-
ally considered it quite informally through, for example, a problem reporting
process.

Once the concept of counter-evidence was described, all the respon-
dents felt that they already had some processes in place for identifying
counter-evidence, however the rigour of these processes varied. In a num-
ber of cases this would be done as part of the safety case evaluation pro-
cess, rather than as part of the safety case development.
The respondents generally agreed that counter-evidence could have a useful role in the software safety case, however all felt that much more guidance was needed on the use of counter-evidence. In particular, some interviewees felt that it was very difficult to know when enough had been done in terms of identifying counter-evidence. One observation was that it can be very difficult to weigh counter-evidence against positive evidence.

COTS

For most of the projects discussed, the safety related software was developed as a bespoke in-house product, although most had some COTS software in their systems such as firmware or embedded software in bought-in equipment. In the majority of cases any COTS software used had been previously certified on other systems, however none of those interviews relied solely on this for their acceptance. In most cases the context of usage was very similar and so it was common for evidence relating to the history of usage to be used to support the safety case.

Where previous usage wasn’t felt to be sufficient or appropriate, some of the projects relied on generating their own additional evidence or, where possible, putting additional mitigations in place. There was a mixed response regarding the ease of obtaining additional evidence from the supplier to support the safety case for COTS software. Some of the respondents reported that they generally didn’t have a problem, however for others it proved very difficult. The response from the COTS vendor seemed to depend on the relative size of the customer. It was also noted that for some types of evidence, it was simply too late to ask for it. A number of respondents noted this as another motivation for defining safety requirements as early as possible in the programme.

One of the respondents explained that they acted as a COTS supplier for some of their software products. In this case, they provide a safety case that goes with the COTS solution. If the customer then requires anything in addition to this, they would be happy to provide it, however the customer would be expected to pay the additional cost of this.

A.3.4 Guidance requirements

The interviewees were asked to describe the aspects of software safety case development where they feel that guidance would add most value, based on their own experiences of the challenges they have faced. The majority of the guidance needs related to one area in particular, the sufficiency of the software safety argument.
ARGUMENT AND EVIDENCE SUFFICIENCY

The main area in which the respondents felt that guidance was most required was in determining what activities it was necessary to undertake (such as software safety activities and evidence generation) in order to achieve a sufficient level of assurance in the resulting safety case. The software safety case developers in particular wanted to be able to determine early on what activities would be required, such that the safety case development could be managed effectively. The respondents noted that guidance was also required for assessors of safety cases to ensure that there were consistent expectations. Some respondents noted that, because they were unsure of what would be considered sufficient by the assessors, they would often do more than they thought was actually necessary for the criticality of their software, simply to reduce the risk of the safety case being rejected. One thing that this suggests is that there may be a need for more regular communication between the reviewers and those providing the safety case.

It was noted that it was particularly difficult to know how to modulate the software safety activities for lower integrity software, i.e. it was often unclear which activities may be omitted because less assurance was required in the argument. A number of respondents stated that guidance on how different techniques affect the overall assurance achieved would be very useful in these situations.

Many of the respondents felt that the move towards a safety argument based approach to the safety case was a positive move, allowing them greater flexibility, however they also felt that prescriptive standards were very useful in setting out what activities needed to be performed. There was a consensus that guidance was needed on how to produce a compelling software safety argument, but that this guidance should be flexible, and not overly prescriptive.

OTHER GUIDANCE NEEDS

A number of interviewees felt that there was a wide range of guidance available on the process aspects of software safety, but much less on the product-focussed activities. It was felt to be crucial to the success of a software safety argument that there was much more understanding of software failures, and the way in which software might contribute to hazards. One particular aspect of this that was noted was the identification of software safety requirements. This was particularly problematic due to interactions
required for this between the system and software level safety teams.

**A Suggested Approach**

It has been considered under SSEI task 6, to generate guidance in the form of a set of argument patterns and anti-patterns to indicate examples of good and bad practice in software safety argument construction. These argument patterns would capture both the argument structure, and the rationale for the structure. The patterns could then be applied in a flexible way to individual projects when developing software safety arguments.

The interviewees were asked if guidance of this form were to be developed, how useful they thought that would be to them. All of the respondents felt that guidance of this form would be extremely useful. A number of suggestions were made as to how this guidance could be made most effective:

- Various respondents’ experiences with the existing patterns is that, although they seem fairly simple at first sight, they can be very difficult to apply successfully on realistic large-scale systems. It was suggested that realistic examples of how to apply the patterns is crucial.

- It was also felt to be important that it was very clear how to tailor the patterns for systems with different sizes, complexities and risks.

- Many interviewees felt that the information about why a particular strategy would be chosen is perhaps the most important thing, as it is currently missing. They thought this would encourage the provision of proper justifications for the safety of the system.

- On a similar theme, it was suggested that rather than just showing how to do it, some more general guidance on constructing compelling software safety arguments would be very useful.

- It was noted that to be of most value, the guidance should be aimed at assessors as well as developers. If the guidance can facilitate agreement on what is sufficient, then it could lead to big savings.

**A.4 Conclusions**

The main results from the consultation exercise are summarised below:
• All the respondents felt that software safety cases were an effective way to demonstrate that the software aspects of the system are safe.

• There was a perception that the cost of producing a software safety case is high.

• It was felt that guidance was most required in determining what activities it was necessary to undertake to support the software safety case, in particular, how to ensure that the argument is appropriate to the risk.

• All of the respondents felt that guidance in the suggested form would be extremely useful.

• It was suggested that realistic examples of how to apply software safety argument patterns is crucial.

• It was felt that to be of most value, the guidance should be aimed at both assessors and developers.
Here it is described how the 6-step method was used to develop a software safety argument for a simplified example system. The main body of the report describes how a software safety argument for such a system could be greatly improved through a more systematic approach to software safety argument construction.

The example used is a Long Range Air-to-Air Missile (LRAAM), which is an aircraft launched missile propelled by a rocket motor towards an airborne target. The details of this example have been extracted from [29]. The software aspects of this system are contained within the Electronics and Power Unit (EPU) which is the subject of the argument. The argument presented here is an initial development of the argument resulting from the application of the existing method, making use, where applicable, of existing software safety argument patterns (such as those from [29] discussed in section 4). All of the GSN figures referenced in this example can be found at the end of this annex.

The argument for the EPU software is based upon a consideration of the potential contributions of the software to the LRAAM hazards. There are a number of hazards associated with the LRAAM including:

- Premature detonation
- Premature launch
- Hitting the launcher due to incorrect trajectory

The potential software contributions to all of these hazards must be considered in the software safety argument. In this example we focus in particular on the hazard ‘Premature launch of missile’. The original proposition for the high-level structure of the software safety argument is shown in figure 28 (Annex B). It should be noted that if the argument were fully developed, ‘Goal: Other Hazards’ would in fact be a number of separate claims, one for each hazard, similar to the chosen example ‘Goal: Hazard 1’. The software safety argument for the EPU must form part of a larger safety case for the LRAAM system as a whole. It is at the system level that the identification of the system hazards is dealt with. It is necessary to make an assumption in the software safety arguments that all of the system hazards for the LRAAM have been correctly identified at the system level. The system level safety case would provide an argument which would support this assumption.
Because the software safety argument must integrate with a system level safety case, the high level structure of the argument is to some extent governed by the low level structure of the system safety case (that is that the objectives of the software safety case must fulfill the requirements of the system level). This is discussed in more detail in section 4.

The software may contribute to system hazards in either a positive or negative manner. Positive software contributions are those things that the software must do in order to mitigate the effects of some potentially hazardous failure in the system. In this case the hazardous failure has occurred in some other (perhaps non-software) component in the system, and the contribution of the software is to mitigate the effects of that failure such that it doesn’t result in a hazard, or the severity of the consequences are reduced. Negative software contributions are failures within the software itself which could potentially lead to a hazard. Of course failure to provide a positive contribution is in itself negative. The key difference is that the initiating failure is not a failure in the software, but in some other component.

As is the case with the hazards, the software contributions to those hazards can only be identified at the system level. At the system level, these software contributions would be considered as ‘basic events’, that is they would not be developed any further. An example of one such basic event that may be identified at the system level is ‘failure of software premature launch prevention function’. This may be identified, for example as a base event in a fault tree considering the ‘premature launch of missile’ hazard. To develop such an event further requires detailed consideration of the software. Such an event would therefore be left undeveloped at the system level. At the software level, these contributions are the ‘top events’, which are the high-level objectives which the software safety case must consider.

Figure 29 shows the structure of the argument that was originally proposed for considering each contribution to the example hazard. It is argued that the software contributions to the hazard are correctly identified, and that these identified software contributions are all acceptable. It should be noted that the correct identification of the software contributions could, in the same way as the identification of the hazards, be argued at the system level (in which case an assumption about their correct identification would be made in the software safety argument). In this case it is felt that, since the identification of the software contributions is so crucial to the software safety case, an argument about the software contributions identification is included in the software safety argument. The identification of the software contributions to hazards is effectively the ‘handing over point’ from
the system level argument to the software argument. As for the hazards, an argument is made over each of the software contributions identified for the hazard. Again, a fully developed argument would present a number of claims instead of ‘Goal: Other identified Contributions’ relating to each contribution. This example focusses on one identified software contribution, the premature launch prevention function.

To show that the software contribution under consideration is acceptable it is necessary to consider how the software might fail, such that those failures which might contribute to the hazard can be identified. There are a huge number of different failure modes for software, what is important is that all of those failures which could result in hazardous outcomes are identified such that they can be controlled. Such failures shall be referred to as hazardous software failure modes (HSFMs). Identifying a sufficiently complete set of HSFMs is crucial for the development of a compelling software safety argument. It is possible to introduce different types of errors at each level of refinement as the software is being developed. For example, at the requirements level, where the software design is still essentially a black-box, it is possible to introduce errors through an incorrect specification of the requirements. Such requirements errors could be carried through into the implementation, resulting in potentially hazardous software behaviour. Similarly at the design level, errors may be introduced during the design specification, even if the requirements being used for that design are correct. There may be many layers of decomposition in the architecture and design of the software before implementation occurs. At each level of refinement new errors could be introduced, all of which could lead to hazardous software behaviour. It is therefore necessary to consider HSFMs at each level of refinement in the software development.

All the identified HSFMs must be sufficiently mitigated in order to ensure that the system is sufficiently safe. HSFMs can be mitigated either by eliminating them from the software by ensuring that the HSFMs will not occur, or by detecting the occurrence of the HSFM and handling it such that it will not result in a hazardous outcome. The primary focus in this example will be on the elimination of identified HSFMs, and this shall be achieved predominantly by defining derived software safety requirements (DSSRs) which are requirements that, if met by the software, will prevent the occurrence of a HSFM. Just as the HSFMs may be identified at any level of refinement in the software development, so DSSRs must be identified and implemented at each level in order to mitigate the identified HSFMs. This is described in more detail in section 4.

Figure 30 shows the structure of the argument that was originally proposed for the LRAAM example, where each level of refinement in the EPU
software development is considered. In this example we assume a simple
development process with just three distinct tiers of refinement: require-
ments, design and implementation.

Firstly we shall consider the requirements level. As shown in figure
31, it is necessary to demonstrate that the DSSRs are identified and to
demonstrate that those DSSRs are satisfied. The DSSRs are derived
from the HSFMs. In order to identify HSFMs it is necessary to consider
all the possible failure modes, and consider their potential consequences,
such that those failure modes with a potentially hazardous effect can be
identified. At the requirements level, there is no detailed design or archi-
tectural information available, therefore the identification of failure modes
will focuss on the functions defined by the requirements, and consider po-
tential deviations from intended behaviour. Functional failure analysis is a
very commonly used technique for hypothesising failures and considering
their effects. The technique (described as Functional Hazard Assessment
(FHA)) is described in [22]. The following deviations from intent are nor-

• Function not provided
• Function provided when not required
• Function provided incorrectly

This technique can be used to identify those failures relating to the
contribution which are potentially hazardous. For the LRAAM system, the
following example HSFMs may be identified at the requirements level relat-
ing to premature launch prevention. This is the only failure mode identified
at this level of refinement that could result in the hazard ‘Premature launch
of missile’.

• Software fails to block premature launch (function not provided)

A DSSR can then be specified which can be used to ensure the failure
mode does not occur.

• Software shall block premature launch\(^7\)

\(^7\)It should be noted that not all DSSRs identified in this manner will necessarily be
‘functional’ in nature. The ‘function provided incorrectly’ deviation can include timing re-
lated failures such as early or late provision of the function under consideration.
Figure 32 shows the initial proposition put forward for an argument for the identification of DSSRs at the requirements level. The DSSRs can be captured in a DSSR specification. To ensure that these DSSRs are satisfied, any of the DSSRs which are not already captured as software requirements are included in the software requirements specification. This should ensure that the DSSRs are implemented in the software. It is possible to perform specific tests at this level to check that these DSSRs have been implemented correctly by the software. This is shown in figure 33.

At the design level, more detailed information is available about the design of the software and therefore HSFMs relating to this design can be identified. At this level, the flow of information through the software design can be considered, to identify potential deviations from that design which could be hazardous. It is possible to use an HAZOP-style technique to consider deviations in the information flows. In [23], Pumfrey introduces the SHARD technique for analysing software designs. Such a technique can be used to identify HSFMs at the design level. It is possible to interpret the SHARD guidewords for different design notations. For the LRAAM example, a software design is developed in MASCOT [27]. Figure 26 shows the design of the EPU software. The aspects of the design relevant to premature launch prevention are indicated in red.

Pumfrey provides an interpretation of the SHARD guidewords for MASCOT designs, as shown in figure 27. These guidewords can be used to identify HSFMs relating to premature launch prevention.

An example HSFM which might be identified using SHARD analysis is:

- Software interlock handler fails to write to interlock pool (Omission failure type)

A DSSR can again be specified which can be used to ensure the failure mode does not occur.

- Omission of software interlock handler writes to interlock pool, shall not occur

The proposed argument provided at the design level had a similar structure to that of the requirements level, as seen in figures 34 and 35. Since the satisfaction argument at the design level is more complicated than at the requirements level, a divide-and-conquer style approach is taken by arguing separately for each of the DSSRs identified for the contribution. In figure 36 the argument has been developed just for the DSSR

---

8Software Hazard Analysis and Resolution in Design
‘Omission of software interlock handler writes to interlock pool, shall not occur’. In order to demonstrate that the DSSR is satisfied, it is necessary to show that the software is designed such that it fulfills the DSSR. This alone is not necessarily sufficient however, as it is possible that other failures could occur within the software which prevent the fulfillment of that DSSR. Therefore an argument is also made that other failures will not prevent the DSSR being fulfilled.

Figure 37 shows the proposal that was made for an argument that could be made that the software design fulfills the DSSR. As for the requirements level, it is also possible to perform testing at the design level which provides evidence that the DSSR is fulfilled. However, since we know, from the SHARD analysis, the nature of the HSFM that gave rise to the DSSR (in this case an omission failure), it is also possible to perform some specific analysis of that failure type. The example in figure 37 shows how static code analysis of the interlock handler function can be used as evidence that there are no omission failures by showing that each feasible control path through the function has a unique output statement. For the other failure types of commission, early, late and value, different claims about the properties of the function can be made, appropriate to that failure type.

There are a number of different sources of failures that could result in
an omission failure. Firstly the function must be able to execute successfully. The first thing to consider is if there are sufficient resources available. The resources that need to be considered include the availability of sufficient memory, plus any other resources required by the particular function such as availability of sufficient bandwidth. In order to execute, the interlock handler function must also be scheduled correctly. The argument made here will be dependent upon the particular scheduling philosophy which is chosen. The argument must provide evidence that the interlock handler function will execute when required.

Additionally, even if the function has sufficient resources available and is correctly scheduled, it is still possible that omission may occur due to failures outside of the function itself. It may be possible for failures in other software functions, or failures in the hardware, to prevent the interlock handler function from executing. The failures within the hardware can be considered to be outside the scope of the software safety argument being considered here (a safety case for the hardware aspects would consider these failures), however hardware failures may result in additional derived requirements on the software, and the way in which the software responds to failures in the hardware does come within the scope of the software

---

**Table 7** shows an early attempt to interpret the failure classes from Table 6 for some combinations of communications protocol and data type for MASCOT 3. The intention was that for each flow analysed, the protocol and data type would be used to select the appropriate row of the table; the entries in that row (e.g. “No write, Unwanted write” etc.) would then be used as the guide words for that flow.

![Figure 27: Example SHARD guidewords for MASCOT](image-url)
safety argument. It is necessary to demonstrate that, wherever it is possible for the software to respond to a failure in the hardware, it does so in a safe manner. This may require some form of hardware health monitoring to which the software responds. Physical or logical partitioning between software functions can be used to prevent failures in one function causing omission failures in another function.

Figure 38 shows the example argument that was originally proposed to deal with these other failures. The sources of failures that may result in other types of failure mode will vary. For example, when considering commission failures, lack of resources are not a potential cause.

There are some HSFMs that only emerge at the implementation level, that is when the design is translated into code. At this level the types of errors that may be introduced include type-mismatches (where data elements of the wrong type are assigned to variables), divide-by-zero errors (where the code attempts to divide a value by zero) or dead code (the invocation of code which was not intended to be used). All of these, and many other types of errors made in the code itself can result in exceptions or undefined behaviour from the software. If these errors are not identified and dealt with they could therefore lead to hazardous effects. The argument must therefore show how such failures are prevented.

In addition, it is necessary at the implementation level, as at the other levels considered, to demonstrate that the code correctly implements the DSSRs that have been defined. This includes all of the DSSRs that have been identified at the higher levels of refinement. Figure 39 shows an example of the type of argument that was proposed for the implementation level.

At the code-level the HSFMs can be identified by inspecting the code for the relevant function. It can be seen how different types of code-level failures may be mitigated, such as through exception handling, or use of a strongly-typed language. Testing can be performed at the code-level in the form of unit testing to verify that each DSSR has been correctly implemented by the code.

This section has shown how, using an existing argument development method, it was possible to construct a software safety argument for an example system. In the next section we consider whether the resulting argument would be considered suitably compelling and suggest an extended development approach.
Figure 28: LRAAM example software safety argument view 1
Goal: Hazard 1
Software contributions to hazard 'Premature launch of missile' are acceptable

Goal: contIdent
Software contributions to hazard 'Premature launch of missile' are correctly identified

Goal: contMit
Identified software contributions to hazard 'Premature launch of missile' are acceptable

Goal: cont 1
Software contribution from premature launch prevention is acceptable

Safety requirements
Software contribution may have been defined as an explicit safety requirement

Con: sw contributions
List of the identified software contributions

Strat: contMit
Argument over each software contribution to hazard

Goal: Other Identified Contributions

Figure 29: LRAAM example software safety argument view 2
Goal: contMitRqts
Software contribution mitigated in requirements

Strat: contMitRqts
Argument through derived software safety requirements (DSSRs)

Goal: RqtsDSSRsIdent
DSSRs identified at requirements level

Goal: RqtsDSSRSatisfied
DSSRs satisfied at requirements level.

Figure 30: LRAAM example software safety argument view 3

Goal: contMitDesign
Software contribution mitigated in design

Strat: contMitDesign
Argument over potential contributions from requirements, design and implementation phases

Figure 31: LRAAM example software safety argument view 4
Goal: RqtsHSFMIdent
HSFMs identified at the requirements level

Goal: RqtsDSSRsIdent
DSSRs identified at requirements level

Goal: RqtsDSSRderived
DSSRs derived to mitigate identified HSFMs

Goal: FHA
FHA of premature launch prevention function identifies HSFMs

Sol: FHA
FHA results

Sol: DSSRspec
DSSR specification

Figure 32: LRAAM example software safety argument view 5

Goal: RqtsDSSRSatisfied
DSSRs satisfied at requirements level.

Goal: RqtsTrace
DSSRs correctly captured as software requirements

Goal: RqtsSatisfied
Software requirements are satisfied

Sol: SRS
Software requirements specification

Sol: SystemTest
System/Requirement test results for DSSRs

Figure 33: LRAAM example software safety argument view 6
Figure 34: LRAAM example software safety argument view 7

Figure 35: LRAAM example software safety argument view 8
Goal: DesignSatisfied
Software design fulfills the DSSR.

Goal: DesignDSSRSatisfied
Design correctly fulfills DSSRs.

Strat: designDSSRSatisfied
Argument over each DSSR.

Goal: DSSR 1 satisfied
DSSR ‘omission of software interlock handler writes to interlock pool, shall not occur’ is satisfied.

Goal: Other DSSRs satisfied

Goal: DesignFailures
Failures at design level do not prevent DSSR being satisfied.

Goal: DesignSatisfied
Software design fulfills the DSSR.

Goal: InterlockOmission
Omission failures do not occur in software interlock handler function.

Sol: IntTest
Software design integration test results for DSSR.

Goal: InterlockOmission
Omission failures do not occur in software interlock handler function.

Goal: DesignFailures
Failures at design level do not prevent DSSR being satisfied.

Sol: StaticAnalysis
Static code analysis results for software interlock handler function.

Goal: Omission failure
All feasible control paths through software interlock handler function include a unique output statement.

Sol: StaticAnalysis
Static code analysis results for software interlock handler function.

Figure 36: LRAAM example software safety argument view 9

Figure 37: LRAAM example software safety argument view 10
Goal: DesignFailures
Failures at design level do not prevent OSSR being satisfied.

Conclusion: Resources
The resources required for the execution of the function are...

Strategy: DesignFailures
Argument over sufficiency of resources, the successful execution of the function and the absence of other failures which could cause omission.

Conclusion: Execute
Choosing execution/scheduling philosophy.

Goal: Resources
Necessary resources exist to support the execution of the software interlock handler function.

Goal: OtherFailures
Failures in other software functions will not prevent the execution of the software interlock handler function.

Goal: Non-Interference
Software partitioning ensures other functions do not prevent the execution of the interlock handler function.

Goal: Execution
Software handler function executes.

Goal: Connect
Software response to hardware failures in a safe manner.

Figure 38: LRAAM example software safety argument view 11

Goal: CodeHSFMMit
Code-level HSFMs sufficiently mitigated.

Conclusion: CodeHSFMs
Code HSFMs will include e.g. Divide-by-zero errors, type mismatches, dead code etc.

Solution: Code inspection
Manual code inspection of premature launch prevention code.

Goal: Exception handling
Software exception handling is sufficient to mitigate HSFMs.

Goal: Language
Software is implemented using a strongly typed language.

Goal: codeTrace
DSSRs implemented in the software code.

Goal: codeHSFMIdent
HSFMs identified in premature launch prevention code.

Figure 39: LRAAM example software safety argument view 12
Annex C       Non-MOD Safety Standards

This Annex contains a discussion of the application of DS 00-56 to non-MOD standards. The material herein is taken from the SSEI deliverable “Working Paper on the Evidential Approach, Addressing Non-MOD Standards”. All references in this are to internal sections of this Annex, and citations are to the works listed in Annex C.6.
EXECUTIVE SUMMARY

Overall, this task is intended to produce a statement of best practice for the assurance of software systems in the context of Defence Standard 00-56 Issue 4. The key objective of the task is to produce guidance on the production of the software aspects of safety cases.

This working paper is a precursor to this eventual statement of best practice. From the proposal, it is required to “examine the evidential requirements for satisfaction of DO178B and how this maps to evidence levels as described in Defence Standard 00-56, part 2, Issue 4”. A similar analysis has been made of the use of IEC61508 in the context of the Defence Standard.

DO178B and IEC61508 are the focus of this working paper, as these are the two software safety standards most frequently applied on non-MoD safety-related applications, and most often quoted by COTS suppliers to MoD.

The paper provides an outline summary of the requirements of Defence Standard 00-56, DO178B and IEC61508.

It then provides a comparison between the requirements of the Defence Standard and DO178B, and the Defence Standard and IEC61508, in terms of the topics addressed.

It also provides an initial comparison between the Defence Standard’s required levels of confidence and DO178B’s Design Assurance Levels, in terms of the evidence expected and levels of independence (this will be considered in more depth in future documents).

It concludes with a report of Quintec’s experiences applying Defence Standard 00-56 Issue 3 on various software intensive systems, notably Watchkeeper.

DO178B focuses on development assurance against previously identified software requirements (including safety requirements). It says nothing about how those requirements are identified or about safety management activities, essentially assuming that system and software safety assessment, hazard identification and integrity target setting are performed before the software development process starts. Where there are common objectives in DO178B and the Defence Standard, there is broad agreement on the approach, though for high criticality systems the Defence Standard requires diversity, whilst DO178B stresses coverage and independence.

IEC61508 more closely aligns with DS00-56, particularly in terms of the requirement for system risk assessment etc, as well as satisfaction of safety requirements. One difference is that the Defence Standard requires an explicit safety argument and safety case. Another difference is the prominent position of the ALARP principle in the Defence Standard.
The report concludes that both DO178B and IEC61508 emphasise the need to have evidence that demonstrates the satisfaction of identified safety requirements, which is one of DS00-56’s objectives. Where they differ is in the relationship between the satisfaction evidence and overall system safety assessment.

Thales' experience in applying Defence Standard 00-56 on some software projects suggest that sub-contractor have problems following it, as they lack the visibility of the system boundary needed to make a system-wide risk assessment. This leads to a reversion to more prescriptive techniques. There is also a tension between many system safety standards’ quantitative approach and the ability to justify quantitative failure rates for software.
# TABLE OF CONTENTS

Executive Summary ........................................................................................................... 209
Table of Contents ............................................................................................................. 211
C.1 Introduction .................................................................................................................. 212
C.2 Summary of DS00-56 .................................................................................................. 212
  C.2.1 Principal requirements ......................................................................................... 212
  C.2.2 Required levels of confidence ............................................................................. 214
C.3 RTCA DO178B ............................................................................................................ 215
  C.3.1 Background of RTCA DO178B ......................................................................... 215
  C.3.2 The Link between Software Development Process and Safety ....................... 216
  C.3.3 DO178B Overview ............................................................................................ 218
  C.3.4 Potentially Available Evidence from DO178B ................................................ 218
  C.3.5 Comparison of DO178B with DS00-56 ............................................................... 224
  C.3.6 DO178B and MoD ............................................................................................ 226
C.4 IEC 61508 .................................................................................................................. 229
  C.4.1 Introduction ........................................................................................................ 229
  C.4.2 Background of IEC 61508 ................................................................................ 229
  C.4.3 Comparison of IEC61508 with DS00-56 ............................................................ 234
  C.4.4 Thales’ approach to evidence requirements for software ................................. 237
C.5 Conclusions and Concerns ......................................................................................... 245
C.6 References .................................................................................................................. 247
C.7 Glossary ...................................................................................................................... 247
C.A Def Stan 00-56 Mapping to IEC61508 ..................................................................... 249
C.1 INTRODUCTION

Task 11 is intended to produce a statement of best practice for the assurance of software systems in the context of Defence Standard (DS) 00-56 Issue 4 [1]. The key objective of the task is to produce guidance on the production of the software parts of safety cases, so the focus is on arguments and associated evidence.

This working paper, milestone 11.2, is a precursor to this eventual statement of best practice. From the proposal [2] “QinetiQ will examine the evidential requirements for satisfaction of DO178B and how this maps to evidence levels as described in Defence Standard 00-56, part 2, Issue 4”. In addition, Quintec will provide a similar analysis of the use of IEC61508\(^1\) in Thales.

DO178B [3] and IEC61508 [4] are the focus of this working paper, as these are the two software safety standards\(^2\) most frequently applied on non-MoD safety-related applications, and most often quoted by COTS suppliers to MoD.

This working paper aims to be strictly factual ‘what the standards say’ and ‘how they have been used’. It identifies a number of areas where subjective comparisons need to be made (e.g. does satisfying the objectives for test coverage defined in DO178B provide the “rigorous and comprehensive” evidence required by DS00-56?), but arguing these comparisons is left for future documents.

C.2 SUMMARY OF DS00-56

The following is a, necessarily brief, summary of DS00-56. It focuses on those aspects related to the system under development, as opposed to management organisation etc.

The notation [R/S] is used to refer to section S of reference R. Sections shown in italics will not be considered further in this report, as they are introductory or organisation material:

C.2.1 Principal requirements

- [1/0] Introduction
- [1/1] Scope and Applicability

---

\(^1\) this isn’t a direct quote from the proposal, as the contractual arrangements have changed since this was written. However, the intention was always to provide an analysis of the application of IEC61508

\(^2\) strictly, DO178B is a guidance document, not a standard
[1/6] General Requirements:
- All relevant safety legislation, policy etc. shall be identified and complied with.
- The system, its boundaries and operating environment shall be defined.
- Compliance with all safety requirements shall be demonstrated.
- A configuration management process shall be implemented, to maintain consistency between safety documentation and system configurations.

[1/7] Roles and Responsibilities

[1/8] Safety Management:
- A Safety Management Plan (SMP) shall be produced.
- The SMP shall be reviewed and updated regularly through the life of the system.

[1/9] Safety Case:
- A Safety Case shall be produced;
- It shall contain a structured argument demonstrating that the evidence contained (or referenced) is sufficient to show that the system is safe;
- It shall be developed, maintained and refined through life.

[1/10.1-5] Risk Management, Hazard Identification and Hazard Analysis:
- A Safety Risk Management process shall be implemented;
- A Hazard Log shall be created;
- Hazard Identification and Hazard Analysis\(^1\) shall be carried out to identify credible hazards and potential accidents;
- The suitability and adequacy of the processes used shall be justified;
- Hazards and their analysis shall be reviewed and revised throughout life.

[1/10.6-8] Risk Reduction:
- The ALARP principle shall be applied;
- The effectiveness of the process for selecting mitigation strategies shall be demonstrated;
- Records showing traceability between hazards/potential accidents, and the associated safety requirements shall be maintained.

[1/11] Safety Requirements & the Provision of Evidence:
- Records to show traceability between each safety requirement and the source of that requirement shall be maintained, be they from applicable legislation or standards, proposed risk mitigation strategies or derived safety requirements.

\(^1\) including risk estimation
• Within the Safety Case, compelling evidence that the safety requirements have been met will be provided;
• The quantity and quality of the evidence shall be commensurate with the potential risk posed by the system, its complexity and any unfamiliarity (see “Required confidence levels”, below);
• Diverse evidence that safety requirements have been met shall be provided, such that the overall safety argument is not compromised by errors or uncertainties in individual pieces of evidence;
• The Safety Case shall integrate the evidence that shows that (derived) safety requirements have been met for elements of the system, to demonstrate that the overall safety requirements have been met.
• [1/12] Interfaces
• [1/13] Managing Change and Feedback:
  • A change control system shall be operated so that the safety impact of any change is identified and assessed, and any necessary remedial action taken;
  • A process for identifying and assessing defects or failures of the system shall be implemented. Similarly incident, accident and near-miss reports shall be analysed. In either event, any necessary remedial action shall be taken;
  • A process for recording and analysing relevant data from the use of the system shall be implemented in order to validate (and strengthen) the Safety Case.
• [1/14] Safety audits:
  • Safety audits shall be planned, carried out and documented
  • The auditors shall be independent of the team being audited
  • MoD may appointment an Independent Safety Auditor
  • Any remedial actions needed to rectify non-conformities found in safety audits shall be identified and implemented

C.2.2 Required levels of confidence

The fourth bullet of ‘Safety Requirements & Provision of Evidence’ (diversity) is expanded in [5/Annex C]. This defines three ‘Required levels of confidence’: high, medium and low.

For High:
• Diverse forms of evidence should be combined.
• The safety argument should be based on:
  • integrating evidence from:
    o testing, to a high degree of statistical confidence
    o experience of use (again giving a high statistical confidence)
    o analytical arguments, non-destructive examination, etc,
qualitative arguments of good/best practice and other confidence building measures such as process metrics or quantitative modelling.

- Evidence should be rigorous and comprehensive (e.g. 100% test coverage);
- The functionality, performance and behaviour of the system should be understood in its entirety in its operating context;
- For COTS components, the cooperation of the supplier is essential;
- Evidence should be subjected to rigorous, detailed independent scrutiny.

For Medium:

- May rely on a single primary argument with high confidence plus some supporting evidence or on the combination of arguments (each with a lower confidence);
- Evidence should include some analytical and quantitative data;
- Coverage need not necessarily be 100%, provided that effort is directed towards areas of greatest risk and the results do not show significant counter-evidence;
- Analyses may include elements of judgement (e.g. functional failure analysis);
- For COTS components, the cooperation of the supplier is normally necessary, unless comprehensive non-destructive examination is practicable;
- Evidence should be independently scrutinised, but a sampling approach may be used.

For Low:

- The system attributes can be demonstrated with adequate confidence from testing, field data and standards compliance.
- For COTS components good engineering practice needs to be demonstrated. Design and manufacturing evidence may be used if cost-effectively available;
- A sampling approach may be used in scrutinising evidence.

C.3 RTCA DO178B

C.3.1 Background of RTCA DO178B

RTCA is an association of aeronautical organisations within the USA and which is a company in its own right. The documents it produces are therefore by consensus of the members and participating organisations and do not form any government policy unless stated otherwise by a government or agency.
EUROCAE has a similar role within Europe and jointly, the two organisations issued “Software Considerations in Airborne Systems and Equipment Certification – RTCA DO178B/EUROCAE ED12B” [3].

The document relates to the development of airborne avionics equipments and engine control systems. It is specifically intended for use on equipment for airborne use, but is often cited for non-airborne applications. It is a process oriented document with objectives for each part of the software development and verification process, which it must be demonstrated have been satisfied.

It classifies software by required Design Assurance Level (DAL), depending upon criticality. There are five classification, A to E. DAL-A being for the most critical applications, DAL-D for applications with a minor safety-related element and DAL-E used to refer to applications with no safety requirements. According to the target DAL, differing requirements are set for, coverage, recommended techniques and independence. The amount of effort required for DAL-A is considerable, and there is not much reduction between DAL-A and DAL-B. DALs C & D require considerably less effort.

The aim of DO178B (to quote para 1.1) is to “provide guidelines for the production of software ... that performs its intended function with a level of confidence in safety that complies with airworthiness requirements”. However, DO178B assumes that system and software safety requirements, and consequently, any software safety requirements have been identified before the software management and development processes it describes start. It makes no mention of how it assumes this has been done. In the context of civil avionics, it is understood that safety analysis is performed following ARP 4754 [6], but this is not referenced within DO178B.

With RTCA’s background in civil aeronautics, DO178B assumes that the system (including the software of concern) is being developed within a specific certification framework. In particular, DO178B assumes the existence of a Certification Authority (CA), responsible for the acceptance of the system. Furthermore, it is assumed that, early in development, the CA will review the project’s “Plan for Software Aspects of Certification”, in effect, how the project aims to interpret and implement DO178B. The liaison between the project and the CA, is usually referred to as the project’s Designated Engineering Representative (DER), though this terminology is not used in DO178B.

C.3.2 The Link between Software Development Process and Safety

The main objectives of any safety-related software development process are to show:

- the software development process and methods are appropriate, given the system’s integrity requirements.
• the validation and verification activities carried out demonstrate that the software meets its safety requirements, to an appropriate level of confidence.
• all tools and techniques used in the process are appropriate (i.e. give the evidence required to support the safety argument, to the required level of confidence). This includes all development and verification (analysis and test) tools as well as the software language upon which the verification is carried out.
• all staff are competent for their intended roles.
• the documentation produced is appropriate to support the safety argument, and the software’s production and maintenance.

These are essentially the topics addressed by DO178B (other than staff competency, [3/1.2] explicitly excludes “personnel qualification criteria”). However, this assumes that, ab initio, the software has been classified as safety-related, and that specific software safety requirements have been identified.

Where previously developed software is being considered that is ‘certified to DO178B’, it is imperative to gain an understanding of the way in which risks and hazards have been identified at the system level and apportioned to software prior to the start of software development. This is because the definition of what is acceptable in terms of safety can differ between UK MoD and the typical users of DO178B. The most obvious concern is that criticality is a system property that depends upon the planned use of the system, and cannot finally be determined without knowledge of the operating environment.

Where DO178B is being considered for a new software development, some hazard identification process has to be developed/selected, as DO178B explicitly excludes such considerations from its scope.

For typical users of DO178B, DALs are derived from ARP4754 [6] where System Safety Analysis is required to include quantitative analyses for failure conditions based on component failure rates. Its purpose is to provide “a common international basis for demonstrating compliance with airworthiness requirements”, but note that, like DO178B, “The contents are recommendations and are not mandated”. The accompanying guidance document ARP4761 [7] deals extensively with how to estimate risk and how to combine risks using Failure Modes and Effects Analysis (FMEA) and Fault Tree Analysis (FTA).

---

1 not to mention that DS00-56 requires the hazard identification process to be justified as part of the safety case.
C.3.3 DO178B Overview

DO178B expects a software development lifecycle to be specified before any work commences. It does not recommend any particular lifecycle, but notes that most life cycles include some combination of Planning, Development and Integration, and its structure reflects an assumption that these will occur essentially in that order. Transition criteria may be specified, indicating when it is acceptable to move between processes in the lifecycle.

The elements of planning also include a relationship to a certification authority. In the USA, the Federal Aviation Administration appoints Designated Engineering Representatives (DER) to act as certifiers on its behalf. They can be independent of a particular manufacturer or be employees.

One of the first documents required by a DER is the Plan for Software Aspects of Certification (PSAC). This document sets out for the certifier the proposed software lifecycle so that he can determine if it is commensurate with the rigour required for the level of software being developed. It also sets out the means of compliance and includes: system and software overviews, the software lifecycle and highlights the data that will be made available as well as the schedule for reviews and any other additional considerations. The developer has to obtain agreement on the PSAC.

Once development has finished, a Software Accomplishment Summary contains the same types of information as the PSAC, changed to reflect exactly how the software was developed and to demonstrate that this relates to the particular version being certified (i.e. that the system is under appropriate configuration control). It will also contain evidence\(^1\) relating to change history, specific characteristics of the executable code (such as timing) and a list of outstanding problems. It also has a specific statement regarding the justification for compliance to the DAL as set by the system safety requirements and defined by DO178B.

Within all of this activity, there are numerous documents required over and above the PSAC which form part of the evidence generation; these are discussed further at [3/Annex A]. All of these documents could provide evidence to support a software and hence system safety case, but in themselves do not necessarily do so entirely.

C.3.4 Potentially Available Evidence from DO178B

Annex A to DO178B contains a series of tables that list the various objectives of the software development process by DAL. They define what activities are

\(^1\) usually by reference to other documents, rather than explicitly containing the data itself
required (at each DAL) and the level of control required over the configuration management process.

Control Categories CC1 and CC2 define which activities have to be undertaken for configuration management. The activities required by each are defined in [3/Table 7.1], with CC1 being more stringent than CC2. In general, a higher DAL means control is more likely to be CC1 than CC2.

The tables also define whether the achievement of the objectives should be independently confirmed. Independence is defined by DO178B as follows:

“Separation of responsibilities which ensures the accomplishment of objective evaluation. (1) For software verification process activities, independence is achieved when the verification activity is performed by a person(s) other than the developer of the item being verified, and a tool(s) may be used to achieve an equivalence to the human verification activity. (2) For the software quality assurance process, independence also includes the authority to ensure corrective action”

The objectives relate back to relevant paragraphs within the main body. However, what neither the objectives nor the main body do is give a rationale for each objective and therefore the contribution to system safety. Accordingly, it is merely implicit that if the objectives are met, then the software is safe.\(^1\)

Figure C.3.1, below, shows the overall software development process and the relationship to tables A3-A6 (the colour merely indicates grouping). This diagram is essentially a summary of Table A2. The evidence generated by this process is direct evidence that the product satisfies the identified software requirements (with the one main exception of conformance which is process evidence).

