RiSE Project: Towards a Robust Framework for Software Reuse

Eduardo Santana de Almeida[1], Alexandre Alvaro[1], Daniel Lucrédio[2], Vinicius Cardoso Garcia[2], Silvio Romero de Lemos Meira[1]
[1]*Federal University of Pernambuco and Recife Center for Advanced Studies and Systems*
[2]*Federal University of São Carlos*
{esa2, aa2, srlm}@cin.ufpe.br, {lucredio, vinicius}@dc.ufscar.br

Abstract

Software reuse is a critical aspect for companies interested in the improvement of software development quality and productivity, and in costs reduction. However, achieving it is a non-trivial task. In this paper, we present a robust framework for software reuse, based on previous success factors, in order to guide organizations in the effective reuse. Non-technical and technical aspects compose the framework.

1. Introduction

Systematic software reuse is a technique that is employed to address the need for improvement of software development quality and efficiency [1]. Quality could be improved by reusing all forms of proven experience, including products and processes, as well as quality and productivity models. Productivity could increase by reusing existing experience, rather than creating everything from the beginning [2].

There is a vast literature on reuse, including efforts involving domain engineering, component-based development, and, currently, product-lines. However, these focus on isolated reuse aspects without considering the interactions among them. On the other hand, outside the academia, several success factors and best practices may be found, mainly related to previous experiences, success and failure models, myths and inhibitors. Often, these are also not generic enough to be applicable outside its original context.

This is our motivation to develop a robust framework for software reuse. We have surveyed some of the existing works for software reuse success, and based on their strong and weak points, we propose our framework, which attempts to group all the different aspects of reuse, including those pointed out inside and outside academia, in order to guide organizations in the adoption of a reuse program.

2. A Brief Survey on Software Reuse

The idea of software reuse is not new. In 1968, during the NATO Software Engineering Conference, generally considered the birthplace of the field, the focus was the software crisis – the problem of building large, reliable software systems in a controlled, cost-effective way. From the beginning, software reuse was touted as a means to overcome the software crisis. An invited paper at the conference: "*Mass Produced Software Components*" by McIlroy [3], ended up being the seminal paper on software reuse. In McIlroy's words: "*the software industry is weakly founded and one aspect of this weakness is the absence of a software component sub-industry*" (pp. 80), which is a starting point to investigate mass-production techniques in software, in a comparison with the construction industry.

For almost four decades, extensive research tries to follow the fuzzy map of the area. The results have been presented in many forums, such as Software Engineering and Software Reuse Conferences, as well as in many journals.

This research material contains important information, which may lead the way in solving the reuse problem. However, one of the reasons that may explain the gap between McIlroy's ideas and the current state-of-the-art is, ironically, presented by McIlroy himself: "*To develop a useful inventory, money and talent will be needed. Thus, the whole project is an improbable one for university research*" (pp. 84).

2.1. Success Factors in software reuse

Because software reuse provides a competitive advantage, there is little incentive to share the learned lessons across companies. Moreover, there has been little research to determine if an individual company's software reuse success factors are applicable to other organizations. Thus, this section presents and discusses the main success factors related to software reuse available in the literature, and try to establish a relationship among them.

0-7803-8819-4/04/$20.00 ©2004 IEEE.

2.1.1. Informal Research. During more than one decade, companies wanted to achieve systematic reuse. They pursued a domain focused, repeatable process, primarily concerned with the reuse of higher level lifecycle artifacts, such as requirements, design models and source code. The following works represent some attempts of researchers aiming to obtain systematic reuse:

i. According to Frakes [4], a key concept in systematic reuse is the domain, which may be defined as an application area or, more formally, a set of systems that share design decisions.

Frakes relates that there are six critical factors for systematic software reuse: *Management*, *Measurement*, *Legal issues*, *Economics*, *Design for reuse* and *Libraries*.

ii. Glass [5] presents and discusses the question that software reuse has problems when put in practice. Glass considers that, if there is a motherpie-and-applehood topic in software engineering, reuse is such topic. His point of view is that software reuse has good potential, but this has not been achieved yet. While other researchers point to factors such as management and culture, *Glass believes that the main problem is that there are not many software components that can be reused.*

2.1.2. Empirical Research. There is little empirical research about software reuse. In this section, we present the three main studies available in the literature.

i. Rine [6] researched the fact that there is no set of success factors that are common across organizations and have some predictability relationships to software reuse.

