

Towards a Practical and Efficient Process for Software Reuse

Eduardo Santana de Almeida*, Silvio Romero de Lemos Meira (Advisor)

Federal University of Pernambuco and C.E.S.A.R. – Recife Center for Advanced Studies and Systems
Av. Professor Luiz Freire, Cidade Universitária, Recife, Pernambuco, Brasil
Zip Code: 50.740-540, Phone: 55-81-2126-8430
{esa2, srlm}@cin.ufpe.br

Abstract. Software reuse is seen as a key factor for companies interested in the improvements of quality, productivity, and time-to-market. However, without adequate processes, it is too difficult to obtain the desired benefits. Moreover, current reuse processes present lack of details in important activities and are most concentrated in specific stages of the reusable software development process. Thus, this work presents a proposal for a practical and efficient reuse process. The process is composed of guidelines, metrics, economics aspects, and a well-defined set of requirements.

Keywords: software reuse, software reuse processes, domain engineering, software product lines.

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 and productivity 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 using existing experience, rather than creating everything from the beginning [2].

Through the years, several research works, including company reports [3, 4], informal research [5] and empirical studies [6, 7] have shown that an effective way to obtain the software reuse benefits is to adopt a *reuse process*. However, the existing reuse processes present crucial problems, such as gaps in important activities like development *for* and *with* reuse, and putting more emphasis on some specific activities (analysis, design and implementation). Even today, with the ideas of software product lines arising, there is still no clear consensus between the activities (inputs, outputs, artifacts) and the requirements that an effective reuse process must have.

In this context, we agree with Bayer et al. when they said [8] (pp. 122): “*Existing methods have been either not flexible enough to meet the needs of various industrial situations, or they have been too vague, not applicable without strong additional interpretation and support. A flexible method that can be customized to support various enterprise situations with enough guidance and support is needed*”.

Under this motivation, this research proposes a practical and effective process for software reuse. The process will provide a systematic way to develop reusable software through a set of guidelines and tasks to plan, specify, model, design, implement and document components and applications in a problem domain.

This research is part of the Reuse in Software Engineering (RiSE*) project [9]. RiSE’s goal is to develop a robust framework for software reuse, in conjunction with the industry, involving processes, methods, environment and tools.

The main contribution of this work will be the development of a systematic reuse process in order to allow the development of components and applications. The process will be based on the analysis of the state-of-the-art in the area, including processes for domain engineering and software product lines. We believe that the key aspect of the process is the combination of guidelines, reuse metrics, economic aspects, and a set of requirements for an effective software reuse process. The results can assist organizations in developing reusable software to meet their reuse goals.

* 1st year Ph.D. student (start: March 2004)

* www.cin.ufpe.br/~rise

The remainder of this proposal is organized as follows. Section 2 describes background and related works. Section 3 discusses the proposed work and research approach. Section 4 discusses the work status, and, finally, Section 5 presents some concluding remarks.

2 Background and Related Works

This section provides background and related works in the following field: domain engineering and product lines.

2.1 Domain Engineering Processes

Domain engineering is the activity of collecting, organizing, and storing past experience in building systems or parts of systems in a particular domain in the form of reusable assets, as well as providing an adequate way for reusing these assets when building new systems [10].

Among the works of the early 80's and 90's, such as [11, 12, 13, 14, 15, 16], a special focus is placed on the domain engineering processes to develop reusable software.

An example of this work may be seen in [11]. In this work, Neighbors proposed the *first domain engineering approach*, as well as a prototype - Draco - based on transformation technology. The main ideas introduced by Draco include: *Domain Analysis*, *Domain-Specific Languages*, and *Components* as sets of transformations.

Neighbors' work has given an important contribution for the domain engineering field, presenting concepts such as generative programming, transformation systems and components. Nevertheless, his approach is very difficult to apply in the industrial environment due the complexity to perform activities such as writing transformations and using the Draco machine. Even with some advances related to his work, many of these problems still remain unsolved. Thus, even presenting a potential possibility to develop reusable software, the software reuse community and the industry needed a major guidance to achieve this concept in an effective way.

In this context, in 1992 [12], the Software Technology for Adaptable, Reliable Systems (STARS) developed the Conceptual Framework for Reuse Processes (CFRP) as a vehicle for understanding and applying the STARS domain-specific reuse-based software engineering paradigm. The CFRP established a framework for considering reuse-related software engineering processes, how they interrelate, and how they can be integrated with each other and with non-reuse-related processes to form reuse-oriented life-cycle process models that are tailorable to organization needs.

However, the CFRP by itself was a very generic framework and organizations will find the CFRP too generic to serve as the sole basis for developing a tailored domain-specific reuse-based software engineering life cycle model. Thus, the ROSE Process Model (ROSE PM) [12] was developed by the Paramax STARS team, specifically to bridge the gap between the high-level, generic framework and the detailed, prescriptive methods.

The key objective of the ROSE model is to integrate software maintenance and reengineering in the context of domain-specific reuse to produce a general reuse-oriented approach to software evolution. However, the main problem with ROSE is that the process model presents the activities in a generic way, without specifying, in details, how they should be performed. For example, in Domain Engineering, there is an activity defined as *Develop Software Components*, but how to perform it is not defined. The same problem happens in other activities, such as *Develop Software Architecture* and *Develop Applications Generators*.