\(^1\) i.e. insofar as it has met the identified safety requirements
C.3.4.1 Table A1 – Software Planning Process

Table A1 sets out the documentation that should be available for review which could support a safety case. These include (see [3/11] for detail):

- The Plan for the Software Aspects of Certification: Sets out the system context, intended means of compliance etc.;
- The Software Development Plan (sometimes included within the PSAC): Identifies standards and the specific software development life-cycle including transition criteria between phases for the project. It also identifies the software development environment to be used, including tools (e.g. compilers, and design/verification tools), the programming language (including coding standards/subsets to be used) and the hardware on which the software will run;
- The Software Verification Plan: How it is planned to demonstrate that the software’s safety requirements have been met, the description of the verification methods and compiler assumptions;
- The Software Configuration Management Plan: Outlines the relevant information pertaining to the version of the software and the link to
certification. Includes aspects such as supplier control, tools and software
development environment;

- The Software Quality Assurance Plan and SQA Records: The identification
  of the QA procedures, the establishment of authority and independence,
  the methods, frequency/programme of inspections and a definition of the
  records required. The Record itself documents decisions in the form of
  minutes, reports and release authorisations;
- Software Requirements Standards: Defines how high level requirements
  are to be established, including derived requirements;
- Software Design Standards: Defines how low level requirements and
  architecture are to be developed. This includes naming conventions,
  restrictions on tools and techniques (and why) as well as complexity and
  other aspects of design;
- Software Code Standards: Defines the language and the subset to be
  used, with justification;

Software Verification Results: Identifies the procedure undertaken (review,
analysis or test) and the result achieved (pass/fail) for a particular version of
software

C.3.4.2 Table A2 – Software Planning Process

Table A2 outlines the hierarchy of artefacts typically to be expected from a
software development process. Some of this is direct evidence that the product
meets the identified software requirements and some is process evidence.
Figure C.3.1 shows the process and the links to tables A3-A6 from this process.

C.3.4.3 Table A3 – Verification of Outputs of Software Requirements
Process

The objectives in table A3 are to carry out reviews and analyses to detect and
report any requirements errors that may have been introduced during the software
requirements process. Topics covered include compliance with system
requirements and it is here that there maybe links to system safety requirements
and hence the root for mitigation within a system safety case, particularly as it is
explicitly mentioned within paragraph [3/6.3.1.f] “High level requirements are
traceable to system requirements”. All results of these activities should be
documented in the Software Verification Results.

As already noted, DO178B excludes any mention of system and software safety
activities. This becomes significant, when for example, the first objective of this
table is to assure that the “software high-level requirements comply with system
requirements” As the software development process described in DO178B is
disjoint from any system safety assessment process, it is not clear how this
compliance can be demonstrated.
C.3.4.4 Table A4 – Verification of Outputs of Software Design Process

The objectives in table A4 are to check that reviews and analyses have been carried out which will detect and report requirements errors that may have been introduced during the software design process and in the development of the software architecture. All results of these activities should be contained in the Software Verification Results.

As noted above, the lack of visibility of system and software safety activities means that the activities described in this table can only confirm compliance with the initial software requirements, and cannot assess the impact of design decisions on system hazards.

C.3.4.5 Table A5 – Verification of Outputs of Software Coding & Integration Processes.

The objectives in table A5 are to check that reviews and analyses have been carried out which will detect and report errors that may have been introduced during the software coding process. Primary concerns include correctness of the code with respect to the software requirements and the software architecture, and conformance to the Software Code Standards. The Software Verification Results contain all necessary detail for review, but focus tends to be on correct and complete traceability between Low Level requirements and the architecture, as well as the integration process.

C.3.4.6 Table A6 – Testing of Outputs of Integration Process.

Whilst this table is called “testing”, it covers all planned verification activities, including reviews and analysis. The justification for the procedures undertaken, will be in the PSAC as well as in the Software Accomplishment Summary. The objectives in table A6 are largely grouped into two categories: compliance and robustness\(^1\). Compliance aims to show that the software works under normal conditions and checks are carried out between the Executable Object Code and both the High and Low Level Requirements. This aim is to trap implementation errors and hence the focus on requirements based testing. Robustness addresses testing under abnormal conditions. The justification for the test cases used is documented in Software Verification Cases and Procedures and, once again, the Software Verification results are an important source of information to support a safety case.

\(^1\) there is an additional objective to show that the software is compatible with the target hardware
C.3.4.7  Table A7 – Verification of Verification Process Results

The objectives in table A7 are to confirm that the testing activities have been carried out in such a way as to establish the necessary properties of the software. That is, that the tests have been correctly implemented, produce the expected results, and provide adequate coverage. More so than table A6, this table is slanted towards dynamic testing, rather than formal analysis (e.g. coverage is not a particularly meaningful concept for formal methods). This is also the only section of DO178B that specifies techniques and as such there is often discussion between the DER and the developer over the rationale rather than the letter of its objectives. The rationale should be to examine whether the right verification activities were performed to demonstrate the required behaviour to the necessary level of confidence, and that the actual results have been compared with the expected results. This is documented in the Software Verification Cases and Procedures and results should be contained within the Software Verification Results. There is considerable focus on test coverage both for requirements as well as coverage of branches, statements and decisions within the code.

C.3.4.8  Table A8 – The Software Configuration Management Process

The set of objectives implicit in table A8 is what should be expected from any competent software development process. That is, to ensure that the development and verification records relate to the correct version of the deployed software. The relevant documents are the Software Configuration Management Record, the Software Configuration Index, Problem Reports and the software Life Cycle Configuration Index.

C.3.4.9  Table A9 – The Software Quality Assurance Process

The set of objectives implicit in table A9 is again what should be expected from a competent software development process. All the required documents should be available from developers who have CMM\(^1\) Level 3 \([10]\) or equivalent. The key aspect is to ensure the independence of the QA assessment from the development/verification activities. The main output document is the Software Quality Assurance Record.

\(^1\)  The Capability Maturity Model (CMM) was originally developed as a tool for objectively assessing the processes used by US government suppliers in developing contracted our software products.
C.3.4.10 Table A10 – The Certification Liaison Process

Although there is no mandated procedure analogous to the PSAC, Def Stan 00-56 advises on the appointment of an Independent Safety Auditor (ISA). Whilst superficially these two roles would appear to have much in common, it should not be assumed that they are identical. The precise terms of reference of an ISA’s are set by the project, whilst those of the DER are set by Federal regulation.

C.3.4.11 Table Summary

The impact of selection of DAL on the development process is illustrated in the following Table (Table C.3.1). For each of the tables in DO178B Annex A, it shows the total number of objectives (Obj.), then for each of the four Design Assurance Levels, how many of these objectives are Required with Independence (RI), Required (R) or Optional (O).

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Obj.</th>
<th>DAL-A</th>
<th>DAL-B</th>
<th>DAL-C</th>
<th>DAL-D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>RI</td>
<td>R</td>
<td>O</td>
<td>RI</td>
</tr>
<tr>
<td>A1</td>
<td>Software planning process</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A2</td>
<td>Software development process</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A3</td>
<td>Verification of outputs of software</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>A4</td>
<td>Verification of outputs of software</td>
<td>13</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>A5</td>
<td>Verification of outputs of software coding and integration process</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A6</td>
<td>Testing of outputs of integration process</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A7</td>
<td>Verification of verification process results</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>A8</td>
<td>Software configuration management process</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A9</td>
<td>Software quality assurance</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>A10</td>
<td>Certification liaison process</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>66</td>
<td>25</td>
<td>41</td>
<td>0</td>
</tr>
</tbody>
</table>

Table C.3.1: Summary of DO178B Annex A tables

C.3.5 Comparison of DO178B with DS00-56

Table C.3.2, below, provides a rough comparison of the provisions of DO178B with the requirements of DS00-56

As may be expected from the preceding discussion, DO178B (potentially) addresses most of DS00-56’s requirements in terms of providing evidence that software safety requirements have been met, managing change, and allowing for

---

1 Only those sections of DS00-56 identified in section 2.1 of this report as “related to the system under development, as opposed to management organisation etc” are considered in table 3.2
independent audit. DS00-56 general requirements, in terms of identification of relevant safety legislation/standards etc., definition of system boundaries and demonstration of compliance with safety requirements are also addressed.

The term “potentially” was used above, because DO178B has some specific exclusions, that are in scope of DS00-56\(^1\), and for high assurance DS00-56 emphasises the need for diversity of evidence, which may be available from DO178B, but which isn’t explicitly called for. Note that whilst Table 3.1 highlights where DO178B requires independence, this is distinct from diversity. Independence is concerned with different objectives being addressed by different people (or tools), so that common cause errors\(^2\) do not lead to complementary errors in, for example, development and verification. By contrast, diversity is concerned with a single objective being justified by multiple evidence, to protect against deficiencies or limitations of any single technique\(^3\).

What DO178B clearly doesn’t address are any of DS00-56 requirements related to safety management, hazard identification and analysis, and risk reduction. To some extent this should be unsurprising, as DO178B starts from the assumption that the software safety requirements and target integrity levels have been identified. That is, that the hazard analysis has been performed prior to the process described in DO178B. Similarly, how DO178B’s software development activities fit into an overall safety management and risk reduction process are outside its scope. This means that it is not clear how the impact of software design decisions on system safety properties can be determined.

Within the civil aviation domain that was DO178B’s original target, ARP4754 [6] and ARP4761 [7] define many of the safety management activities, not addressed by DO178B. However, whilst its common for COTS software suppliers to describe products as ‘developed in accordance with DO178B’, it is far less common for the supplier to advertise how broader safety management aspects have been addressed, or to define their assumed ‘context of use’ (essential for risk identification and analysis).

\(^1\) e.g. from [3/1.2] “... the operational aspects of the resulting software are not discussed. For example, the certification aspects of user-modifiable data are beyond the scope of this document”

\(^2\) such as an invalid assumption about required behaviour

\(^3\) the value of diversity therefore depends upon the confidence one has in a technique and the competence with which it has been applied.
Table C.3.2: Comparison of DO178B and DS00-56

It should also be noted that DO178B is focussed on “assurance”, whilst DS00-56 talks about “Integrity”. This is a distinction that may require further discussion in the statement of best practice, as the usual interpretations are that ‘integrity’ is in the realm of risk, whilst ‘assurance’ is concerned with strength of evidence. It is usually accepted that whilst it is appropriate to map from a broadly acceptable risk to a target development assurance level, it is not appropriate to map from an achieved assurance level to a predicted failure rate. Indeed, DO178B explicit prohibits this (2.2.3) “Development of software to a software level does not imply the assignment of a failure rate for that software”.

C.3.6 DO178B and MoD

MoD is likely to be involved with DO178B in three contexts:

- procurement of systems including DO178B certified software
- projects planning to use DO178B during the development of new software
- purchase of COTS software components ‘developed to DO178B’
As DO178B doesn’t directly address safety and risk management issues, when procuring systems including DO178B certified software it is important to determine how the safety assessment was performed, and particularly, what assumptions and limitations on use were considered when developing the software’s safety requirements. It then is beholden on the project to establish that MoD’s proposed use is consistent with those assumptions and limitations. As DS00-56 is a requirement for all MoD projects, all projects will have to perform an assessment of the risks involved with the system in the specific context of its planned use by MoD. The activities and evidence from the DO178B software development process may then provide the bulk of the safety case evidence, but a ‘gap analysis’ will be needed to identify any: short comings in the available evidence, mismatched assumptions, or differences in scope\(^1\). An assessment must also be made of whether the level of evidence from the software verification process meets the confidence requirements from DS00-56. For example, does statement and decision coverage (as required by [3/table A7] for DAL-A and B) meet DS00-56’s requirement for evidence to be “rigorous and comprehensive” where High confidence is required? As DS00-56 doesn’t provide an objective definition of “rigorous and comprehensive” this becomes a subjective judgement that must be justified. Consideration of this will form part of the statement of best practice.

Similarly, does the evidence available meet DS00-56’s diversity requirement? This issue is particularly acute as none of the requirements of DO178B are mandatory, but are all recommendations. The assumed certification process has the developer negotiating an agreed interpretation and plan to achieve compliance with the DER. This is recorded in the PSAC and conformance to the plan is documented in the Software Accomplishment Summary (SAS). These are both signed-off by the DER. It is therefore important to see these agreements.

Where a project is planning to use DO178B, it is assumed that MoD is the direct customer (otherwise, its effectively the procurement of a previously developed/certified system, as discussed above). As described above, the project will need to perform safety management and risk assessment activities (outside the scope of DO178B). If this indicates that the system is sufficiently critical for MoD to appoint an Independent Safety Auditor (ISA), it is then they that will in effect take on the role of DER and negotiate the appropriate interpretation and compliance plan (also taking into consideration the views of other stakeholders). If the project doesn’t have an ISA, this is a negotiation that will occur directly with the IPT. Either way, it can be assured that the evidence generated for the software develop satisfies the needs of DS00-56.

Many developers of high-integrity software components aimed at the civil aerospace market appear to describe their products as ‘developed to’ or ‘certified to’ DO178B (usually quoting a DAL). This is at best a shorthand for ‘having the

\(^1\) e.g. have issues of safe disposal been planned for?
evidence necessary to allow certification to DO178B DAL-X'. Whilst the product may have been used in a system certified to DAL-X, as has already been discussed, any safety assessment depends upon the context of use and the risks posed by the system. In effect, the software’s DAL should be a derived safety requirement based on the system specification and context of use. Also, as DO178B does not include any mandatory\(^1\) clauses, a reasonable interpretation may be that all recommended activities for DAL-X have been performed, but again this is something that ought to be confirmed.

\(^1\) [3/1.4 third bullet] “This document recognizes that the guidelines herein are not mandated by law, but represent a consensus of the aviation community. It also recognizes that alternative methods to the methods described herein may be available to the applicant. For these reasons, the use of words such as ‘shall’ and ‘must’ is avoided”
C.4  IEC 61508

C.4.1  Introduction


The purpose of this section is to identify what evidence is required for software developed for highest level of criticality in the context of IEC61508 and how this maps to evidence levels as described in Defence Standard 00-56, part 2 [5]. Section C.4.4 also describes Quintec’ experiences on a number of safety related software projects.

C.4.2  Background of IEC 61508

IEC61508 is divided into 7 parts. The identification of requirements for High Integrity systems is covered in the first 3 parts which include:

- Pt1: General requirements
- Pt2: Requirements for Electrical/Electronic/Programmable Electronic Safety Related Systems (E/E/PES)
- Pt3 Software Requirements

IEC61508 is a generic standard recognising that there is a great variety of applications in a variety of sectors covering a wide range of complexity, hazard and risk potential. It is also cognisant of sector specific standards and their future development. It is intended that the standard can be applied to the safety of personnel, the environment and equipment.

The overall approach to requirements is illustrated in Figure C.4.1. In many ways this approach is not dissimilar to the V life cycle with overall safety requirements being established as described in Pt1 leading to the allocation of safety requirements for E/E/PES systems. The application or realisation is achieved as in Pts 2 and 3 before safety validation is carried out as again described in Pt1.
Conformity to the standard depends heavily upon the satisfaction of required criteria, such as technical requirements. Clearly such criteria have to be agreed and set for the system in question, and the standard acknowledges that these criteria will depend upon a number of factors and the stage of the design lifecycle. As with DO178B, there is also an assumption that following the process will deliver a safe system.

Figure C.4.1: Overall Requirements Framework for IEC61508
C.4.2.1 Part 1 technical requirements

It is commonly acknowledged that whilst software cannot cause harm in isolation, when embedded in systems there are many ways in which it can cause harm through incorrectly or inadvertently permitting unintended operation, either directly or indirectly, for example by misleading the system’s operators.

Therefore, IEC61508 considers it important to identify the system safety requirements, which can influence the relationship between hardware and software.

The overall safety lifecycle requirements are built around a framework with the intention of systematically achieving the required safety integrity level for E/E/PES and software. The objective of these requirements is to gain an understanding of the equipment or system under consideration, and its operating environment (including legislative requirements) to enable other safety lifecycle activities to be carried out.

The technical requirements of part 1 require the following aspects to be addressed:

- **Scope:** The overall scope determines the system boundary and scope of the hazard and risk analysis;
- **Hazard and Risk Analysis:** There are three objectives for this analysis.
  - Determine the hazards and hazardous events for the equipment or system
  - Determine the event sequences leading to the hazard
  - Determine the risks associated with the hazard.
- **Overall Safety Requirements:** The purpose of these requirements is to produce an overall system safety specification in terms of safety functions and safety integrity requirements. Safety integrity levels shall be established for each safety related function.
- **Safety Requirements Allocation:** the allocation of safety functions to either safety related systems, or external risk reduction facilities. It also covers the allocation of Safety Integrity Levels (SIL) to each function;
- **Operation and Maintenance Planning:** This activity is to ensure that suitable plans to ensure continued system integrity are in place to cover operation and maintenance;
- **Safety Validation Planning:** needs to include who should conduct the validation and when, together with which operating modes need to be evaluated;
- **Realisation of the system**
- **Installation and Commissioning**
- **Safety Validation:** the objective is to ensure that the E/E/PES meets its safety specification in terms of safety functionality and safety integrity. This should involve following the planned validation activities...
- Operation, Maintenance and Repair: this should implement the planned Maintenance and Repair activities.

## C.4.2.2 Part 2 technical requirements

Part 2 of IEC61508 covers safety related programmable electronic systems. It specifically excludes software, but is included in this paper to show the relationship between hardware and software, as illustrated in Figure C.4.2. It can be seen from this diagram that software safety requirements are derived either from the safety requirements specification or from the architecture. There is also an input from Part 2 at the hardware integration stage, but it is the requirements from the specification and architecture which this task will to concentrate upon.

![Figure C.4.2: Hardware and Software Requirements Relationship](image)

## C.4.2.3 Part 3 technical requirements

Unlike previous requirements of this standard, the objective of the Part 3 General requirements is to structure software development into defined activities. In doing so the standard permits a relatively high level of tailoring.

The following bullets identify the requirements of the software safety lifecycle:
Software Safety Requirements Specification: The objective of this activity is to specify the requirements in terms of software safety functions and software safety integrity. If these requirements have already been completed from pt 2 then they do not need to be repeated. All software safety requirements need to be commensurate with the SIL in terms of: clarity, precision, verifiability, testability, maintainability, traceability and free of ambiguity.

Software Safety Validation Planning: the validation of safety software is closely aligned for the validation requirements for hardware mentioned above. There are some additional considerations for safety related software, which includes:
- the validation of each mode before commissioning commences,
- the technical strategy for validation which should cover measures (techniques) and procedures.
- pass/fail criteria
- policies and procedures for evaluating results and in particular, failures.

Software Design and Development: In addition to fulfilling the safety requirements in terms of functionality and SIL, there are some additional objectives to be fulfilled for this activity:
- the selection of tools, including languages and compilers
- the software needs to be designed to be analysable, modifiable and verifiable.
- the division of responsibility between supplier and user, or both. This needs to be determined during the planning phase.
- safety related parts of the software should be minimised. If it is impossible to segregate the safety related software, or demonstrate adequate independence between functions, then all of the software shall be treated as safety related.

Software Architecture Requirements: the software architecture shall include a description of: the selection and justification for the techniques and measures during the safety lifecycle phases, information for each sub-component covering its source (and whether it has had any previous safety assessment), software and hardware interactions;

Support Tools and Programming Language Requirements: All tools, languages, compilers, configuration management tools and automatic testing tools need to be selected to be appropriate to the safety integrity level.

Design and Development Requirements: The requirements specification, description of software architecture and validation plan should be available prior to the start of any detailed design. For each major sub-system the software, any refinement should be partitioned into modules, which should each be specified.

Code Implementation Requirements: The source code shall possess defined desirable properties
Module Testing Requirements: All modules shall be tested as specified in the software design, and they are to show how each module performs its intended function and does not possess unintended functionality.

Software Integration Testing Requirements: Software integration tests are to be specified concurrently during the design and development phase.

Integration of Programmable Electronics: The requirements of the hardware and software integration tests are identical to those listed.

Software Safety validation: The requirements for software validation should be established as part of the software safety validation process and the results of the validation documented.

Software Modification: Software modification requirements are effectively a sub-set of the software safety lifecycle requirements, with an emphasis on impact assessment and analysis.

Software Verification: Software verification is a planned activity concurrent with the development of each phase of the software safety lifecycle. Code shall be verified by static methods to ensure conformance to the design, the coding standards and the requirements of safety planning. The verification planning should refer to criteria, techniques and tools in order to address:
- the evaluation of safety integrity requirements,
- the selection of verification strategies, activities and techniques,
- the selection and use of verification tools,
- the evaluation of results and
- the corrective actions taken.

C.4.3 Comparison of IEC61508 with DS00-56

The different philosophies behind IEC61508 and Def Stan 00-56 are bound to lead to significant differences in requirements. However, this paper has found that there are also many similarities. These conclusions are therefore divided into 2 parts to firstly highlight the principle differences and then the similarities.
C.4.3.1 Principal Differences between the Standards

Legislation

IEC61508 is an international standard and thus, whilst it does require developers to abide by appropriate legislation it is not bound by the specific requirements of UK Health and Safety legislation and the principle of ALARP.

Safety Management

Def Stan 00-56 establishes core safety requirements, but expects developers to use their own expertise to develop safety requirements that are appropriate to the system in question under a safety management system, which itself is not over specified. This leaves the onus on the developer to justify any decisions taken on the embodiment of a safety management system. IEC61508 does not explicitly require a safety management system. However, the application of IEC61508 is itself a means of managing safety. Thus it actually is a more prescriptive way of managing safety, explaining safety management requirements in section 6 of Pt 1 and further guidance in Annex b to Pt 1. Similarly, MoD’s Project Orientated Safety Management System (POSMS) [11] is an attempt to provide a prescriptive approach to managing safety, consistent with the requirements of DS00-56.

Safety Case

This is a clear difference between the two standards. Whilst IEC61508 makes frequent references to the need for evidence, it is principally with regard to compliance with requirements. In effect this becomes the safety argument in IEC61508, whereas DS00-56 takes a much wider look at how evidence can be used to support a safety argument. This ‘argument’ approach is missing in IEC61508. However, it is fundamental to Def Stan 00-56 because the standard expects developers to provide justification for all aspects of safety. The safety argument then becomes a means of expressing this justification.

The safety argument is the ‘glue’ that holds together the evidence contained in the safety case and is the basis for the safety case report. None of these are mentioned as specific requirements in IEC 61508, but it clearly states a number of requirements covering the production of evidence and demonstrating compliance. There is also a degree of latitude provided in some areas, which allows a degree of interpretation. So, whilst IEC61508 does not explicitly cover safety cases, it has the ability to lend itself to such approaches. However, this latitude also creates ambiguity in the extent to which claiming compliance with IEC61508 satisfies the integrity requirements of DS00-56.

‘Strength’ of objectives
As discussed in the comparison between DO178B and DS00-56, there are differences between DO178B objectives being ‘required’, ‘required with independence’ or ‘optional’, and DS00-56 objectives being required with or without diversity. There is a similar distinction in IEC61508, where techniques etc. are classified as 'highly recommended', 'recommended' or 'not recommended'.

C.4.3.2 Similarities between the Standards

The table in Annex C.A attempts to show the degree of correlation between the two standards. The table does assess the degree of correlation in terms of High (H), Medium (M) and Low (L), which is somewhat subjective and would benefit from further scrutiny. However, the timescales of this task have prevented a more detailed analysis and mapping of requirements between the two standards. Nevertheless, the summaries below do provide a good overview of how the two standards compare.

The principle areas where there is a high degree of similarity are covered below

Scope and Safety Process

Both standards provide similar requirements for identifying the scope and system boundary. They also both emphasise the safety process, although IEC61508 is much more prescriptive. Def Stan 00-56 stresses the need for process, but stops short of specifics. However, there is ample guidance provided in part 2.

Risk Management

The principles behind the identification of hazards, accidents and classification of risks are dealt with in very similar ways by both standards. This also extends to hazard analysis and risk estimation.

Risk reduction and acceptance are not too dissimilar except that Def Stan 00-56 does mandate the use of ALARP.

Safety Requirements and Evidence

There is a high degree of correlation covering the traceability of requirements and the need for compelling evidence to demonstrate compliance. This is stressed throughout IEC61508 and whilst the approach in Def Stan 00-56 is less prescriptive, the objective is broadly the same.

Interface

With the exception of the reference to the safety case, IEC61508 emphasises the need to identify interfaces between systems, hardware and software and between operators and the system, which is also expressed in Def Stan 00-56.
Managing Changes and Feedback

There is a high degree of correlation here both in terms of recognising the need to address changes and modifications and ensuring that safety is not compromised. IEC61508 does benefit from having more emphasis on software development, perhaps recognising the issues surrounding software modification.

C.4.3.3 Summary

The above comparison should be subjected to further scrutiny in order to obtain a wider consensus. However, in acknowledging the relatively short timescales to achieve this task it does suggest that such scrutiny could be best applied by a more detailed analysis of the specific requirements in each standard. In particular, there has been little opportunity to review the HR/R tables within 61508 against any similar requirements in 00-56. Such further analysis should explore the level of evidence that each standard is expecting to satisfy the requirements.

Therefore, it is recommended that this paper is taken forward as the catalyst for further study to provide this level of detail.

C.4.4 Thales’ approach to evidence requirements for software

C.4.4.1 Introduction

The following paragraphs have been obtained from consultations with a number of business units within Thales. The views and opinions expressed are derived from individuals who have current experience of developing software on safety critical and safety related projects. However, the limited time available to complete this aspect of this task, has not enabled all of these views and opinions to be backed up by hard evidence. Nevertheless, the information provided has been supplied by individuals who have genuinely held views, are experienced in software development and have influence over the software development processes used.

Very few of programmes within Thales UK are categorised as safety critical. However, whilst the intent of Def Stan 00-56 is to cover all safety related systems, it was felt that the information from systems involving higher integrity levels would provide the most useful source of information for this paper. Thales UK operates a number of programmes which involve the use of High Integrity systems. One of these is the Watchkeeper UAV programme which aims to comply with DS00-56 Issue 3, which was current as an interim standard at the project’s start. A decision has been taken that there will be no benefit in taking steps to comply with Issue 4, and that the project’s compliance with Issue 3 provides adequate compliance with Issue 4. Further details of how this programme has sought to comply with Def Stan 00-56 Issue 3 is contained below, together with a number of issues that have been identified.
There are many more programmes within Thales UK which address safety issues for lower integrity levels. It was deemed worthwhile providing some feedback from these programmes regarding the issues they have applying current MOD safety standards. This feedback is provided at sections C.4.4.3 and C.4.4.4.

Watchkeeper is a multinational programme, so some additional information is provided describing the issues surrounding the merging of safety requirements and safety cases from suppliers who are not usually within the DS00-56 sphere of influence. This feedback is provided at section C.4.4.5.

An overall summary is in C.4.4.6.
C.4.4.2 Thales Aerospace, Watchkeeper PCMO

4.4.2.1 Def Stan 00-56 Issue 3 Experience

Safety Analysis

Safety analysis is carried out (by Praxis, contracted by PCMO) in a manner that is consistent with 00-56 Issue 3. Accidents and hazards have been identified, classified and managed. Whilst some hazards were initially classified (Using Def Stan 00-56 Iss 2 definitions) as level B, subsequent controlling measures have reduced all of those to class C or D. Fault Tree Analysis has been used to identify functional behaviour or failures, that may lead to hazards, at the level of subsystem decomposition. This has allowed mitigating safety requirements to be placed upon subsystems to reduce the likelihood of those failures, or to reduce their severity, for example by ensuring detectability.

Thereafter, a key facet of the Watchkeeper Safety Case is the claim that all safety requirements have been satisfied. For each safety requirement, the satisfaction of that claim is provided by the subsystem supplier to whom the requirement was given. In general, then, the Watchkeeper Safety Case is tree structured, with the various supplier’s Safety Cases forming the lower part of the structure (by reference rather than integration) to substantiate the claim that all safety requirements are satisfied.

Thus Watchkeeper PCMO passes responsibility to each supplier for the achievement of safety requirements and for the provision of a Safety Case making the claim for the adequacy of that achievement. This is entirely conventional and consistent with DS00-56 Issue 3.

Safety Targets

Watchkeeper is required to meet numerical overall safety targets. In order to achieve numerical safety targets for hazards at the top of fault trees, it is necessary to state numerical targets for all the causal base events at the bottom of the fault trees, and thus to state numerical criteria for the mitigating safety requirements.

Watchkeeper is a software-intensive system, and functional safety analysis has shown a significant dependence upon software functionality in several subsystems in meeting those targets. Thus failures that are predominantly systematic in nature have to be given numerical failure rate targets, and safety requirements that must be implemented mostly in software must be stated with numerical achievement targets. The demonstration of numeric failure rate targets when allocated to software can be problematic.

SIL Allocation
Watchkeeper PCMO passes functional safety requirements to suppliers that are typically stated with both a numerical failure rate and an equivalent SIL are stated per requirement, not per subsystem. Throughout Watchkeeper a consistent equivalence is used from the SMP of $\text{SIL1} \equiv 5.5 \times 10^{-3}$, $\text{SIL2} \equiv 5.5 \times 10^{-4}$, $\text{SIL3} \equiv 5.5 \times 10^{-5}$. Responsibility for showing that both the numerical and SIL criteria of the requirements are satisfied is placed firmly upon the suppliers, although that is not always easy to achieve as described below in section C.4.4.3.

How a supplier chooses to make a claim that a requirement is satisfied to SILn is not prescribed to suppliers. Informal guidance has been given that they might choose to claim compliance with 00-55 Issue 2 [8], and that approach is being taken by several Watchkeeper supplier, particularly those that are Thales organisations. Other suppliers have chosen different approaches. For example, one key supplier, Elbit, who is to supply the UAV itself including the SIL3 Vehicle Management System and the SIL2 Ground Flight Control Computer, is apparently basing its claim for achievement of SILs on comparison with achievement of DO-178B DALs, partly supported by the CAA in Israel.

Furthermore, whether a supplier chooses to ignore the SIL criterion of a requirement and attempt to claim that a requirement is satisfied with adequate numerical reliability is not prescribed. Only one supplier, Thales Optronics, suppliers of the exploitation subsystem, seems to have considered that approach but their initial attempts to construct such an argument were not convincing when presented to independent assessors, and the approach was dropped in favour of arguing SIL compliance.

4.4.2.2 Compliancy Issues with Def Stan 00-56 Issue 3

Watchkeeper PCMO has been forced to provide suppliers with dichotomous requirements. They are required to comply with DS00-56 Issue 3. They are also required to meet requirements to SILn, which DS00-56 Issue 2 facilitated but DS00-56 Issue 3 does not. Statements of Work provided to suppliers do, in fact, reference DS00-56 Issue 2 as well as DS00-56 Issue 3 to allow for this.

Whilst it can be argued that this dichotomy is not considered problematic for Watchkeeper PCMO or its suppliers, there is no doubt that it has struggled – and ultimately not completely succeeded – to manage the project wholly consistently with DS00-56 Issue 3, and has had no choice but to continue to depend upon DS00-56 Issue 2 and DS00-55 Issue 2.

This seems to show, very clearly, that there is a serious inconsistency between the more qualitative approach, represented by the change from DS00-56 Issue 2 to DS00-56 Issue 3, and the continued quantitative dependence upon numerical overall safety targets for military systems.

Watchkeeper PCMO is presently preparing the Safety Case, initially for UK flight trials. Making an ALARP argument is, of course, a significant aspect of that case.
Effort is concentrated predominantly upon making the ALARP case for the class C risks, with the class D ones being considered to be acceptably low without further measures. It is not considered that any of the class D risks could be removed completely by any further measures taken, so any such measures could only leave a class D risk as still a class D risk. On the other hand, the ALARP argument must show there are no further practicable measures to reduce the class C risks to class D.

C.4.4.3 Thales Underwater Systems, Watchkeeper GMCC(M) Software Development

4.4.3.1 Background

Thales Underwater Systems, TUS, is a primary subsystem supplier (i.e. it is contracted directly to Watchkeeper PCMO rather than some other supplier). It is noteworthy partly because its contractual delivery is of software only.

The GMCC(M) Software – Ground Management Control Computer (Mission component software) – has been given some requirements with a SIL1 and/or $5.5 \times 10^{-3}$ criterion.

The first full delivery of GMCC(M) Software has been made, with a complete Safety Case. The purpose of the first delivery does include use in control of flight, but the Safety Case does not provide assurance for that purpose. Accordingly, some evidence is incomplete at delivery, so the Safety Case does state limitations on use.

As a subsystem supplier, TUS has no visibility at all of the possible hazardous consequence of any failure of the GMCC(M) Software to fulfil its specified functionality, nor of the extent of control of such failures outside its own scope and thus of the probability that any such failure might result in an accident.

Thus TUS has no foundation upon which to base a risk classification. That situation pertains to other suppliers, too, except where a supplier’s hazard is manifest at the same boundary as an accident may occur at (i.e. the supplier’s equipment can directly cause an accident).

Therefore, TUS considers a GMCC(M) Software hazard to be “a failure, manifest at the output boundary of the software, that may, partially or wholly, cause or fail to prevent a Watchkeeper Hazard and thus may, partially or wholly, cause or fail to prevent an Accident.” In other words, a GMCC(M) Software hazard is just a potential cause of a Watchkeeper hazard.

Safety Analysis Approach
As a subsystem supplier whose deliverable has no direct accident boundary (being wholly software), TUS can only identify hazards as manifest failures of itself, and it cannot perform risk classification.

TUS is aware that its necessary definition, above, of a hazard is different from that in DS00-56 Issue 3, “A physical situation or state of a system, often following from some initiating event, that may lead to an accident.”, and that it is unable to fully comply with some aspects of DS00-56 Issue 3 by having no foundation upon which to base either risk classification or justification of risk reduction.

Instead, TUS has identified all failures at its output boundary as hazards, because it doesn’t know there are not dangerous consequences of any such failure. Some of those identified hazards are seen to be mitigated by safety requirements that have already been stated to TUS. The rest are stated to PCMO as unclassified hazards, each with an assumption that any hazardous consequence is adequately controlled outside the scope of the GMCC(M) Software.

For those hazards that are mitigated by given safety requirements, TUS has performed detailed safety analysis to identify and defend against any potential fallibility in its fulfilment of those safety requirements, and has generated some derived safety requirements that it has implemented. The derived safety requirements have been given the same SIL as the safety requirements they are derived to support.

ALARP

Since TUS has been unable to classify the risks associated with the hazards it has identified, it cannot make a claim that it has reduced those risks, or that it considers them to be ALARP. Instead, TUS places a hazard in ALARP status (the desirable formal Hazard Log status) if either “it is considered that the satisfaction, to the specified integrity level, of Given Safety Requirements will make the occurrence of the Hazard adequately unlikely” or “assumptions have been made and stated to PCMO that the Hazard is adequately controlled outside the scope of GMCC(M) Software”.

In other words, all TUS hazards are ALARP if all the given safety requirements are met, along with derived safety requirement to defend against them not being met, and the other identified potential hazards not mitigated by those given safety requirements are formally assumed to be adequately mitigated elsewhere.

Since TUS has fully implemented all given and derived safety requirements, it may claim it has done everything it possibly can, and while it cannot claim it knows risks are as low as practicable if it cannot classify them, it does know there are no identified steps it can take to reduce them any further. So risks, classified or not, are as low as TUS can make them – which is the best possible ALARP claim that can be made.
4.4.3.2 Compliancy Issues with Def Stan 00-56 Issue 3

While being aware that there are ways it cannot, as a subsystem developer without an accident boundary, possibly comply with some aspects of DS00-56 Issue 3, TUS has sought to assure the safety of its deliverable and has followed a process that is consistent with DS00-56 Issue 3 where possible (i.e. mostly) and not inconsistent with DS00-56 Issue 3 at all. Its Safety Case includes a DS00-56 Issue 3 Compliance Matrix accordingly.

This approach to DS00-56 Issue 3 was made very clear to Watchkeeper PCMO, in TUS’s first delivered Safety Management Plan and subsequent safety audit, and considered appropriate.

In order to satisfy the SIL1 criterion of its safety requirements, TUS has claimed adequate compliance with DS00-55 Issue 2.

C.4.4.4 Thales Aerospace Technical Business Unit (TBU), Watchkeeper Client Server

4.4.4.1 Background

The Watchkeeper Client Server is a major subsystem with several component subsystems. TBU is implementing a majority of the components, which are collectively referred to as the Infrastructure. Three other major components are identifiable: Flight Plan Validation, being implemented by TAO; Exploitation Client, being implemented by TOL; and Mission Planning, being implemented by TBU itself. Integration of the various components is also carried out by TBU. Safety assurance of Client Server is in TBU’s scope and has been subcontracted to Ebeni.

4.4.4.2 Compliancy Issues with Def Stan 00-56 Issue 3

The Client Server, like the GMCC(M) made by TUS, is a wholly software system, though Ebeni do consider the PC hardware when assuring it. In general, then, TBU and Ebeni are in a very similar situation to TUS, described above. Unlike TUS, they initially tried to perform risk classification and argue risk reduction. Some confusion resulted, with attempts to analyse and manage matters that were outside Client Server scope of visibility and responsibility, leading to a clarification of safety boundary. As for the TUS GMCC(M), PCMO is not depending on any claims of Client Server risk classification or reduction, only that the Client Server safety requirements are met. Hazard identification and management have been performed in a similar manner to TUS.

Thus, like TUS, TBU and Ebeni have been unable to comply fully with the risk classification and reduction of DS00-56 Issue 3, yet might claim that risks are as low as possible. However, unlike TUS, it is possible that TBU have made some (no doubt very pragmatic and justifiable) compromises on the implementation of
safety requirements, in which case some ALARP justification may be necessary within the Client Server Safety Case.

C.4.4.5 Elbit, Watchkeeper UAV Including VMS And GFCC

4.4.5.1 Background

Elbit is a supplier of main Watchkeeper components, including some that are safety critical. Elbit is an Israeli company, so is not formally mandated to DS00-56 in ways that UK suppliers are.

Elbit has accepted requirements with SIL1, SIL2 and SIL3 criteria. It is not yet clear how they will argue satisfaction of those criteria, since they have not yet delivered a Safety Case. Apparently their safety process is based on Mil-Std-882, and their claims for SIL achievement will be based on meeting DO-178B, without formal certification but with some measure of Israeli CAA underwriting.

4.4.5.2 Compliancy Issues with Def Stan 00-56 Issue 3

In general, it is not yet clear how, or to what extent, Elbit will show DS00-56 compliance. Needless to say, this is a matter of continuing interest to Watchkeeper PCMO! On the other hand, it is very clear that Elbit have an exceptional history of satisfactory UAV production and assurance.

There is an obvious dichotomy. UK suppliers are in the formal scope of DS00-56. They know they must comply with it, and they are generally able to do so because it reflects and builds upon established UK military development and assurance methods – which are historically and culturally of a high standard. Foreign suppliers are not in that DS00-56 scope, and they use established development and assurance methods that might be of an equivalent standard but are different and not necessarily consistent with DS00-56 or the culture it reflects.

Watchkeeper is a UK military procurement project, for which DS00-56 is mandatory, that is partly being developed outside the UK, where DS00-56 is not mandatory or culturally established. Thales’ experience is that it is not clear that DS00-56, either Issue 3 or Issue 4, is wholly suitable for this multinational purpose or, perhaps, that there is enough understanding during early procurement – when international collaboration is initiated - of the extent to which a non-UK supplier may have to adapt their whole lifecycle to meet the acceptability criteria of the UK MoD.

C.4.4.6 Thales Def Stan 00-56 Evidence Summary

It is clear from the experience on the Watchkeeper programme that, despite best intentions, it has proven difficult to apply Def Stan 00-56 Issue 3 for the following reasons:
• Watchkeeper has found it necessary to continue applying SILs despite them not being supported by Def Stan 00-56 Issue 3;
• Sub-contractors who supply software, but are not cognisant of the whole system boundary, have difficulty complying with Def Stan 00-56 Issue 3 for even low integrity levels and have had to resort to compliancy with Issue 2;
• It is very difficult to apply ALARP to software only systems, although this issue is applicable to all safety standards;
• It is not clear how multinational projects should apply Def Stan 00-56 Issue 3;
• It is not clear what is best practice for the development of safety related and high integrity software.

C.5 CONCLUSIONS AND CONCERNS

The following paragraphs identify the conclusions of the comparison between the standards and the concerns expressed by the Thales.

Both DO178B and IEC61508 emphasise the need to have evidence that demonstrates the satisfaction of identified safety requirements, which is one of DS00-56’s objectives. However, the level of evidence has not been possible to ascertain within this paper. It is recommended that further work is undertaken to examine in more detail the evidential requirements of each standard, and to what extent satisfaction of specific DAL/SIL requirements meets 00-56’s integrity requirements.