In this way, in 1995, a survey was conducted to investigate which software reuse success factors are applicable across all application domains. The organizations that achieved the highest software reuse capability had the following characteristics: *product-line approach, architecture which standardizes interfaces and data formats, common software architecture across the product-line, design for manufacturing approach, domain engineering, reuse process, management which understands reuse issues, software reuse advocate(s) in senior management, state-of-the-art reuse tools and methods, experience in reusing high level software artifacts such as requirements and design, instead of just code reuse, and the ability to trace end-user requirements to the components.*

ii. To assess the market for component-based software engineering, the Software Engineering Institute (CMU/SEI) studied industry trends in the use of software components. The study [7], conducted from September 1999 to February 2000, examined software components from both technical and business perspectives.

The report concluded that the market perceives the following key inhibitors for reuse adoption, in decreasing order of importance: *lack of available components, lack of stable standards for component technology, lack of certified components, and lack of an engineering method to consistently produce quality systems from components.*

iii. The Morisio's research [8], maybe the most detailed practical study about software reuse available in the literature, presents and discusses some of the key factors in adopting or running a company-wide software reuse program. The key factors are derived from empirical evidence of reuse practices, as emerged from a survey of projects for the introduction of reuse in European companies: 24 such projects conducted from 1994 to 1997 were analyzed using structured interviews.

The projects were undertaken in both large and small companies, working in a variety of business domains, and using both object-oriented and procedural development approaches. Most of them produce software with high commonality between applications, and have at least reasonably mature processes.

According to Morisio's research, successes were achieved when, given a potential for reuse because of commonality among applications, *management committed to introducing reuse processes, modifying non-reuse processes, and addressing human factors.*

iv. In [9], Rothenberger investigates the premise that the likelihood of success of software reuse efforts may vary with the reuse strategy employed and, hence, potential reuse adopters must be able to understand reuse strategy alternatives and their implications.

In order to engage the reuse phenomenon at a level closer to the organizational reuse setting, the author focuses on the systematic reuse of software code components. Rothenberger used survey data collected from 71 software development groups to empirically develop a set of six dimensions that describe the practices employed in reuse programs.

According to study, the high level of reuse is associated with the high levels of: *planning and improvement, formalized process, management support, project similarity, and common architecture.*

3. Towards a Robust Framework for Software Reuse

With basis on the survey and the identified success factors, we propose a framework to enable the adoption of a reuse program. The proposed framework (Figure 1) has two layers. The first layer (on the left) is formed by best practices related to software reuse. Non-technical factors, such as education, training, incentives, and organizational management are considered. This layer constitutes a fundamental step before the introduction of the framework in the organizations. The second layer (on the right) is formed by important technical aspects related to software reuse, such as processes, environments, and tools.

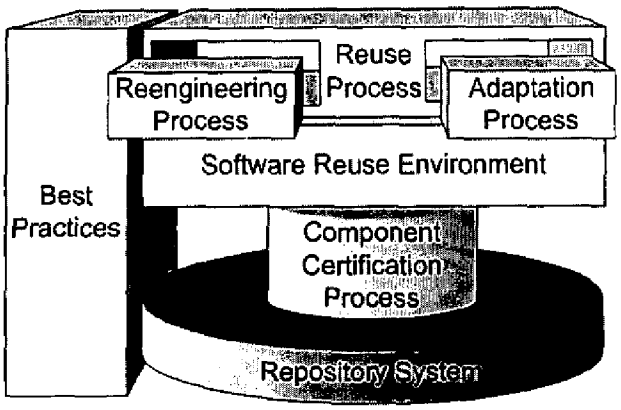

Figure 1. Framework for Software Reuse.

This framework constitutes a solid basis for organizations that are moving towards an effective reuse process. Its elements not only help the organization in adopting reuse, but also guide it in the migration process, reducing its risks and failure possibilities. Next we present each element of the framework.