Where they differ is in the relationship between the satisfaction evidence and overall system safety assessment:

• DO178B says nothing on safety management, essentially assuming that system safety assessment, hazard identification and integrity target setting are performed before the software development process it describes starts. In the civil aviation domain, the system safety aspects are addressed by ARP 4754 and 4761, but this approach is not mandated. Where there are common objectives in DO178B and the Defence Standard, there is broad agreement on the approach, certainly for low and medium confidence levels. At high level, the Defence Standard requires diversity, whilst DO178B stresses coverage and independence;
• IEC61508 more closely aligns with DS00-56, particularly in terms of the requirement for system risk assessment etc, as well as satisfaction of safety requirements. This is possibly unsurprising, as the original version of DS00-56 was written as the UK military interpretation of the forerunner of IEC61508. One difference is that the Defence Standard has more focus on the safety argument and the safety case than IEC61508 (which essentially assumes that if appropriate safety requirements are identified and satisfactorily implemented, the system will be safe). Another difference is
the prominent position of the ALARP\(^1\) principle in the Defence Standard, which, whilst it is described in IEC61508 (part 5 Annex B) as an example of a method for the determination of safety integrity levels, is not mandated. Conversely, IEC61508 has more focus on the design development process.

Thales' experience in applying Defence Standard 00-56 issue 3 to some software projects suggest that sub-contractor have problems following DS00-56 issue 3/4, as they lack the visibility of the system boundary needed to make a system-wide risk assessment and ALARP argument. This has led Thales to reverting to more prescriptive techniques, such as the use of SILs and the adoption of Defence Standard 00-56 issue 2. There is also a tension between many system’s need for achievement of quantitative safety targets and the ability to justify quantitative failure rates for software.

\(^1\) ALARP is inherent in UK safety legislation. Whilst a recent European ruling has agreed that its use in the UK is consistent with pan-European requirements, there is no suggestion of it being adopted more broadly. However, the requirement for continuous improvement is recognised internationally in related contexts, such as CMM [10].
C.6 REFERENCES


C.7 GLOSSARY

<table>
<thead>
<tr>
<th>CA</th>
<th>Certification Authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAA</td>
<td>Civil Aviation Authority</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>CC1/2</td>
<td>Control Category 1 or 2</td>
</tr>
<tr>
<td>CMM</td>
<td>Capability Maturity Model</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial Off The Shelf</td>
</tr>
<tr>
<td>DAL</td>
<td>Design Assurance Level</td>
</tr>
<tr>
<td>DER</td>
<td>Designated Engineering Representative</td>
</tr>
<tr>
<td>DS</td>
<td>Defence Standard</td>
</tr>
<tr>
<td>FAA</td>
<td>Federal Aviation Authority</td>
</tr>
<tr>
<td>FMEA</td>
<td>Failure Modes and Effects Analysis</td>
</tr>
<tr>
<td>FTA</td>
<td>Fault Tree Analysis</td>
</tr>
<tr>
<td>ISA</td>
<td>Independent Safety Auditor</td>
</tr>
<tr>
<td>PSAC</td>
<td>Plan for the Software Aspects of Certification</td>
</tr>
<tr>
<td>QA</td>
<td>Quality Assurance</td>
</tr>
<tr>
<td>SAS</td>
<td>Software Accomplishment Summary</td>
</tr>
<tr>
<td>SIL</td>
<td>Safety Integrity Level</td>
</tr>
<tr>
<td>SMP</td>
<td>Safety Management Plan</td>
</tr>
<tr>
<td>SQA</td>
<td>Software Quality Assurance</td>
</tr>
</tbody>
</table>
## Annex C.A  Def Stan 00-56 Mapping to IEC61508

<table>
<thead>
<tr>
<th>Def Stan Requirement Heading</th>
<th>Def Stan 00-56 Requirement</th>
<th>IEC 61508 Requirement</th>
<th>Degree of Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General Requirements</strong></td>
<td>The contractor shall identify and demonstrate compliance with all relevant safety legislation, regulations and MOD policy relevant to the system and any contractual safety requirements.</td>
<td>The objective of the conceptual requirements is to gain an understanding of the equipment or system under consideration, and its operating environment (including legislative requirements, to enable other safety lifecycle activities to be carried out</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>The contractor shall define the system boundary and operating environment in agreement with the Duty Holder (MOD) and implement measures to ensure effective stakeholder representation, particularly during risk management activities.</td>
<td>The overall scope determines the system boundary and scope of the hazard and risk analysis.</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>The contractor shall provide visibility of the safety process, passing all information to the Duty Holder or other organisations as agreed, ensuring that it is under a suitable configuration management process that is consistent with the system configuration.</td>
<td>The overall safety lifecycle requirements are built around a framework with the intention of systematically achieving the required safety integrity level for E/E/PES. This lifecycle is used as the basis for conformance with the standard.</td>
<td>H</td>
</tr>
<tr>
<td><strong>Roles and Responsibilities</strong></td>
<td>The Project Manager shall be the primary point of contact between the contractor and Duty Holder. The Project Manager is responsible for meeting the requirements of the standard.</td>
<td>The standard details a number of aspects for consideration, which cover policy, procedures, training, techniques and identification of suitable organisations and personnel to carry out these activities</td>
<td>L</td>
</tr>
<tr>
<td></td>
<td>The Project Safety Manager shall be appointed by the contractor to manage project safety on behalf of the Project Manager and act as primary point of contact for safety issues</td>
<td>No requirement specified</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>The contractor shall establish a safety committee to allow participation of all relevant stakeholders and oversee, review, and endorse safety management and safety engineering activities</td>
<td>No requirement specified</td>
<td>N</td>
</tr>
<tr>
<td><strong>Safety Management</strong></td>
<td>The contractor shall produce a Safety Management Plan</td>
<td>There is no specific requirement for an overall safety management plan. However, there are various references to planning activities throughout the standard, but these cover specific safety and software activities</td>
<td>L</td>
</tr>
<tr>
<td><strong>Safety Case</strong></td>
<td>The contractor shall produce a safety case consisting of a structured argument supported by a body of evidence that provides a compelling, comprehensive and valid case that a system is safe for a given application in a given environment</td>
<td>IEC 61508 has no specific requirement for a safety case as described above, but does refer to the need for evidence to demonstrate that requirements have been met.</td>
<td>L</td>
</tr>
<tr>
<td></td>
<td>The contractor shall produce safety case reports that summarise the safety case at a particular instant. This</td>
<td>No requirement specified, but there are numerous references to safety and software assessment and validation. It</td>
<td>L</td>
</tr>
</tbody>
</table>

*Standard of Best Practice UNCLASSIFIED Page 252 of 397*
<table>
<thead>
<tr>
<th>Def Stan Requirement Heading</th>
<th>Def Stan 00-56 Requirement</th>
<th>IEC 61508 Requirement</th>
<th>Degree of Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>provides visibility and assurance to the Duty Holder that safety is effectively managed and highlights areas of safety that require management attention</td>
<td>has to be presumed that this information is communicated to stakeholders in a comprehensible manner</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>Risk Management</td>
<td>The contractor shall identify all hazards and potential accidents, so far as is reasonably practicable, and manage their risks as appropriate.</td>
<td>There are three objectives for hazard analysis. Determine the hazards and hazardous events for the equipment or system Determine the event sequences leading to the hazard Determine the risks associated with the hazard.</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>All safety risks shall be reduced to levels that are ALARP.</td>
<td>No requirement specified</td>
<td>N</td>
</tr>
<tr>
<td>Hazard Identification and Hazard Analysis</td>
<td>A Hazard Log shall be implemented and managed as part of the Safety Management System</td>
<td>IEC 61508 does not refer to a Hazard Log as such, but it does require that the source of hazards are determined and documented</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>Hazard identification and analysis shall be carried out to identify potential hazards and determine the potential accident sequences. The contractor shall also demonstrate the adequacy and suitability of any processes and techniques used</td>
<td>There are three objectives for hazard analysis. Determine the hazards and hazardous events for the equipment or system Determine the event sequences leading to the hazard Determine the risks associated with the hazard.</td>
<td>H</td>
</tr>
<tr>
<td>Hazard Identification and Hazard Analysis cont</td>
<td>Certain hazards may be identified that are outside the scope of the contractor’s control. Such hazards shall only be closed with the agreement of the owner of the hazard and when the risk has been reduced ALARP and broadly acceptable or tolerable</td>
<td>This is really covering the management of hazards and is not covered by IEC 61508 explicitly</td>
<td>N</td>
</tr>
<tr>
<td>Risk Estimation</td>
<td>The contractor shall carry out risk estimation to determine the severity and likelihood of any consequences for the hazards and potential accidents and estimate the overall risk for the system</td>
<td>There are three objectives for hazard analysis. Determine the hazards and hazardous events for the equipment or system Determine the event sequences leading to the hazard Determine the risks associated with the hazard.</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>The effectiveness of the risk estimation process and suitability shall be demonstrated including the recording of assumptions, data and calculations.</td>
<td>This is really covering the management of hazards and is not covered by IEC 61508 explicitly</td>
<td>N</td>
</tr>
<tr>
<td>Def Stan Requirement Heading</td>
<td>Def Stan 00-56 Requirement</td>
<td>IEC 61508 Requirement</td>
<td>Degree of Correlation</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Risk and ALARP Evaluation</td>
<td>The contractor shall establish tolerability criteria based upon relevant legislation, standards and MOD policy, which shall form the basis for making an assessment of whether a risk is broadly acceptable, tolerable or unacceptable.</td>
<td>No requirement specified</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>The contractor shall undertake risk and ALARP evaluation for the identified hazards and accidents against the tolerability criteria and review and revise this throughout the life of the contract.</td>
<td>IEC 61508 does not cover ALARP and thus there are no requirements which cover ALARP evaluation</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>The contractor shall also demonstrate the effectiveness and suitability of any processes and techniques used.</td>
<td>IEC 61508 does not cover ALARP and thus there are no requirements which cover ALARP evaluation</td>
<td>N</td>
</tr>
<tr>
<td>Risk Reduction</td>
<td>Where the risk is not assessed to be broadly acceptable or tolerable and ALARP, the contractor shall carry out risk reduction by identifying and implementing a combination of mitigation strategies until the tolerability criteria are met.</td>
<td>Risk reduction requirements do have some broad alignment with the requirements of Def Stan 00-56.</td>
<td>M</td>
</tr>
<tr>
<td>Risk Acceptance</td>
<td>The contractor shall agree a process for risk acceptance with the Duty Holder. Risks shall be accepted when there is agreement that sufficient evidence has been provided that risks have been reduced ALARP.</td>
<td>The standard effectively sets its own risk acceptance criteria, thus a contractor would be deemed either compliant or non-compliant.</td>
<td>M</td>
</tr>
<tr>
<td>Safety Requirements and Evidence</td>
<td>The contractor shall maintain records that show traceability between each safety requirement and the source of that requirement. Compelling evidence, commensurate with the risk and complexity of the system, shall be provided demonstrating that safety requirements have been met.</td>
<td>IEC 61508 makes numerous references to the validation of safety requirements.</td>
<td>H</td>
</tr>
<tr>
<td>Safety Requirements and Evidence cont</td>
<td>Safety requirements that lead to the realisation of mitigation strategies shall provide evidence that is commensurate with the level of risk reduction resulting from that requirement.</td>
<td>IEC 61508 makes numerous references to the validation of safety requirements.</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>Evidence shall be integrated within the safety case to show that all safety requirements have been met.</td>
<td>Although IEC 61508 does not specifically refer to a safety case, it does require records to be kept that demonstrate that safety requirements have been met.</td>
<td>M</td>
</tr>
<tr>
<td>Interfaces</td>
<td>The Safety Management Plan shall define processes and procedures for managing interfaces. The contractor shall identify and provide such information as is necessary to allow interfaces between systems to be safely implemented. Any dependencies shall be recorded in the safety case.</td>
<td>With the exception of the reference to the safety case, IEC 61508 places great emphasis on the need to identify interfaces between systems, hardware and software and between operators and the system.</td>
<td>H</td>
</tr>
<tr>
<td>Def Stan 00-56 Requirement Heading</td>
<td>Def Stan 00-56 Requirement</td>
<td>IEC 61508 Requirement</td>
<td>Degree of Correlation</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------</td>
<td>-----------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Managing Changes and Feedback</td>
<td>The contractor shall develop and implement plans for identifying and addressing changes to ensure the safety of the system, operating a change control system, so that the safety impact of the change is assessed</td>
<td>IEC 61508 also recognises that changes or modifications are necessary, particularly where software is concerned. This is covered from the perspective of ensuring that the software is modifiable.</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>In all cases, where necessary, remedial action shall be taken to ensure the safety of the system in accordance with the risk management requirements of the standard</td>
<td>IEC 61508 specifically requires that any modifications or changes are treated as a sub-set of the overall safety requirements</td>
<td>H</td>
</tr>
<tr>
<td>Safety Audits</td>
<td>The contractor shall carry out safety audits as specified in the contract, or within reasonable requirements of the Duty Holder</td>
<td>IEC 61508 has some requirements for auditing</td>
<td>L</td>
</tr>
<tr>
<td></td>
<td>The appointment of an ISA shall be at the discretion of the MOD and shall have access at reasonable times to all premises and records of the contractor or sub-contractors</td>
<td>It also makes a number of recommendations for independent validation and verification for both hardware and software design as a means of improving overall safety effectiveness</td>
<td>M</td>
</tr>
</tbody>
</table>
Annex D  COTS and Legacy

This Annex contains a discussion of the application of DS 00-56 to COTS and legacy software. The material herein is taken from the SSEI deliverable “Working Paper on the Evidential Approach, Addressing COTS and Legacy Software”. All references in this are to internal sections of this Annex, and citations are to the works listed in Annex D.9.

Executive Summary

The overall purpose of this task is to produce guidance on software for use in the context of DS 00-56 Issue 4. This working paper is a precursor to the eventual guidance, focussing on the use of COTS and legacy software.

We firstly examine some of the problems associated with satisfactorily demonstrating the safety of COTS products. We then discuss how safety arguments should be constructed to alleviate these problems, emphasising the need for different types of evidence to support these arguments. Finally, we present a sample method of selecting and evaluating COTS products for use in safety-critical and safety-related systems. In addition, we briefly describe the Watchkeeper project to illustrate some of the issues surrounding the use of COTS, and to demonstrate where effort should be concentrated to adequately assure the safety of commercial products.

In general, the use of COTS (Commercial Off-The-Shelf) software is becoming widespread in UK MOD systems. The perceived advantages of COTS products over bespoke products include the reduced cost to obtain COTS, the faster deployment time, the proven safety or reliability of these products, and the guarantee of software support [16].

However, the use of COTS components can also be problematic from a safety perspective. Evidence about the safety of a COTS component is typically harder to obtain than evidence about the safety of a bespoke component. Consequently, systems using COTS may be placed into operation without adequate assurance as to their safety. This presents a problem because consequently, the safety of MOD systems containing COTS components cannot be assured as necessary.

We have found that COTS vendors do not always supply sufficient evidence about the software development process and the internal implementation of their component. Furthermore, when evidence is provided it may be developed to show compliance to a different standard, and also may be prohibitively expensive. In addition, we have found that the methods used for COTS selection vary significantly from one project to another, with very few considerations which are common to all methods. Most of the selection methods typically do not explicitly consider safety requirements when selecting a COTS product. Consequently, COTS products will sometimes be selected for use in a system where it cannot be shown that they do not contribute unacceptably to system hazards.
To address this problem, this working paper provides guidance on how to conduct a gap analysis to identify assurance deficits for COTS components. This includes a discussion of the types of evidence which are typically available for COTS components, and the ways in which this evidence may not be suitably rigorous for assuring safety as required. We also provide guidance on how to address these assurance deficits by generating new evidence to increase confidence in the COTS component. Finally, we present a customizable method of COTS selection which explicitly addresses the issue of safety requirements.

We recommend that this, subject to validation and in combination with the future work to be performed by March 2009, be incorporated into a Statement of Best Practice for compliance with Defence Standard 00 56 Issue 4.
# TABLE OF CONTENTS

- **Executive Summary** ........................................................................................................253
- **Table of Contents** ...........................................................................................................255
- **D.1 Introduction** ................................................................................................................256
- **D.2 The Evidential Approach** ............................................................................................258
- **D.3 COTS and Evidential Requirements** ..........................................................................260
  - **D.3.1 Assessing Available Evidence** ..............................................................................261
  - **D.3.2 Generating New Evidence** ..................................................................................278
- **D.4 Assurance Deficits and Providing "Enough Evidence"** .............................................284
  - **D.4.1 Evidence Types and Outcomes** ...........................................................................285
  - **D.4.2 Assessing Evidential Outcomes** .........................................................................291
  - **D.4.3 Sample COTS Evidential Rigour Outcomes** .......................................................296
- **D.5 Selecting a COTS component** ....................................................................................299
  - **D.5.1 Existing COTS Selection and Evaluation Methods** ............................................300
  - **D.5.2 A Contract-based Approach** ..............................................................................300
- **D.6 Conclusions** ...............................................................................................................315
- **D.7 Recommendations** ....................................................................................................316
- **D.8 Acknowledgements** ...................................................................................................316
- **D.9 List of References** .....................................................................................................316
- **D.10 Definitions** ................................................................................................................319
- **D.11 List of Abbreviations** ...............................................................................................320
- **D.12 Annexes** ....................................................................................................................320
  - **ANNEX D.A** A Contractual Language for COTS Safety Requirements ......................321
  - **ANNEX D.B** Watchkeeper: An Illustration .................................................................323
  - **ANNEX D.C** Gap Analysis Evaluation - Thales ............................................................329
  - **ANNEX D.D** Gap Analysis Evaluation - IBM ...............................................................332
  - **ANNEX D.E** Further Gap Analysis Evaluation - IBM ..................................................334
D.1 INTRODUCTION

The use of COTS (Commercial Off-The-Shelf) software is becoming increasingly prevalent in military systems. “COTS” is an umbrella term which has been used to describe a number of different types of software. These include:

Legacy software: this is software which was previously designed and implemented in-house. Source code and documentation for legacy software is likely to be available, although it is possible that the tools and techniques used in its development have since been found to be flawed. The software must then be assessed to determine whether the flaw could have affected its safety properties. It should be noted that not all legacy software is COTS (and neither is all COTS legacy!) although in some cases they may share common characteristics.

Commercial-Off-The-Shelf Software (COTS): software which has been previously developed by an external vendor. Notable characteristics of such software include a lack of access to documentation and process evidence, a lack of certification in a specific environment, and the presence of features which may be unnecessary for the proposed usage.

Option-selectable software: COTS products may provide a number of different configuration options, or a different configuration option may need to be added to a COTS product for use by the MOD.

Freeware: This is software which is in the public domain. Source code for this software is available, but there is often a lack of process evidence, or of evidence of good practice in development.

SOUP: Software Of Uncertain Pedigree [1] is typically non-commercial software where the ownership and responsibility for support is transferred to the acquirer along with the software. This includes both freeware and legacy software.

In this paper we will not attempt to describe the characteristics of all of these types of software. The software we describe will primarily be commercially-produced software which has been previously developed by an external vendor. In some parts of this document we will also consider freeware and legacy software; these sections will be clearly marked as discussing a potentially wider range of products.

In more detail, the types of COTS products that we will be considering in this paper share certain characteristics which are important when considering how their safety may be demonstrated [1, 13]. Firstly, the product already exists and is not intended to be modified by the acquirer, with the exception of freeware. Although in some cases the rights to the code can be obtained – or reverse-engineering can be used – altering the product in any way is likely to invalidate the conditions of sale and support. Secondly, any COTS product will usually...
contain functionality which is not required for the specific purpose for which it is purchased. While unnecessary, the presence of this functionality may nevertheless have an impact upon the component’s safety properties. Thirdly, almost all COTS products are subject to continuous change and upgrades performed by the supplier, each of which may impact their safety properties. Finally, there are limitations upon the available development and assessment data. Specifically, access to design documentation, source code, testing procedures and fault identification documents is typically limited.

Given these characteristics, there are some types of software which it is important to note have not been explicitly addressed in this paper. In particular, legacy software has not been considered to the same degree as COTS software. While there are similarities between COTS and legacy, there can also be pronounced differences. Both can be vulnerable to a lack of documentation – for legacy software, this documentation may no longer be available while for COTS the documentation may never have been available. Similarly, both may contain unneeded functionality for the proposed usage. However, legacy software differs in that it is sometimes possible to obtain the cooperation of the original developers. In this case, more evidence as to the safety of the product may be obtained. Furthermore, legacy software can usually be modified without the consequent invalidation of a warranty. However, there may be problems with performing such modification – for example, the language used to write the original software may no longer be in common use. This can be an issue both for coding and for compiling and executing the software. A detailed exploration of the similarities and differences between legacy and COTS software is proposed in the future for the SSEI. Additionally, we also propose that case studies to determine the types of use of legacy software in industry will be included to inform the upcoming Standard of Best Practice.

There is another specific type of COTS product which we have not considered in this paper. Software of this type is that which has been developed specifically for a particular safety-critical or safety-related system, in partnership with the developers or owners of that system. An example of this is when a supplier is contracted to deliver a COTS product meeting certain safety requirements. When developing these products, the supplier(s) and acquirer(s) may work in close collaboration, and access to documentation, code and lifecycle data may be permitted to a greater degree. These products may also be customized for the particular safety needs of the application, and re-design and extension of the product is not unknown. Because these COTS products do not necessarily share the characteristics listed above, we propose – instead of including a discussion in this paper – that a case study of some of these is included in the upcoming Statement of Best Practice.

There are a range of perceived benefits associated with the use of COTS products as characterised above. These include a reduced cost of development, a faster deployment time, in some circumstances software support from the vendor (although this perceived benefit is less often the case for safety-related
software, as some vendors explicitly state that certain software should not be used in safety-related roles), and an increased measure of confidence since the software in question has been proven in use [16]. However, there are also problems associated with the use of COTS components, primarily when they are used in safety-critical or safety-related roles. Not all the perceived benefits eventuate in practice. For example, COTS vendors may cease to support a product, which can have a significant effect if a system has been developed to a stage where it is “locked in” to using that product. Similarly, development costs may be higher than for a bespoke product, or proven-in-use software may be intended by the vendor to be used in a manner which is different to the MOD’s proposed usage. From a safety perspective, evidence assuring the safety of COTS products is harder to obtain, and the costs associated with this alone can outweigh the benefits.

There are two different development scenarios involving the use of COTS products. Firstly, developers may choose to make use of COTS components in a bespoke system which is being developed. In this case, the bespoke components may be customised or altered as necessary in order to mitigate risks posed by the COTS component. That is, the software architecture can be refined and altered to enable the safety requirements arising from the use of COTS to be satisfied.

Secondly, developers may choose to update an existing system with COTS components in order to provide some extra functionality. In this case, the system architecture must be explicitly analysed for its ability to withstand the evolution associated with COTS products [27]. If the architecture is not resilient to change, extra assurance as to the safety of the COTS component itself will usually be required. This is the software architecture cannot necessarily be relied upon to mitigate risks posed by the commercial products either now or after upgrades.

In this paper we will discuss the issues faced when attempting to construct a safety argument for COTS products. Section D.2 introduces the evidential approach recommended by Defence Standard 00-56 Issue 4 [26]. Section D.3 discusses the evidence requirements for the safety of COTS components, describing how these can be satisfied by a combination of existing evidence and newly-generated evidence. Section D.4 describes what must be considered when judging whether sufficient evidence has been presented, as well as providing a discussion on the suitability of different types of evidence for COTS components. Section D.5 then presents a method for evaluating and selecting COTS components, by emphasising the safety and evidential requirements on these.

## D.2 The Evidential Approach

DS 00-56 Issue 4 recommends an evidence-based approach to software safety. That is, any claim about the safety of the software must be supported with rigorous evidence, and with a compelling safety argument demonstrating how this evidence supports the claim. This represents a departure from the perspective of
other standards, such as DO-178B [34], IEC 61508 [19] and DS 00-56 Issue 2 / DS 00-55.

These standards recommend the use of safety levels, sometimes referred to as SILs (Safety Integrity Levels) or DALs (Development Assurance Levels). Different safety levels are achieved by completing certain objectives within the software development lifecycle. Depending on the number and type of objectives completed, the system is attributed a particular “safety level”. These safety levels are then sometimes mapped to a particular failure rate, implying that the completion of a particular set of lifecycle objectives will produce a system which satisfies a target failure rate. It should be noted that this mapping is not widely regarded as good practice. Furthermore, DO-178B expressly states that this mapping should not be performed, owing to the two problems we list below.

These are two major problems with this approach. Firstly, there is no evidence that a good process will result in a good product (although there is a correlation between bad processes and bad products!). That is, there is no indication that completion of a given set of lifecycle activities really will produce a system with a target failure rate as claimed. Secondly, while the notion of “failure rate” has some credence in terms of hardware, it is almost meaningless when applied to software. Software failures are systematic, rather than random. That is, where an error exists in software, performing the same function with the same inputs will consistently result in the appearance of the error. Thus, the failure rate generally depends upon the exact environment and does not vary with age, physical conditions, or any of the other factors which affect hardware safety.

Using an evidential approach to software safety addresses the first of these issues. This approach does not permit any implicit assumptions as to the safety of a system produced using a particular lifecycle. Instead, evidence must be explicitly used to show that the software satisfies its safety requirements. The major feature of this approach is the presence of a compelling argument, supported by evidence, to demonstrate satisfaction of these safety requirements. One of the most subtle questions when assessing this approach is the determination of whether evidence is “good enough”. This gives rise to the notion of assurance, being the extent to which we are confident that a safety claim is true. A related concept is that of an assurance deficit, being the difference between the confidence we have in a safety claim and the confidence that we require in that claim.

The assurance we require of software will vary in proportion to its criticality. That is, if software is safety-critical, then we will require greater confidence that it satisfies its safety requirements than if it is only safety-related. This also provides a tentative solution to the issue of systematic failures in software. Instead of requiring the software to meet a target failure rate, we can instead require that the software does not fail in defined failure modes (otherwise known as Hazardous Software Failure Modes, or HSFMs) under known circumstances. Furthermore, we require this absence of failures to be supported with an appropriate degree of
confidence, or assurance. This approach provides a qualitative assessment of safety, rather than a quantitative one.

**D.3 COTS and Evidence Requirements**

When being used for COTS, the evidential approach as required in DS 00-56 can present some problems. These are largely to do with the difficulty involved in obtaining evidence to demonstrate the safety of COTS products. This does not, however, imply that the alternative “safety level” approach will be any more appropriate for assessing the safety of a COTS product. Evidential requirements can be applied regardless of the provenance of the software in question. On the other hand, process-based standards must be either waived entirely for COTS products, or applied retrospectively (an exception to this is when a COTS component is provided with a certification pack relating to a process-based standard). Neither of these solutions is satisfactory, and can result in the inclusion of unsafe components into systems.

As mentioned in Section D.1, the COTS products which are the focus of this paper share certain characteristics, which can make it difficult to obtain evidence to demonstrate their safety. The inability to modify a COTS product means that code written specifically to enhance its safety properties cannot be added to it. Additionally, the continuous change to which COTS components are subject means that new failures are potentially possible with each upgrade. Finally, the lack of access to source code, design documentation and development processes means that these cannot be used as evidence to demonstrate the safety properties of the product. Consequently, there is often a real lack of rigorous evidence to support a safety argument to the required confidence, or assurance.

However, not all COTS products demonstrate these shared characteristics to the same degree. Some products are rarely upgraded, some products may contain very little additional functionality beyond that required, and some COTS vendors make data relating to their product available to differing degrees. These differences have led to these products being characterised as either “thick SOUP” (where little information about development processes is provided) or “clear SOUP” (where source code and fault histories are provided, if not detailed development documents) [1]. In addition, many of the COTS components purchased by the UK MOD will have been assessed according to some standard (usually DO-178B or IEC 61508). This in itself requires the provision of a certain level of information about the product. The primary question when selecting and evaluating a COTS component using DS 00-56 is whether there is “enough” evidence that it meets its safety requirements. That is, the question is whether sufficient evidence can be obtained to assure the safety of the system to the level required.

There are several questions implicit in that statement. Firstly, what does “enough” evidence mean, and how can this be judged? Secondly, what is the “level of
assurance" required for a component? Finally, there is the question of acceptance, or the question of what form a safety argument should take to justify the claim of system safety, given the choice and extent of the evidence presented.

D.3.1 Assessing Available Evidence

The primary purpose of this section is to discuss what evidence is typically available to potential purchasers of a COTS product and to what extent that information can be trusted. This section will discuss only the evidence which is made available to potential purchasers of a COTS product. That is, we will postpone a discussion of the techniques which are used to generate new evidence until later sections.

It is important to note that information about the COTS product and evidence of the safety of the COTS product are not the same thing. Part of the software code may be made available; this constitutes information about the COTS product but does not in itself constitute evidence. In order to obtain evidence to support a safety claim, we would need to perform tests or analysis on this code. However, the provision of the code does make it easier to generate the evidence to support a claim.

This section is also intended to serve as guidance for conducting gap analysis to determine what evidence is available for the COTS component, and identify where this may not be sufficient to provide confidence in the safety of this component. Firstly, we discuss commonly available evidence sources, and recommend that all these sources be accessed for potential information. Secondly, we identify some common reasons why available evidence may not be suitable when assessing the component. This is presented in the form of a number of questions which must be asked to determine whether the environment in which the evidence was generated is “similar enough” to the target environment. These questions also serve to identify what new evidence would be needed to fill these evidence gaps. Finally, we discuss the trustworthiness of the available evidence. This trustworthiness assessment forms the final part of the gap analysis.

D.3.1.1 Commonly Available Evidence Sources

The evidence which may be available for a COTS product can vary greatly. The vendor may provide the results of tests conducted against particular safety requirements, such as worst-case execution times or the absence of particular failure modes. In addition, testimonials and reviews may be available from third parties. As a general rule, process evidence is typically not available, unless the COTS component is developed to an accepted standard and purchased with a certification pack. Consequently, there will often be no guarantee that the product was developed in accordance with good practice. Furthermore, the source code
for a COTS product is not always readily available, nor is any formal analysis of
the product’s performance.

In this section, we make some general observations about the evidence that is
likely to be provided, and the major sources of existing evidence for COTS
products. Note that we defer discussion of new evidence to Section D.3.2.

**Certification Packs**

The first major source of evidence for a COTS component is the certification pack
[18]. Certification packs may be provided to demonstrate compliance to a
particular standard, such as DO-178B or IEC 61508. These packs typically
include such things as design documentation, evidence of external assessments,
testing criteria and operational limits, software development plans, and
information about the verification environment. If available, these packs can be
very useful. However, the primary disadvantages of these packs are possible
doubt over the trustworthiness of their contents, and the probability that the
evidence may not be applicable to the component’s operation in the proposed
environment.

The trustworthiness of the evidence in a certification pack must also be
considered. If the assessor and certification system is unknown to the MOD, then
the rigor of the evidence which is required for that certification may be doubtful.
Known certification systems and authorities will be regarded with more trust, but
again certification to a known standard (e.g. DO-178B) does not automatically
mean the product is safe for the intended usage by the MOD. In general,
certification packs are likely to be trustworthy in practice – most COTS products
which are provided with certification packs and are being seriously considered by
the MOD will include sufficient evidence to certify to well-accepted standards –
but this trustworthiness should not be assumed in all cases. Similarly, a
certification pack which has been provided for certification to a specific standard
may not be useful unless that standard and its requirements are also assessed.
Furthermore, in terms of the applicability of the evidence provided, standards
such as DO-178B certify entire software systems rather than individual
components. Because of this, the evidence provided in the pack may relate to a
different usage and environment profile. Section D.3.1.3 discusses this in more
detail and provides guidance for conducting a gap analysis to address this
deficiency.

Certification packs also have the disadvantage that they will only contain
evidence for the aspects required – and to the rigour required – by the standard
in question. In the case of civil standards, this extent and rigour of evidence may
not suffice for military requirements. Even where military standards have been
used, the standard may not address all issues which will be relevant to the use of
the COTS product. Consequently, some military requirements – such as the
testability and supportability requirements of MIL-STD-2165 and MIL-STD-2084
which are discussed in the next section and which are not always satisfied by the
design of non-military software – for the software will most likely need to be lessened if a certification pack is to serve as the primary means of evidence [16].

Further Tests and Analyses

The second major category of evidence for a COTS component is evidence which is available from the manufacturer, which may not in all cases be in the form of a certification pack to a specified and widely-accepted standard (such as DO-178B or IEC 61508). The types of evidence available from the manufacturer vary immensely, and the extent of the evidence is often constrained only by cost (for a sufficient fee, it is usually possible to buy the entire rights to the product, including the source code!) In practice, most COTS suppliers will provide user manuals where relevant, and data sheets establishing recommended operational limits. Mean times before failure are sometimes also included, as well as testing or analysis results.

COTS vendors may also include the results from testing and analysis performed by third parties, which we cover in more detail in the next section. This evidence may in some cases be considered more trustworthy, as a third party has no vested interest in the safety or performance achievements of the COTS component. Section D.3.1.4 discusses this in more detail, noting that the degree of independence of the vendor and third party is important when considering the effect of third-party evidence. In practice, if any testing evidence is supplied, it is typically in the form of information about test coverage – for example, guaranteeing path coverage, or modified decision coverage. Testing results which demonstrate data integrity may also be supplied. However, unless required as part of a certification pack (or obtained for an additional fee), COTS vendors will typically not supply process evidence. It is also unusual for any analysis evidence, such as a formal model of the system and the subsequent analysis, to be provided.

As well as establishing broad requirements for the use of the component, this type of evidence also provides information about the environment in which the component is expected to operate. That is, the operational limits and established mean times before failure apply in a particular environment, which will usually be specified. While there may be doubts as to the degree of confidence which can be placed in these results (see Section D.3.1.3 and Section D.4) for more detail, the information provides a starting point to identify where further evidence may need to be generated.

The primary problem with this evidence is that COTS suppliers may not provide sufficient evidence for the satisfaction of military safety requirements [30]. This is especially relevant where the COTS component was not designed with the intent of being used in safety-related systems (various commercial operating systems are an example of this). Section D.3.2.1 discusses the effects of this in more detail, but here we briefly cover the causes. The first of these is that COTS components might not specify “that military test capabilities be incorporated into
designs” [30]. To take a general example of how military requirements are not always matched by civil requirements, as mentioned earlier MIL-STD-2165 (Testability Program for Electronic Systems and Equipments) and MIL-STD-2084 (General Requirements for Maintainability of Avionic and Electronic Systems and Equipment) specify testability and supportability requirements relating to fault detection and isolation. In a case where these standards were applicable, a COTS product may not have been designed to support these testability requirements. Consequently, evidence showing satisfaction of the standards may be more difficult to acquire. Secondly, the thoroughness of the tests is often unknown – for example, military safety-critical may require extensive fault testing whereas a commercial application which is not designed for safety-related work may have undergone only normal range testing. The test methods of the COTS supplier may also not be known, or not be considered to be rigorous enough for military safety-critical usage. One example of this is the types of faults detected – the vendor may only have evidence showing the detection of certain faults, such as omission failures.

This type of evidence is essential, however, in that it is the only source of information about proprietary extensions to software that is otherwise developed to an accepted standard, or open source. Obviously this information will not be needed in a certification pack for the standard in question, and so is unlikely to be included under the first category discussed above. The utility of this information is that it enables a purchaser to identify which parts of the system are “trusted” according to the criteria of a particular standard. Information about proprietary extensions to software also serves to highlight problems with the robustness of the safety case, in that the systems making use of the COTS product may now be locked into use of a product line which could potentially be discontinued [16].

**Third-party Evidence**

The third major source of evidence comes from information about the safety culture of the vendor, and user testimonials from third parties. This may take the form of certification (e.g. CMMI 5) of the vendor, or even of public discussion (e.g. internet forums) of their products. It should be noted, however, that the certification (e.g. CMMI) by itself will not necessarily prove that the product is sufficiently safe! Instead, the effect of the certification must be assessed by ascertaining precisely what it denotes – in some cases, for example, that certain procedures have been followed. The contribution of these procedures to the safety or integrity of the product will then need to be assessed in order to determine exactly what the certification denotes. Similarly, when making use of user testimonials, it is necessary to assess the value of these by contrasting the proposed environment and usage of the product with the environment and usage which were relevant to this testimonial.

Third-party evidence is unlikely to provide sufficient confidence in the component by itself, but it can be used to justify a claim that the vendor typically works in accordance with the principles of good practice. One important consideration
when judging the effect of third-party evidence is the degree of independence of the third party and the vendor. If there is little independence, then third-party evidence may not contribute significantly more to the confidence with which the COTS product is regarded than additional vendor-supplied evidence would. Similarly, the lack of any independent testimonials as to the quality of the product or the track record of the supplier may highlight a potential problem with the trustworthiness of the vendor-supplied evidence.

Third-party evidence may also include the results of tests and analyses performed upon a COTS product. These can be more trustworthy than the results of similar analyses carried out by the COTS vendor. However, they suffer from a similar disadvantage in that the criteria against which the COTS product has been tested, and the evidence which has been produced, may be insufficiently stringent for military safety-related (or safety-critical) use [30]. Because COTS products evolve over time, it is also necessary to demonstrate configuration consistency if these results are to support a safety argument. That is, there should be some evidence provided to demonstrate that they were obtained by examining the version of the COTS component which is being considered by the procurer.

The adequacy of this evidence is also clearly dependent upon the nature of the third party providing this evidence. Reviews from accredited certification authorities are more likely to be trustworthy than anonymous internet forums. In other words, many of the aspects which affect trustworthiness of a COTS vendor (such as safety culture, competence and others discussed in Section D.3.1.4.) also affect the trustworthiness of third-party evidence.

**Pre-Existing Safety Cases**

A safety case and safety argument may already have been provided for a particular COTS product, most notably if it was developed in accordance with DS 00-56 Issue 3 / 4. In this case, DS 00-56 recommends that “due cognisance” should be taken of the pre-existing safety case to reduce costs. However, because the safety of any software component is dependent upon the context in which that component will be used, the relevance of a pre-existing safety argument may be minimal.

In order to ascertain the extent to which a pre-existing argument can be used to demonstrate safety of a COTS component, some form of gap analysis is necessary. This gap analysis should identify the differences between the proposed COTS context and the context of the pre-existing safety case. These differences may invalidate some evidence that was used to support the pre-existing argument, meaning that this evidence cannot be used to support a new safety argument. Consequently, the pre-existing safety argument can only be used in the presence of a second argument designed to support the claim that the proposed context is “sufficiently similar” to the context used for the original safety argument.
Some COTS vendors do consider the necessity of justifying contextual assumptions, most notably when explicitly producing modular safety cases [22]. In these cases pre-existing safety cases may provide significant support. Additionally, the Accomplishment Summaries which are recommended by DO-178B can sometimes provide significant support to safety arguments. Section D.3.1.3 contains further discussion, including the provision of guidance questions to aid in conducting such a gap analysis.

D.3.1.2 In-service Evidence

There is one major form of evidence available which may be available for COTS software but not for bespoke products: in-service evidence. In-service (or historical) evidence is evidence which is generated from the use of the product. Clearly, this evidence is not available if the product has not been used before. To expand, this evidence is not obtained by explicit testing or analyses performed during development. Instead, it is obtained by actual use of the product in other systems. Examples of in-service evidence include fault logs, configuration management reports, modifications and subsequent regression analysis, maintenance reports and so on. There are both advantages and disadvantages to in-service evidence as we discuss below.

Advantages of In-service Evidence

The primary advantage of in-service evidence is that, as an evidence type, it is unlikely to be vulnerable to inaccurate assumptions or simplifications in the same way that evidence types obtained from examining a system model are vulnerable. These assumptions could be detrimental to safety where, for example, a formal model used to generate evidence about the safety of a system does not take into account all parameters which affect the system. Because in-service evidence is generated from an environment in which all these parameters are encountered, over-simplification is unlikely to be a problem. This does not imply that any particular item of in-service evidence is invulnerable to all assumptions (for example, in the next section we discuss the effects of assuming that data collection quality is higher than it is), simply it is not vulnerable to the same “simplifying” assumptions which can be erroneously made when constructing a model.