3.1. The Software Reuse Process and Best Practices

Since McIlroy's work, the goal of research in the area of software reuse is to develop and support systematic approaches for effectively reusing existing assets, in order to maximize quality and productivity. Although successful software reuse experiments are increasingly common, success is not the norm. Software reuse is not a matter of routine practice, the promises of software reuse remain for the most part unfulfilled, and a number of issues remain worthy of further research [10].

A number of different reuse approaches have been presented in the literature, considering aspects such as libraries, domain engineering, and, currently, product lines. However, there are still some limitations, including:

i. How assets are represented. An asset that embodies a function should be represented by a function that abstracts its most relevant functional (semantic) properties, whereas an asset that embodies a structure should be represented in a way that highlights its relevant structural (syntactic) properties. Current research on reusable assets does not properly acknowledge this distinction and its impact.

ii. How assets are developed. Design for black-box reuse is primarily a specification issue and is determined by the generality of its implementation. Design for white-box reuse, on the other hand, is primarily determined by design issues, such as modularity, simplicity, and structuredness. Current design life cycles do not make this distinction. Moreover, the development for reuse processes present gaps among the phases of analysis, design, and implementation. According to Szyperski [11], assets such as components, for example, are already a reality at the implementation level, and now the concept must be found at the earlier phases of the development lifecycle. Doing so, reuse principles and concepts should be consistently applied through the whole development process, and consistently followed from one phase to the other.

iii. How assets are reused. The development with reuse activity is also very important for software reuse. Methods to search assets, make adaptations, and integrate them into current project are needed. Thus, this activity must be well integrated with the development with reuse. Current reuse processes have presented few advances towards this integration.

iv. Metrics. As in any engineering activity, measurement is vital for systematic reuse. In general, reuse benefits (improved productivity and quality) are a function of the reuse level achieved. A reuse process must define what, where, and when to measure, however, reuse processes with a metric program are not also explored.

v. Costs. Economics considerations are an important aspect of reuse, since most decisions that arise in software reuse can and must be justified by some scale economic. Nevertheless, even with important works in this area, such as [12], current reuse processes do not consider cost aspects as, for example, ways to estimate the cost of development for reuse considering domain engineering or product lines.

vi. Lack of best practices. Section 2 presented a brief survey about software reuse. In this survey, important non-technical aspects related to software reuse were identified, such as education, training, incentives, and organizational management. Current reuse processes have explored few these aspects and considered them as essential for software reuse success.

These six issues are being investigated and analyzed by our research group, in order to be incorporated into an effective reuse process. Moreover, two other essential processes compose our reuse process: reengineering and adaptation.

3.1.1. Reengineering Process. Software reengineering is being used to recover legacy systems and allow their evolution [13]. It is performed mainly to reduce maintenance costs, improve development speed and systems readability.

However, the importance of reengineering goes beyond such technical aspects. Legacy systems represent much of the knowledge produced and maintained by an organization, who cannot afford to lose it. Thus, reengineering allows this knowledge to be reused, in the form of reconstructed code and documentation.

There are two major stumbling blocks to the practice of reengineering. One is the lack of support tools to aid the entire process. The other is the lack of a well-defined process for planning an effective approach for

reengineering. The goals in developing such a process are [14]:

- Provide a flexible approach to reuse knowledge;
- Improve commonality and modularity across systems;
- Reduce cost and schedule;
- Facilitate hardware/software evolution;
- Improve the system understanding; and
- Reduce long-term maintenance costs.

Another important characteristic of a reengineering process is its adaptability, i.e., its suitability to different situations. Each organization has its own legacy system, written in different programming languages, with different tools, platforms and different people involved. Thus, a reengineering process must accommodate these particularities, in order to be industrially applicable.

We are currently developing the reengineering process, focused in reusing the knowledge embedded in legacy systems, in order to help organizations in adopting an effective reuse process without losing its past experience. The main objective of our reengineering process is to promote reuse by extracting high-modularized, low-coupled reusable assets, through the separation of concerns concept [15].