Secondly, in-service evidence can provide information about an entire system. There is the risk, in a modular safety case, that the interactions between components have not been fully assessed (that is, evidence has been provided as to the safety of each component, but owing to limited visibility by subcontractors, the true extent of component interactions and consequences of these has not been understood). In-service evidence is obtained from the entire system, and therefore by default takes into account these interactions in the context in which the software was operated. Because of this, in-service evidence can often act as counter-evidence, demonstrating that safety claims cannot be demonstrated in practice. It is important, however, to note that in-service
evidence will only provide information about the interactions of components in the environment in which the evidence was generated. That is, when a component is placed in a different environment, it may undergo different interactions to those indicated by in-service evidence.

Finally, in-service evidence can be used to satisfy strict quantitative safety requirements, where testing would not be sufficient to demonstrate these. This is because taking in-service evidence from a large number of sources can approximate a statistical distribution which may be used to model failure rates. Similarly, in-service evidence can provide information about failures of the component over a longer operational time than testing itself would permit. However, it should be emphasised that the utility of in-service evidence in demonstrating quantitative safety requirements will vary with the quality of data collection. That is, if failures have occurred but have not been reported, the resultant in-service evidence may show that a more strict quantitative safety requirement has been met than is strictly the case.

**Disadvantages of In-service Evidence**

The primary disadvantage of in-service evidence is that it applies only to the environment and context in which it was generated. This has been referred to as “relevance of usage environment” [18]. Of course, this is a factor for all evidence, but analysis and testing techniques will often produce results which can be extrapolated to other environments. By contrast, it is often difficult to extrapolate any conclusions from service history because in-service evidence may not contain sufficient detail about the causes of faults. In this situation, it is difficult to determine whether the component is likely to exhibit the same fault in the target environment. Furthermore, determining whether the usage environment is sufficiently similar to the target environment can be a complex and time-consuming process. Section D.3.1.3 provides guidance for conducting a gap analysis to determine whether the environments are “similar enough”, but in some cases this gap analysis cannot be justified. If this is the case, then the in-service evidence cannot be used, as its relevance cannot be sufficiently demonstrated.

Secondly, it is necessary to determine the effectiveness of problem reporting. It may be the case that not all faults have been recorded in fault logs, leading to the question of whether absence of evidence of a fault implies absence of that fault. Similarly, incidents (failures which do not lead to accidents) may not be reported, leading to misplaced confidence in the integrity of components. Section D.3.1.3 discusses the problems inherent in using existing evidence where component failures have been detected and masked. Another important consideration here is whether all the relevant in-service evidence is available. Inadequate provision of evidence (especially where this inadequacy is not made clear) may give the erroneous impression that the software has undergone fewer failures in operation than is the case.
Finally, we must ascertain whether the in-service evidence is obtained from the correct version and configuration of the component in question. This consideration is related to the consideration of relevance of usage environment, but for reasons of thoroughness should be considered explicitly. A supplementary consideration is also the quantity and extent of in-service evidence. A COTS product which has not been deployed clearly will have no in-service history. Similarly, where a limited number of instances of a product have been deployed, the in-service history will naturally be limited.

D.3.1.3 Applicability of Existing Evidence

One major issue with all evidence which is made available is the possibility that it does not pertain to the software as it is to be used. The safety of software is dependent upon the environment in which the software is used, meaning that the available evidence can only be of use to a safety case if the usage and circumstances were “similar enough”. It is important to understand the distinction between the two related questions:

- Is the usage and circumstances in which the existing evidence was generated similar enough to the proposed usage to allow this existing evidence to be used in a safety case?
- Is the evidence provided enough to assure the system?

Although these questions appear to have a superficial resemblance, they are addressing very different aspects of COTS safety assurance. The first asks whether the evidence provided can be of use to a safety case, and if so, then in what capacity. The second asks whether the evidence provided is enough by itself to support a safety case. Obviously, the answer to the second will depend upon the answer to the first – if the evidence provided is only of limited use or applicability, then it is unlikely to be able to support a safety case on its own. In this section we will address only the first question.

Identifying precisely what defines usage which is “similar enough” is a complex question. To aid in answering this, we have provided some guidance for conducting gap analysis. The intent of this guidance is to identify the reasons why existing evidence may not be applicable (and thus should not be used to support a safety case for the target system). These “gaps” may then be filled by providing newly-generated evidence, either obtained from examination of the COTS component or obtained from examination of the protection afforded by the proposed environment.

Categorising Potential Evidence Mismatches

Evidence mismatches (reasons why the existing evidence is unlikely to provide sufficient support to a software safety case) can be judged to fall into one of six major categories, which we will detail below. Each of these corresponds to a way in which the existing evidence may not be relevant. Evaluating the existing evidence for a COTS component against these categories will therefore identify
the situations in which this evidence does not apply. Each category contains a number of supplementary “guide” questions designed to prompt further discussion and analysis of the COTS component. The lists of questions are not intended to be exhaustive, but simply to serve as a starting point for examination of the software. Any claim about the sufficiency of existing evidence must therefore be justified by an argument over the lack of applicability of the following categories.

It is important to be aware that in many cases there is simply not enough information to conduct a thorough gap analysis. That is, while evidence about the safety of the COTS component may have been provided, it is common that no justification is forthcoming as to why this evidence is applicable. A common example is the provision of evidence without any proof that this applies to the version of the COTS software which is being considered for purchase (the “Adequacy” category below). In this case, we emphasise that the evidence must be considered inapplicable. That is, the use of existing evidence must be justified by an appeal to gap analysis findings which show that there is no significant usage mismatch. Where the extent of usage mismatch is unknown, it must be considered to be sufficient to render the evidence inapplicable.

The categories are in an approximate order of importance, as identified by respondents to evaluation survey (Annex D.C). They are not completely orthogonal, but the survey has indicated that the distinctions are sufficiently clear to require all the categories as separate considerations.

Failure Consequences and Integration Assumptions

The failure consequences and integration assumptions refer to the likely effect on the system if the software fails, as well as any assumptions which have been made about the manner of failing or the ability of the component to integrate into a system. In order for the existing evidence to be useful in a safety case, these assumptions must be similar to valid assumptions which can be made about the proposed environment. This category has been deemed the most important when providing guidance, due to the difficulty in integrating safety cases for multiple components (e.g. a COTS module within a bespoke application).

The following guide questions are applicable when considering the potential for this type of evidence mismatch:

- If the COTS software does fail, is it required to fail in the same way (e.g. gracefully, silently, visibly signalled to an operator) in the proposed environment as the environment in which the existing evidence was generated?
- Are components in the rest of the system required to fail in the same way in the proposed environment as the environment in which the existing evidence was generated? If not, is there also additional evidence to show that the performance of this software does not rely on any assumptions made about the way in which failures occur in the rest of the system?
• Are there any human or physical factors which affect the original evidence but may not be applicable to the target environment? For example, the original evidence may demonstrate that all failures are acceptably mitigated by showing that some types of failure are signalled to an operator who is qualified to correct these. If the operator in the target environment cannot be shown to necessarily correct the failures in time, this evidence cannot be used to support a safety case that the software is acceptably safe.

• If the software does fail, is it intended to continue operation after this failure in the same way in the target environment as it was in the proposed environment? For example, the target environment may continue to execute functionality provided by the software after a particular failure, whereas the original environment may have expected the software to cease operating. Consequently, existing evidence will only be able to describe the effects of the first failure of the software but not of any subsequent failures.

• Have any assumptions been made about the consequences of a particular failure of this software, based on limited visibility? Subsystems implemented by contractors often have only limited visibility of the consequences of failure. Consequently, when generating evidence, they may make assumptions about the effect on the rest of the system. Such assumptions must be justified if the existing evidence is to be used.

• Did the original environment explicitly provide any functionality to the software which will not be provided by the target environment? For example, were external software libraries and interfaces provided? If so, is there any justification to show that that the omission of these libraries cannot affect the safe functioning of the software?

• Is there any evidence about the integration of this component into other existing systems? This evidence may emphasise the possible effects upon other components, leading to further targeted investigation within the “Resources Availability and Interference” category.

Adequacy

The next category of usage mismatch refers to the adequacy of the evidence provided. Evidence may be inadequate either through a lack of trustworthiness, or simply because it has been restricted in ways which are inappropriate considering the proposed usage. The majority of this section (Section D.3) can be thought of as a discussion of the adequacy of evidence, as this question is at the heart of assessing COTS software.

Relevant guide questions when determining the adequacy of existing evidence include:

• How comprehensive is the evidence? For example, when considering in-service evidence, does lack of evidence of a failure imply the logical absence of that failure mode within the software? Or does it simply mean that the conditions leading to the fault were not experienced or not reported?
Are there any failure modes of the software for which evidence (detailing the absence or acceptable handling of the failure) has not been provided?

What operational modes were used to generate this evidence, and are they similar to the proposed modes of use?

Does the evidence presented reflect the amount of time the software will spend in each operational mode, as well as reflecting the criticality of each?

Is there any compelling reason to believe that this evidence relates to the software version which is being considered for purchase? That is, in addition to the evidence about safety, is there also evidence about configuration consistency? This issue of configuration consistency may be satisfied by evidence that any updates which have taken place since the original evidence was generated have not affected the characteristics of the software which were demonstrated by that original evidence.

If the evidence in question originates from a certification pack, what is the justification for trusting that this pack contains comprehensive failure mode information (e.g. has it been accredited by a trusted third party)?

Have any unjustified assumptions been made when generating the evidence (for example, simplifications when producing a formal model of the system)?

Is there “enough” evidence? That is, is the extent and quality of the evidence commensurate with evidence provided in existing safety cases for similar software, where these safety cases have been judged representative of good practice by the MOD? For example, systems where the functionality resides entirely within the software may require a more sizeable or rigorous software safety case than those where the functionality is shared between software and hardware.

Does the evidence accurately correspond to the safety claims which you wish to make? This is especially relevant where evidence is sourced from existing safety cases which rely on different safety claims to those which are being made for the proposed system. Section D.4.2 describes how assuring the “wrong requirement” can lead to a loss of confidence in the system.

Does the evidence align with the questions from all categories? For example, if there is evidence showing that the software in question can interfere with other components, there should also be evidence showing the consequences of this interference. This is explored further in the “Resources Availability and Interference” category.

Degree of Protection

The degree of protection refers to the likelihood that a failure of the software will progress to a system hazard. If the environment from which the existing evidence was gathered provides a higher degree of protection than the proposed environment, then this existing evidence will show fewer system hazards which have been caused by this software. This may then lead to the false belief that the software actually undergoes fewer failures, rather than the correct belief that the failures are simply being detected and handled by the environment before they
can progress to system hazards. This particular usage mismatch is most useful when assessing in-service evidence, or evidence available from actual use of the product. Existing evidence obtained from testing and formal analysis of the software in isolation is unlikely to suffer from any usage mismatch relating to the degree of protection.

The following guide questions will help determine whether differing degrees of protection between the environments where the evidence was produced and the target environment mean there is likely to be a usage mismatch.

- Is the evidence generated from an environment where the architecture may have masked faults in the product? Some particular examples include:
  
  a) If multiple-version programming was used, where is the evidence to show that it is this software which produced the results shown in the evidence?
  
  b) Was a wrapper component used to filter out software faults? If so, does the evidence show those faults?
  
  c) Was a watchdog timer used to compensate for timing errors in this software? If so, where is the evidence to show which results are generated from this software, and which from the software called by the watchdog timer?
  
  d) Was there any timestamp logic in the environment which would compensate for timing errors? If so, is this same logic going to be present in the proposed environment?

- Is the evidence generated from an environment where the hardware may have masked failures of the software?

- Were there any human factors (such as cross-monitoring) in place in the original environment to provide extra mitigation against failure, where these will not be present in the proposed environment?

- Were there any other system-level mitigations against failure?

- Is there compelling evidence to accurately determine the precise effect of each mitigating factor? In more detail, is it known under precisely what circumstances the original environment provided each protection factor? Mitigations themselves can also fail, so it is useful to know the integrity of the architectural mitigation solutions which provided this evidence.

There are many other types of software architecture which can be implemented to provide fault tolerance. If it is unclear whether the evidence has been generated with the fault tolerance mechanisms in place, then this evidence can only be used to support a safety case if equivalent fault tolerance mechanisms will be present in the proposed environment.

**Inputs and Data Flow**

In order for the existing evidence to be of use to the proposed safety case, the data flows and inputs must be equivalent, or compatible in some way. If this is not
the case, one result may be an increase in commission or value failures affecting this software. That is, these failures may be caused due to functionality being invoked by inputs which were not present in the environment in which evidence was generated. Relevant guide questions to ask in order to determine whether the existing evidence is vulnerable to this type of evidence gap include:

- Is there any compelling justification to show that all possible sources of input have been represented within the evidence?
- Is the range of acceptable inputs a subset of the original?
- If the software receives an unacceptable input, will it be required to respond in the same way in the proposed environment as in the original?
- In the environment where the evidence was generated, are the inputs and interfaces to the product the same as in the proposed environment? (This will often be the case for software which offers the ability to write “plug-ins”, or re-use interfaces)
- Are there any assumptions which were made when producing the existing evidence about the types or nature of possible input (e.g. units of measurement, positive vs. negative integers, inputs considered as abnormal)?
- Did the previous environment protect the software against inputs that were out of range? If so, will this protection be present in the proposed environment?

Resource Availability and Interference

If evidence is generated from an environment where there is no possibility of interference from other software, then this evidence is of limited use when attempting to show the safety of the software in a system where such interference is a possibility. Equally, if the evidence is generated in an environment where there is no difficulty over competition for resources, then the software may not be as reliable as the evidence indicates (for example, it may be vulnerable to timing failures).

This category of usage mismatch can also serve to identify possible inadequacies in the evidence which have not been identified as part of the “Adequacy” category. This is achieved by comparing the evidence of interference between components with the evidence showing the degree of protection. Furthermore, evidence of interference should then lead to the provision of further evidence about consequences of failure. If this is not the case, the evidence is inadequate in extent, which should be identified by the investigation into the “Adequacy” category of usage mismatch.

Some considerations for this category of usage mismatch are as follows.

- Will the software have to compete for resources in the same way (or in a less intensive manner)) as it did in the environment where evidence was generated? If not, is this difference likely to lead to more or fewer timing failures?
Will the proposed environment be partitioned in the same way (or such that it provides a greater degree of isolation) as the environment where evidence was generated? There are three subsidiary questions to this:
  o Will the software be competing in a different manner for resources and therefore liable to starvation or input failures?
  o Will the software be vulnerable to interference by other processes in ways in which it was previously not?
  o Will the software potentially interfere with other components in ways which the original environment did not consider?

Will the software be interacting with other components and functionality in ways that it did not in the environment used to generate existing evidence? That is, can the COTS product cause failures in other components, due to unforeseen interactions?

Does the evidence about interference align with the evidence provided in the previous categories “Degree of Protection” and “Failure Consequences and Integration Assumptions”? For example, is there evidence showing an apparent lack of interference which cannot be explained by mitigating factors in the “Degree of Protection” category? Additionally, is there evidence of interference which has not then been considered in the “Failure Assumptions and Consequences” category?

Standards and Applicability

This type of usage mismatch comes about when the techniques used to generate the existing evidence are no longer in use, or when the types of evidence presented do not cover the full scope of the proposed usage. In addition, the following guide questions determine whether adequate configuration control was exercised, an important consideration for COTS products.

- If the evidence includes a claim of good practice during the development, does this satisfy the MOD as reflecting good practice as applied to current standards?
- What was the configuration of the system (and particularly, this software) used to generate evidence, compared to the proposed configuration?
- Will the software be running on different hardware? If so, what effect will those differences have?
- If there is an existing safety case which makes use of ALARP principles, do these principles still apply, given new technology and circumstances? (If not, the evidence may still be used, but not the ALARP argument). It should be noted that not all component safety cases make use of the ALARP principle, as this can be considered a system design issue
- Has the evidence been generated using any tools or techniques which have since been identified as flawed? If so, is there justification to show that these flaws would not have affected the evidence?
• Was the existing evidence generated in an environment in which the relevant tools were qualified to be used in a safety-related role? If so, are these same tools qualified to be used in a safety-related role in the new environment?
• Are the compilers, linkers, libraries etc. used in the original environment known to be the same as in the target environment? If not, where is the justification that the functionality of the software is not affected by these changes?

To summarise, the above guide questions identify possible reasons why evidence provided (in-service evidence, evidence from an existing safety case, or evidence provided by the COTS vendor or supplied from a third party employed by that vendor) may not be suitable to support a safety argument for use of the COTS product by the MOD. These guide questions in each category should be interpreted as a form of gap analysis, to be undertaken during COTS assessment as described in Section D.5. It is important to note that it is unlikely that “pleasing” answers will be obtained to all these questions, as this would indicate that the existing evidence was obtained from almost identical usage—a situation which rarely arises when considering COTS products.

We also emphasise that the actual evidence required to support a safety case must be commensurate with the integrity requirements. That is, the extent and quality of evidence required to assure each safety claim must be assessed individually. It is extremely unlikely that pre-existing evidence will be sufficient in quantity, even if the gap analysis identified few usage mismatches. Additionally, the importance of the different categories must be assessed for each COTS component and each item of pre-existing evidence.

The results of the gap analysis will help determine how much of the evidence provided can be used in a safety case. However, to complete the gap analysis (and determine the extent of the assurance deficit), it is necessary to consider how much evidence is needed to adequately assure the safety of the software. Section D.4.1 discusses this in more detail.

D.3.1.4 Trustworthiness of Existing Evidence

Evidence supplied for COTS products may not always be trustworthy. There are two major reasons for this. The first reason is due to the possibility that representative evidence has not been supplied. One reason that this may happen is that a COTS vendor cannot analyse the component in every possible application and environment. Consequently, the evidence supplied may not be representative for the particular context in which the acquirer intends to use the component. This may give the impression that the COTS product displays safety properties which, in fact, cannot be relied upon.

The second reason is due to inadequate quality processes when producing the evidence. In some cases this can be due to the application of standards and
techniques which have been subsequently found to be flawed (see Section D.3.1.3). In other cases it may be due to sources whose accuracy cannot be verified – for example, user tests on open-source software modification. It is important to note that untrustworthy evidence may not in fact contain errors. However, because the possibility is there, untrustworthy existing evidence is unlikely to provide sufficient sole support to a software safety case.

The reasons why existing evidence may be deemed untrustworthy include the following: lack of evidence of a safety culture, lack of independent assessment of processes or tools used, lack of evidence of external assessment, lack of ability to duplicate or validate existing evidence, use of unproven tools and known substantial counter-evidence to the claims. In addition to these, there are several situations which in practice act as indicators that the evidence provided is untrustworthy.

The first of these is simply whether this vendor has developed similar systems in the past and is a trusted collaborator. An established COTS vendor has made a significant investment in their reputation and is hence unlikely to compromise their standing with the customer base by providing faulty evidence. However, unknown or new COTS vendors may not place such importance on maintaining a high reputation, and hence the trustworthiness of their evidence cannot be assumed. The second situation which may indicate untrustworthy evidence is an unwillingness to engage in a discussion or provide increased – but nonproprietary – evidence for a fee. For example, if a COTS vendor will not provide information about the language used to code their product, or will not provide any documentation about their development processes, then the evidence that is supplied is potentially untrustworthy.

Additionally, a claim which is not credible indicates that the evidence is fundamentally flawed. For example, a claim that no errors were found in branch testing of a large (thousands of lines of code) product is not credible, except where supported with evidence that this absence is due to errors having been previously found and corrected. Without this supplementary evidence, this claim may indicate that the branch testing has not been carried out correctly.

Some uncertainties which reduce the trustworthiness of evidence can be easily resolved (for example, identifying the extent of external assessment by querying the assessors). However, others are more difficult to quantify without significant work. For example, it may be impossible to verify the existing evidence obtained by the COTS vendor from static analysis, without reverse-engineering the product – and the lack of replicability gives rise to a potential lack of trust in the static analysis. However, the most common and easily performed method of increasing the trust placed in the available evidence is by validating it. This involves performing the tests, analyses or assessments that were claimed to be performed successfully by the COTS vendor. Obtaining the same – or equivalent – results will increase the trustworthiness of the vendor-supplied evidence. It is usually not required to validate all the evidence, simply a representative sample which should
indicate if the safety properties are demonstrably false in the proposed environment.

If none of the existing evidence is able to be validated, then it is unlikely to support a safety case. This is because the trustworthiness of the evidence cannot be directly estimated (although the trustworthiness of the vendor can be judged to a certain extent based on the factors mentioned above). In Section D.3.2 we discuss the extent and nature of the types of validation which can be performed. However, it is not felt useful to attempt to give generalised guidance for quantifying the effect of a lack of trust in evidence. That is, trust can be diminished by a number of factors, from vendor familiarity to inability to replicate the supplied evidence. The extent to which untrustworthy evidence creates an evidence “gap” or deficit will have to be judged in each system individually. Nevertheless, a gap analysis which does not consider the trustworthiness of evidence presented is potentially fundamentally flawed.

D.3.1.5 Counter-evidence

Counter-evidence is also a significant factor when determining the validity of existing evidence. Counter-evidence weakens claims by providing evidence to show that in a particular case – or under particular circumstances – the claim does not hold. DS 00-56 mandates a search for counter-evidence which, if found, must be justified. That is, the provision of counter-evidence will weaken the assurance of the claim. To negate this, justification must be provided to state why the counter-evidence is inapplicable.

Counter-evidence is typically not thought to affect the trustworthiness of the remaining evidence items [13] – if anything, it may increase the trustworthiness. This is because for many claims it is not credible that they hold in all situations, or that there will not be a single example of failed testing, as would be implied by a complete absence of counter-evidence. However, depending on the nature of the counter-evidence, the trustworthiness of the remaining evidence may in fact be affected. One situation where this may hold is when considering COTS components. This is the situation where a claim has been made by the COTS vendor, and evidence provided to support this (but no counter-evidence has been supplied by the vendor). Upon performing further tests, the purchaser identifies items of counter-evidence under circumstances which it can be estimated should have been addressed by the items of evidence from the vendor. In this case, the trustworthiness of all items of evidence provided by the vendor is reduced. Section D.3.1.4 discusses further reasons for this further.

Because of the potential impact of this situation, it is the recommendation of this guidance that counter-evidence be searched for with respect to all appropriate vendor claims. This can be done during the process of verifying these claims (by generating further evidence). The precise effect on trustworthiness of any counter-evidence found during this process will depend on several things. These include a consideration of whether it is reasonable to expect the vendor to have identified
the counter-evidence, the importance of this counter-evidence to the operational scenario, and the trustworthiness of the counter-evidence itself. Just as for evidence, counter-evidence has the greatest effect when multiple diverse items are presented. Multiple pieces of counter-evidence which do not display independence will have a lesser negative effect on the trustworthiness and assurance of the claim.

D.3.2 Generating New Evidence

In this section, we discuss the need to generate new evidence for a safety case addressing use of a COTS component. One important note when constructing a safety case for a system containing COTS components is that there should be no undue expectation that the COTS component does, in fact, satisfy safety requirements. That is, evidence that the component satisfies safety requirements may be lacking simply because the component does not actually satisfy the requirements! If this situation becomes apparent, then mitigation strategies must be used to lower the criticality of the COTS component.

As discussed in Section D.3.1.1, even when evidence is provided by COTS vendors, it may not adequately address military safety-related or safety-critical requirements [30]. Similarly, in-service evidence can also be inadequate for the proposed environment, as discussed in Section D.3.1.3, or untrustworthy as discussed in Section D.3.1.4. Section D.3.1 constitutes guidance for conducting a gap analysis, by identifying where existing evidence is likely to be inadequate. This must be read in conjunction with Section D.4, which discusses the concept of “enough evidence”. These two sections enable the purchaser to determine how much evidence ought to be presented, and how much has been presented as existing evidence. These gaps must then be addressed by generating new evidence to adequately satisfy the safety requirements in question. Standard examination of the product can be performed, including testing, analysis – if sufficient code is provided – examination of design documents and fault injection at component boundaries. If sufficient assurance cannot be obtained from direct assessment of the product, mitigation strategies can be used to decrease the assurance requirement on the COTS component.

However, new evidence is not only required to “plug the gaps” in the safety case, but to verify the evidence which has been provided. It has often been stated that – no matter what written guarantees or assurances are obtained from the COTS vendor – it is still necessary to evaluate the COTS component before selection [9]. That is, the component must be tested to determine whether the supplied evidence is trustworthy. Section D.3.2.4 discusses this in more detail.

The three most commonly used techniques for obtaining new evidence to shown the safety of COTS components are: black-box testing, fault injection testing and operational testing [36]. These are discussed in more detail in the following sections.
D.3.2.1 Fault-injection Testing: Generating New Environmental Evidence

Fault-injection testing [36] examines the effects of component failure on the system. If necessary, architectural mitigation strategies (such as the use of a wrapper component) may be implemented to ensure that failure of the COTS component does not contribute unacceptably to any system hazards. Thus, fault-injection testing does not assess the quality of the COTS component, but rather assesses the ability of the environment to acceptably handle possible failures of the component.

There are two reasons why fault-injection testing may be used to support a safety case addressing integration of a COTS component. Firstly, it may not be possible to gain enough evidence about the COTS component to assure the relevant safety requirements to the degree which is needed. That is, lacking sufficient evidence, we cannot assume that the COTS component will not fail unacceptably. Secondly, it may actually be the case that the component does not satisfy the safety requirements. In both these cases, fault-injection testing will demonstrate that failure of a COTS component can be acceptably detected and handled by the environment.

The use of fault-injection testing has been described in many systems and may lead to the use of architectural mitigation strategies. That is, in order to negate the effects of failure of a COTS component (as demonstrated via fault-injection testing), it may be necessary to implement additional functionality or components to monitor COTS components, partition them from the rest of the system, handle their failures as expected, or provide suitable responses where the component has failed.

Some common architectural mitigation strategies include providing a wrapper to acceptably detect and handle all failures of the component, or alternatively providing n-version programming by using a backup bespoke component to perform the same functions as the COTS component. In these cases, the software providing architectural mitigation is bespoke. Thus, it is typically easier to generate rigorous and compelling evidence for it than for the COTS component, and the assurance requirements on the COTS component can be correspondingly relaxed. After implementing these mitigation strategies, it will be necessary to perform fault-injection testing again to verify that failures of the COTS components are now acceptably detected and handled.

It should be noted that if architectural mitigation strategies are used, then the COTS component is being assessed in the context of the rest of the system. That is, the safety case which is produced cannot be used for the COTS component in isolation.
D.3.2.2 Black-box Testing: Generating New Testing Evidence

As mentioned earlier, generation of new evidence is typically gained from testing. Testing can typically only be performed at the boundary of the component, and its use is therefore limited to those properties which are visible there. Assurance of this form may be sufficient only for those systems with low safety criticality [6] where requirements can be expressed solely in this form. If the safety requirements make it necessary to provide evidence about the internal architecture of the COTS component – and this information is not available from the COTS manufacturer – then a convincing safety case cannot be produced.

Tests which are relevant include timing tests, storage of signatures for checksum tests, normal-range testing, boundary value testing and failure testing. It is not in the scope of this document to provide a comprehensive discussion of all testing techniques, but typically all black box techniques can be used on a COTS product. Black-box testing has been used successfully to identify faults in Unix releases [25], but is heavily reliant on the provision of an extensive testing profile and on the accurate identification of incorrect behaviour.

Black-box testing cannot determine whether a component sufficiently implements partitioning, or isolation, requirements [1]. That is, as for fault-injection testing, a COTS component must be assessed in the context of the rest of the system. This is the only way in which this type of testing can determine possible interference between a COTS component and the rest of the system.

D.3.2.3 Operational Testing

Operational testing assesses the performance of the entire system in circumstances which are as similar as possible to the proposed usage profile. It is therefore clear that operational testing can only be performed with a COTS candidate which has already been integrated into the system. Operational testing is perhaps the most expensive type of testing, and requires a significant amount of time and resources to produce worthwhile results.

However, as discussed below in Section D.3.2.5, operational testing has the advantage of showing the effects of component failure on the system. That is, operational testing will provide compelling evidence of the level of tolerance of the system to failures of the COTS component [36]. Consequently, it is recommended for verification of vendor claims, for safety-critical systems and for systems which are not urgently required to be placed into full service (operational testing can take a long time). It is important to note, however, that operational testing may not be extensive and is dependent on the quality of fault reporting. Thus it is possible that absence of evidence of a fault does not imply absence of that fault.
D.3.2.4 Reverse-engineering Software

Reverse engineering is the process of examining software to determine its design and implementation properties. Reverse engineering can be particularly advantageous for demonstrating the safety of COTS products, as information about these properties may not be supplied by the COTS vendor.

The most common motivation for reverse engineering is the necessity of providing some form of analytical evidence to assure the safety of the system. Section D.4.1.1 describes how analytical evidence can be used to aid in assuring safety claims, and explains how this is one of the evidence types recommended by DS 00-56. Analytical evidence must be provided by the COTS vendor or generated from the source code, design or implementation by the purchaser. Because COTS vendors do not always supply either the results of their own analysis, or sufficient details about the design or implementation of the software, reverse engineering can be the only way to reliably obtain this type of evidence.

Another reason why reverse engineering may be performed is to perform white-box testing. Unlike black-box testing, the exact coverage of white-box testing is known. Consequently, where exhaustive testing of a COTS product is required (for example, structural coverage may be required to identify dead code), this can be achieved via reverse engineering.

Reverse engineering is typically performed upon COTS software which is stable and mature [33]. Part of the reason for this is the intention that the data obtained from reverse engineering will be used as evidence for later safety cases which make use of this software. If the software is in constant flux, each new upgrade will require another process of reverse engineering. In addition, not all mature COTS products are suitable for reverse engineering. Design recovery [5] techniques can be used to recover the static design of software, but this does not necessarily convey any information about the dynamic software properties (such as the ordering of different events). Consequently, a COTS product for which the safety requirements relied on correct ordering of execution sequences might not be a suitable candidate for reverse engineering.

Commercial services such as JumpReverse-178B exist to reverse engineer software to achieve compliance with standards such as DO-178B. This is possible because reverse engineering with reference to DO-178B refers solely to the practice of generating design information – such as software lifecycle data – instead of source code. However, DS 00-56 typically requires more detailed information – for example, about the structure and design of the product itself – to adequately assure its safety. As a result, reverse engineering for COTS software intended to be compliant with DS 00-56 is a more involved process, often resulting in the production of viable source code and information about dynamic properties [38].
Reverse engineering can also be used with legacy items, where partial code is available. This is because there is the potential for cooperation with the original developers, and consequently increased knowledge of the design process. Reverse engineering a product where this is the case is significantly less expensive, and results in more reliable information. However, it should be noted that access to the original documentation and developers is not always possible with legacy code.

D.3.2.5 Validating Existing Evidence

The sections above have discussed the use of a number of techniques to generate new evidence to “plug the gaps” where there is no existing evidence. However, new evidence may also be needed to verify that, where evidence does exist, it is sufficiently trustworthy to provide assurance of a safety argument. More generally, it has been noted [9,10] that verification of vendor claims is both necessary to gain confidence in a system, and difficult to achieve without access to the source code. The degree to which the existing evidence requires validation will vary depending upon the trustworthiness of the evidence, the credibility of the claim in question, and the quantity and quality of evidence supporting this claim.

The reason that credibility of the claim is a factor is because it is not unknown for safety properties to be stated where these cannot be sufficiently rigorously demonstrated by the supplied evidence. The presence of these claims which cannot be supported from the available evidence will typically lower the trustworthiness with which the COTS supplier is regarded. Similarly, a claim that a product complies with a particular standard or has achieved a particular “safety level” (such as a given DO-178B DAL) when insufficient evidence has been provided to demonstrate this may lower the trustworthiness with which the product is regarded.

Cryptographic techniques [10] allow verification of results without the need for the vendor to disclose the source code or other proprietary information, but impose a cost upon the vendor. Consequently, this technique has not yet been widely adopted. Other grey-box techniques [11] require a certain amount of disclosure, which again cannot be relied upon. These restrictions on disclosure mean that, although analytical evidence is most strongly recommended by DS 00-56, it can also be difficult to verify.

In practice [12], the technique used for verifying vendor claims is often operational testing, sometimes via a demonstration. However, it should be noted that this is likely to be at least in part because operational testing is likely to be able to be performed on a COTS product, whereas analysis may not be. As discussed earlier, evidence from operational testing can be limited and does not constitute analytical evidence as defined by DS 00-56 [26]. Consequently, this type of verification may need to be supported by other analyses of the product where possible. However, operational testing has the advantage of accounting for operational parameters – at least in the context in which it is used – and being
typically resilient to incorrect assumptions or simplifications. It is also advantageous in that if any discrepancy in vendor claims is found, this testing technique permits an immediate assessment of the effect of the discrepancy on the system. That is, it may be the case that although the satisfaction of the vendor claims cannot be shown, the safety requirements which can be verified as being satisfied are still sufficient for the system. In this case, although untrustworthy, the existing evidence can be used if the software safety requirements can be relaxed, or if they are already less rigorous than those claimed by the COTS vendor. The likelihood of this situation arising is not uncommon, given that the decision to use a COTS component is best taken where requirements are known to be flexible [12, 16].

In general, it is recommended that verification of vendor claims should embody the principle of “first fit” rather than “best fit” [7]. First fit is a selection strategy which considers whether the minimum requirements are met, while best fit considers the gain in achieving more than the minimum necessary. Because it is generally irrelevant whether a COTS product satisfies more rigorous safety requirements than is necessary, first fit is a better option for verification. This general principle is echoed in our recommendation of operational testing as the primary means of verification of vendor claims.

As mentioned before, DS 00-56 places the most emphasis on analytical evidence. Thus, we recommend that – where this evidence is provided and where verification of vendor claims relating to analysis is possible – these should be verified. However, in some cases sufficient access to code and documentation will not be possible. To remedy this, we also recommend that verification of vendor claims by operational testing should be performed where possible. This may be particularly important where this is the only form of verification which can be performed. The lack of any verification – whether this be analysis or testing – must be justified in the safety case by a claim establishing the trustworthiness of the supplied evidence (either from in-service evidence which reinforces the claim or from an appeal to the safety culture and track record of the vendor). The degree of verification which should be performed will vary with the criticality of the system, the degree of trust in the vendor, the vendor’s familiarity with the domain and the type of solution offered. The McDermid square [26] can be used in some cases, notably those where familiarity with the domain is an issue, to guide decisions on how much verification is necessary.

In cases where analytical verification cannot be performed (that is, there is not sufficient access to code and documentation), operational testing should be used to verify the evidence. However, there are situations where this operational testing must be supplemented with other verification – typically testing, as this is the type of verification which is most easily performed. This may occur when the claim lacks credibility (for example, a claim of a failure rate of $10^{-8}$), where the history of the vendor is unknown, where there is no evidence to support a claim (unlikely if a certification pack has been provided), or if doubts have been raised about the component by a trusted third party. In these cases, the verification may
also include black box testing and fault injection testing. In this case it is advised to use a similar test profile to that used by the vendor (if known) to avoid the possibility of inadvertently testing different properties. Other works have recommended the performance of black box testing in all cases [37]. However, this is not to verify the results supplied by the COTS vendor but rather to address perceived deficiencies in the testing profile and regime. These deficiencies should have been identified in the gap analysis described in Section D.3.1, and therefore can be used to drive the black box testing process.

D.4 Assurance Deficits and Providing “Enough Evidence”

In this section we discuss some of the difficulties with determining whether sufficient evidence has been provided to adequately assure the safety of a COTS component. In order for an assurance deficit to be justified, it must be within some tolerable region (at least as confident as some minimum allowable value). Once within the tolerable region, the confidence is to be increased as far as is reasonably practical. This means that any assurance deficit must be justified by showing that the cost of addressing this deficit is grossly disproportionate to the benefit gained.

DS 00-56 states that the quantity, quality and rigour of evidence supporting an argument must be commensurate with that argument [26], and also with safety integrity requirements. Extrapolating from this, the confidence required in a claim must be commensurate with the potential consequences of that claim not being true. When constructing safety arguments relating to the inclusion of COTS components, it is necessary to identify the HSFMs (hazardous software failure modes) of these components. This can be done using a number of established techniques, such as HAZOP, SHARD, FTA and so forth. Section D.5 discusses this in more detail.

The relevant safety claims for the COTS product will then be that these HSFMs either do not occur, or are acceptably detected and handled. The degree of confidence we require in each of these claims is determined by the effect of each of these failure modes occurring. This can be determined by a criticality analysis [2] which takes into account such factors as the severity of the consequences of each failure, and the degree of architectural mitigation. As discussed later in this section and in the deliverables for Task 6, we do not assign a SIL, or criticality level, to components and functions. Rather, we identify the degree of confidence which is needed in each claim by assessing the potential impact of not providing that confidence.

Confidence in a claim may be increased by increasing the quantity and rigour of the evidence supporting that claim. This evidence may be either pre-existing (Section D.3.1) or newly-generated (Section D.3.2). Because COTS components often differ from bespoke components in terms of available evidence, safety arguments for the two can look quite different. Firstly, it may be more expensive (in terms of cost and resources) to obtain evidence for a COTS component than
for a bespoke component for comparable systems and requirements. Consequently, a greater assurance deficit may be judged acceptable for a COTS component due to the cost of obtaining more evidence being grossly disproportionate to the benefit gained. It is important to note, however, that the concept of tolerable confidence does not differ between COTS components and bespoke components. That is, it is not acceptable to accept a COTS component for which the safety cannot be tolerably assured, simply because it is too “expensive” to assure it.

One example of this is with the use of analysis evidence. Bespoke components can typically be provided with analysis evidence, as the code is available to aid in constructing formal models. For COTS components, however, analysis evidence can only be provided (if not from the COTS vendor, which is unlikely) by reverse-engineering the product. If sufficient other evidence to show that the COTS component is tolerably safe, then a software safety argument can be used to justify the exclusion of analysis evidence. By comparison, a safety argument which attempts to justify the exclusion of analysis evidence for a bespoke component may not be acceptable.

It has sometimes been proposed that this categorisation of assurance deficits (tolerable, intolerable, broadly acceptable) may be used to justify a greater assurance deficit for complex systems than simple ones. This is because it is typically more expensive to generate evidence for complex systems, and this expense may in some cases be grossly disproportionate to the benefit. It is the position of this guidance that this argument cannot be used for COTS. In more detail, it is common for a COTS component to provide more functionality and complexity than is actually required for the target application. Consequently, the cost of generating evidence may be higher. We emphasise that this cost cannot be used as a justification for providing less evidence to assure the functionality which is required by the target application. In other words, when judging whether enough evidence has been provided to assure the functionality used, only the complexity of the required functionality should figure in a justification of assurance deficits, not the complexity of the entire product. However, in order to demonstrate safety of the COTS component, it is necessary to also demonstrate that this unused functionality cannot contribute unacceptably to system hazards.

D.4.1 Evidence Types and Outcomes

As stated in DS 00-56, there are five different types of direct evidence: analytical evidence, demonstration evidence, quantitative evidence, review evidence and qualitative evidence. We examine each of these and discuss the outcomes which may be expected from the provision of each type of evidence to assure a COTS component. This section is therefore intended to serve as guidance for determining which evidence-gathering processes should be undertaken in order to achieve the desired assurance. Further research into this topic is also anticipated as part of the deliverables for SSEI Task 6.
D.4.1.1 Analytical Evidence

DS 00-56 states that “all analyses are dependent on some form of model” [26], where the code may also serve as a model (e.g. for static analysis). The construction of these models therefore requires access to the source code or design information. Analytical evidence may include formal modelling of the system (e.g. using Z, Haskell etc), the results from static analysis of the source code, tool-supported analysis to identify unused code and so on. Analytical evidence can also include evidence which serves to define the safety requirements for the COTS components, by modelling the context in which the component will operate.

**Outcomes**

Analytical evidence can provide information about almost all functional properties of a system. It can be used to credibly show the absence of all failure modes (with the possible exception, in some situations, of timing failures). Analytical evidence can also be used to show the absence of anomalous behaviour [34], something which can only rarely be achieved with the other types of evidence. Thus, analytical evidence may be used to answer both questions “what does this system do” and “what does this system not do”.

Analytical evidence is the only evidence which can state anything about the internal composition of the software. This includes information such as the extent of code accessed when performing a given function, the timing of internal communications, and the precise algorithms used to accomplish communication with the target application.