There are six main steps in the process, which were defined after a wide study of the related literature. The initial steps (i-iv) compose the reverse engineering, and the followings steps (v-vi) the forward engineering.

i. Define objectives. A strategic planning, that establishes the main objectives of the reengineering process, must be defined before its execution.

ii. Analyze legacy system. In order to capture the information that will be used in the reengineering, it is necessary to perform a full analysis of the system elements. There are a variety of elements to be collected, including: programs, library routines, business rules, informal documentation, models, user manuals, history records, among others. These constitute the knowledge that must be reused.

iii. Organize and decompose. Since the reengineering product must be prepared for reuse, an intermediate step, which organizes the recovered information in order to increase its modularity, is performed. This facilitates the task of recovering the documentation and reconstructing the system, which are performed in the next steps.

iv. Reconstruct Documentation. Reuse success depends on the ability of correctly determining if some asset can or cannot be reused in some situation. In order to achieve this, good documentation is essential to inform to the Software Engineer about the functionality contained in some asset. Hence, the reengineering process must obtain such documentation from legacy systems, where they are often of poor quality or inexistent.

v. Redesign. After organizing and recovering the legacy system assets, the Software Engineer may introduce new design specifications, in order to make them more reusable and/or introduce new functionalities.

vi. Reimplementation. Finally, the knowledge recovered from the legacy system is materialized in executable assets, which can be directly reused.

The reengineering process is currently under development, with Aspect-Oriented Software Development techniques [15]. Its main steps and guidelines have been already specified. Next, case studies will be applied in order to refine it and evaluate its validity within a reuse adoption program.

3.1.2. Adaptation Process. Many organizations are actively pursuing software process maturity, in order to increase their competitiveness. Usually, they use measures such as the Capability Maturity Model (CMM) to structure their process improvement initiatives. One method often cited for accelerating process improvement within an organization is to replicate a standard, reusable organizational process within other projects. Hollenbach defined this approach as a reusable process [16]. According to Hollenbach, process reuse is the use of one process description in the creation of another process description. This should not be confounded with multiple executions of the same process on a given project.

However, this method presents some problems when the boundary to reuse the process is not the organization, but other software factories. Moreover, the problem is still higher when the process in question is a software reuse process. Thus, what is needed is an effective process to capture guidelines that lead the adaptation of an organization's process to become reuse-centered.

We believe that the initial Hollenbach's work can offer some directions to define the aspects of the adaptation process. However, we deem that the formalization of a Reuse Maturity Model (RMM) with levels of reuse and key practices can offer important insights. Thus, organizations can be classified according to RMM levels and the process can be adapted more accurately. Currently, studies are being made toward the RMM. After that, the adaptation process will be developed.

3.2. Software Reuse Environment

CASE tools have always helped in software development, providing improved productivity and quality. However, the benefit of CASE is bigger when in the form of integrated environments, with several tools that add its own contribution to the process, guiding the Software Engineer throughout the whole lifecycle.

With software reuse, it could not be different. Inside a Software Reuse Environment, tools are used in the development and reuse of assets. There are some issues that must be considered inside a Software Reuse Environment:

i. Distinction between development "for reuse" and development "with reuse". These are different activities, which must have distinct tool support.

ii. Tool integration. The tools of the environment must have different levels of integration [17], including data and presentation integration, but mainly process integration, i.e., they must be focused on a single process, in this case, a reuse process.

iii. Reusability. Not just code, but every kind of asset must be reused, including models, test cases, requirements, among others.

iv. Referential Integrity. Reuse often causes assets to reference each other. In order to avoid confusion, these references must be correctly managed.

v. Software Configuration Management. As happens in any development environment, the control of the software evolution reduces the risks in development and maintenance, and so this is a major requirement in a reuse environment.

vi. Technology and language independence. The environment must not be fixed in a single technology or language, or else it would be of little use to a software factory, which must be able to use any technology and/or language in its projects.

We are currently developing a prototype environment, composed by some tools that focus on Component-Based Software Engineering [18], which will serve as a basis to give automated support for the reuse process.