**Potential Disadvantages**

The assurance provided by analytical evidence which is based on a formal model of the system (e.g. a Z specification) depends entirely upon the accuracy of the model. When constructing the model, it is typical that assumptions or simplifications are made. These may cause the model to behave differently to the original system, with the consequence that the analytical evidence – while providing assurance for the model – may be flawed when applied to the system itself. To overcome this disadvantage, any simplifications or assumptions made when constructing the model must be justified explicitly.

**Availability for COTS**

Because source code is rarely provided by COTS vendors, it is reasonable to suppose that analytical evidence will be unable to be furnished for a safety case addressing use of a COTS component. This is not strictly true, however, as it is possible to analyse the environment in which the COTS component will operate,
and thereby obtain safety requirements. However, it should be noted that this will result in analytical evidence at the boundary of the COTS component only, and therefore may not be considered suitable for a safety-critical system [6]. As mentioned above, analytical evidence in the form of formal modelling is used to increase confidence that anomalous behaviour will not occur. Consequently, due to the lack of analytical evidence, COTS products will typically give rise to an assurance deficit where it is necessary to show the absence of anomalous behaviours.

Analytical evidence cannot be obtained via in-service evidence, and therefore can only be obtained as existing evidence if a model – with sufficient justification of assumptions and limitations – is provided by the COTS vendor. Alternatively, it may be obtained by first reverse-engineering the code, then conducting a formal analysis on this.

D.4.1.2 Demonstration Evidence

Demonstration evidence can be obtained either from testing or in-service evidence, and is used to demonstrate that the behaviour of the system in question is safe. If it is obtained via testing then DS 00-56 requires that any difference between the test environment and operational environment is justified. The potential differences between these environments are discussed in Section D.3.1.3 for demonstration evidence which is either obtained from the COTS vendor or as in-service evidence. That is, Section D.3.1.3 can be used to help identify those differences which must be justified.

Outcomes

Demonstration evidence is primarily used to show that the system behaves correctly when given a particular input. It can also – in the form of exhaustive testing – be used to show that a system cannot behave anomalously. In the first case, the test profile can be generated from the requirements. In the second case the test profile would have to be generated by referencing the source code. Demonstration evidence is usually used to answer the question “what does the system do in this situation”.

Demonstration evidence – in the form of conducting tests in a new environment – can also be used to confirm supplied evidence, by showing that the component’s behaviour remains unchanged. In this way, demonstration evidence is useful for addressing potential usage mismatches.

Potential Disadvantages

In order for demonstration evidence to be of use, an oracle [36] is required. That is, it is necessary to know in advance what behaviour is expected, given a certain input, so that the success of the test can be judged. In many cases providing such an oracle is difficult.
When using demonstration evidence to show that anomalous behaviours do not occur (i.e. the absence of certain faults of behaviours), it is necessary to have knowledge about either the internal workings of the component, or all possible inputs to the code in question. This will then allow the tester to verify that all possible decision paths in the code have been taken, and none of them lead to the undesired behaviour.

**Availability for COTS**

Demonstration evidence is the type of evidence most likely to be available from the COTS vendor, although it may not be judged sufficiently rigorous for compliance with DS 00-56. Demonstration evidence which is instead newly-generated from testing the COTS component (rather than using existing evidence) must address known problems as well as show that the behaviour of the component is “as expected”. Consequently, it is necessary to obtain information about the correct behaviour of the component. It is also necessary to obtain information about the possible inputs and stimulus to which the component is expected to react. This information will enable the creation of acceptably rigorous test cases, and is often available from the COTS vendor in the form of data sheets and specification descriptions. However, if no evidence whatsoever is provided from the COTS supplier, it is unlikely that sufficient demonstration evidence can be newly generated. Demonstration evidence can be obtained via in-service evidence.

**D.4.1.3 Quantitative Evidence**

Quantitative evidence refers to evidence which is obtained to satisfy numerical safety requirements, such as target failure rates or availability rates. It can be obtained from in-service evidence or statistical testing. Some examples of statistical testing which produce quantitative evidence are failure free models, distribution models from faults observed and reliability growth models.

**Outcomes**

Quantitative evidence is the primary type of evidence for satisfying strict numerical safety requirements, such as target failure rates below a certain threshold. Other safety integrity requirements such as reliability and maintainability can also best be shown using quantitative evidence.

In some cases quantitative evidence of a low quality can be obtained via testing without using these statistical methods (e.g. black box testing). However, there are problems with this as discussed below.

**Potential Disadvantages**

One important characteristic of quantitative evidence is that if it is obtained for safety-critical systems via testing or operational data only (i.e. with no formal
analysis or predictions using statistical models) there is likely to be an assurance deficit. The reason for this is that it is unlikely that the component in question has been in any system for long enough to obtain convincing evidence of its reliability, failure rate etc. Similarly, testing can only determine failure rates of higher than $10^{-3}$ [24]. Additionally, poor data collection methods can have an impact upon the quality of quantitative evidence. If failures are not consistently reported, the evidence may imply that the software in question experiences fewer of these than is the case.

**Availability for COTS**

It is not unknown for developers to claim – or to be required to show – satisfaction of more stringent numerical safety requirements than can be directly shown via testing. COTS vendors typically do not perform statistical testing, and without data as to the implementation of the component, statistical tests are unlikely to be able to be performed cost-effectively after purchase.

Quantitative evidence can be obtained via in-service evidence, but as mentioned earlier, there is likely to be an assurance deficit. This deficit may be addressed by making use of statistical distribution testing, but this must be explicitly noted and justified in the safety case.

**D.4.1.4 Review Evidence**

Review evidence consists of external or internal assessment of the software development processes. The purpose of review evidence is to show that the software in question is capable of satisfying its safety requirements. Review evidence typically consists of formal reviews which can be examined against available criteria. Review evidence does not include user testimonials (see Section D.4.1.5).

**Outcomes**

Review evidence can be used to demonstrate traceability of requirements, validity of requirements, robustness and compliance with good practice (all required by DS 00-56). This and qualitative evidence are the only types of evidence which can be used to provide these outcomes.

Review evidence is also the only type of evidence which can be used to easily demonstrate compliance (for COTS components) with a standard. This is important because of the position of DS 00-56 to be “as civil as possible and only as military as necessary”. That is, demonstrating compliance with an existing standard can sometimes provide significant assurance to a safety argument. Review evidence can also be used to identify discrepancies between the high-level requirements and the specification of the software, or the way in which these
high-level requirements have been interpreted. Additionally, review evidence can be used in a supporting role when combined with the other forms of evidence

**Potential Disadvantages**

There are two major problems with review evidence. Firstly, it is subjective – it is generally very difficult to objectively demonstrate compliance with good practice, for example. Secondly, it cannot be used to directly demonstrate either of the most common safety claims, these being “the system does what it should” and “the system does not do what it shouldn’t”. Review evidence of a process cannot provide this, and review evidence of the product can only determine that the safety requirements are translated into derived safety requirements. That is, it cannot objectively assess the performance of the software in an operational environment.

Where review evidence includes a claim of good practice, this may have to be qualified. Good practice differs between domains, and also between assessors. Consequently, review evidence showing that standards of “good practice” are met may or may not be relevant, depending on whether those properties which are denoted as good practice by the reviewers are also denoted as such by the acquirer of a COTS component. Furthermore, review evidence of a product must consider the operating environment of the product. If the review has taken place under different conditions than would be expected in the proposed environment, it may not be relevant to demonstrate safety of the software.

**Availability for COTS**

Review evidence cannot typically be newly generated for the COTS product, especially if reviews of the development process are required. Typically, review evidence will be obtainable if the COTS product has been developed to an existing standard such as DO-178B or IEC 61508. These mandate traceability, compliance with good practice, validity and robustness, which are the properties required by DS 00-56 (if another standard has been used for the COTS component, then the review evidence must be explicitly assessed to ensure that compliance to this standard demonstrates the required properties).

Naturally, depending on the trustworthiness of the COTS vendor, the claim that a product has been developed to one of these standards may or may not be sufficient to contribute in a safety case. That is, a vendor claim that the product meets the requirements of DO-178B may not be sufficient to be included as review evidence in a safety case. However, in the majority of cases the COTS vendor – to reach the stage where the product is being seriously considered – is likely to be a trustworthy source. Additionally, third-party review evidence may not be susceptible to these issues of trustworthiness to the same degree. Section D.3.1.1 covers this in more detail.
In-service evidence cannot be used as review evidence. Consequently, COTS products – to satisfy the DS 00-56 requirement to provide review evidence – must either be produced to an accepted standard or must make design documentation and assessments of these available.

**D.4.1.5 Qualitative Evidence**

Qualitative evidence is evidence that shows that good practice has been followed in the design of the component. It differs from review evidence in that there are generally no set criteria to measure good practice or “good design” in the same way as there is to measure compliance with a standard. In fact, good practice may in some cases be interpreted as compliance with an appropriate standard. In this way qualitative evidence is valuable because it can be used to demonstrate conformance with those standards. It also can increase confidence in those aspects of the design process which are not explicitly assessed by reviews.

**Outcomes**

Qualitative evidence is the only type of evidence which shows the consistent application of good practice throughout the entire software development lifecycle. While review evidence also assesses good practice principles, these are assessed against set criteria. Hence, review evidence will not assess any principles which are not explicitly required by the conditions of the review.

**Potential Disadvantages**

The potential problems for qualitative evidence are the same as those for review evidence – a lack of direct applicability, and a lack of objectiveness. In the case of qualitative evidence, the problem of what constitutes “good practice” is more complex than the similar question for review evidence. This is owing to the fact that there are no set criteria to assess the processes against, while for review evidence these criteria are provided. What is defined as good practice can also change, and is domain-dependent as well as differing for many assessors. Consequently, this type of evidence is best supported by a justification that the techniques, processes, tool usage etc. it demonstrates do conform to good practice as accepted by the MOD or a relevant assessment authority. Future SSEI work (Task 6) will consider this in more detail.

**Availability for COTS**

If sufficient information is provided about the design and lifecycle of the COTS component, it is possible to generate new qualitative evidence. However, in the main this evidence will have to be provided by third parties or the COTS supplier.

**D.4.2 Assessment of Evidence**

Ascertaining whether “enough evidence” has been provided for each claim is one of the most difficult questions when constructing a safety case. While it is certainly
possible to assign a criticality level to software (e.g. the DALs of DO-178B), it is much less simple to determine exactly what implications this has for the evidence which must be provided.

Unlike earlier standards (DO-178B, DS 00-55), DS 00-56 is goal-based. This means that to comply with 00-56, it is insufficient to simply perform certain development activities without also providing an argument as to why these activities support safety requirements. More subtly, it is also insufficient to provide a list of types of evidence which must be provided at each “criticality level”. The reason for this is that such a list does not provide any justification as to why these evidence types have been selected for each level. Furthermore the rigour of the evidence at each level would also have to be considered, regardless of the evidence type.

To address this issue, we recommend that when considering a COTS component for inclusion into a system, an assessment should be performed to determine what evidence is needed to support a compelling argument for its safety. This process will be referred to as assurance-based argument development. It requires that a criticality analysis be performed to determine the importance to safety of this component and its functions. Subsequent to that, further analysis (the argument development) must be performed to determine exactly what is required to provide that confidence. Thus, instead of simply stating – for example – that formal analysis should be performed, it is necessary to state what benefit (increase in confidence) is expected from the formal analysis, as well as what expectations are placed upon the rigour of the analysis. Other work [17] covers the topic of assurance-based argument development more fully, but a brief description of it is included here to understand how different COTS components may be evaluated for selection and inclusion in a system. SSEI Task 6 is intended to produce a report thoroughly detailing the issues faced when constructing arguments, which will provide more detail to the primarily COTS-focussed discussion below.

D.4.2.1 Assurance-Based Argument Development

Assurance-based argument development is the term we will use to describe the process of constructing a safety argument – and providing evidence to support this – which will result in sufficient assurance of the safety of a COTS component. This process is based on the six-step method [21], which can be briefly summarised as follows. The first step consists of “phrasing the goal”, or determining precisely what property the evidence should support or demonstrate. The second step is “defining the basis for the goal”, and determines under what circumstances the goal is appropriate and must be satisfied. The third step is to “identify the strategy” for satisfying the goal, and determines how the goal can be adequately supported by breaking it down into sub-goals. The fourth step is to determine whether all assumptions made have been justified, while the fifth step “elaborates the strategy” by identifying those goals which will implement the chosen strategy (step 3). Finally, the sixth question is intended to identify
evidence which provides adequate support to this goal. This step determines what rigour of evidence will be required.

This top-down strategy is extended in [17] – and elaborated in the deliverables for SSEI Task 6 – to examine the sufficiency of the argument, given the findings of a criticality analysis to determine how much confidence is needed in a claim. This extended process ensures that sufficient assurance is provided at each level of the argument. This is achieved by considering why assurance might be lost at each step of the argument, and what argument structure (for example, deciding on the sub-claims for each safety claim) is necessary in order to minimise this loss. This loss of assurance can then be justified by assessing the benefit gained and the costs associated with addressing the loss. The difficulty with applying this strategy to COTS products is that there will inevitably be a compromise between the ideal strategy and evidence and the evidence which is actually obtainable. As described earlier, not all types of evidence are readily-available for COTS components. Hence the assurance deficits – while perhaps not being larger than those for bespoke products – typically originate from a different cause.

In order for COTS components to be cost-effective, the maximum use must be made of the existing evidence. Where this is not sufficient to assure the safety of the system, new evidence can be generated to address this gap. However, this adds another dimension to the argument strategy described above – that the availability of the evidence must be kept in mind throughout the argument construction. Consequently, the argument must be structured to make maximum use of the available evidence, and make it clear where new evidence is required. Furthermore the rigour of evidence must also be kept in mind throughout the construction. The complete argument structure should not be considered finalised before the rigour of the evidence provided by the COTS vendor is assessed (although a preliminary safety case structure may be considered finalised). The reason for this is that if the argument structure depends on the provision of certain evidence for the COTS product, then if this evidence turns out to be unobtainable (or not to meet the required standard of rigour), the relevant part of the argument structure will need to be redefined. Refinement of the argument at a late stage can result in projects being completed after the deadline or over budget. More importantly, failing to consider the rigour of evidence may result in the selection of a COTS candidate which was seemingly the best, but on later examination proved to be a poor choice in terms of safety assurance.

The quality and quantity of evidence which is deemed necessary to support a COTS safety claim can also be construed as COTS selection criteria. That is, from a safety perspective the best candidate COTS component to choose is the one which most nearly provides this required quality and quantity of evidence. Of course, in practice the safety considerations usually have to be balanced with the functional considerations (a candidate COTS component may be demonstrably safer than the others, but if it does not provide the required functionality then it may not be the best choice). Nevertheless, identifying the required rigour and extent of evidence to demonstrate safety can be considered as identifying safety-
related COTS selection criteria. This is the approach taken in Section D.5, where a contractual language is used to represent these selection criteria.

These safety-related COTS selection criteria will be referred to in this paper as “COTS Evidential Outcomes”. They are obtained by applying the six-step strategy referenced above [21], while keeping in mind the caveats on available evidence for COTS products.

D.4.2.2 COTS Evidential Outcomes

There are two major types of COTS evidential outcomes which must be identified in order to form safety-related selection criteria for COTS components. The first type relates to properties of the COTS software which must be demonstrated. That is, evidential outcomes of the first type will be obtained by considering the strategy which will be used to show the COTS component is safe. We will refer to these evidential outcomes as COTS evidential strategy outcomes, which are most closely linked with Step 3 of the six-step strategy. The second type relates to the rigour of the evidence which will be used to support the argument that the COTS component is safe. These evidential outcomes are most closely linked to Step 6 of the six-step strategy.

For example, consider a COTS database providing Add, Edit and Delete functionality. One safety claim which might need to be satisfied for the safe operation of this software is that “Omission of ‘Add’ functionality does not occur”. A strategy to support this claim may involve showing that neither the implementation nor design permit the ‘Add’ functionality to be omitted when it is requested. These would be expressed as sub-claims “Software design proscribes omission of the ‘Add’ functionality”, and “Software implementation proscribes omission of the ‘Add’ functionality”. To provide more assurance that the ‘Add’ functionality is not omitted, the strategy may also require that the following sub-claims are demonstrated: “ Necessary resources exist to support execution of ‘Add’ functionality”, “Other components do not interfere with execution of ‘Add’ functionality” or “Communication between ‘Add’ functionality and target application cannot be corrupted”. These five sub-claims then form the strategy – or the COTS evidential strategy outcomes – for demonstrating the top-level claim.

The chosen strategy for demonstrating a claim will vary with the confidence required in that claim. When identifying the sub-claims which make up the argument strategy, it is important to consider the effect of not providing one or more of these sub-claims. For example, omitting the sub-claim “Software design proscribes omission of the ‘Add’ functionality” means we will have no assurance about the design of the software. That is, we will not be able to demonstrate traceability of the ‘Add’ function from high-level requirements down through design to implementation, nor will we be able to show that software architectural solutions make omission of this function impossible. Whether this assurance deficit is acceptable will depend upon the criticality of the COTS component and the cost of providing the assurance. SSEI Task 6 covers this in more detail.
However, it is also important when setting these evidential strategy outcomes to consider what is likely to be provided for a COTS component. A claim that the ‘Add’ functionality is present in the software design will increase the assurance of the top-level claim, but is itself very unlikely to be supported for a COTS component (design documents are not often provided). Consequently, a different strategy to support this top-level claim – for example, showing via operational testing or reverse-engineering that this functionality is present in the code – may provide more assurance overall.

The other type of COTS evidential outcome relates to the rigour and extent of evidence which is necessary to assure the safety claim. We will refer to these evidential outcomes as “COTS evidential rigour outcomes”. Evidential rigour outcomes constrain such characteristics as the trustworthiness, replicability, independence of sources, maintainability and extent of coverage of the evidence. Where a COTS evidential rigour outcome is set, this must be justified by an argument demonstrating what increase in assurance is gained from this outcome. Where an evidential rigour outcome is set which implies that some assurance is lost (for example, where testing coverage is not required to be exhaustive) this assurance deficit must be justified.

For example, a COTS evidential rigour outcome might be that evidence supporting the claim should be replicable, meaning that in-service or qualitative evidence is not acceptable. The assurance gained by setting this outcome lies in the fact that replicable evidence can be verified by the purchaser, and therefore greater trust will be placed in it after this verification. Another COTS evidential rigour outcome could be that the evidence supporting a claim should be sourced from demonstrably independent sources. The assurance gained from this is due to the fact that common cause failures are less likely where evidence is sourced independently.

In general, every COTS evidential rigour outcome must be generated in the context of a specific claim. That is, evidential rigour outcomes relating to the required extent of test coverage must address a particular safety claim. To take the previous example, when attempting to demonstrate the claim that ‘Add’ functionality in a database cannot be omitted, one possible evidential rigour outcome may be that testing for this functionality in all possible environmental modes is required. A less stringent evidential rigour outcome may be that testing for this functionality in the most common environmental modes is required, with a justification that shows that the cost of obtaining information about the rarely-used modes is grossly disproportionate to the information gained.

While it is not the purpose of this paper to provide a full discussion of the issues involved in constructing a safety argument (this will be provided in the deliverables for SSEI Task 6), we will provide examples of what is meant by “rigour of evidence” for COTS components specifically. The examples that follow are in the form of two different sets of COTS evidential rigour outcomes (or requirements on the evidence) which might be required for individual claims made about COTS
components. These correspond to the evidential rigour outcomes which might be required for particular safety claims if the COTS component is safety-critical (Section D.4.3.1) or safety-related only (Section D.4.3.2). These two different sets of COTS evidential rigour outcomes do therefore roughly correspond to two different levels of confidence, or assurance, in the system. However, these are demonstrative only. As explained before, the presence of any COTS evidential rigour outcome must be justified by stating exactly what extra information – and therefore what extra assurance – is gained. Likewise, any assurance deficit associated with this evidential rigour outcome must be justified. Consequently, there is no single defined “assurance level” (SIL / DAL) for the system.

D.4.3 Sample COTS Evidential Rigour Outcomes

In this section we present some examples of evidential rigour outcomes for COTS components, intended primarily to give the “flavour” of what these should look like. We emphasise that all evidential outcomes can in practice only be used in the context of a particular safety requirement, and additionally must be justified by considering the effect on assurance if this outcome is not met. However, our research (Annex D.B) has shown the need for providing examples of how the required rigour of evidence might vary between safety-related and safety-critical systems. Consequently, the examples below highlight how the changes in evidential rigour outcomes affect assurance. Of course, each claim must be individually considered in the context of the entire system to determine whether any consequent assurance deficit can be justified.

D.4.3.1 COTS Evidential Rigour Outcomes Set 1

This set of COTS evidential rigour outcomes includes those which might be required to demonstrate satisfaction of different safety requirements for a safety-critical COTS component.

The first outcome is that diverse and independent evidence must be provided to support claims (this provides more assurance by reducing the likelihood of common cause failures). A second outcome might be that if any numerical target failure rates more stringent than 10^-3 are presented as safety requirements, then quantitative evidence – in the form of statistical modelling, or in-service evidence – is required to adequately demonstrate satisfaction of these. The justification for this is because demonstration evidence cannot show failure rates lower than this, and hence such evidence cannot be considered compelling.

A third outcome might be that if a claim is presented about the functionality which the system is to provide, then evidence of that functionality being explicitly exercised must be present (typically, this evidential outcome will require that in-service or testing evidence is used). This evidential outcome provides increased confidence that the environment does not interfere with this function execution. A further evidential rigour outcome might be that it is also necessary for the evidence to show that the functionality is present and executable when the
software is placed in an operational context (typically, this will mean in-service evidence is required, as well as any suitable “usage mismatch” discussion of operational context). This outcome gives greater assurance about the true consequences of failure of the functionality, which then permits verification of any models used.

A possible fourth evidential outcome is that the evidence presented to support a claim must be shown to be representative of functionality of this COTS product (rather than another version). The justification for this outcome is that there is no certainty that evidence originating from another version will demonstrate the same failures as evidence generating from this version. Consequently, this evidential outcome increases our confidence in the configuration consistency of the evidence. Satisfaction of this outcome could be achieved by the provision of source code, or the provision of process documents showing that all upgrades to this component were performed before the evidence was generated. Alternatively, configuration consistency documents such as those required for DO-178B might be used to satisfy this outcome.

A fifth evidential outcome might be that the evidence presented to support any claim must fully support that claim in all relevant software modes and environments. For example, consider a claim about the ability of the COTS component to provide a certain safety-critical function. According to this fifth evidential outcome, if this claim is supported by in-service evidence then this evidence must demonstrate that all methods of executing this function have been invoked during the operational use. If not, other evidence (e.g. demonstration evidence) will be required to “plug the gap”. Similarly, if the claim is that anomalous behaviour will not occur, then this fifth evidential outcome implies that the evidence must be sufficient to demonstrate absence of anomalous behaviours under all circumstances (so, for example, formal analysis or exhaustive testing may be sufficient while normal-range testing would not be).

A sixth outcome could require that every claim must be supported with evidence which is trustworthy. This is particularly relevant where the claims are supported with evidence sourced from a COTS vendor. Ensuring trustworthiness can be partially achieved by performing operational tests as in Section D.3.2 to verify all quantitative and demonstration claims made by the vendor. Alternatively, trustworthiness of the vendor may be demonstrated by appeal to external reviews (which provide a measure of independence) or qualitative evidence.

Finally, a seventh outcome might be that counter-evidence must be searched for, for particular named claims. If any is found, then the reasons why it is inapplicable must be stated (for example, it was generated from a different operational environment). If the counter-evidence is shown to be both applicable and trustworthy, then the safety claim cannot be supported.

D.4.3.2 COTS Evidential Rigour Outcomes Set 2
These evidential outcomes are examples of those which might be applied against safety requirements for a non-critical component. That is, these evidential requirements are less stringent because failure of the component to satisfy its safety claims will have a less serious consequence. It is important to emphasise that in each case, the evidential outcome must be justified. That is, if assurance about certain properties is missing, this absence must be justified given the criticality of the software. Once again, this is not an exhaustive list of outcomes, but simply an example of what outcomes might look like and how they might be satisfied.

The first outcome, as before, may be that diverse evidence must be provided to support claims (hence reducing the likelihood of common cause failures). The second example evidential rigour outcome relates to numerical target failures rates, and states that if numerical target failure rates more stringent than $10^{-3}$ are presented as safety requirements, then these must be supported by an appeal to the integrity of the software (in practice, this may require process review evidence and in-service evidence showing all failures). Note the difference between this evidential outcome and the corresponding one in Section D.4.3.1, which required demonstration by statistical modelling techniques. The justification for this difference may be that the reliability of statistical tests is questionable, and the expenditure required to perform them is significant.

A third outcome might be that if a claim is presented about the functionality which the system is to provide, then evidence of that functionality being explicitly exercised must be present. This is similar to the corresponding evidential outcome in Section D.4.3.1, but lacks the additional requirement that the functionality must be shown to be present within an operational context. The consequent assurance deficit relates to the fact that we now have no evidence showing that the functionality is not masked or interfered with in the actual operating environment. This deficit could be justified by demonstrating that the testing environment is equivalent in terms of stimulus and conditions to the operational environment, and that therefore results from one may be extrapolated to the other.

A fourth outcome might state that the evidence supporting a claim must be shown to come from an equivalent COTS component, if not this version. The assurance deficit here originates from the fact that we appear to have no knowledge of how this evidence relates to the evidence which would have been generated from the version in question. However, to justify this deficit we might perform a gap analysis (Section D.3) to show that the environment of the original version and the one in question are similar enough. Furthermore, we may supplement this with configuration consistency documentation showing that no upgrades have impacted this functionality.

A fifth evidential outcome might be that the evidence presented to support any claim must be shown to completely address the areas of greatest risk. For example, if the COTS component is a database, consider the safety claim “The database will never fail to update an entry when required”. If the database is
divided into numeric entries and text “comments” on these, then it may be sufficient to show that the database will update all numeric entries when required, and not to necessarily show that all comments are updated also. The assurance deficit here comes from the fact that we do not know if the comments have been updated at any stage. This may be justified by demonstrating that the accuracy of the comments does not have any importance to safety. Evidential outcomes of this type should be supplemented by a caveat that if the areas of greatest risk cannot be determined, then the evidence must address the whole of the claim.

A sixth outcome might be that counter-evidence must be searched for, for particular named claims. If any is found, then the reasons why it is inapplicable must be stated (for example, it was generated from a different operational environment). If the counter-evidence is not shown to be both applicable and trustworthy, then its effect on the claim need not be assessed.

We emphasise that the evidential rigour outcomes given above are examples only. In practice, each evidential rigour outcome must be associated with a specific claim. In addition, the evidential rigour outcomes are selected by considering their effect on the assurance of the claim in question. Consequently, each claim and each system must be considered individually.

D.5 Selecting a COTS Component

In this section we will discuss the evaluation and selection of a COTS component, from a safety-related perspective. Current COTS selection and evaluation methods often fail to adequately consider safety requirements, and to address this we will propose the adoption of a contract-based method of selection [13].

This contract-based method can be viewed as an extension to existing COTS selection methods, in that it explicitly requires the consideration of safety requirements as well as functional ones. While this requires a slight increase in costs and time during the selection and evaluation phase, it reduces the risk of selecting a COTS product which does not meet the demands of a safety-critical system. We will also present a language which can be used to express the contracts which are central to this method.

The COTS selection and evaluation method which we present is not intended to be mandated for use on all projects. Rather, it demonstrates one way to explicitly consider safety requirements during selection. It may be customized for use with existing selection procedures, or extended to consider project-specific requirements. Provided that safety requirements have been considered and the use of the selected COTS component can be justified in an adequately compelling safety case, the intent of this guidance will have been satisfied.

D.5.1 Existing COTS selection and evaluation methods
There are many existing approaches to selection and evaluation of a COTS component [23, 28, 7], the majority of which emphasise functional requirements over non-functional aspects. That is, these techniques provide guidance for establishing functional requirements for the COTS component, in terms of specifying “what must this component do”. The position of this guidance is that this type of requirements establishment is necessary, but it is not sufficient.

The reason for this is that many safety requirements are best expressed as a mixture of functional and non-functional requirements (including considerations such as certifiability and robustness). The selection methods above therefore may not emphasise safety to the level required during the selection process. Consequently, there is the likelihood of selecting a COTS product for which an acceptable safety case cannot be made. That is, the existing methods may result in selection of a COTS component which is not “safe enough”.

Similarly, the methods above also emphasise cost as an important factor when evaluating and selecting a COTS component. While this is certainly true, the requirements of DS 00-56 require that all risks should be reduced until they are ALARP. If cost is considered as one of the most important factors in selecting a component – that is, before a detailed risk assessment of each component can be carried out, and to a greater extent than the consideration of “reasonably practical” when reducing risks to ALARP – it will be almost impossible to adequately demonstrate that the system risks are ALARP. This is because the candidate components are unlikely to contribute identical measures of risk, so selecting one before detailed risks assessments are carried out means that that component selected is not necessarily demonstrably safe. That is, the existing methods may result in selection of a COTS component which – while it may actually be “safe enough” – cannot necessarily be shown to be safe enough to the rigour required by DS 00-56.

The consequence of both of these disadvantages is that significant additional work may be required after the selection of the COTS component, to ensure that its use in a safety-critical system can be justified. This work takes the form of providing mitigation such as component wrapping [15] and redundant components to ensure diversity. The extent of this cannot be adequately judged during selection, hence may result in selecting a COTS component which imposes a higher cost than would have been incurred by developing a bespoke solution. Because cost is generally one of the major motivating factors for use of COTS, this is an unsatisfactory outcome.

D.5.2 A Contract-Based Approach

The approach we present here considers safety requirements as an essential focus during selection and evaluation processes. At a high-level it is based on a two-stage process where candidate components are first selected against some relatively high-level criteria, then more carefully evaluated using detailed criteria [9]. However, the individual stages proposed – and guidance for the processes
which must be undertaken at each stage – are based on a detailed contract-based framework centred on criticality analysis [13, 14].

The framework as presented consists of four stages:

- Reference COTS Component Definition
- Evaluation and Selection Criteria Establishment
- COTS Component Evaluation and Selection
- Safety Justification of COTS Component Use

A fifth stage, System Safety Maintenance, is not explicitly discussed in this paper. It is not the intent of this guidance to present an extensive discussion of the theoretical underpinnings of this method. Instead, we will simply describe each stage and show how the processes undertaken result in an awareness of safety requirements throughout the entire selection and evaluation lifecycle.

In brief, after the decision to consider use of a COTS component is made, the required COTS component is defined in terms of its functionality and interfaces. This process is similar to that undertaken during most existing selection and evaluation methods. That is, the process consists firstly of establishing all functional requirements, as well as the agreed interfaces between the COTS component and the rest of the system. A “reference” COTS component is defined, being simply a hypothetical component satisfying these functional requirements.

Secondly, safety-related evaluation and selection criteria are established. This is achieved by performing a safety analysis upon the reference (hypothetical) COTS component. The possible failure modes are identified, and their consequences determined. Determining the consequences of these failures can be done by performing a criticality analysis on the target application. This allows a judgement to be made about the quantity and rigour of evidence that will be required (either existing or newly generated) to justify use of a COTS component. The requirements relating to the provision of evidence then become the safety-related evaluation and selection criteria for the candidate COTS components.

Thirdly, the individual candidate COTS components are assessed on how well they meet these criteria. Any deviation from the specification of the reference component, including the provision of unneeded functionality or the lack of desired functionality, is also considered here. The component selected will be the component for which there is the least “gap” between the desired and delivered safety and functionality requirements. A contract-based approach is used for the safety requirements, and any project or organisation-specific approach can be used for addressing gaps between the functional requirements and the candidate COTS component. It should also be noted that in many cases there is only one feasible candidate for a COTS component. The assessment against the criteria
will still need to be carried out, however, to determine whether this solution will be ultimately more beneficial than a bespoke solution. Additionally, the assessment will identify areas in which more evidence as to the safety of the product is required.

Fourthly, a safety justification is presented for the use of this COTS component. All necessary evidence identified by the selection criteria must be presented here, and any assurance deficit justified. This stage may also include any project or organisational-specific justification for the use of COTS and for the relaxation of any functional requirements which may have been performed.

As can be seen, this method extends many existing methods [7] by explicitly considering safety requirements in step two. COTS components are assessed on how well they satisfy the safety criteria. The contract-based approach which we recommend allows the capture of any assumptions made, as well as the capture of information about the extent of the evidence which will be required.

**Timing of Contract Steps**

The contract steps as outlined above are described serially (that is, where earlier steps provide input for later steps). This does not imply, however, that we recommend a serial approach to development. Most typical development processes are iterative, meaning that development phases are often repeated several times throughout the project. There are several advantages for this, such as the ability for a design to be modified as new requirements emerge, or implementation considerations can affect the “hard” and “soft” requirements. From a safety perspective, at the time of establishing a contract (and selecting a COTS supplier) it is unlikely that detailed risk assessment will have been carried out. Consequently, many of the safety requirements will be unknown, and will emerge as more comprehensive safety assessments are undertaken.

The approach as described here can be used iteratively, to take advantage of these benefits (although it is likely that the fourth and fifth steps will not be undertaken until a relatively late stage of development). That is, when defining a reference component in order to obtain selection criteria, it is not necessary to know all the required characteristics of the component. Instead, a preliminary reference component may be defined, where only high-level knowledge of required functionality and safety properties is required. This preliminary reference component can then be used to establish preliminary selection and evaluation criteria. Although initially these are unlikely to be detailed, they can be helpful in establishing the scope and structure of the eventual safety case. Similarly, preliminary safety cases for components which will interact with the proposed COTS software may make use of the preliminary reference component. The use of a contract language – such as that proposed here – may allow a modular approach to be taken to the safety case, as the contract will help to provide a well-defined interface between the COTS component and the system [3].
The first three stages (and, to a lesser extent, the fourth) may then be performed iteratively as more information about safety requirements and functional properties emerges. Candidate COTS components may then be assessed in more detail against these emergent requirements. As the safety requirements become more detailed, information about how these may best be demonstrated will also become more detailed. That is, as the project progresses it will be possible to assess whether each candidate COTS solution will provide the evidence which is needed to demonstrate its safety.

One issue which has been identified with the assessment of COTS products and the selection of a supplier is the difficulty with timing. For contractual reasons it may be necessary to select a supplier at a very early stage of the project, when detailed hazard and risk assessment is not possible. In this case, it is possible that detailed software safety requirements will not be known. Rather, the safety requirements for the COTS product may have been “flowed down” from system safety requirements. Consequently, the evaluation and selection criteria may be less detailed than we describe below. Other SSEI work (Tasks 2 and 3) is currently being undertaken to assess the administration and timing aspects of establishing contracts, which may result in more detailed evaluation of COTS solutions.

D.5.2.1 Definition of Reference COTS Component

The first step in this stage is to establish relatively high-level functional requirements for the COTS component. These can be established from a number of sources, including use cases, stakeholder interviews, and knowledge of the operational environment. One particular class of functional requirement which it is vital to establish at this stage is the specification of the interfaces which will be used by the system to communicate with the COTS component.

The reference COTS component is intended both to satisfy the needs of the application and to be representative of the available functionality on the market [13]. In some cases it may not be possible to achieve this second point, because there may not be sufficient commonality across all candidates. If this situation arises, then it is sufficient to use an “ideal candidate” to determine the qualities of the COTS component. For example, the specification of a particular standard interface may be used for the reference COTS component if there is not sufficient commonality across the market in terms of the interfaces offered by COTS components. This must, however, be recorded in the process documentation as it will result in further work when considering the individual COTS components (and the particular interfaces they offer) as described in Section D.5.2.3.

During the establishment of functional requirements it will almost certainly be necessary to modify these to the functionality offered on the market [12, 16]. Determining which requirements can be modified to suit the market and which are fixed can be a very difficult process. A full discussion of this is out of the scope of this document, but we note that there is much available guidance (e.g. [7]) for this.
In general, requirements can be categorised as *hard requirements* or *negotiable requirements*. Hard requirements include critical functionality and any software characteristics which are legally required, while negotiable requirements might include those which specify the optimum desired performance.

If the entire system is currently being developed – rather than an existing system modified to make use of a COTS component – then it may be necessary to establish a preliminary software architecture to help elicit the functional requirements and required interfaces. This has the advantage of building modifiability into the system architecture from the beginning, resulting in a system which is more robust to the constant change and upgrading which is a property of COTS components. If the system is currently being developed, then it may be advisable to propose alterations to the system architecture to facilitate these eventual changes. The alterations may include such things as the introduction of a wrapper component to restrict interactions between the COTS component and the rest of the system, hence providing a constant interface to the system even if the COTS component changes. If this is the case, then this updated system architecture must be used when eliciting the functional requirements.

Finally, a reference COTS component is defined as being a hypothetical component which satisfies these functional requirements and provides the desired interfaces. Note that the internal workings of the reference component are unknown at this point.

**D.5.2.2 Establishment of Evaluation and Selection Criteria**

The second stage consists of establishing the evaluation and selection criteria for the COTS component. Because a reference component has already been defined in the first stage, we can assume that the only requirements left to elicit are those addressing safety.

In order to establish these requirements, it is necessary to carry out software safety analysis to determine the effects of failure of the COTS component. There are a number of established techniques for this, including HAZOP [32], FTA [35], SHARD [31] and Functional Failure Analysis – in addition to many organisation-specific techniques. All these techniques are designed to identify the possible failures or failure modes of the reference component, and determine the effect of these upon the system. However, this working paper does not specifically recommend a single technique or combination of techniques which are to be used.

This safety analysis will identify potential failures of the COTS component, and will lead to the identification of several safety requirements. That is, for each failure there will be a consequent safety requirement that this failure is either eliminated or acceptably detected and handled. These safety requirements will eventually become safety selection criteria. They will be used to select the COTS component
for which evidence can be most easily supplied to provide the required support for a safety argument.

Because the analysis is being carried out on a hypothetical reference component – or more accurately, upon a specification – it is obvious that the findings (safety selection criteria) will be applicable only to those candidate COTS components which provide all the functions and interfaces provided by the reference component. That is, the failures of the reference component are intended to be generic, meaning that the safety criteria established from the analysis are applicable to all the candidate components. In some situations, however, this may not be the case – for example, if there is not sufficient commonality across the market to define a single reference component. In this case, as described in Section D.5.2.1, the reference component makes use of “ideal” functionality, which must be documented. Where this “ideal” functionality has been used, there are two choices.

Firstly, the safety analysis can be carried out on the “idealised” reference component, and the work of distinguishing the actual candidate COTS components from the reference deferred until the next stage. That is, further safety analysis must be carried out when evaluating individual COTS components.

Secondly, the “idealised” reference component can be replaced at this stage with a number of “actualised” reference components. Each of these components will have the “ideal” functionality replaced with actual functionality obtained from a candidate COTS component on the market (obviously, where a group of candidate COTS components provide the same functionality, a single reference component can be used to represent this group). Safety analysis must be conducted on each of these new reference components, and the actual COTS components to which they relate must be explicitly noted in project documentation.

**Component Criticality Analysis**

The process described above will identify the potential failures of the reference COTS component by carrying out safety analysis. These failures will then give rise to related safety requirements. That is, each of these failures will be associated with a safety requirement that the failure is absent or acceptably detected and handled. The candidate COTS component which most adequately shown to satisfy these safety requirements will then be the best possible choice from a safety perspective. However, in order to identify this candidate component, we must first consider the second aspect of satisfying a safety requirement – the assurance, or confidence, that is needed in the satisfaction of that requirement. In order to determine how much confidence we will require in each of these safety requirements we must carry out a component criticality analysis on the reference COTS component.
A criticality analysis will identify the “risk responsibility” of the reference COTS component. That is, this analysis will determine the degree of contribution made by each failure of the reference COTS component to system-level hazards. By determining the importance to safety of each of these failures, a criticality analysis enables us to identify those failures which are the most critical. More critical failures will need more assurance of their absence or acceptable mitigation than less critical failures will need.

There are a number of different ways in which criticality can be determined. One of these [4] is to assign an index based upon the probability of each component failure, the probability of such a failure progressing to a system hazard, the consequences of this system hazard and the probability of the system hazard occurring under any circumstances. While this results in a defensible and mathematically sound measure of criticality, it is often impossible to ascertain the required probabilities. For example, determining the failure rate – and therefore the probability of failure per year – of software is often very difficult to achieve. Software failures are systematic instead of random [24], and testing evidence can be insufficient to show satisfaction of a low failure rate. The other probabilities are also unlikely to be established without performing analysis of the source code – an activity which can often not be performed upon COTS products, and certainly cannot be performed upon a hypothetical reference component.

Other criticality discussions have emphasised the need to consider the frequency of component failure, the potential impact of the component failure, the degree of isolation of the component, and the potential effect on other components [2]. Again, this would provide a good estimate of the criticality of a component, but the practicality of carrying out such an assessment is doubtful. In particular, identifying the frequency of component failure is problematic for the reasons discussed earlier. Similarly, there is conceptual overlap between the impact of the component failure and the potential effect on other components, which could lead to an unclear analysis.

An alternative, and more practical, method of criticality analysis is to consider only the degree of protection afforded to each of the failure modes of the reference COTS component, and the severity of the hazards which may be caused by each of these failures [14]. The degree of protection refers to the level of protective measures provided for a failure of the reference component, and can be ascertained from fault tree analysis of the component. For example, the provision of a monitor component which will detect and acceptably handle timing failures of the COTS component constitutes a single degree of protection.

As input, this last method of criticality analysis will require at a minimum the identification of all system hazards and their severity levels, as well as knowledge of the possible consequences of each failure of the reference component. The first of these – identification of system hazards and their severities – can be achieved via the use of the Hazard Log mandated by DS 00-56. The second can be provided by the use of fault tree analysis with the failures – which were
previously identified in the safety analysis – as basic events. Fault tree analysis is an established technique for assessing the degree of protection accorded to each component failure, and suitably-qualified tools [8] may be used to increase its efficiency. Note, however, that a justification as to the suitability of these tools in a safety-critical application must be presented.

As mentioned before, the criticality of each of the reference COTS component failures will determine the degree of assurance needed in the related safety requirement (as discussed earlier, the related safety requirement states that this particular failure does not occur, or is acceptably detected and handled). Conducting such an analysis is therefore essential to distinguish between the COTS candidates which provide sufficient evidence that they satisfy these safety requirements and the COTS candidates which do not. However, due to the goal-based nature of DS 00-56 we do not provide guidance for assigning “criticality levels” to safety requirements.

Instead, the assurance needed for each safety requirement is reflected in the argument strategy which is chosen to demonstrate satisfaction of this safety requirement, and in the rigour of evidence needed to support this argument. SSEI Task 6 will produce a report which describes precisely how assurance affects these, and how it is determined. Using the terminology of Section D.4, the assurance information which we obtained from the component criticality analysis will then be used to identify COTS evidential outcomes for each safety requirement. That is, the criticality analysis forms part of the assurance-based argument development.

**Selection Criteria and Evidential Outcomes**

By this stage we have obtained safety requirements for the COTS component, via the safety analysis carried out upon the reference COTS component. We have also obtained a measure of the criticality of each safety requirement, and therefore of the required assurance. These two can be combined to form selection and evaluation criteria for selecting the safest COTS component. That is, the component we select must demonstrate satisfaction of the safety requirements with the required confidence (assurance). In order to describe exactly what will be required from the candidate COTS components, we return to the discussion of COTS evidential outcomes in Section D.4.

Section D.4 presents two types of COTS evidential outcomes – those relating to the strategy for demonstrating a claim, and those relating to the rigour of evidence required to demonstrate this claim or its sub-claims. The results of the component criticality analysis provide evidential outcomes of the second type (COTS evidential rigour outcomes). That is, the required assurance for a safety claim is easily seen to directly affect the rigour of the evidence which must be presented to support that claim. Thus, simply from the criticality analysis it is possible to ascribe evidential rigour outcomes to each safety requirement. Examples of these outcomes can be found in Section D.4.3, and include restrictions on test
coverage, demonstration of claims, presentation of historical evidence, and required independence of evidence. The precise evidential outcomes depend on the criticality of this safety requirement, and the circumstances in which it can contribute to a system-level hazard. In each case, the COTS evidential rigour outcomes must be justified by a consideration of the effect on assurance if this rigour were not provided.

In addition to evidential rigour outcomes, it is also useful to identify evidential outcomes of the first type (COTS evidential strategy outcomes). Establishing these will mean it is easier to accurately distinguish the extent to which different candidate COTS components satisfy the relevant safety requirements. In order to establish evidential strategy outcomes, it is necessary to consider the ways in which each safety requirement might be demonstrated. There are often multiple strategies for demonstrating a safety property, and these usually vary in the amount of assurance that they provide. The number and extent of these outcomes will then be dependent upon the assurance needed for the satisfaction of this requirement. That is, the evidential strategy outcomes will be partly determined – just as the evidential rigour outcomes were determined – by the component criticality analysis. The strategy chosen will, again, be dependent on the criticality of this safety requirement and the circumstances under which (or reasons why) it can contribute to a system-level hazard.

To determine whether a particular evidential strategy outcome is required, it is necessary to consider what assurance will be lost by not providing that evidential strategy outcome. If its omission will result in inadequate assurance for the safety requirement, the evidential strategy outcome must be included. Section D.4.2.1 provides more information on how to determine possible evidential strategy outcomes, and further SSEI work on Task 6 will directly address this process.

By this stage, we will now have a number of safety requirements obtained by considering the possible failures of the reference COTS component. Associated with each of these safety requirements will be a number of evidential rigour outcomes and evidential strategy outcomes. These outcomes – for each requirement – are then said to form the safety selection criteria for the COTS component.

D.5.2.3 Evaluation of Candidate COTS Component

During this stage, the candidate COTS components are evaluated against the selection criteria using a contract-based approach. The advantages of using a contractual language to describe this evaluation are threefold. Firstly, the contract serves as an explicit record of the service provision and environment provided and expected by the COTS vendor. Secondly, establishing a contract reduces the likelihood that an inadequate component will be selected due to mismatched assumptions which were not made explicit. Finally, the contract can be used in the safety justification argument to show traceability of the safety requirements.
When using a contract-based approach to assess a particular candidate COTS product, a separate contract document should be prepared for the application side and for this candidate COTS component per each safety requirement. On the application side, the contract states the assumptions made by the application about the operational environment of the COTS product. In addition, the contract also states what the required assurance is, in the form of evidential outcomes (see Section D.4.2) which must be achieved. On the COTS side, for each candidate COTS component, the contract will state the assumptions made by this COTS product about the operational environment it will be placed in. It will state the available evidence (both existing and that which is anticipated to be able to be generated) which can be obtained to show satisfaction of each safety requirement by this COTS product. In this way, the expectations and requirements of the application can be measured against the expectations and capabilities of each COTS component in a formal setting. The provision of such a contract can also clarify the expectations of the acquirer, and can therefore be used even when there is only one feasible COTS candidate.

**Contractual Language**

Before a safety contract can be written, an adequate notation should be established. The notation should be sufficient to identify each safety requirement and the required degree of assurance with which its satisfaction must be demonstrated. In addition, it should be able to unambiguously describe the operational assumptions of the COTS component and target application. Previous work [13] has suggested a tabular approach, but here we propose the use of a language as described in Annex D.A.

This has the advantage of being able to prioritize safety requirements. In addition, the language we propose indicates additional features, such as information which need not be addressed during the COTS selection process but must be articulated in a safety case. The provision of this information may provide guidance as to how the information gathered during COTS evaluation will eventually be used. Finally, this language can be extended to include functional requirements, as well as safety requirements. The functional requirements will typically be those which were used to define the reference component, as well as any additional desirable requirements which were not judged to have sufficient market commonality to be included in the reference component. This formalisation emphasises that both types of requirement should have the same priority, instead of the more usual prioritizing of functional requirements over non-functional.

The language format is described in Annex D.A and briefly summarised below. Most of the language construction is fairly self-explanatory, but we provide some minor discussion in the interests of clarity. Obviously, this can be extended to permit the explicit consideration of functional requirements as well.

Firstly, the language specifies a [Requirement] element. This is a safety requirement which was obtained during establishment of the COTS selection
criteria. It will typically be of the form “HSFM X does not affect functionality Y”. The next element, [Acceptable Likelihood], is an indicator of the target failure rate associated with this particular HSFM. It should be noted that not all failure modes will have a target failure rate associated with them, as this is often flowed down from system safety requirements. This element will be left blank on the COTS side, and if an acceptable likelihood has not been provided as a safety requirement, will also be blank on the application side.

The next two elements, [Severity] and [Degree of Protection] are obtained from performing criticality analysis. These refer to the severity of the hazards to which this HSFM can contribute, and the degree of protection – or existing mitigation strategies – which are already in place. These will typically be filled in on the application side only, and left blank on the COTS side.

The [Evidential Outcomes] element refers in the case of the application side to the results of the assessment described in Section D.5.2.2. This assessment produces a number of different evidential outcomes for each safety requirement. These may relate to the evidence required to demonstrate that the [Acceptable Likelihood] is met. However, if the acceptable likelihood is too low to be adequately shown via testing, then these outcomes will relate to the general integrity of the COTS component, and may refer to the degree of rigour of evidence which is needed to adequately demonstrate sufficient integrity of the software. These outcomes may be further detailed to refer to the sub-goals which must be satisfied (the strategy) to demonstrate satisfaction of this safety requirement. On the COTS side, this element refers to the evidential outcomes which can be satisfied by existing evidence for this COTS product. These may be provided directly by the COTS supplier or inferred from the evidence which is deemed likely to be provided.

The [Assumptions] element provides information about relevant operational assumptions made by the application (on the application side of the contract) or the COTS product (on the COTS side of the contract). These assumptions may include such issues as access to memory and system resources, or modes of data transmission. It may be the case that some operational assumptions are not deemed relevant when considering particular safety requirements. However, this should always be justified. Finally, the [Mitigation] element is used to describe specific mitigation strategies if the evidence provided for the COTS product is not sufficient. This element may be empty where specific mitigation strategies have not been identified.

**Contract Proposal**

Once the contract for each requirement is drawn up, it can be filled in on the application side. On the COTS side, each potential COTS candidate will require a separate contract for each requirement, as obviously the evidence available for each candidate component will be different. At this stage, the contracts for the
potential COTS components are also drawn up and filled in up to the best estimate available.

At this stage, no significant expenditure should be made to obtain evidence for the COTS component. This means that the information that can be filled in on the COTS side of the contract is only a “best guess”, and the cost of ascertaining what information is available should be kept to a minimum. Evidence which is known to be available for that potential candidate – such as a certification pack – can be included in the contract, but at this stage there is no requirement either to pay fees to obtain more existing evidence or to conduct tests and analyses on the component to obtain newly generated evidence. Thus, COTS components which are known to come with evidence to support safety claims, or with a certification pack, are at an advantage. If it is known that certain evidence can be obtained for a cost (either monetary or resources), this can be tentatively included in the contract. However, this stage of the process the aim is to reduce the field of potential candidates by eliminating those for which the application contract seems least likely to be met.

**Contract Verification**

At this stage the remaining COTS candidates are individually assessed more thoroughly. Firstly, any additional functionality offered by a candidate component over that specified in the hypothetical reference component is examined, and safety requirements established. These requirements, if satisfied, will ensure that the additional functionality does not contribute unacceptably to system hazards. These are then added to both the application contract – for the relevant COTS candidate – and the COTS contract for that candidate.

Furthermore, at this stage the evidence that was projected to be available must be obtained. This includes the purchase of any evidence from the COTS vendor which is deemed cost-effective, as well as the further testing and analysis which was proposed. In some cases, it may have been originally believed that only minimal new evidence was needed, only to find at this stage that the evidence from the vendor is untrustworthy or otherwise lacks sufficient rigour. Because of this, COTS candidates may fall short of their contracts in ways which were not anticipated previously. Where this is the case it is possible either to reject this candidate, or to conduct further tests and analysis to attempt to obtain the new evidence. Counter-evidence must also be searched for at this stage.

There will be only a few remaining COTS components – possibly only one – which will progress to the next stage of contract establishment. The candidates remaining will be those components which most nearly fit the terms of the contract.
**Contract Finalisation**

At this stage the mismatches in the contracts between the remaining candidate COTS components and the application are addressed. The selected COTS product will be the one for which these mismatches can be reduced to the smallest magnitude (or in the case of multiple components which match completely, the component for which this match is most cost-effective). Of course, if functional requirements have not been included in the contract in any form, then the selection of the COTS component will also be dependent upon the success with which it meets the functional requirements. Existing organisational techniques can be used to determine this, provided the safety contract match is also given an equally high priority.

A mismatch between a COTS and application contract is defined to be a situation where the evidence supporting the safety claim (on the COTS side) is not sufficient to satisfy the requested evidential outcomes (on the application side). In the previous stage of Contract Verification, all evidence that could reasonably be gathered to support the claim was provided, so there are two possibilities. Firstly, the COTS component does satisfy the claim, but sufficient evidence cannot reasonably be gathered to show this with enough confidence. Secondly, the COTS component may not actually satisfy the claim – hence the lack of evidence to show this.

In either case, mitigation strategies must be used to reduce the effect of the failure of the COTS component. That is, the combined evidence about the mitigation strategies and COTS component must together be sufficient to satisfy the claim with the confidence required. The contractual language provides an element for recording these mitigation strategies, which may already have been identified if SHARD was used to conduct the safety analysis. There are a number of commonly-used mitigation strategies, and it is not the purpose of this document to compare and contrast these. However, we will later present a brief discussion of the most common.

Once a mitigation strategy has been identified for a particular safety claim, it may be necessary to change the Outcomes element on the application side of the contract. This is because some of the required evidential outcomes for this safety claim will now relate to the mitigation strategy instead of simply to the COTS product alone. Most notably, the strategy for satisfying the safety requirement will have changed, which is likely to lead to changed evidential strategy outcomes. Less evidential rigour may also be needed to demonstrate the absence of COTS, resulting in changed evidential rigour outcomes. However, the “total” evidence provided to support this safety requirement (that is, evidence about the mitigation strategy combined with evidence about the absence of the failure) must still be of the same rigour as initially identified. That is, the gap between the evidence available for the COTS component and the evidence required by the evidential rigour outcomes will be made up by the evidence obtained from the mitigating component.
Architectural mitigation strategies can be categorised as those which provide design diversity and those which work by wrapping or annotating outputs. Amongst the first category is n-version programming. This involves using multiple versions of independently-implemented software to perform a required function. Thus, a failure of one component can be detected (via a voting mechanism) and masked by using the functionality from one of the other n-1 components. Because these are all independently implemented, common-cause failures are unlikely to occur. The multiple components may all be COTS products, or may also include bespoke components.

A second type of architectural mitigation is the use of controller-monitor components. Here, the monitor component implements a very simple version of the COTS functionality, while the COTS component itself acts as the controller. If the output of both components agrees to some determined equivalence, then the COTS output is accepted as correct. If not, then either the application will accept the output from the monitor component, or may perhaps accept no output at all. The monitor component in this case is likely to be bespoke, in order to ensure that it is providing correct – though degraded – functionality. Finally, a third common use of architectural mitigation strategies to provide design diversity is by providing recovery blocks. These are multiple independently-implemented versions of the component in question which are executed sequentially. If all results are equivalent – within a certain degree – then the executed functionality is deemed accurate.

The other major type of architectural mitigation is the use of wrappers. Wrappers are – usually bespoke – components which are inserted between the COTS component and the rest of the application. Wrappers are designed to ensure that the COTS component can only communicate with the application via known channels. Wrapping may also help to isolate the COTS component from resource competition, and to ensure that a consistent interface is presented to the application even when the COTS component is upgraded. Wrapping is sometimes referred to as isolation, and is not a specific fault tolerance technique.

In addition to architectural mitigation strategies, there are also system-level mitigation strategies that can be used. Some of these relate to human factors (such as providing personnel to monitor failures), while others may rely on interactions between hardware and software. The presence of these system-level mitigation strategies must be noted in project documentation if they are included as part of the COTS contract. Furthermore, their effect on the COTS component – that is, the degree of protection which they are likely to provide – must be directly assessed.

D.5.2.4 Justification of COTS Component Use

Once a COTS component has been selected, and all required evidence gathered, a safety justification for the use of this component must be presented. This will form part of the final safety case, and should consist of a compelling argument
that the COTS component does not contribute unacceptably to system hazards. The safety justification should be sufficient to show that the safety requirements identified for the COTS component are valid, sufficient, satisfied and traceable.

There are multiple ways to present a safety justification, but one of the most accessible is by the use of GSN [21]. However, DS 00-56 does not mandate a particular technique to be used.
D.6 CONCLUSIONS

Using COTS components in many systems can be an advantageous choice. However, assuring the safety of these components can be problematic, as evidence is typically less easily available than for bespoke systems. In order to assure their safety adequately, three types of evidence will typically be required: evidence available from the COTS vendor, in-service evidence, and newly-generated evidence.

In Section D.3.1 we examined the types of evidence which are typically available for COTS components, and discussed the salient features of each type. Section D.3.1.1 defined four common sources of evidence: certification packs, evidence otherwise available from the COTS vendor, third-party testimonials and existing safety cases. Section D.3.1.2 then discussed the advantages and disadvantages of in-service evidence. The advantages include fewer assumptions made during the evidence-gathering, and the ability to show exactly what effect a failure of the component will have on the rest of the system.

The disadvantages of in-service evidence are disadvantages which it holds in common with most existing evidence – the possibility of “usage mismatches”. A usage mismatch occurs when the circumstances in which the evidence was generated are insufficiently similar to the circumstances of proposed usage as to render some of the evidence inapplicable. In Section D.3.1.3 we presented guidance for conducting gap analysis based on six categories of usage mismatch. This gap analysis will identify where existing evidence is insufficient due to the circumstances under which it was generated not being “similar enough” to the target circumstances. Furthermore, in-service evidence is vulnerable to failures of reporting, as discussed in Section D.3.1.2.

Section D.3.2 discussed the generation of new evidence to assure the safety of the COTS component, where existing evidence is insufficient in extent or rigour. We also identified a need to validate the evidence supplied from the COTS vendor, where this may not be considered trustworthy. Questions of counter-evidence were also discussed, and we noted that the presence of counter-evidence may serve to diminish the trustworthiness of the COTS vendor (provided it can be reasonably assumed that it should have been found during their product examination).

Section D.4 introduced the concept of justifiable assurance deficits and presented some issues which arise when determining how much evidence is enough to adequately assure the system. In this section, we have characterised the five different types of direct evidence introduced in DS 00-56 according to the advantages which each offers. We also discussed the potential disadvantages of each, and the likely availability when considering a COTS product.
Section D.4.2 and D.4.3 then presented an introduction to the concept of “evidential outcomes”, being the outcomes – for each safety requirement – that would need to be demonstrated to result in a given assurance for that requirement. These evidential outcomes refer to the strategy for showing the satisfaction for a requirement, as well as the rigour of the evidence which must be provided to support the argument.

Finally, Section D.5 described a method by which safety requirements can be considered when evaluating and selecting a COTS product. This method provides a contractual language for stating the safety requirements, and for comparing the degree to which each candidate COTS product can be said to satisfy these. Additionally, architectural mitigation strategies were discussed to ensure that sufficient total evidence is provided to assure the chosen COTS product to the degree necessary.

Annex D.A comprises the contractual language discussed in Section D.5 for evaluating and selecting COTS components. Annex D.B contains a brief overview of a project which makes use of COTS components. In it we show how the current perspective on COTS makes their use within a project limited to non-safety-related purposes. We also provide references to the text where guidance can be found for each issue. Finally, Annex D.C contains a discussion of the review that was performed on the gap analysis techniques discussed in this paper.

D.7 Recommendations

It is our recommendation that the findings from this work be assessed further as required. If validated successfully, we recommend that they are then incorporated into the Statement of Best Practice for DS 00-56, scheduled March 2009.

D.8 Acknowledgements

We would like to thank Thales and QinetiQ for their assistance in providing case studies to validate this work. We would also like to thank the UK MOD for their support and funding for this task.

D.9 LIST OF REFERENCES

[1] Adelard: Justifying the use of software of uncertain pedigree (SOUP) in safety-related applications


**D.10 DEFINITIONS**

Assurance: the amount of confidence in the safety of a system

Assurance Deficit: the difference between the assurance which has been obtained and the assurance which is required

Evidential Outcomes: characteristics of the evidence or properties of the argument which must be demonstrated in order to support a particular claim

Evidential Rigour Outcomes: evidential outcomes which relate to the rigour of the evidence

Evidential Strategy Outcomes: evidential outcomes which relate to the strategy chosen to satisfy a particular requirement
D.11 List of Abbreviations

COTS: Commercial Off-The-Shelf Software
HSFM: Hazardous Software Failure Mode
SOUP: Software of Unknown Pedigree

D.12 Annexes
Annex D.A  A Contractual Language for COTS Safety Requirements

[Requirement]

The software safety requirement which must be met (for example, “Failure mode X of this component shall not occur”)

[Acceptable Likelihood: rate of occurrence]

Acceptable occurrence likelihood, or target failure rate. Numerical failure rates might not be provided for all software safety requirements. In cases where the acceptable likelihood is too low to be shown through testing, the argument must instead appeal to the integrity of the software. Requirements for this can be detailed in the [Evidential Outcomes] element

[Severity: catastrophic | major | … [project defined]]

The severity of the hazard contributed to by this HSFM

[Degree of protection]

This reflects whether there is any external mitigation for the hazard.

[Evidential Outcomes]

A description of the evidential outcomes which should be demonstrated to show satisfaction of this safety requirement. These may take one of two forms. A proposed GSN argument can be used to record the evidential strategy outcomes (by showing the sub-claims beneath this safety requirement). Evidential rigour requirements can be shown textually, or alternatively using GSN. Evidential rigour outcomes must relate either to showing the acceptable likelihood is met (if possible) or to the general integrity of the software if acceptable likelihood is too low for this to be possible

[Assumptions]

Any contextual assumptions about the operation of the COTS product
[Mitigations]

Any mitigation strategies identified for this requirement. This element may be empty unless specific mitigation strategies are identified early.
Annex D.B  Watchkeeper: An Illustration

The information in this section is provided by Thales, with reference to the Watchkeeper project [17]. It is intended to illustrate some of the ways in which COTS components are currently used, as well as some of the issues associated with their use. Throughout this section we will provide references to the rest of this working paper, in order to demonstrate how current industrial thinking has provided the motivation for much of this guidance. Much of the information in this section has been provided in response to surveys carried out for the purpose of validating this working paper.

D.B.1 Watchkeeper Introduction

Watchkeeper is the development of an Unmanned Air Vehicle system for the UK MoD. It is a multinational project and is intended to comply with DS 00-56 Issue 3. However, where individual suppliers are concerned, their processes and products may also be legally required to comply with other standards. Watchkeeper includes both airborne equipment (air vehicles and payloads) and ground control systems that provide facilities which are partly equivalent to cockpit and mission planning systems for a manned aircraft.

COTS items are extensively used throughout Watchkeeper in two categories: tools and functional components. The first category consists of those tools which have been purchased off the shelf and used in the development of safety-related equipment including software. These include software items such as high-level language compilers, UML design modelling environments, code generators and programmable logic simulators. The second category consists of those components which have been purchased off the shelf and included within the delivered safety-related equipment.

In more detail, some of the airborne components and most of the ground control components make use of off-the-shelf processing platforms. The two most common are Microsoft Windows and Vx-Works. Additionally, most sub-systems include large amounts of software developed in a commercially-available high level language (C, C++, Ada etc.). The compilers and linkers for these languages are considered tools (and so placed in the first category, above). The COTS libraries and software components required to use these languages (for example, maths libraries or board support facilities), however, are considered as functional COTS components themselves.

D.B.2  Watchkeeper and COTS Safety Analysis

In this section we discuss the general principles of COTS safety analysis as implemented on Watchkeeper. This safety analysis is informed and motivated by the necessity of adequately demonstrating the safety of all COTS components on the project.
D.B.2.1 Watchkeeper and Existing Evidence

In practice, it is generally considered on the Watchkeeper project that COTS components do not have credible claims for integrity. That is, they cannot be depended upon to perform (or not perform) safety-related or safety-critical functions. Consequently, all safety evidence relating to a COTS component has to be generated by the procurer, a process which can result in both inadequate evidence and a significant increase in cost. It is this perspective that has motivated the guidance contained in Section D.3, where we attempt to qualify the degree of trustworthiness of a vendor and of supplied evidence. The function of this guidance is to ensure that if supplied evidence is trusted, this is for supportable and justified reasons. We also present guidance on some criteria which, if satisfied, may increase the trust that can justifiably be placed in existing evidence. This, in turn, can reduce project costs and ensure that a safety argument is more robustly supported.

Because of the significance of this position taken at Watchkeeper – that COTS components cannot be depended upon in safety-related roles – we have also provided additional validation guidance. Section D.3.2.5 discusses validation activities that may be performed to determine whether the results claimed by the COTS supplier are correct. It is the position of this working paper that – where evidence is available – the trustworthiness of this evidence can be to some degree estimated by performing validation. Should the validation uncover readily-available counter-evidence, this will decrease the trustworthiness with which the rest of the supplied evidence is regarded. However, should the validation of a random sample of supplied evidence concur with the COTS supplier’s claims, this is an indication that the supplier may be more trustworthy than assumed.

Discussion of the Watchkeeper project also identifies a significant difference between safety arguments relating to those COTS components which are purchased with a certification pack (e.g. WindRiver for VxWorks is supplied with a DO-178B certification pack) and those which are not (e.g. Microsoft Windows). For those products which come with a certification pack, some analysis must be performed to determine what – if anything – this implies about the compliance of this product with DS 00-56 [24]. In general, these packs are regarded on the Watchkeeper project as sufficient evidence for the certifying authority to accept as proof that the COTS component has been developed to the relevant standard. That is, the evidence is felt to be sufficiently trustworthy that – for example – test and analysis results included in the pack are considered acceptable. Section D.3.1 expands on this topic, identifying the presence of a certification pack to be a point in favour of the trustworthiness of the vendor.

One of the issues with accepting a certification pack as proof of the adequacy of a COTS component as regards safety is the specific usage which is proposed. If the COTS component is not used as specified, the evidence in the certification pack becomes irrelevant. Experience on Watchkeeper has identified that the responsibility to use a COTS component only as specified is not always
adequately appreciated by procurers. This has informed the guidance we provide in Section D.3.1.3 for conducting a gap analysis of the target and proposed environments.

In addition, the use of third-party testimonials to the safety of COTS software is accepted to a limited degree on Watchkeeper. One of the primary concerns is that the third party’s requirement for the COTS component will not be adequately similar to the proposed usage on Watchkeeper. This, again, has contributed to the guidance in Section D.3.1.3 to determine the extent to which the usage can be considered relevant – and therefore the extent to which the testimonial is considered valuable. Third-party test results are also felt to be useful in showing the absence of failures. However, if they are intended to support a safety argument with any confidence, they must show what testing was done, how the testing was done, any standards that were complied with, and the version of the software that was tested. This experience has informed and motivated the guidance given in Section D.3.1, discussing the use of third-party evidence.

D.B.2.2 Watchkeeper and In-service Evidence

As Thales contributors to Watchkeeper place the minimum of trust in COTS products, the use of in-service evidence is not accepted on this project. The reasons for this include the following:

- A significant amount of testing or usage of the COTS component would be needed to support any safety argument
- Analysing the environments and circumstances in which the in-service evidence would need to show that the component did not fail would be complex and time-consuming
- It was considered both less expensive and more defensible (from a safety perspective) to simply assume that the COTS component did fail – unless explicitly shown in tests not to – and consequently build mitigations into the architecture. Consequently, in-service evidence could be of no value.

These points are all valid, and the position taken on the Watchkeeper project is to err on the side of safety, as required by DS 00-56. We discuss the extent to which in-service evidence can be used in Section D.3.1.2, and note that the examples identified by Watchkeeper are common to the use of in-service evidence on all projects.

D.B.2.3 Watchkeeper and New Evidence

The Watchkeeper project relies on newly-generated evidence to demonstrate safety of almost all the COTS products used. This is due to the position described earlier, that little or no trust is to be placed in the safety of COTS components, or the trustworthiness of existing evidence.
Furthermore, experience on Watchkeeper is that COTS suppliers will generally refuse to provide enough evidence – let alone enough trustworthy evidence – to demonstrate adequate satisfaction of the safety requirements. In general, prevailing industry opinion is that a COTS vendor – for a price – will provide additional evidence. However, as discussed earlier, the quantity and quality of this evidence may not always be that required. Additionally, the timeliness and price of delivery may also prohibit this as a practical solution. As we have discussed earlier, the trustworthiness of evidence is a primary consideration. It is possible that the reluctance to supply evidence which has been encountered on this specific project has contributed to the adoption of a policy that little or no trust is to be placed in the safety of COTS components. We reflect this in Section D.3.1.4, when discussing factors which can lower the trustworthiness of a vendor or of existing evidence. For Watchkeeper, the policy of relying on newly-generated evidence to demonstrate safety of COTS products means that the trustworthiness of the COTS vendor and evidence is less important than otherwise. This does not mean that the suppliers chosen are untrustworthy, simply that vendor-supplied evidence is not used, and so an assessment of its trustworthiness is not explicitly carried out. It may also be the case that lifecycle evidence is simply not available, as is often the case with legacy software. Section D.3.2 discusses the use of reverse engineering to attempt to provide this information.

Because any evidence supplied as to the safety of COTS components is not generally considered admissible on Watchkeeper, most of the safety evidence is newly-generated. For example, software written in both C++ and Ada is required by Watchkeeper to perform safety-related calculations. Consequently mathematical functions and libraries are also required, and are obtained via the use of COTS components. However, the trustworthiness of the COTS functions is not assumed. The functions are tested to confirm that correct answers are returned appropriately for each call, taking account of the circumstances under which function calls can be made. The correctness of these test results is then validated by comparison with results from a different mathematical tool.

We have attempted to reflect this need for generating new evidence by the guidance provided in Section D.3.2. This section discusses the different types of testing or analysis which may be performed. Additionally, it also provides guidance on validating existing evidence, as demonstrated by the checking of function calls on Watchkeeper software.

D.B.2.4 Watchkeeper COTS Assurance Methods

In general, all Thales contributors to Watchkeeper implement their own methods of selecting, evaluating and assuring COTS components. The only requirement is that all contributors should meet a common standard of assurance. The most common method for contributors to achieve this is to use the following generic process (subject to customization as required) for evaluating and assuring COTS components.
Firstly, the hazardous failure modes (HSFMs) of the COTS component are identified. These are the ways in which the COTS component may fail to satisfy high-level safety requirements. Once these have been identified, one of two approaches is taken. The project personnel may either:

1. introduce derived safety requirements relating to the system architecture to defend against the consequences of such failures; and/or

2. carry out adequate testing of the COTS item (most likely in an integrated environment) to show the absence of these hazardous failure modes.

When selecting a COTS component for use on Watchkeeper, the presence of existing evidence as to its safety is not considered as a point in its favour. However, third-party testimonials as to the safety of a particular component have been considered.

The guidance we provide for selection and evaluation of a COTS component proposes an evaluation method which is not dissimilar to that used on Watchkeeper. Section D.5 discusses how evidence as to the safety of a COTS component can either relate directly to the component itself, or to the architectural mitigations. We also present a method for selecting and evaluating components which takes into account the presence of existing trustworthy evidence of the safety of candidate components. However, if this evidence is not acceptable (as on Watchkeeper), the method we propose simplifies to the common approach discussed here. Because there are very few detailed standard approaches to COTS selection—as demonstrated by the different approaches taken by Watchkeeper contributors—Section D.5 presents generic guidance rather than specific.

**D.B.3 Watchkeeper Recommendations**

In addition to providing the information above, Watchkeeper personnel have also expressed an interest in specific topics to be included in guidance for compliance with DS 00-56. These are discussed below, with references to those sections in this working paper where the relevant guidance can be found.

Firstly, there is a need for guidance or discussion on the use of gap analyses. In particular, the following comment was made: “A technical gap analysis carried out either by the procurer or by a suitably qualified independent analyst to report on what it has achieved in comparison with criteria the procurer states may be very useful”. Consequently, we have provided guidance for conducting this type of gap analysis in Sections D.4 and D.5. Section D.4 discusses how the procurer should set criteria, specifically those relating to adequacy of evidence. Section D.5 then demonstrates how a candidate COTS component may be compared – technically – against those criteria. In addition, Section D.5 describes ways in which the gaps thus identified can be addressed.
In addition, guidance on determining the trustworthiness of COTS components was requested. The Watchkeeper project in particular maintains a policy towards COTS components which assumes their inherent lack of safety properties. While this is certainly the most defensible safety-related position to take, there may be significant benefits in adopting a more trusting and flexible approach to COTS products. Section D.3 describes how this may be done without compromising the required standard of assurance on projects.

Finally, one of the strongest requests pertained to numerical safety requirements. Just as for bespoke products, assigning target numerical failure rates to COTS components is common, yet demonstrating any but the most lax of these is very difficult. The types of examination which can be performed on COTS components – for the most part, testing – cannot possibly demonstrate satisfaction of these requirements. Thus, it was strongly felt by all respondents that these types of safety requirements should be replaced by qualitative safety requirements. Accordingly, we have used qualitative safety requirements throughout this working paper, and intend to include a discussion on this point in the eventual Statement of Best Practice.
Annex D.C  An Evaluation of Evidence Mismatch Gap Analysis

The evidence usage mismatches of Section D.3.2 were evaluated by Thales, in order to determine their sufficiency and correctness. Owing to timing issues, these usage mismatches could not be validated against an existing project. However, the survey respondent was able to draw on varied experiences to assess the mismatch categories.

Most of the projects on which the respondent had worked were SIL 1 or SIL 2, with a few being considered High Integrity. Consequently, the validity of these evidence usage mismatch categories when producing a safety-critical system is unknown. Current experience suggests that in high integrity and safety-critical systems very little weight is placed in such circumstances on evidence that has not been directly generated by the purchaser, so the likelihood of these categories applying in such a situation is low.

The respondent had also worked on a wide variety of projects, including mission planning, situational awareness on the battlefield, communications and safety and security of encrypted systems. These projects had used COTS products extensively, often in a similar manner to Watchkeeper, as discussed in Annex I.B. Consequently, while not applied to a specific single project, the evidence usage mismatches (and associated gap analysis recommendations) have been evaluated for utility against a wide variety of possible situations.

The respondent’s perception of the evidence usage mismatches and the associated guidance on gap analysis was that these were comprehensive, sufficient and useful. One minor point noted was the necessity for ordering these categories to indicate the severity of their associated usage mismatch. Accordingly, we provided an ordering as described in Section D.3.2. This ordering was based on the respondent’s experience and further analysis of the consequences of each type of mismatch.

The necessity of providing all these evidence usage mismatches was another topic on which we sought input from Thales. The consensus was that guidance for all the categories was necessary, although the degree to which each was relevant would vary from system to system. The respondent was also of the opinion that the guide questions – while not completely orthogonal between categories – did not suffer from too much repetition.

The other point which was recommended by the respondent was the need for further detail on the different levels of assurance provided by different types of evidence. Some discussion of evidence types has been provided in Section D.4, but this was not included in the survey given to the respondent. Consequently, our overall recommendation for this working paper is that it be considered as a whole rather than several discrete parts. That is, this experience shows that the recommendations and guidance can only be understood and validated in the context of the subsequent discussions. This finding will inform the construction of
eventual software guidance for DS 00-56. In addition, later deliverables for SSEI Task 6 concentrate on a discussion and analysis of evidence and assurance. We recommend that the findings of this task be incorporated into the eventual guidance to aid in conducting gap analysis and software assessment.

Finally, the overall opinion of the respondent was that this guidance on gap analysis was useful and, notwithstanding these additional points, should certainly be provided. While the respondent was unable to state whether this would have caused a different COTS product to be chosen for any specific project upon which he had worked, he noted that such guidance would almost certainly have given him greater confidence in the various COTS products which were eventually selected.
Annex D.D  An Evaluation of Evidence Mismatch Gap Analysis

IBM also undertook a survey in order to determine the sufficiency and correctness of the gap analysis categories of Section D.3.2. The raw survey data has been included here (with any identifying comments removed).

INITIAL RESPONSE TO COTS SURVEY 2

PREPARED BY IBM UK LTD

Question 1

*If possible, please provide a brief (1-paragraph) summary of your project. There is no need to provide commercial information, or any details which you feel would compromise the confidentiality of your work. However, it would be useful to know the general purpose of the system, and the role of the COTS component within it.*

The project selected is the replacement national air defence command and control system. The system delivered under the project provides ‘decision aid’ information to air defence operators conducting the air surveillance, policing and defence of the UK. System functions include supporting operators to guide aircraft into tactically advantageous intercept positions. Numerous COTS components are used, ranging from hardware (e.g. servers, workstations) and software (e.g. database), through to a major subsystem (albeit with some development).

Question 2

*Can you briefly describe how the existing evidence for COTS products in your system was evaluated?*

For each COTS product selected, suppliers were asked to fill in a safety related questionnaire (COTS Evidence Profile). Suppliers were generally content to answer the questions although the detail provided varied considerably. Much of the evidence provided was difficult to verify and was taken on trust. However, the information was used primarily as a general indication as to the quality, stability and maturity of the product, and hence the degree if testing to be applied. Statistics were also gathered on component failures during testing and in-service use. However, these were not directly compared against the original evidence provided for the product.

Question 3

*Please read the attached guidance and consider its applicability to your project. Do you think these categories are sufficient to evaluate the existing evidence for the COTS products in your system? If not, can you provide examples where*
existing evidence is inapplicable for a reason not included in these categories? (Again, no commercial or sensitive details are requested).

We suggest that the definition of ‘Adequacy’ be expanded to explicitly mention that there is a quantitative aspect to the provision of evidence. In other words, one piece of evidence – although it may be adequate in every other respect mentioned by the checklist – may not be credible solely because it is only a single piece of evidence.

**Question 4**

*Do you consider all these categories are necessary for evaluating existing evidence for COTS components in your system? If not, which categories do you consider to be the less useful ones?*

All categories would be used.

**Question 5**

*If you had performed gap analysis – using this guidance – of existing evidence for COTS components in your system, would this have given you extra information during the COTS selection and evaluation phase? Additionally, would it have given you more confidence in the existing evidence or in the vendor?*

Generally not. COTS vendors were generally not in a position to provide information on Degree of Protection; Consequences of Failure; and Resource Availability. Such information could probably only be supplied by the user of the product, and commercial sensitivities will often preclude such detailed information being provided.

**Question 6**

*If your answer to either part of 4) was affirmative, how would this extra information have affected your project? Would you have selected a different COTS product for use? Would it have given you greater confidence in the COTS product you did select?*

Unsure if we would have selected a different product. Better information would certainly have resulted in greater confidence in the COTS products selected, which may have reduced the amount of testing performed on the product in the target environment.

**Question 7**

*Do you find the sample questions in each category useful? Are they clear enough in establishing the intent of the category?*

Yes.
Question 8

Do you have any further comments on the guidance? Is there anything else that you would like to see included in this?

See Question 3. The questions generally focus on the provision of quality data, seemingly anticipating that very detailed information will potentially be available. In practice we have generally found that this is not the case – only high level data is available.

In contrast, some weighting should be given to the quantity of evidence (perhaps in the Adequacy question), even if this is at a higher level of detail. For example, if a COTS product has a wide user base and a good reputation in a number of deployments that superficially appear similar (without detailed information on eg Degree of Protection; Consequences of Failure; and Resource Availability being available), this still increases confidence.

If one COTS product had a single, but detailed set of evidence available, and its competitor had a much higher level of evidence detail available but supplied from numerous customers, which would be the lower risk product?
Annex D.E  IBM Evaluation of Evidence Mismatch Gap Analysis

This annex contains the results of a case study conducted by IBM to validate the findings of the SSEI research into COTS and Legacy software [4]. This case study is an extension of the initial survey, results of which are presented in Annex D.D. The recommendations of this case study have been incorporated into the gap analysis technique proposed, and will be reflected in future updates to the SoBP.