3.3. The Component Certification Process

In a software reuse process, it is important to assure quality in the assets that will be stored into repository systems. According to Wohlin [19], components that are to be retrieved from a repository must have a quality stamp in terms of what level of reliability can be expected from them as they are put into a system. Thus, the certification stands out as an essential area to evaluate the component reliability level.

In the literature, we found some divergent definitions about component certification. However, Council [20] established a satisfactory definition about it:

"Third-party certification is a method to ensure that software components conform to well-defined standards; based on this certification, trusted assemblies of components can be constructed".

Besides, certification should provide evidence that the component is adequate to fulfill a given set of requirements. In other words, a component is certifiable if it has properties that can be demonstrated, in an objective way, to adhere to the component. Any such property can be the subject of certification. However, the component properties should be described in sufficient detail, and with sufficient rigor, to enable their certification [21].

Nevertheless, certification can face some difficulties, particularly due to the relative novelty of this area. To increase this problem, the reuse community is still far from reaching a consensus on how certification should be carried out, what its requirements should be or even on who should perform it [22].

In this way, we are investigating the component certification area in order to define a component quality model (analyzing the key component requirements), define a metrics framework to track the properties of the components and establish a certification process to group the concepts. The plan is, clearly, to develop it to the point where it can be used as a component certification standard to a software factory, in order to achieve a Components Certification Center.

3.4. The Repository System

The software development process involves great amount and variety of information. These include requirements, graphical analysis and design models, annotations, drafts, and executable code. This information constitutes an important knowledge basis of the software development process, influencing directly in its quality.

In this sense, repository systems play a central role in reuse. By storing and managing previously built reusable assets, development teams may take advantage of this knowledge in new projects, saving effort and time.

We are developing a repository system based on the XMI standard [23], which allows any kind of metadata to be represented, instead of just executable code. It also provides some semantic information, which is an important requirement for reuse.

The repository has three main features:

i. Storage. As any database mechanism, the repository allows assets to be included, removed and updated;

ii. Search. The repository has a search mechanism that speeds up the process of finding reusable assets. There are several issues related to this subject, as we present in [24], such as performance, query formulation, precision and recall ratios, among others, which must be taken into consideration when implementing this kind of search;

iii. Management. This involves the management of the repository users, version control and assets mining. This last item is of great importance. By gathering information from the repository, it is possible to obtain very useful data, such as which assets are being more reused and which developers are achieving more reuse. This data may be used to infer other questions, such as which assets must be modified to become more reusable and how many training the development team needs in order to put reuse into practice, among other information that may define the success or failure of a reuse program.

Our repository is based on CVS, a widely used version control system, which makes it suitable to a large number of companies. We are currently developing the

search mechanism, with metrics to improve the results [25]

4. Concluding Remarks

During the last years, several organizations are trying to achieve software reuse, in order to increase their productivity and quality, and to reduce development time and costs. However, the adoption of a reuse program is a difficult task, with technical and non-technical aspects that must be carefully analyzed and combined.

In this paper, we present a framework for software reuse that guides organizations in the adoption of an effective reuse program. The framework is based on an exhaustive study [26] that identified critical aspects for reuse success. These aspects were integrated in a coherent way, in order to work together. Moreover, another important point is that the framework is being developed in conjunction with a large Brazilian software factory[1], where the results could be applied in an industrial environment. In this sense, we agree with Ali Mili [10], believing that this is fundamental for the research in the software reuse area, where small, non-realistic examples often takes place.

The framework is part of RiSE Project[2] (Reuse in Software Engineering), and is still under development. In future papers, we will explore each framework's element in details.

References

[1] C.W. Krueger, Software Reuse, ACM Computing Surveys, Vol. 24, No. 02, June, 1992, pp. 131-183.

[2] V.R. Basili, L.C. Briand, W.L. Melo, How Reuse Influences Productivity in Object-Oriented Systems, Communications of the ACM, Vol. 39, No. 10, October, 1996, pp. 104-116.

[3] M.D. McIlroy, Mass Produced Software Components, NATO Software Engineering Conference Report, Garmisch, Germany, October, 1968, pp. 79-85.

[4] W.B. Frakes, S. Isoda, Success Factors of Systematic Software Reuse, IEEE Software, Vol. 12, No. 01, January, 1995, pp. 14-19.