All references are to local sections within Annex D.E, and citations are to the list of references contained within the annex as shown by the Table of Contents on the following page.
# TABLE OF CONTENTS

Table of Contents.................................................................................................................. 338

1 **Introduction**.................................................................................................................. 341
   1.1 Background...................................................................................................................... 341
   1.2 Purpose of this Document.............................................................................................. 341
   1.3 Classification.................................................................................................................. 341
   1.4 Structure of this Document ............................................................................................ 342

2 **The UK ASACS Command and Control System (UCCS)** ........................................... 343
   2.1 Context and Purpose of the System .............................................................................. 343
     2.1.1 Context - UK ASACS............................................................................................. 343
     2.1.2 Purpose of the UCCS............................................................................................ 344
     2.1.3 UCCS Requirements Framework ......................................................................... 345
   2.2 UCCS System Description.............................................................................................. 346
   2.3 Safety Framework and Requirements.......................................................................... 348
     2.3.1 UCCS Safety Framework....................................................................................... 348
     2.3.2 UCCS Safety Cases............................................................................................... 349
     2.3.3 Defence Standards (DEF STANs)......................................................................... 350

3 **Use of COTS within UCCS**........................................................................................... 351
   3.1 Design Drivers .............................................................................................................. 351
   3.2 MOD Requirements Governing COTS Usage in UCCS.............................................. 352
   3.3 Overview of COTS Software Products within UCCS............................................... 353
     3.3.1 Introduction............................................................................................................. 353
     3.3.2 How They were Selected – General Principles......................................................... 354
   3.4 Relationships with COTS Suppliers............................................................................ 355

4 **COTS Items for Case Study – Description, Selection and Supplier Evidence Overview** .................................................................................................................. 356
   4.1 Voice Communications Subsystem (VCS).................................................................... 356
     4.1.1 Role within the System............................................................................................ 356
     4.1.2 Selection Process..................................................................................................... 356
     4.1.3 Supplier Relationship.............................................................................................. 357
     4.1.4 Supplier Evidence Provided..................................................................................... 357
     4.1.5 Conclusion............................................................................................................... 358
   4.2 Microsoft Windows 2000.............................................................................................. 359
     4.2.1 Role within the System............................................................................................ 359
     4.2.2 Selection Process..................................................................................................... 359
4.2.3 Supplier Relationship.................................................................360
4.2.4 Supplier Evidence Provided..................................................360
4.2.5 Conclusion .............................................................................360
4.3 Map Processing Software Toolset .............................................360
  4.3.1 Role within the System.........................................................360
  4.3.2 Selection Process.................................................................361
  4.3.3 Supplier Relationship..........................................................361
  4.3.4 Supplier Evidence Provided.................................................361
  4.3.5 Conclusion .............................................................................361
4.4 Database Frontend Application ................................................362
  4.4.1 Role within the System.........................................................362
  4.4.2 Selection Process.................................................................362
  4.4.3 Supplier Relationship..........................................................362
  4.4.4 Supplier Evidence Provided.................................................362
  4.4.5 Conclusion .............................................................................363
5 Supplier Evidence and Gap Analysis ..........................................364
  5.1 Introduction .............................................................................364
  5.2 Assessment of Evidence Provided Using Working Paper Definitions ..366
    5.2.1 Certification Packs............................................................366
    5.2.2 Further Supplier Tests and Analyses ..................................367
    5.2.3 Third Party Evidence .......................................................369
    5.2.4 Pre-Existing Safety Cases ...............................................369
    5.2.5 In-Service Evidence .......................................................369
  5.3 IBM Examination of Supplier Evidence ......................................371
  5.4 Use of the Proposed Gap Analysis ............................................372
    5.4.1 Introduction .....................................................................372
6 IBM Generated Evidence ..........................................................382
  6.1 Introduction .............................................................................382
  6.2 VCS ..........................................................................................382
    6.2.1 Introduction .....................................................................382
    6.2.2 VCS Software Safety Case Claims ....................................382
  6.3 Windows 2000; Map Processing Software Toolset; Database Frontend Application ..................................................383
    6.3.1 Introduction .....................................................................383
    6.3.2 Microsoft Windows 2000 ..................................................384
    6.3.3 Map Processing Software Toolset ....................................385
    6.3.4 Database Frontend Application ....................................385
  6.4 User Testing .............................................................................386
    6.4.1 Introduction .....................................................................386
    6.4.2 Evidence ............................................................................386
6.5 Other Aspects .................................................................................................................. 386

7 Evaluation of Possible Process for Selecting a COTS Software Component
........................................................................................................................................ 388

7.1 Summary of IBM Selection Process for COTS Software Products in UCMP388

7.2 Evaluation of Working Paper Section 5 – A Process for Selecting a COTS Component .................................................................................................................. 389

7.2.1 Introduction .................................................................................................................. 389

7.2.2 Comments .................................................................................................................. 389

8 List of References ............................................................................................................. 392

9 List of Abbreviations ....................................................................................................... 392

ANNEX A UCMP COTS Evidence Profile Proforma .......................................................... 395
INTRODUCTION

1.1 Background

The Software Systems Engineering Initiative (SSEI) is a MoD funded strategic initiative intended to enhance through life capability management for software intensive defence systems, and thus to reduce risks, delays and cost overruns. The emphasis of the SSEI activities is on those systems which contain high integrity or safety critical software.

QinetiQ with support from the University of York are undertaking SSEI Task 11, “Software Guidance for DS 00-56 Issue 4”. The intent of this task is to provide pragmatic guidance for system suppliers on producing software safety evidence in conformance with DS 00-56 Issue 4. IBM is acting as a sub-contractor to QinetiQ to provide industry input and validation during the formulation and definition of Statements of Best Practice in support of this guidance.

The University of York as a deliverable of Task 11 have prepared the report “Working paper on Evidential Approach, Addressing Legacy and COTS Software” (Ref [1]). IBM has been tasked with providing input to this report in the form of a case study of a relevant UK project. This case study both provides input to and validates any approaches or techniques described in the report giving the report a sound practical basis.

1.2 Purpose of this Document

This report constitutes the case study described above. The overall aim of this case study, as described in a clarifying memo (Ref [2]), is to determine the sufficiency, or otherwise, of the methods proposed in the working paper (Ref [1]). The case study has been generally structured in accordance with the clarifying memo (Ref [2]) to achieve this.

1.3 Classification

This case study is UNCLASSIFIED. However, the subject of the case study, the UK ASACS Command and Control System (UCCS), contains elements that are protectively marked UK RESTRICTED and above. To produce an unclassified document it has been necessary to sanitise these elements, or where they are of a higher protective marking, to avoid reference to them altogether. This should be taken into account when reading this document.
1.4 Structure of this Document

This document is structured as follows:

- Section One is this Introduction;
- Section Two provides a brief description of the UCCS and its safety context;
- Section Three describes, in general terms, the use of COTS within the UCCS;
- Section Four describes in more detail the subset of UCCS COTS software components examined in this Case Study;
- Section Five discusses in more detail the guidance for conducting gap analysis presented in Section 3.1 of the Working Paper, Ref [1];
- Section Six summarises the evidence generated by IBM for the COTS components identified in Section Four;
- Section Seven provides an evaluation of a possible selection mechanism (described in Section 5 of the Working Paper, Ref [1]);
- Annex A provides a blank COTS Evidence Profile proforma, as used by the UCMP safety team.
2 THE UK ASACS COMMAND AND CONTROL SYSTEM (UCCS)

2.1 Context and Purpose of the System

2.1.1 Context - UK ASACS

The UK Air Surveillance and Control System (UK ASACS) is the UK’s ground based component of the NATO Integrated Air Defence System (NATINADS). The UK ASACS provides early warning information to UK and NATO commanders on the air situation in the UK’s area of responsibility, and enables commanders to initiate an appropriate response to any incursion of UK airspace in peace, crisis and war. The UK ASACS is actually a ‘system of systems’, comprising of a range of systems that gather, process and disseminate information on all aircraft within the UK’s area of responsibility, and specific maritime contacts. It comprises of radar sensors, voice and data communications, command and control and battle management entities.

The static command and control and battle management elements are provided by an IBM UK system, the UK ASACS Command and Control System (UCCS). The first sites became operational in 2004, with another major site becoming operational in 2006. The system operates on a 24/7/365 basis, and is supported in service by IBM UK.

Figure 1 on the next page provides a high level depiction of the ASACS system context, and the position of the UCCS in it.
2.1.2 Purpose of the UCCS

The operational tasks that the UCCS must fulfil can be summarised as:

- Early Warning;
- Air Policing;
- Operational Training.

Early Warning comprises of the detection of air vehicles representing a potential threat or of special interest. This is achieved by the production and interpretation of a Surface and Recognised Air Picture (SRAP) for the UK’s area of responsibility, and by the production of an air and surface track picture within a larger display area. The Early Warning mission also requires the exchange of Recognised Air Picture (RAP) data and management messages with adjacent NATO Air Defence Ground Environment (NADGE) and Systeme de Transmission et de Representation des Information de Defense Arienne (STRIDA) sites.

Air Policing includes supporting appropriate response measures to air vehicles detected by the Early Warning mission. This mission is supported by the SRAP, but also requires the provision of battle management facilities and ground-to-air
communications facilities. It is of interest from a safety perspective that ‘appropriate response measures’ include the interception of air vehicles by fighter aircraft, with the fighters under the control of RAF operators using the UCCS.

Operational Training is required for both the above missions and for UK forces prior to deployment to operational theatres. This mission requires the provision of simulation facilities to support both individual and team training. Live intercept missions are also continually practised using a variety of scenarios. These missions generally occur in airspace being used by civilian traffic.

### 2.1.3 UCCS Requirements Framework

The UCCS programme was provided through the United Kingdom Air Defence Capability Maintenance Programme (UCMP). It was intended that the system delivered under UCMP would be an interim system, bridging the ‘capability gap’ between the demise of the existing system and the arrival of a new system some years in the future. Under the ethos of ‘capability maintenance’, there were no new major functional requirements and any increases in operational effectiveness were anticipated to be the by-product of the use of modern design and development techniques rather than a design goal\(^1\).

The competition to win UCMP had also been conducted against a backdrop of a publicly declared budget, and a clear MOD expectation that COTS products would be extensively used in order to stay within this budget.

The Demonstration and Manufacture phase of the UCCS programme commenced at the end of 2001, with a contract let against a comprehensive requirements documentation set.

The documentation set consisted of:

- A System Requirements Document (SRD) containing approximately 900 mostly functional requirements;
- A Statement of Work (SOW) containing over 500 non-functional requirements (including safety requirements);
- System Security Policies, containing approximately 150 functional and non-functional security requirements;

\(^1\) However, it should be borne in mind that the system that the UCCS replaced – and from which the requirements were derived – used bespoke software applications throughout along with significant elements of bespoke hardware.
An Integrated Logistics Support Plan containing over 200 non-functional requirements regarding the reliability, availability, maintainability, testability and support of the system;

A Safety Case Part 1 containing over 150 mostly non-functional safety requirements (see Section 2.3);

A Works Services Brief, containing approximately 300 functional and non-functional requirements for aspects such as lighting, air conditioning, and civil works activities;

A Contract, imposing budget, schedule, and terms and conditions.

Of relevance to this case study is the fact that the Statement of Work contained several requirements expressly addressing the suitability of COTS software. These are discussed in Section 3 (Use of COTS within UCCS).

### 2.2 UCCS System Description

The main system elements are:

- Two Control Centres, each with a significant number of operator positions.
- One Executive Centre which is an overview centre responsible for command of operations.
- Additionally, UCCS equipment is positioned at a significant number of other sites to provide system access to radars and radios.

At a Control Centre, the UCCS consists of a Command and Control Subsystem (CCS) and a Voice Communications Subsystem (VCS), supported by a communications infrastructure provided as Government Furnished Equipment (GFE).

The Command and Control Subsystem (CCS) produces the Surface and Recognised Air Picture (SRAP) using local and external data, and provides the facilities for weapons and battle management.

The Voice Communications System provides secure and insecure ground-to-ground and ground-to-air communications.

The GFE communications infrastructure provides the connectivity between the major and minor UCMP elements, and between UCCS and external entities, for the transfer of voice and data.

**Figure 3** on the next page provides an overview of the UCCS system.
Note that the CCS and VCS are only loosely coupled for safety reasons, so that the failure of one subsystem should not also result in the failure of the other.

**Figure 3: UCCS System Overview**
2.3 Safety Framework and Requirements

2.3.1 UCCS Safety Framework

The safety framework against which the UCCS was developed consisted of the following elements.

The overarching ASACS Safety Case (Ref [3]). This document was used as a background document rather than a requirement, as elements dealing with the ASACS command and control system were influenced by the existing, obsolete system that UCCS was to replace, and additional work on safety targets was still ongoing at the time of UCMP contract let. The document was produced by the RAF ASACS Safety and Standards Unit (ASSU) who acted as operational safety advisers to the ACCS Integrated Project Team (IPT) and the operational end user community.

The ACCS IPT, part of what then was the MOD Defence Procurement Agency (DPA), are the UCMP contract holders. As such they were responsible for ensuring that the UCCS was delivered against the requirements, including the safety requirements. The ACCS IPT employed the services of a safety contractor to act as their advisers throughout the development of the system.

In addition to the IPT’s safety advisers, an Independent Safety Auditor/Adviser (ISA) was also appointed by the IPT.

As part of the preparation for UCMP, a significant amount of preliminary safety work was performed by the MOD’s agents. This included:

- Analysis to define safety targets for the system, and guidance on the apportionment between the major system functions. As this work was undertaken prior to the solution being known, apportionment was at a true functional level and did not break down into hardware and software apportionment.

- Conducting a preliminary safety assessment (including Preliminary Hazard identification and Analysis), that enabled high level safety requirements to be set.

As a consequence of this initial work, a four part Safety Case approach was adopted for UCMP. This is explained further in Section 2.3.2.
2.3.2 UCCS Safety Cases

2.3.2.1 Safety Case Part 1

The four part Safety Case approach involved the production of the Safety Case Part 1 by the MOD’s safety agent (and its review and approval by the ASSU, ISA, Operational and Support Authorities). Part One presented the high level safety targets and the required safety integrity levels (SILs).

The preliminary safety analysis had determined that the most serious accident which could occur as a consequence of a failure of the UCCS, is a collision between a military aircraft under the control of the UCCS and a civilian passenger aircraft. Other types of potential accidents included military aircraft accidents (e.g. collision between 2 military aircraft, or controlled flight into terrain), and the accidental engagement of an aircraft with weapons. The top level safety target for the UCCS is therefore defined in terms of the maximum number of incidents per year in which prescribed aircraft separation standards have been eroded (violations of minimum separation criteria, or “minseps”).

This overall UCMP safety target was apportioned between human error, Government Furnished Equipment within the UCMP system boundary, and equipment provided by IBM under the UCMP contract. The UCCS safety target was apportioned further in terms of the number of ‘dangerous failures’ of key functions.

Due to a significant proportion of the safety target being apportioned to human error, a number of UCMP safety requirements were formulated addressing Human Factors.

In practical terms, the SIL levels of the system were assessed as:

- Command and Control Subsystem (CCS) components – SIL 1;
- Voice Communications Subsystem (VCS) components – SIL 2.

2.3.2.2 Safety Case Part 2

The Safety Case Part 2 was produced by IBM, predominantly during the Demonstration and Manufacture Phases of the programme. It presents the safety argument, based on a number of claims, and the accompanying evidence that the design and implementation satisfies all the safety requirements (including IBM originated ‘Derived Safety Requirements’ - DSRs) and that the safety risks have been identified and reduced to ALARP.

2.3.2.3 Safety Case Parts 3 and 4

The Safety Case Part 3 was produced by the ACCS IPT at the end of the Demonstration and Manufacture Phases, with the aim of demonstrating to the
RAF end user that the UCCS is safely integrated into the wider ASACS environment, is fit for use, and is supportable throughout its intended operational life.

The Safety Case Part 4 was produced by the RAF prior to the release of the UCCS into operational service, with the aim of demonstrating that operations could safely commence on the system and that the level of safety could be maintained throughout the life of the system.

2.3.3 Defence Standards (DEF STANs)

Production of the Safety Case Part 2 was against the following DEF STANs:

- DEF STAN 00-56 Issue 2, Parts 1 and 2 (Safety Management Requirements for Defence Systems);
- DEF STAN 00-55 Issue 2, Parts 1 and 2 (Requirements for Safety Related Software in Defence Equipment).
3 USE OF COTS WITHIN UCCS

3.1 Design Drivers

Analysis of the UCMP requirements by IBM resulted in the identification of a number of design drivers, and the subsequent definition of a number of design decisions early in the programme.

Design drivers and major decisions relevant to this case study are summarised in Table 1 below.

<table>
<thead>
<tr>
<th>Serial</th>
<th>Driver</th>
<th>Decision</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Meet the In Service Date (or better)</td>
<td>Minimise the creation of new 'custom' software by IBM or sub-contractors</td>
<td>This was one of a number of decisions taken to ensure that the In Service Date was met, and helped drive the selection of COTS products over bespoke alternatives on a number of occasions</td>
</tr>
<tr>
<td>2</td>
<td>Design safety into the system from the start, for ISA approval</td>
<td>Use only proven hardware and software components</td>
<td>This decision aligned with explicit MOD requirements (see Section 3.2 below) that COTS products used in the UCCS should be mature</td>
</tr>
<tr>
<td>3</td>
<td>Maximise use of standard products to minimise support costs</td>
<td>Maximise the use of proven, COTS software components.</td>
<td>Total through life cost was a significant MOD evaluation criteria for the award of the contract. The support budget was also publicised by MOD</td>
</tr>
<tr>
<td>4</td>
<td>Provide high system availability</td>
<td>Where appropriate, only use software components that support off-the-shelf high availability solutions</td>
<td>The UCMP requirements included requirements for high availability (as expected for a national defence system)</td>
</tr>
</tbody>
</table>

Table 1: Major Design Drivers and Design Decisions
3.2 MOD Requirements Governing COTS Usage in UCCS

As previously stated, there was a clear MOD expectation that the UCMP solution would employ COTS hardware and software to meet the requirements. Explicit MOD requirements regarding the employment of COTS were included in the SOW. Requirements of relevance to this case study can be summarised as:

- Where the COTS product implements some agreed (with MOD) product standard, the COTS product is actually certified as compliant to the product standard.

- The Contractor (i.e. IBM) will provide evidence, to the satisfaction of the MOD and the ISA, that the quality controls of COTS software suppliers provide:
  - A suitable quality management system;
  - Adequate development processes and configuration control;
  - An adequate problem reporting and correction infrastructure;
  - A documented record of known and corrected problems.

- The Contractor (i.e. IBM) will provide evidence, to the satisfaction of the MOD and the ISA, that the COTS product is mature, that is:
  - The product has a large user base;
  - The product is functionally stable (i.e. the functional changes in a new version are relatively small compared to the overall functionality of the product, and the interval between major versions is large (>1 year);
  - The version used is mature (i.e. uses the most up-to-date “service upgrade” of the version that incorporates fixes to reported problems);
  - In the Safety Case Part 2, the Contractor (i.e. IBM) will ensure that evidence of reliability includes experience using the COTS products during development and testing, including recording any faults detected.
3.3 Overview of COTS Software Products within UCCS

3.3.1 Introduction

A wide variety of COTS software products were potentially available for use within the UCCS. The following subset has been selected to indicate the range and scope of the items under consideration.

Voice Communication Subsystem. UCMP required a highly available, large scale voice communications system, able to demonstrably meet the SIL 2 requirement apportioned to this part of the overall system. Potential suppliers existed amongst companies able to offer large (national) scale voice systems for:

- Control of emergency services;
- Provision of national Air Traffic Control (ATC) services.

However, it was clear that regardless of which product was selected, an element of customisation would be needed to meet the specific UCMP requirements. Thus a modified COTS product would be needed. The potential supplier base at that time was deemed to be approximately 6 companies world-wide.

Interestingly, it was determined that systems employed by other militaries were either bespoke systems built to a specific set of requirements (and therefore unsuitable for UCMP), or modified COTS systems.

Multi-Sensor Tracker. UCMP required a sophisticated multi-sensor tracker software product. This product would lie at the heart of the production of the Surface and Recognised Air Picture (SRAP) within the SIL 1 Command and Control Subsystem (CCS). Again, the potential supplier base at that time was deemed to be approximately 6 companies world-wide, with any product having to be customised to meet the UCMP requirements.

Operating Systems. Operating systems were needed for a range of platforms, ranging from RAF operator workstations to relatively high end servers. Potential suppliers included Microsoft, and various suppliers of UNIX.

Database. Although the UCCS required a moderate-sized database, the requirements (including safety) restricted the choice to either Oracle or IBM DB2.

Map Processing Software Tools. The UCCS CCS is required to extract, process and display a number of map formats provided by several sources. Additionally, IBM felt that an appropriate tool would also support the display of dynamic objects on the RAF operator’s Geographic Display. Thus such a software component would play a significant role in the operator HCI (SIL 1). The number of suppliers was assessed at less than 6.
System Management Software Tools. Software tools were required to enable RAF engineering staff to understand and manage the CCS hardware and computer resources. Products from two potential suppliers were examined.

Database Frontend Applications. It was believed that a COTS database frontend application could be used to present information to operators in a suitable format. Thus such a software component would play a significant role in the operator HCI (SIL 1). Several such tools were believed suitable.

Software Development Tools. Included in this category are compilers, static analysis and runtime analysis tools, and configuration management tools. A significant number of products were potentially available.

3.3.2 How They were Selected – General Principles

IBM’s policy for the use of COTS software was defined and documented early in the project. The policy reflected the MOD requirements for the use of COTS software components (see Section 3.2), and can be summarised as:

- Use of only proven software components;
- Maximising the use of proven, off-the-shelf software components;
- Use of software components that support high off-the-shelf availability;

These policy statements required that selected COTS software products should be mature, reliable and robust, should have sufficient integrity, and should not be demonstrably unsafe. The policy was used during the development to guide the derivation of more detailed COTS product selection criteria that were then applied to the selection of specific products.

In accordance with the IBM UCMP Safety Plan, a COTS Evidence Profile proforma was developed and sent to potential COTS software suppliers. A blank proforma is included in Appendix A.

Each profile was examined by the IBM Safety Team to determine the extent to which the candidate products complied with the selection criteria. Examination of the completed profiles showed that the candidate COTS products generally had insufficient information to confirm that the product met all (or even most) of the required safety criteria, particularly in the area of documented acceptable reliability performance (and hence potential software dangerous failure rate when used in the CCS). The Profiles did, however, give confidence that all of the products were suitably mature, were well supported and had a wide user base. The Profiles also included evidence of a suitable development process for most products, giving confidence that the software was of suitable commercial quality. It was concluded therefore that the selected potential COTS products were
suitable for use in the CCS, subject to the collection of further evidence of their reliability (see Section 6).

### 3.4 Relationships with COTS Suppliers

At the time of entering the COTS software selection process, relationships with the potential suppliers varied considerably. Of significance was IBM’s previous experience in the air command and control systems domain, with systems such as FIADGE\(^\text{2}\) and FPDS\(^\text{3}\), providing some exposure to suppliers. This exposure varied from, in one instance a close and enduring working relationship, to the other extreme of merely ordering and receiving a ‘shrink-wrapped’ COTS software package.

As employees of a major software development company, UCMP staff had a range of personal experiences of some of the suppliers, and corporate policy also encouraged the use of some suppliers over others. By way of contrast, there were also a number of potential suppliers who were only known via publicly available sources, such as websites, product data sheets, press releases etc.

Included in the pool of potential suppliers was IBM itself. IBM as a supplier, and the associated COTS products, are not considered further for this case study as the internal relationship can be considered atypical of the usual COTS vendor/buyer relationship.

In summary, where sufficient UCMP team members had had a very positive previous experience with a potential supplier, and their product was clearly a contender for use in the UCCS, then this resulted in an early evaluation of their product. Two suppliers fell into this category.

---

\(^{2}\) FIADGE – Falkland Islands Air Defence Ground Environment, a much smaller and simpler predecessor to the UCCS.

\(^{3}\) FPDS – Flight Plan Dissemination System. A system that pre-dated the UCCS and used to buffer, filter and convert the flow of flight plan data into the air defence command and control system.
4 COTS ITEMS FOR CASE STUDY – DESCRIPTION, SELECTION AND SUPPLIER EVIDENCE OVERVIEW

4.1 Voice Communications Subsystem (VCS)

4.1.1 Role within the System

At its simplest level, the VCS provides secure and insecure ground-to-ground and ground-to-air communications. From a software perspective, the role of the VCS can be summarised as:

- Supporting 2 Control Centres and an Executive Centre with a significant (>100) number of operator positions.
- Provision of a VCS HCI for operators, complimentary to the Command and Control Subsystem (CCS) HCI (and used in parallel with it). HCI functionality was determined by the role of the operator.
- Control of voice switching throughout the UCMP system (including remote radio sites).
- Control of radios (local and remote).
- Provision of security enforcing functionality.
- Provision of system management functionality, to allow RAF engineering staff to manage the VCS at all sites from a central location.
- Interfacing with other systems. Although some of these interfaces were in accordance with various industry standards, several others would require the development of bespoke interfaces.

Although it had been determined that the core functions of the VCS, such as voice switching and control of radios, could largely be met by a COTS product, it was also clear that a degree of software modification would be required to meet some of the UCMP requirements.

4.1.2 Selection Process

Initial evaluation of the contending products eliminated those designed only for the support of emergency services, on the grounds that they were unlikely to be able to demonstrably meet the SIL 2 requirement without so much modification that they would effectively become bespoke systems. This left products with an Air Traffic Control (ATC) heritage and which had been developed for a safety-related
environment, and their respective suppliers who were consequently familiar with the production of safety cases (although not necessarily familiar with DEF STAN 00-56 and 00-55).

Although good evidence was available from suppliers and third parties regarding previous employment of their products in a civil ATC capacity (providing evidence around some of the core functions such as voice switching), evidence regarding the use of the product in a similar, military environment was more limited. The reasons included:

- Unwillingness to identify other customers (customer confidentiality);
- Evidence was from earlier versions in the product line, not the currently fielded version;
- Lack of detail about the exact role and modifications of the system in other military employments, due to security.

Final selection was based on a limited competition between several non-UK suppliers, against an IBM provided Request For Proposal package that included a suitable flowdown of requirements. These not only included functional requirements, but given the role and SIL of the VCS and the clear evidence gaps that existed, also required the successful supplier to undertake their own safety programme to generate the requisite evidence for the VCS, in accordance with the UK DEF STANs 00-56/2 and 00-55/2.

The supplier responses were formally evaluated by IBM and a subcontract awarded on the basis of this evaluation.

### 4.1.3 Supplier Relationship

Until the commencement of the programme, the supplier was unknown to IBM. Given the pivotal role that the supplier would play however, and the SIL 2 requirement for the VCS, a close working relationship would clearly be necessary between IBM and the supplier. This ‘soft’ factor was taken into account during the formal evaluation of the suppliers.

### 4.1.4 Supplier Evidence Provided

In response to the Request For Proposal, the suppliers provided evidence of compliancy against requirements; evidence of the proposed software and hardware architectures; evidence regarding process and good practice (both development and safety management), including any certification; evidence of user base; evidence of product line development and improvement; evidence of domain experience; and an indication of the types and extent of field data that could be provided.
Once selected and contracted, the supplier provided evidence throughout the development phase included:

- A safety plan detailing their approach to the provision of the required safety evidence (for acceptance by IBM).

- A Safety Case Part 2. As this safety case was not going to be used as a standalone safety case, but would be incorporated into the full UCCS Safety Case Part 2, the goal of the VCS safety case was to provide enough coherent evidence for the UCCS safety case, including evidence that the VCS safety requirements had been met. The VCS Safety Case included a separate software safety case tailored to meet DEF STAN 00-55/2 (Ref [4]).

- Test results.

- Field data performance both on system components (from previously fielded versions), and on the UCMP VCS system itself. This was gathered for the period of test at the supplier factory (with unrepresentative interfaces but simulated high user loads), and for the interval between installation in the target environment and final acceptance by IBM (where real interfaces were connected but user loads were generally low).

- Analysis results.

- Evidence regarding the supplier safety culture, quality and configuration management, staff competency, etc.

Although some of this evidence pre-existed the UCMP subcontract, other evidence had to be generated specifically for UCMP. Of particular interest from a safety perspective was the evidence regarding the UCMP specific modifications to the core product, estimated to affect approximately 5% of the software. In some cases the provided evidence was deemed inadequate, necessitating further work by the supplier.

The core VCS product did not include any embedded COTS products provided by other suppliers.

A key point is that because of the approach followed, IBM could observe the generation of some of the evidence at first hand (e.g. by witnessing the testing), and periodically review other evidence as it was collected to ensure that it was suitable. The supplier could be re-directed if felt necessary.

### 4.1.5 Conclusion

Despite the VCS being developed explicitly for a safety-related environment (with associated evidence available), IBM nevertheless required the supplier to
generate a significant amount of safety evidence, documented in a safety case, for IBM's acceptance. It is estimated that approximately 50% of the UCMP VCS safety case was generated by the supplier under the terms of the subcontract, with the remainder pre-existing. The newly generated evidence included a significant amount of software analysis.

4.2 Microsoft Windows 2000

4.2.1 Role within the System

Microsoft Windows 2000 operating system, in the Server, Professional and Security Enhancement (SE) versions, is used extensively throughout the UCCS. For example, there are significantly more than a 100 instances of the Windows 2000 SE version used in UCCS operator workstations alone.

The key functions performed by the operator workstation are deemed SIL 1.

4.2.2 Selection Process

At the preliminary design stage, consideration was given to using either UNIX or Windows as the operating system for operator workstations. UNIX was deemed more robust, but would entail the development of considerably more bespoke software, posing a risk to the required In Service Date. It would also have undesirable implications for the selection of hardware platforms.

Windows NT4.0 had been successfully used by IBM on several other systems within the same or related domains, including a SIL 1 system, providing IBM with a body of field experience. However, Windows NT4.0 was obsolete and near the end of its support life, and its use would be problematic given the anticipated life of the system. Consideration was therefore also given to Windows 2000.

Microsoft were requested to complete COTS Evidence Profile proformas for both operating systems. No response was received. A proforma was completed by the IBM team using publicly available information, but this did not include information on aspects such as development processes, quality and configuration management.

The following selection criteria were used:

- User base. Both operating systems had a wide user base, but at the time of evaluation, little evidence could be found that Windows 2000 had accrued a significant period of use in similar or related environments.

- Field Experience. Although IBM had field experience of NT4.0 in similar and related domains, in examining the historical fault data it proved difficult
to distinguish between faults with a root cause in the operating system, and faults generated by the application. This data proved far less useful than anticipated. However, it was possible to obtain general software Mean Time Between Failure (MTBF) data for IBM systems using NT4.0.

- Third Party Trials. Published data was obtained from 2 third party trials of Windows 2000. The first was a comparison of the reliability of Windows 2000 and NT4.0, using a large number of PCs running “everyday business applications.” This concluded that Windows 2000 was more reliable than NT4.0 (in the trial environment). The second trial examined the reliability of Windows 2000, and concluded that it was reliable enough for large scale “enterprise” solutions. Due to the differences between the trial and UCMP domains however, this evidence was treated merely as indicative of possible improved reliability in Windows 2000.

Windows 2000 was provisionally selected on the probability of being more reliable than Windows NT4.0, and on the basis of being supportable for longer, but it was clear that the existing evidence was unsatisfactory to support a safety argument. Consequently, IBM would need to generate its own safety evidence for Windows 2000.

4.2.3 Supplier Relationship

The relationship with Microsoft did not extend beyond that of a normal buyer-vendor relationship.

4.2.4 Supplier Evidence Provided

None.

4.2.5 Conclusion

Other than the evidence of a large user base, and indications that Windows 2000 was more reliable than NT4.0 (albeit in an office environment), virtually no suitable safety evidence existed for Windows 2000. This necessitated IBM generating its own.

4.3 Map Processing Software Toolset

4.3.1 Role within the System

The toolset is required to extract, process and display a number of map formats provided by several sources. Additionally, the toolset also supports the display of
dynamic objects on the RAF operator’s Geographic Display. As such the toolset is a critical part of a function deemed by the Safety Case Part 1 to be SIL 1.

4.3.2 Selection Process

A number of products were evaluated using information from public sources, together with relevant field experience available from the use of several of the candidates by IBM. The field evidence provided indications of functional unsuitability or disadvantage, rather than counter-evidence as defined in the Working Paper (Ref [1]).

A candidate product was selected based on its match against functional criteria, and the supplier was then asked to complete a COTS Evidence Profile proforma.

The COTS Evidence Profile confirmed that the base product was mature with a large defence user base, including use in similar applications to the UCCS. Hundreds of development licences and hundreds of thousands of deployment licences had been deployed throughout the world in operational systems. Based on the data supplied in the Profile, the software was assessed as a mature and reliable product.

The supplier also provided information on functional stability, requirements, configuration and quality management, design and development processes, and the data/fault reporting and corrective action system (DRACAS) used.

To meet UCMP functional requirements however, the supplier had to develop two new sub-applications within the toolset. Although these became part of the COTS product commercially offered by the supplier, it nevertheless meant that the product version selected by IBM had to be regarded as a new version of the product.

4.3.3 Supplier Relationship

A close working relationship was developed with this supplier. The supplier was very co-operative in supplying information, and in resolving issues.

4.3.4 Supplier Evidence Provided

The main supplier evidence comprised a detailed COTS Evidence Profile.

4.3.5 Conclusion

Although a mature base product was selected with prior usage in related domains, only generalised evidence was available from the supplier. Reasons why detailed evidence was not available included:
• IBM was the first customer for the product version used;

• Other users of the toolset in applicable domains tended to be non-UK military users. Thus both commercial and security considerations prevented the release of detailed information to IBM.

Given these circumstances, it was necessary for IBM to generate its own safety evidence.

4.4 Database Frontend Application

4.4.1 Role within the System

The product was required to provide the underlying software for the operators to view and manipulate (enter/amend/delete) data contained in the CCS database. As such the toolset is a critical part of a function deemed by the Safety Case Part 1 to be SIL 1. Elements of the data being viewed/manipulated are also safety-related.

4.4.2 Selection Process

A number of products were evaluated using information from public sources. IBM had not used any of the products previously, thus no field experience was available.

A candidate product was selected based on its match against functional criteria, and the supplier was then asked to complete a COTS Evidence Profile proforma. The proforma was completed, but only to a very high level and probably by a member of the sales team. The combination of this and the publicly available evidence could do no more than demonstrate that the product was mature with a wide user base. There was no evidence to show the product had been used in a related or similar domain.

4.4.2 Supplier Relationship

The relationship with this supplier did not extend beyond that of a normal buyer-vendor relationship. Even when faults were identified in the product by IBM, the supplier remained unresponsive.

4.4.4 Supplier Evidence Provided

COTS Evidence Profile – partially completed.
4.4.5 Conclusion

Other than evidence of a large user base, no suitable safety evidence existed for this product. This necessitated IBM generating its own.
5 SUPPLIER EVIDENCE AND GAP ANALYSIS

5.1 Introduction

The Working Paper on Evidential Approach, Addressing Legacy and COTS Software (Ref [1]), identifies (in Section 3.1) the major sources of existing evidence likely to be provided for COTS products. These sources are listed in the table below, and their applicability to the COTS products described in Section 4 are summarised. The remainder of this Section discusses in more detail the evidence provided by the suppliers and the guidance for conducting gap analysis presented in Section 3.1 of Ref [1], in the context of its applicability to the UCMP project.

<table>
<thead>
<tr>
<th>Evidence Type (see Ref [1] Section 3.1)</th>
<th>Applicability to UCMP Case Study COTS Products</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certification Packs</td>
<td>None provided.</td>
<td>The COTS Evidence Profile proforma requested evidence of conformance to standards. Suppliers either did not complete the proforma or could not point to any certification other than to quality and process standards for the product being offered, even though in the case of the VCS, earlier product versions had been certified by various ATC safety agencies.</td>
</tr>
<tr>
<td>Further Supplier Tests and Analyses, and Other Documentation</td>
<td>The VCS supplier was contracted (and funded) to generate further evidence. The other 3 suppliers were requested to complete COTS</td>
<td>The VCS supplier was contracted to supply evidence in the form of a product safety case. The level of detail...</td>
</tr>
<tr>
<td>Evidence Profile proforma; two did so.</td>
<td>provided in one response was good compared to the majority of the responses received. The level of detail in the other response was sparse.</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>Third Party Evidence</td>
<td>Used for 3 out of 4 products. In general, provided confidence that the supplier worked in accordance with the principles of good practice. This evidence helped provide the justification for selecting the supplier in the expectation that IBM could then generate adequate evidence for the safety case.</td>
<td></td>
</tr>
<tr>
<td>Pre-Existing Safety Cases</td>
<td>Not used. One supplier (VCS) had generated safety cases for earlier product versions, giving confidence that they understood how to generate a safety case for the proposed product. However, even with this, evidence gaps still existed.</td>
<td></td>
</tr>
<tr>
<td>In-Service Evidence</td>
<td>Used to gain confidence (or otherwise) in supplier/product selection for 3 out of 4 products. Used to provide MTBF data for 1 product.</td>
<td></td>
</tr>
<tr>
<td>Trustworthiness of Existing Evidence</td>
<td>For 1 product, no evidence was supplied. For another product, very little evidence was supplied (and none regarding development). For the other 2, visibility of the development processes For VCS, trustworthiness was further established by audit (safety and quality) and by collaborative working. This enabled process</td>
<td></td>
</tr>
</tbody>
</table>
provided trust that those processes were as stated by the supplier. evidence (e.g. DRACAS) to be used with confidence.

Counter-Evidence Anecdotal evidence from early use of Windows 2000 in the project suggested unreliability when used in a general office environment. Although this evidence came from a different domain, it resulted in mitigation in the form of a ‘lock-down’ of the operating system. See Section 6.

Table 2: Summary of Supplier Evidence Using Working Paper Categories

5.2 Assessment of Evidence Provided Using Working Paper Definitions

5.2.1 Certification Packs

As can be seen from Table 2, no Certification Packs were available for the COTS products described in Section 4. Product selection occurred in the period 2001-2002, so this was certainly a factor. The use of COTS Evidence Profiles by IBM sought to elicit information about relevant, pre-existing certifications, but this was unsuccessful.

Of particular interest is the VCS, which was a product aimed specifically at a safety related market. Despite this, the VCS safety case made no claims regarding previous certifications of related products. Reasons for this include:

- The product selected was scaled for use as a national system, and the supplier would not divulge other users (civil and military) on the grounds of customer confidentiality.
- UCMP requirements resulted in the modification or addition of software (approximately 5% of the VCS software was modified or new).
- No claim could be made for the use of the same or related products in an environment identical to that in which the UCCS would be employed, as the supplier did not have this detailed knowledge of the UK operational environment.
5.2.2 Further Supplier Tests and Analyses

As identified in Table 2 above, 3 of the suppliers were requested to complete COTS Evidence Profile proforma, but only 2 did so. Of these two responses, one (for the database frontend application) was relatively superficial and was probably completed by a member of the product sales team. The information provided was generally publicly available, and served to confirm a wide user base and a mature product. The supplier would not name any other specific projects however, thus there was no evidence to show the product had been used in a related or similar domain. No information was provided regarding development processes (the Development Summary section was not completed).

The second response – for the map processing software toolset – was comprehensively completed when compared to the other responses received. However, this still meant that responses in the Development Summary section were generally no more than one paragraph in length.

IBM required a safety case part 2 for the VCS, in order to provide enough coherent evidence for the UCMP Safety Case Part 2. The VCS Safety Case included a separate software safety case tailored to meet DEF STAN 00-55/2 (Ref [4]). The VCS Safety Case was a significant undertaking, against an agreed safety plan, which required approximately 2 years to complete.

In terms of software, the supplier was required to complete a software safety case providing evidence that a number of safety claims had been met. The evidence provided included:

- Direct evidence from analysis.
- Direct evidence from testing, and direct evidence from in-service use of previous systems.
- Direct evidence extracted from the review process.
- Process evidence regarding the supplier’s practice in developing and maintaining the product.
- Qualitative evidence for good design.

Further detail is provided below.

5.2.2.1 Direct Evidence from Analysis

Software failure data from the preceding version of the product was used by the supplier to undertake a Fault Tree Analysis of the VCS, to demonstrate that the maximum number of dangerous failures in key functions would be well within tolerable limits.
Analysis was also used to demonstrate that the modified and new software for UCMP did not exceed 5% of the overall product, that the most critical software (controlling basic voice switch functions) had not been changed at all, and that the software architecture provides protection against software faults.

The supplier also undertook a number of analyses (e.g. hazard analysis), and was able to demonstrate how control measures had been put in place to control the consequences and probability of identified hazards (including those flowed down from the wider UCMP hazard analysis).

A specific analysis was undertaken to demonstrate that the software development process was compliant with the requirements of SIL2 in DEF STAN 00-55 (Ref [4]).

5.2.2.2 Direct Evidence from Testing; Direct Evidence from In-Service Use of Previous Systems

Test evidence from System Tests was provided by the supplier. These tests provided 100% coverage of the functional requirements, and were conducted with the software loaded on the target hardware.

The supplier was able to provide significant evidence regarding the performance of the preceding version of the product. As only the supplier can maintain the product software, support arrangements are always placed with the supplier who maintains a central database of all actual and suspected software (and hardware) problems. The support arrangements also meant that the total number of operating hours for each installation was also known.

This data enabled the supplier to calculate the Mean Time Between Failures (MTBF) for the core software. However, this evidence was tempered by the fact that just as the UCCS contained new/modified software to meet UK requirements, most of the fielded installations had also been delivered with minor modifications, albeit not to the core software.

5.2.2.3 Direct Evidence Extracted from the Review Process

Safety audits of the supplier were undertaken by both IBM and the ISA. These focused on the verification of process evidence, test evidence and in-service use evidence.

5.2.2.4 Process Evidence Regarding the Supplier’s Practice in Developing and Maintaining the Product.

As part of the software safety case, the supplier was required to provide evidence that:
- The general process framework was adequate. To achieve this, the business and quality processes were described (including provision for, and evidence of, feedback and process improvement).

- A suitable, mature safety culture existed.

- A robust quality management system was in place and proven to be effective. This included evidence of ISO9001 certification.

- The supplier software development process was adequate (e.g. by means of external certification, use of coding standards, code review processes and independent audit).

5.2.2.5 Qualitative Evidence for Good Design

As part of the software safety case, the supplier was required to provide evidence that the design was suited to the safety related function of the system. This was achieved by demonstrating a history of design improvement in the product series, including the rationale, benefits and limitations of the design. Aspects of this evidence were verified by audit.

5.2.3 Third Party Evidence

Third party evidence was used to assist the selection of Microsoft Windows 2000, although the data obtained did little more than generate some confidence that Windows 2000 was more reliable than NT4.

5.2.4 Pre-Existing Safety Cases

Although safety cases had been generated for earlier versions of the VCS product, these were not directly used or referenced by the supplier. However, some elements of pre-existing safety cases e.g. process evidence, were incorporated in the safety case developed for UCMP.

As far as is known, there were no pre-existing safety cases developed by the suppliers of the other products.

5.2.5 In-Service Evidence

As discussed in section 5.2.2.2, in-service evidence from the preceding version of the VCS product was used to generate the Mean Time Between Failures (MTBF) for the core software.

However, as previously discussed, it was not possible to identify either users of the same product version, or identical operational environments. It is of note that the general UCMP experience was that suppliers of COTS products used in
military projects were, on the whole, unwilling to disclose what those projects were.

In-service evidence gathered by IBM also gave confidence that Windows would prove to be a suitable operating system.
5.3 IBM Examination of Supplier Evidence

In describing the extent to which evidence was examined by IBM, it is best to consider the 4 products in the following two groups:

- Windows 2000, map processing software toolset, and database frontend application;
- VCS

Evidence for the more ‘typical’ COTS products (Windows 2000 etc) was examined to varying degrees, probably because it was rapidly concluded that the evidence was already inadequate, but also because of the difficulty in verifying suppliers’ claims.

For example, claims regarding user base (made in the COTS Evidence Profiles) were checked against the supplier’s website and general industry literature for consistency. Claims regarding development processes were taken at face value, if supplier relationships had generated a degree of trust and a positive working relationship (e.g. map processing software toolset supplier). Where positive supplier relationships had not been developed the amount of evidence available was also sparse.

Historical fault data from IBM projects for Windows NT4.0 was examined in some depth to attempt to generate meaningful reliability evidence.

VCS evidence was examined in some detail. As explained earlier, selection of the VCS product was made following a limited competition between several potential suppliers. The information received from the suppliers was verified, as best as possible, using publicly available sources. It should be noted that whilst some third-party information was available in the public domain regarding relevant previous civil Air Traffic Control (ATC) projects, the only information available regarding military projects (and thus more analogous to UCMP) was placed in the public domain by the suppliers themselves.

Examination of the evidence tended to confirm claims regarding product user base, product stability, and external certification (e.g. ISO9001). Claims regarding good engineering practice could, to some extent, be indirectly confirmed by obtaining information regarding the supplier’s reputation. Generally however, such corroboration was used to build confidence that a supplier could generate adequate evidence if subcontracted to do so.
5.4 Use of the Proposed Gap Analysis

5.4.1 Introduction

The purpose of this section is to investigate the application of the gap analysis explained in Section 3.1.3 of the Working Paper, Ref [1]. The Working Paper presents 6 major categories of evidence mismatch (reasons why the existing evidence is unlikely to provide sufficient support to a software safety case). The categories are discussed in the order they are presented in the Working paper, i.e. in an approximate order of importance. Each category is discussed in turn in the context of the 4 COTS products used in this case study. A brief summary of each category is given, but the reader should refer to Ref [1] for a full description.

5.4.1.1 Failure Consequence and Integration Assumptions

The failure consequences and integration assumptions refer to the likely effect on the system if the software fails, as well as any assumptions which have been made about the manner of failing or the ability of the component to integrate into a system.

5.4.1.1.1 VCS

As previously stated, the VCS was required to meet a SIL2 target, thus in selecting a VCS product particular attention was paid to the suppliers’ claims regarding failure consequences and fault tolerance. At the time of selection, the provided evidence was in the form of design documentation. For the selected product, this indicated that for core VCS functions, the failure modes and consequences could read from the ‘normal’ product environment (civil ATC) across to the UCMP environment, if certain assumptions were made regarding the core software functionality. (It is of note that not all of these assumptions were explicitly made by the supplier; some had to be inferred by IBM.)

The following bullet points assess those assumptions in the context of the guide questions contained in Ref [1]:

- UCMP-specific requirements would necessitate some changes to VCS software. It was assumed that these changes would be ‘small’, and more importantly, would not affect the core software operation and failure modes. From a safety perspective however, there was no justification to accept these assumptions as valid, and IBM deemed it necessary to explicitly address them in the VCS safety case. It is recommended that a guide question be included in this working paper category to force consideration of the impact on the evidence of any proposed modification to the COTS software.

- UCMP-specific requirements would require development of at least one bespoke interface to an external system, as well as bespoke interfaces to
the other subsystem (the CCS). Details of the former were unknown at the time of product selection, and the requirements for the latter had to be finalised between IBM and the selected supplier post-selection. The product claimed robust handling of interfaces, and that there would be no impact on core functionality or failure modes arising from new interfaces. Again, from a safety perspective there was no evidence to justify such claims, and these had to be explicitly addressed in the VCS safety case. In retrospect, consideration of the working paper guide questions would lead to a similar conclusion being drawn by IBM.

5.4.1.1.2 Windows 2000; Map Processing Software Toolset; Database Frontend Application

Evidence for these COTS products was generally restricted to a claim that the product was ‘reliable’ if installed on a suitable hardware platform and (excluding Windows 2000) in conjunction with a suitable operating system. Further evidence therefore had to be generated by IBM.

Retrospective application of the guide questions however, does highlight the need to consider situations where multiple COTS software products will operate together. For example, all 3 products were present in the operator workstation.

5.4.1.2 Adequacy

‘Adequacy’ refers to the adequacy of the evidence provided. Evidence may be inadequate either through a lack of trustworthiness, or simply because it has been restricted in ways which are inappropriate considering the proposed usage.

5.4.1.2.1 VCS

Despite a significant amount of information being provided by the supplier, it was deemed inadequate for a number of reasons already discussed and which the guide questions in the Working Paper would have illuminated. Inadequacies included:

- Unwillingness to identify other customers (customer confidentiality);
- Evidence was from earlier versions in the product line, not the currently fielded version;
- Lack of detail about the exact role and modifications of the system in other military employments, due to security.

The final point regarding role is perhaps worth expanding in the guide questions. As has been found on a number of defence projects, there are several significant barriers to the provision of evidence that a COTS product has been used in environments similar enough to the target one. These include:
Security considerations/classification;

Inadequate knowledge possessed by the COTS supplier regarding the environments in which his product is operationally employed. By their very nature, COTS products possess an element of the generic and general, which perhaps implies a lack of specific military domain knowledge. Their suppliers are also several steps from the end use of their products.

It is suggested therefore, that the guide question on ‘operational modes’ be expanded to challenge the supplier’s knowledge of the domains in which he is claiming usage evidence for, and for the domain that he is trying to supply into.

5.4.1.2.2 Windows 2000; Map Processing Software Toolset; Database Frontend Application

As has been explained, relatively little (or no) evidence was provided by the suppliers of these products. In applying the guide questions, the final question (“Is there ‘enough’ evidence?”) would have been the first and only question asked.

5.4.1.3 Degree of Protection

The degree of protection refers to the likelihood that a failure of the software will progress to a system hazard.

5.4.1.3.1 VCS

There was no explicit claim made by the supplier regarding the degree of protection offered by the product, although the claim was implicit in that the product had been developed for a safety-related environment and was, by design, intended to be fault tolerant.

Applying the guide questions retrospectively provides the following points for consideration:

- Supplier evidence related to an environment (civil ATC) that the supplier believed was analogous enough to carry some weight. As IBM are not specialists in the civil ATC environment it was not possible to reach the same conclusion. Guide questions regarding mitigations in the original environment are especially pertinent here.

- The fact that evidence generally related to the previous released version of the product, and as subsequently became clear, this previous version had been subject to minor modifications to meet individual customer requirements, means that the majority of the questions were unanswerable.
5.4.1.3.2 Windows 2000; Map Processing Software Toolset; Database Frontend Application

Generally speaking, insufficient evidence was supplied for all these products to enable any of the guide questions to be meaningfully answered. Even where IBM had its own field data (from Windows NT4.0), the majority of the questions require a level of detail that was not available.

5.4.1.4 Inputs and Data Flow

In order for the existing evidence to be of use to the proposed safety case, the data flows and inputs must be equivalent, or compatible in some way.

5.4.1.4.1 VCS

The guide questions presented in the Working Paper are particularly relevant to the question of VCS interfaces. As stated previously, several interfaces were undefined at the time of product selection, undermining supplier evidence regarding software behaviour. This situation had been predicted (as a result of requirements analysis), and was one of the reasons behind requiring the supplier to produce a safety case.

5.4.1.4.2 Windows 2000; Map Processing Software Toolset; Database Frontend Application

Insufficient data was provided by the suppliers to allow any of the guide questions to be answered. In considering the questions against the map processing software toolset and the database frontend application, intuition would have suggested (correctly) that given the role of the applications and their user base, it should be possible to generate adequate safety evidence for these products.

5.4.1.5 Resource Availability and Interference

If evidence is generated from an environment where there is no possibility of interference from other software, then this evidence is of limited use when attempting to show the safety of the software in a system where such interference is a possibility. Equally, if the evidence is generated in an environment where there is no difficulty over competition for resources, then the software may not be as reliable as the evidence indicates (for example, it may be vulnerable to timing failures).

5.4.1.5.1 VCS

The VCS supplier was able to present evidence regarding resource availability and interference, as his product was designed to run only on his supplied hardware. The evidence suggested, superficially at least, that this was a low risk area, although the evidence was not accepted at face value due to the need to develop some new software, and required enhancement during the development
of the VCS software safety case. As previously discussed, questions concerning the impact of new interfaces also existed.

However, application of the guide questions would have raised questions concerning the origins of the presented evidence (it was derived from a previous product version), and the impact of scale (the proposed product was 'up-scaled' significantly from its predecessor).

5.4.1.5.2 Windows 2000; Map Processing Software Toolset; Database Frontend Application

No evidence was presented by suppliers other than host platform performance requirements.

5.4.1.6 Standards and Applicability

This type of usage mismatch comes about when the techniques used to generate the existing evidence are no longer in use, or when the types of evidence presented do not cover the full scope of the proposed usage. In addition, the guide questions help challenge whether adequate configuration control was exercised.

5.4.1.6.1 VCS

Application of the guidance questions to the evidence provided would have generated a degree of confidence that good practice had been followed, although this confidence would still have been insufficient to accept the evidence at face value. One point from the application of the questions should be considered:

What do MOD consider ‘good development practice’ as applied to current standards? Many commercial software suppliers have adapted standards to meet their own particular requirements, and claim a degree of (uncertified) conformity with a standard.

5.4.1.6.2 Windows 2000; Map Processing Software Toolset; Database Frontend Application

From the COTS Evidence Profile information provided, there was only sufficient evidence to support a limited claim of good practice during development for one of the 3 products. Nearly all the guidance questions would have received a negative response.

5.4.1.7 Conclusion

IBM had quickly concluded that the evidence provided by suppliers was inadequate to satisfy IBM’s requirements for a safety case, and indeed this outcome had been predicted from the outset of the project. This conclusion was reached however, more by the application of judgement rather than any formal analysis.
Use of the Working Paper categories to provide some rigour to this process would be, we judge, very beneficial. The guidance questions also serve to highlight areas of evidence mismatch which need to be addressed by the system safety case. We would also make the following points for consideration:

- If applied retrospectively to UCMP, nearly all the guidance questions would have received a negative response. The primary reasons for the negative responses can be distilled down to 2:
  - Lack of detail in suppliers’ responses;
  - Lack of detailed supplier knowledge about either the environments their product has previously been used in, the environment their product is intended to be used in, or both.

- The guidance questions can be used to attempt to elicit better information from suppliers. However, there will still be a significant number of cases where suppliers will not provide anything other than standard marketing information.

- Has consideration been given to the use of guidance questions to test evidence mismatches in the area of supplier product improvement? For example, if the supplier claims a Defect Reporting and Corrective Action System (DRACAS) for a product line, then there is a suggestion of continuous improvement. Although guidance questions test whether the evidence being presented is for the current product version, it does not explicitly test evidence regarding the effectiveness of a supplier DRACAS.

- The guidance questions for the category ‘Degree of Protection’ assume very detailed knowledge by the supplier. It is our experience that this is very unlikely to ever be available.

- The category ‘Adequacy’ suggests a binary response for evidence (i.e. adequate or inadequate), although the guidance question regarding ‘enough evidence’ is more indicative of the UCMP experience. One informal perspective we found useful was to assess what ‘weight’ evidence

---

4 In our experience, reasons given include: company policy; desire to avoid legal liability by giving inaccurate or misleading information; or customer purchase is insufficient to warrant the supplier expending effort to supply additional information.

5 The VCS provides an interesting example here. The supplier could claim that because the VCS hardware and software were developed by the supplier, only the supplier could support the product, providing a high degree of visibility of in-service faults which are addressed in later versions of the product.
carried (none, a little, a lot etc). Most COTS supplier evidence carried ‘little’ weight, and it was necessary to generate more evidence until the ‘weight’ of the safety argument was judged enough to convince the MOD.
5.4.1.7.1 IBM Selection of COTS Products for UCMP

Finally, a brief summary is offered of the COTS selection approach used for UCMP.

Firstly, it is important to appreciate that selection of COTS items occurred throughout the early phases of the project, with some of these phases being competitive, and with all phases subject to intense timescale pressure. Table 3 below indicates the phases, the associated major activity, and when the 4 COTS items used in this case study were selected. Note that the phases and activities are fairly typical of an MOD procurement.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Approx Duration</th>
<th>Activity</th>
<th>Selection of COTS Item &amp; Comments</th>
</tr>
</thead>
</table>
| Concept Phase (bidding occurred late in this phase) | Industry involvement (bidding): Jul-Oct 2000 | Bid for Assessment Phase contract (IBM one of 6 bidders). High level design & rough costs for final solution required (implementation budget known) | VCS supplier selected  
Windows NT4.0 selected |
| Assessment Phase             | Jan-May 2001    | Bid for Demonstration and Manufacture (implementation) Phases. IBM one of 2 bidders. Preliminary design and firm price for solution | Decision to overturn Windows NT4.0 selection and use Windows 2000 instead  
Selection of Map Processing Software Toolset  
Safety plans, preliminary hazard analysis and interim safety case part 2 generated |
| Demonstration and Manufacture Phase | Dec 2001 – May 2004 | Detailed design, implementation and test                                  | Selection of database frontend application early in phase  
Full safety case part 2 |

Table 3: Selection of COTS Items by Project Phase
In all cases, COTS items were selected on the basis that IBM believed they could generate sufficient safety evidence for them during the course of the programme. At no time was a COTS item selected on the basis that the supplier evidence was adequate.

Supplier evidence, if available, was used to assist this selection process, and continued to be used in the development of the safety case. Such evidence was used in the selection of the VCS and map processing software toolset, whereas supplier evidence for Windows 2000 and the database frontend application was negligible, and the decision was made using the few items of publicly available information, and ‘engineering judgement’.

Usable supplier evidence was provided for the VCS and the map processing software toolset. Given the SIL level and the role of the VCS, this evidence was verified by IBM during the development of the Safety Case Part 2, by means such as review of the supplier’s internal process documentation and by audit. The supplier evidence for the map processing software toolset was assessed as carrying relatively little weight compared to the anticipated IBM generated evidence, thus no direct verification of the evidence took place. There was some indirect verification using the informal contacts between the supplier and the IBM developer teams.

Note that Windows NT4.0 was originally selected as the operating system rather than Windows 2000, on the basis that:

- IBM had experience of, and field data for, NT4.0;
- At the time, Windows 2000 was a relatively unknown quality regarding its use in the UCMP type of environment;
- Windows 2000 was also some way off receiving security accreditation.

This decision was changed as a result of actual analysis of the NT4.0 field data, information regarding the supportability of NT4.0, a stated target date for security accreditation of Windows 2000, and the limited third party evidence indicating that Windows 2000 would be more reliable than NT4.0.
6 IBM GENERATED EVIDENCE

6.1 Introduction

To produce the evidence deemed necessary for the safety case part 2, further evidence was generated either by IBM, or in the case of the VCS, by the supplier under subcontract to IBM.

6.2 VCS

6.2.1 Introduction

As previously explained, IBM believed it necessary to contract the VCS supplier to produce a safety case part 2 for the VCS, including a software safety case. In keeping with the approach used in the UCMP Safety Case Part 2, the VCS software safety case used a claim based approach.

6.2.2 VCS Software Safety Case Claims

Working in agreement with IBM, the VCS supplier presented evidence for a number of claims. The following subset are provided as an illustration:

- Top level claim - “The VCS software is tolerably safe”.
  - Sub claim – “The VCS Switch software is tolerably safe”. (‘VCS Switch software’ is defined as the core voice switching software, as distinct from the new software developed for UCMP.)
    - Sub sub claim – “Historical data shows that the VCS Switch software failure rate is acceptable”.
    - Sub sub claim – “New software does not adversely affect the VCS Switch integrity”.
    - Sub sub claim – “The VCS Switch software development process is adequate”.

The evidence to support these claims was reviewed by IBM as it was presented in successive drafts of the safety case. IBM audit and verification of the supplier’s process evidence has already been summarised. The following paragraphs summarise IBM activities to verify the accuracy of other elements of supplier evidence.
IBM witnessed all supplier formal test activities, having first reviewed and agreed (in conjunction with MOD) the test specifications. Although all testing was performed with the software loaded on the target hardware, Factory Acceptance Testing was not representative of the target environment due to the use of representations of some interface types.

Thus IBM paid particular attention to the installation and commissioning testing of the VCS systems as they were installed in the operational environment. In particular, both informal and formal interface and end-to-end testing were observed, and far-end users and test equipment were employed by the MOD to verify correct functioning of the software.

Particular attention was also paid to the testing of new software. In certain instances, code reviews were used to further verify correct functioning, and also to provide confirmation of coding standards and good practice.

The supplier was also asked to provide statistics for the number of software defects recorded on projects of similar size (both system size and effort), to assess whether the number of defects detected fell within the normal range.

6.3 Windows 2000; Map Processing Software Toolset; Database Frontend Application

6.3.1 Introduction

Further evidence of product suitability was sought from use of the products in known specific applications (especially safety related applications), and from product initial testing (functional evaluation, product integration & soak testing) conducted as part of the CCS development. This was aimed at generating evidence of acceptable COTS software performance.

In estimating the product ‘on hours’, account has been taken of actual usage time, although it is recognised that the usage was not wholly representative of the actual usage that would be experienced in operational service (e.g. there was less user input via operator keyboard than will be experienced during actual system usage).

Use of the COTS products in the project development environment (used to write and test the CCS code) was not included in the evidence, due to this environment being deemed unrepresentative of the target environment.
6.3.2 Microsoft Windows 2000

The Microsoft Windows 2000 operating system (in the Server, Professional and SE variants) is used extensively throughout the UCCS CCS. Thus residual software faults were considered to potentially have a large effect on the perceived reliability of the system.

The COTS Evidence Profile for Windows 2000 was relatively incomplete (what data there was having been collected by the safety team from public sources), and anecdotal evidence gathered during the early system development suggested that the product was not proving to be particularly reliable in the type of general office environment for which it was originally designed. The following paragraphs describe the evidence generated for Windows 2000.

The Operating System functionality was limited (‘locked down’) to that actually required to support the applications, with all instances of the operating system replicated from the build “master” image.

An early development was a safety-related application running on a stand-alone PC under Windows 2000. This was soak tested using a period of continuous operation involving 14 PCs for over one hundred days (each) of continuous use with various stages of software build, ranging from Windows only to the full software application. A total of 36,500 hours were logged, demonstrating the reliability of the operating system in the intended environment.

During CCS development testing, Windows 2000 was deployed on approximately 70 servers and workstations in the IBM test and training systems. Taking account of outages (e.g. during software re-loads and moving the training system to another location) and the actual system usage (hands-on) time, usage time was estimated to be 280,000 hours.

A number of observations were raised regarding ‘blue screens’ being observed on workstation displays. All of these instances related to a known problem that occurred only at reboot and was therefore not safety related. The problem was caused by a fault in the implementation of the Security Enhancements for Windows and was corrected by the incorporation of a Windows 2000 SE Update by the supplier.

The data collected allowed the IBM safety team to calculate, for the Windows 2000 build used on UCMP and for the operating environment, a high Mean Time Between Failures (MTBF) to 90% confidence1.

1 All MTBF estimates calculated using the STGARM RDEMO® software, v2.1.
6.3.3 Map Processing Software Toolset

The COTS Evidence Profile for this product indicated that it had a wide user base, including in similar environments to UCMP, and was mature and reliable. However, the version fielded for the UCCS was a new version, containing 2 additional functions with no field history. An IBM generated derived safety requirement required that additional testing of these features be carried out to ensure their accuracy and reliability, in addition to the normal range of functional tests.

The software was installed on all operator and maintainer workstations in the IBM test and training systems. Integration and system testing revealed 3 non-safety related software failures. All 3 were resolved by the supplier. One of the problems resulted from an incompatibility with Windows 2000.

Usage data was also collected for the tool, with over 47,000 hours in total usage time being recorded. This gave a high estimated MTBF to 90% confidence.

6.3.4 Database Frontend Application

Very little evidence was provided by this supplier. As for the map processing software toolset, the software was installed on all operator and maintainer workstations in the IBM test and training systems. Integration and system testing also revealed 3 software failures, one of which was deemed safety related.

The 2 non-safety related problems were not resolved by the supplier, who viewed them as anomalies rather than defects. RAF operator training had therefore to be amended to mitigate these issues.

The safety related problem was confirmed and fixed by the supplier, with the fix being included in a major new version of the application. Consequently, the CCS was migrated to this new version and retested.

Usage data for the new version of the product was also collected, with over 33,000 hours in total run hours being recorded and an acceptable MTBF calculated. One further safety-related defect was raised late in the test programme. To correct the performance anomaly and to provide appropriate mitigation, IBM had to implement code in the Human Computer Interface (HCI) which detected and resolved any occurrence of the restriction.
6.4 User Testing

6.4.1 Introduction

A specific safety requirement was for the conduct of a large-scale HCI tests. The scope of this testing was agreed jointly between IBM and MOD during the Demonstration and Manufacture Phase of the programme.

The requirement recognised the limitations of formal testing, whereby specific inputs are made to generate expected specific outputs. Such tests are generally unrepresentative of actual system use during operations.

IBM therefore designed a large-scale user test, primarily aimed at gathering safety evidence for the UCMP human factors argument.

6.4.2 Evidence

In conjunction with RAF operational experts from the Air Warfare Centre, a challenging set of peacetime scenarios were devised and created as simulation exercises. Experienced RAF operators – typically instructors or carrying a ‘Combat Ready’ qualification – were trained on the system. They were then asked to assess the safety of the system, and the HCI in particular, as it was used by other groups of operators to perform their normal tasks. (These too were required to provide assessments.) IBM also performed its own observations and collected various types of data for analysis.

The evidence collected during this testing was significant in the weight that it carried in the Safety Case Part 2. Although primarily aimed at generating evidence for the human factors argument, by extensively exercising all system components and applications together in a realistic and challenging operational environment, evidence could be generated for most elements of the system, including COTS software.

6.5 Other Aspects

Of the 4 COTS products examined in this case Study, one (Windows 2000) was selected in part due to the generation of field evidence for an alternative product (NT4.0) by IBM.

IBM did not perform any reverse-engineering to obtain evidence as to the safety of COTS products.

The extent of the validation of the supplier evidence is discussed in the final part of Section 5. Given the varying degrees of supplier evidence provided, the
approach taken was to gain an adequate degree of confidence that sufficient safety evidence could be generated by IBM (or for the VCS, its subcontractor).
7 EVALUATION OF POSSIBLE PROCESS FOR SELECTING A COTS SOFTWARE COMPONENT

7.1 Summary of IBM Selection Process for COTS Software Products in UCMP

The IBM selection process for software COTS products in UCMP is summarised below. The process was underpinned by 3 basic drivers:

- A high level policy regarding the use of COTS and the re-use of software was formulated as a result of initial requirements analysis. It can be envisaged that this policy might be not to use COTS at all, but for UCMP the policy was to maximise the use of proven COTS software products.

- That evidence provided by COTS suppliers would be inadequate for the purposes of satisfying the safety case. The bulk of the evidence would have to be generated by IBM (or its agents).

- The refinement of the design and the achievement of greater granularity of components occurs throughout the early phases of a project. In conjunction with this activity, client requirements are analysed and additional (‘derived’) requirements are identified by IBM. (E.g, safety analysis produces ‘Derived Safety Requirements’.) The opportunities to use COTS products are identified and assessed throughout these activities, rather than at any single point or during a single phase.

Building out from the initial high level policy, selection criteria for COTS products are formulated. For UCMP, these were in part derived from specific MOD requirements for the use of COTS, but also based on the expectation that IBM would have to generate the bulk of the safety evidence. The criteria are therefore concerned with both product suitability, and to control the risk that suitable evidence cannot be generated for the product. Products that meet at least most of these criteria are viewed as potential candidates for safety analysis and evidence generation.

As software components are identified and their attributes (functional and non-functional requirements) defined, the possibility of using COTS products is considered. This decision making process can then be summarised as:

- High level market research, to identify potential COTS products;

- Initial assessment of suitability against requirements. This assessment tends to be a Go/No Go decision point for the use of COTS versus a bespoke development. If a bespoke development is selected, then the decision tends not to be revisited.
If COTS software is deemed suitable, then more detailed market research is performed, including supplier demonstrations etc. Evidence is requested from the potential suppliers, in response to e.g a COTS Evidence Profile or a Request For Proposal. A cost-benefit analysis is performed and the results documented. This analysis includes an initial safety assessment of the suitability of the product, which is recognised as superficial given the inadequacy of the evidence available at that time; together with a judgement regarding whether or not sufficient evidence can be generated, and the likely cost of this activity. A COTS product may then be selected as a result of this analysis.

Once selected, the software product is then subjected to detailed analysis by the safety team, and the necessary evidence generation processes set up. Analysis of the evidence being generated may result in the COTS product being deselected.

7.2 Evaluation of Working Paper Section 5 – A Process for Selecting a COTS Component

7.2.1 Introduction

The Working Paper (Ref [1]) proposes a process for selecting a COTS product for a safety-critical system (see Section 5 of Ref [1]). The evaluation of this approach comes from the perspective of experience gained only with safety-related systems, and this should be borne in mind when considering this evaluation.

7.2.2 Comments

Our first discussion point for consideration is the starting point of the process. The process assumes that a decision to consider the use of a COTS product is made, followed by the definition of a reference COTS component. In practical terms we believe that a degree of component definition must occur before any decision to use a COTS product is made, as the component attributes (even if poorly defined at this stage) may drive the decision to consider the use of COTS.

We would suggest therefore, that the first stage of the process - definition of a reference COTS component (Ref [1] Section 5.2.1) – should recognise that this step is unlikely to start from a blank sheet of paper, as current wording implies.

Our second point for consideration refers to the assumption that the use of a COTS component implies regular change and upgrade to the COTS component. From the perspective of UCMP, a policy decision was made that once the system was endorsed, the versions of COTS software products would be frozen unless there was very good reason to change them (e.g. identification of a defect with a safety impact), due to the significant cost of re-validating the system. The stability
of selected COTS products was not just an IBM consideration for UCMP, but was also an MOD requirement.

There is a suggestion (see Ref [1] Section 5.2.2) that safety requirements are not elicited until the second stage of the process. Although it is accepted that the software safety analysis will not occur until this phase, it seems counter-intuitive to suggest that safety requirements or safety attributes are also not being included as part of the first stage – the definition of a reference COTS component.

The described process is comprehensive. The one area of weakness, in our opinion, would appear to be the third stage, the Component Criticality Analysis. Three methods are suggested. The recommend one (considering the degree of protection afforded to each of the failure modes of the reference COTS component), would appear from our experience to suffer from a timing precondition and this is address further in the next comment. The other two methods have other weaknesses that are described in the Working Paper.

A general comment, which is identified in this section and which we would emphasis, is the difficulty in timing. Project activities are typically overlapping or paralleled in order to meet contracted timescales, and for a large system we would not be surprised to find COTS selection occurring prior to the completion of the identification of all system hazards. In the specific instance of UCMP, system hazard analysis comprised the analyses shown in Diagram 7-1 below (with a blue background), and elements of some of these were not completed until well after Critical Design Review. Thus the suggestion that the recommended method of criticality analysis will require the identification of all system hazards as a precondition, seems to impose a timing condition which will not necessarily be met in the real world. It would be informative to attempt to plot the guideline for this process against the actual UCMP achieved project milestones, to assess a fit against a real world project.
Based on UCMP experience in the safety related domain, it is unlikely that COTS suppliers and third parties will supply sufficient information to contribute meaningfully to the process described in the Working Paper. However, we still believe the process has considerable merit both in assisting the selection of COTS items, and also in providing a less subjective approach to the determination of the quality of the safety evidence that must be generated.
# 8 LIST OF REFERENCES

<table>
<thead>
<tr>
<th>Reference Number</th>
<th>Title</th>
<th>Version Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Memo from Catherine Menon (University of York) to Andy Brown (IBM)</td>
<td>10 Feb 2009</td>
</tr>
<tr>
<td>3</td>
<td>Air Surveillance and Control System (ASACS) Safety Case</td>
<td>3GP/212056/ATC Version 1.0, 21 Jun 01</td>
</tr>
<tr>
<td>4</td>
<td>DEF STAN 00-55, Parts 1 and 2 (Requirements for Safety Related Software in Defence Equipment)</td>
<td>Issue 2</td>
</tr>
<tr>
<td>5</td>
<td>DEF STAN 00-56, Parts 1 and 2 (Safety Management Requirements for Defence Systems)</td>
<td>Issue 2</td>
</tr>
</tbody>
</table>

# 9 LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCS</td>
<td>Air Command and Control System</td>
</tr>
<tr>
<td>ALARP</td>
<td>As Low As Reasonably Practical</td>
</tr>
<tr>
<td>ASACS</td>
<td>Air Surveillance and Control System</td>
</tr>
<tr>
<td>ASSU</td>
<td>ASACS Safety and Standards Unit</td>
</tr>
<tr>
<td>ATC</td>
<td>Air Traffic Control</td>
</tr>
<tr>
<td>CCS</td>
<td>Command and Control Subsystem</td>
</tr>
<tr>
<td>CLS</td>
<td>Contractor Logistic Support</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial Off The Shelf</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>D&amp;M</td>
<td>Demonstration and Manufacture</td>
</tr>
<tr>
<td>DEF STAN</td>
<td>Defence Standard</td>
</tr>
<tr>
<td>DPA</td>
<td>Defence Procurement Agency</td>
</tr>
<tr>
<td>DRACAS</td>
<td>Defect Reporting and Corrective Action System</td>
</tr>
<tr>
<td>DSR</td>
<td>Derived Safety Requirement</td>
</tr>
<tr>
<td>FIADGE</td>
<td>Falkland Islands Air Defence Ground Environment</td>
</tr>
<tr>
<td>FPDS</td>
<td>Flight Plan Dissemination System</td>
</tr>
<tr>
<td>GFE</td>
<td>Government Furnished Equipment</td>
</tr>
<tr>
<td>HCI</td>
<td>Human Computer Interface</td>
</tr>
<tr>
<td>HF</td>
<td>Human Factors</td>
</tr>
<tr>
<td>IPT</td>
<td>Integrated Project Team</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>MTBF</td>
<td>Mean Time Between Failures</td>
</tr>
<tr>
<td>MST</td>
<td>Multi-Sensor Tracker</td>
</tr>
<tr>
<td>NADGE</td>
<td>NATO Air Defence Ground Environment</td>
</tr>
<tr>
<td>NATINADS</td>
<td>NATO Integrated Air Defence System</td>
</tr>
<tr>
<td>NATO</td>
<td>North Atlantic Treaty Organisation</td>
</tr>
<tr>
<td>NATS</td>
<td>National Air Traffic Services</td>
</tr>
<tr>
<td>RAP</td>
<td>Recognised Air Picture</td>
</tr>
<tr>
<td>SIL</td>
<td>Safety Integrity Level</td>
</tr>
<tr>
<td>SOW</td>
<td>Statement of Work</td>
</tr>
<tr>
<td>SRD</td>
<td>System Requirements Document</td>
</tr>
<tr>
<td>SSEI</td>
<td>Software Systems Engineering Initiative</td>
</tr>
<tr>
<td>STRIDA</td>
<td>Systeme de Transmission et de Representation des Information de Defense Arienne</td>
</tr>
<tr>
<td>UK ASACS</td>
<td>UK Air Surveillance and Control System</td>
</tr>
<tr>
<td>UK ASCACCS</td>
<td>UK Air Surveillance, Command and Control System</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>UCCS</td>
<td>UK ASACS Command and Control System</td>
</tr>
<tr>
<td>UCMP</td>
<td>UKADGE Capability Maintenance Programme</td>
</tr>
<tr>
<td>UKADGE</td>
<td>United Kingdom Air Defence Ground Environment</td>
</tr>
<tr>
<td>VCS</td>
<td>Voice Communications Subsystem</td>
</tr>
</tbody>
</table>
Annex A  UCMP COTS Evidence Profile Proforma

**UCMP COTS EVIDENCE PROFILE:** < COTS Product >

**PROFILE REFERENCE:** < profile ref. >  | **VERSION:** < version >

---

**Part 1 – Product Summary**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Product Evidence of Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturer</td>
<td></td>
</tr>
<tr>
<td>Product</td>
<td></td>
</tr>
<tr>
<td>Version / Release</td>
<td></td>
</tr>
<tr>
<td>Service Upgrades</td>
<td></td>
</tr>
<tr>
<td>Def Stan 00-56/2 Development Summary</td>
<td>&lt; provide reference to development summary &gt;</td>
</tr>
<tr>
<td>Conformance to Standards</td>
<td>&lt; provide evidence of conformance to recognised standards and document any non-compliances &gt;</td>
</tr>
<tr>
<td>User Base</td>
<td>&lt; provide evidence of large user base &gt;</td>
</tr>
<tr>
<td>Functional Stability</td>
<td>&lt; provide evidence of functional stability &gt;</td>
</tr>
<tr>
<td>Proven History in Similar Applications</td>
<td>&lt; provide evidence of a proven history of successful operation in similar applications &gt;</td>
</tr>
<tr>
<td>Supportability</td>
<td>&lt; provide evidence of product supportability throughout the life of UCMP &gt;</td>
</tr>
<tr>
<td>Quality Management System</td>
<td>Refer to Part 2 - Development Summary (ISO 9001)</td>
</tr>
<tr>
<td>Development Process</td>
<td>Refer to Part 2 - Development Summary</td>
</tr>
<tr>
<td>Configuration Management</td>
<td>Refer to Part 2 - Development Summary (Configuration Management)</td>
</tr>
<tr>
<td>Problem Reporting</td>
<td>Refer to Part 2 - Development Summary (DRACAS)</td>
</tr>
<tr>
<td>Detected Faults during Formal Testing</td>
<td>&lt; provide a reference to detected faults during formal integration, system and acceptance testing &gt;</td>
</tr>
</tbody>
</table>
### Part 2 – Development Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>SIL S1 Req.</th>
<th>Applicability</th>
<th>Product Development Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements &amp; Design Specification</td>
<td>Informal (Natural Language)</td>
<td>Hardware/Software</td>
<td></td>
</tr>
<tr>
<td>Configuration Management</td>
<td>Manual</td>
<td>Hardware/Software</td>
<td></td>
</tr>
<tr>
<td>Prototyping</td>
<td>Optional</td>
<td>Hardware/Software</td>
<td></td>
</tr>
<tr>
<td>Structured Design Method</td>
<td>Optional</td>
<td>Hardware/Software</td>
<td></td>
</tr>
<tr>
<td>Design Reviews</td>
<td>Yes (Peer Reviews)</td>
<td>Hardware/Software</td>
<td></td>
</tr>
<tr>
<td>Project Management</td>
<td>Preferred</td>
<td>Hardware/Software</td>
<td></td>
</tr>
<tr>
<td>Coding Standards</td>
<td>Preferred</td>
<td>Software</td>
<td></td>
</tr>
<tr>
<td>Defensive Programming Techniques</td>
<td>Preferred</td>
<td>Software</td>
<td></td>
</tr>
<tr>
<td>Fault Tolerant Techniques</td>
<td>Optional</td>
<td>Hardware/Software</td>
<td></td>
</tr>
<tr>
<td>Approved/Validated Compiler</td>
<td>Yes</td>
<td>Software</td>
<td></td>
</tr>
<tr>
<td>Approved/Validated Software Support Tool</td>
<td>Optional</td>
<td>Software</td>
<td></td>
</tr>
<tr>
<td>Independent Technical Assessment</td>
<td>Optional</td>
<td>Hardware/Software</td>
<td></td>
</tr>
<tr>
<td>Data Reporting and Corrective Action System (DRACAS)</td>
<td>Yes</td>
<td>Hardware/Software</td>
<td></td>
</tr>
<tr>
<td>Static Analysis</td>
<td>Optional</td>
<td>Hardware/Software</td>
<td></td>
</tr>
<tr>
<td>Dynamic Analysis (e.g. automatic testing)</td>
<td>Yes</td>
<td>Software</td>
<td></td>
</tr>
<tr>
<td>Independent Testing</td>
<td>Optional</td>
<td>Hardware/Software</td>
<td></td>
</tr>
<tr>
<td>Environmental / Operational Testing to Def Stan 00-35</td>
<td>Optional</td>
<td>Hardware</td>
<td></td>
</tr>
<tr>
<td>Computer Aided Simulation of Component Tolerances</td>
<td>Optional</td>
<td>Hardware</td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>SIL S1 Req.</td>
<td>Applicability</td>
<td>Product Development Process</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>-------------</td>
<td>-----------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Additional Production Vigilance (e.g. Independent Auditing)</td>
<td>Optional</td>
<td>Hardware/Software</td>
<td></td>
</tr>
<tr>
<td>ISO 9001</td>
<td>Yes</td>
<td>Hardware/Software</td>
<td></td>
</tr>
</tbody>
</table>