[5] R.L Glass, Reuse: What's Wrong with This Picture?, IEEE Software, Vol. 15, No. 02, March/April, 1998, pp. 57-59.

[6] D.C. Rine, Success Factors for software reuse that are applicable across Domains and businesses, ACM Symposium on Applied Computing, San Jose, California, USA, ACM Press, March, 1997, pp. 182-186.

[7] L. Bass, C. Buhman, S. Dorda, F. Long, J. Robert, R. Seacord, K. Wallnau, Market Assessment of Component-Based Software Engineering, Software Engineering Institute (SEI), Technical Report, Vol. 01, May, 2000.

[1] Currently, this company has about 270 employees and is in preparation to obtain the CMM level 3.

[2] http://www.cin.ufpe.br/~rise

[8] M. Morisio, M. Ezran, C. Tully, Success and Failure Factors in Software Reuse, IEEE Transactions on Software Engineering, Vol. 28, No. 04, April, 2002, pp. 340-357.

[9] M.A. Rothenberger, K.J. Dooley, U.R. Kulkarni, N. Nada, Strategies for Software Reuse: A Principal Component Analysis of Reuse Practices, IEEE Transactions on Software Engineering, Vol. 29, No. 09, September, 2003, pp. 825-837.

[10] A. Mili, S. Yacoub, E. Addy, H. Mili, Toward an Engineering Discipline of Software Reuse, IEEE Software, Vol 16, No. 05, September/October, 1999, pp. 22-31.

[11] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, 2nd Edition, ACM Press, 2002.

[12] J.S. Poulin, *Measuring Software Reuse: Principles, Practices, and Economic Models*, Addison-Wesley, 1997.

[13] I. Jacobson, F. Lindstrom, Reengineering of old systems to an object-oriented architecture, Proceedings of the OOPSLA. ACM Press, 1991. pp. 340–350.

[14] J. Bergey et al., Why Reengineering Projects Fail, Technical Report CMU/SEI-99-TR-010, April, 1999.

[15] G. Kiczales et al., Aspect-Oriented Programming. Proceedings of the 11st ECOOP. Springer Verlag, 1997. (LNCS, v. 1241), pp. 220-242.

[16] C. Hollenbach, W.B. Frakes, Software Process Reuse in an Industrial Setting, Proceedings of the 4th International Conference on Software Reuse, IEEE Press, 1996, pp. 22-30.

[17] I. Sommerville, *Software Engineering*, 6 ed: Pearson Education, 2000.

[18] D. Lucrédio, E. S. Almeida, C. P. Bianchini, A. F. Prado, L. C. Trevelin, Orion - A Component Based Software Engineering Environment, JOT - Journal of Object Technology, Vol. 03, 2004, pp. 51-74.

[19] C. Wohlin, B. Regnell, Reliability Certification of Software Components, Proceedings of the 5th International Conference on Software Reuse (ICSR), 1998, pp 56-65.

[20] B. Councill, Third-Party Certification and Its Required Elements, Proceedings of the 4th ICSE Workshop on Component-Based Software Engineering (CBSE), 2001.

[21] K. C. Wallnau, Volume III: A Technology for Predictable Assembly from Certifiable Components, Software Engineering Institute (SEI), Technical Report, Vol. 03, April, 2003.

[22] M. Goulao, F. B. Abreu, The Quest for Software Components Quality, Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC), August, 2002, pp. 313-318.

[23] XML Metadata Interchange (XMI) Specification, Object Management Group, 2002.

[24] D. Lucrédio, E. S. Almeida, A. F. Prado, A Survey on Software Components Search and Retrieval, 30th IEEE EUROMICRO Conference, Component-Based Software Engineering Track, Rennes - France, 2004.

[25] D. Lucrédio, E. S. Almeida, A. F. Prado, C. H. Yamamoto, M. Biajiz, Approaches for Efficient Components Recovery (in portuguese), Third Brazilian Workshop on Component-Based Development, São Paulo, Brazil, 2003.

[26] E.S. Almeida, A. Alvaro, S.R.L. Meira, Software Reuse: The State of the Art, 2004. In preparation.