Four years after the beginning of the efforts in the STARS project, Mark Simos [13] and his group developed the Organization Domain Modeling (ODM) method. Their motivation was that there was a gap in the domain engineering methods and processes available to reuse practitioners.

Although ODM encompasses the steps of domain engineering, it does not present specific details on how to perform many of its activities, as happens in ROSE. However, this limitation is clearly described in the ODM guidebook. According to Simos et al. the method provides a "**general, high-level guidance in tailoring the method for application within a particular project or organization**" (pp. 01). In this way,

critical activities are not properly guided, what can contribute to possible problems for organizations using the method.

Until Simos' work, the research involving software reuse processes were too generic, without presenting concrete techniques to perform tasks such as architecture and component modeling and implementation. Moreover, it was difficult to develop reusable object-oriented software using these processes.

In this context, three software development experts - Jacobson, Griss and Jonsson - created the Reuse-driven Software Engineering Business (RSEB) [14]. RSEB is an use-case driven systematic reuse process based on UML notation. The method was designed to facilitate both the development of reusable object-oriented software and software reuse. Similar to the Unified Process, RSEB is also iterative and use-case centric.

Key ideas in RSEB are: the explicit focus on modeling variability and to maintain traceability links connecting representation of variability throughout all models, i.e., variability present in use cases can be traced to variability in the analysis, design, and implementation object models.

RSEB has separated processes for Domain Engineering and Application Engineering. However, despite the RSEB focus on variability, the process components do not include essential domain analysis techniques such as domain scoping and feature modeling. Moreover, the method does not describe a systematic way to perform the asset development as proposed. Another shortcoming of RSEB is the lack of feature models. In RSEB, variability is expressed at the highest level in the form of variation points, which are then implemented in other models using variability mechanisms.

Based on the limitations presented with the RSEB utilization, Griss et al. developed FeatuRSEB [15], which is a result of integrating FODacom, an object-oriented adaptation of FODA [17] for the telecom domain, into RSEB, through cooperation between Hewlett-Packard and Intecs Sistemi.

Although FeatuRSEB had presented important considerations related to domain analysis, such as the process of extracting functional features from the domain use case model, for example, the limitations discussed in FeatuRSEB were not solved.

Concurrently with Griss et al.'s work, a respected researcher in the domain analysis area, Kyo Kang, in conjunction with his colleagues, presented the thesis that there were many attempts to support software reuse, but most of these efforts have focused on two directions: *exploratory research* to understand issues in domain specific software architectures, component integration and application generation mechanisms; and *theoretical research* on software architecture and architecture specification languages, development of reusable patterns, and design recovery from existing code. Kang et al. considered that there were few efforts to develop systematic methods for discovering commonality and using this information to engineer software for reuse. It was their motivation to develop the Feature-Oriented Reuse Method (FORM) [16], an extension of their previous work [17].

The core of FORM lies in the analysis of domain features and in the use of these features to develop reusable domain artifacts. The domain architecture, which is used as a reference model for creating architectures for different systems, is defined in terms of a set of models, each one representing the architecture at a different abstraction level. Nevertheless, aspects such as components specification, design, implementation and packaging are little explored.

After the domain development, the application engineering process is performed. Once again, the emphasis is in the analysis phase, with the use of the developed features. However, few directions are defined to select the architecture model and to develop the applications using the existing components.

2.2 Product Lines Processes

Until 1998, the software reuse processes were only related to domain engineering issues. However, in this same period, a new trend started to be explored: the software product line area [18]. Software product lines began to be seen as one of the most promising advances for efficient software development. However, until the late 90's, there were few guidelines or methodologies available to develop and deploy product lines beyond existing domain engineering approaches.

In this context, based on the lack of product-lines approaches, Bayer et al. proposed the Product Line Software Engineering (PuLSE) methodology [8]. The methodology was developed with the proposal of enabling the conception and deployment of software product lines within a large variety of enterprise contexts. One important feature of PuLSE is that it is the result of a bottom-up effort: the methodology cap-

tures and leverages the results (the lessons learned) from technology transfer activities with industrial customers.

PuLSE methodology presents an initial direction to develop software product lines. However, some points are not well discussed. For example, the component PuLSE-DSSA supports the definition of a domain-specific software architecture, which covers current and future application of the product line. Nevertheless, aspects such as specification, design, and implementation of the architecture's components are not presented. Bayer et al. consider it an advantage, because PuLSE-DSSA does not require a specific design methodology nor a specific Architecture Description Language (ADL). We do not agree with this vision, because, in our opinion, the lack of details is the main problem related to software reuse processes. The same problem can be seen in the Usage phase, which defines activities to specify, derive and validate product lines members, but does not present explicit details on how to perform them.

According to Atkinson et al. [19], PuLSE has been applied successfully in various contexts for different purposes. Among the benefits, it proved to be helpful for introducing sound documentation and development techniques into existing development practices. However, in circumstances where there were no pre-existing processes or well-defined products, the introduction of PuLSE turned out to be problematic. In such cases, the customization of PuLSE was actually more concerned with the introduction of basic software engineering processes than with the adaptation of the product line ideas into existing processes.

From this perspective, the Kobra approach was proposed [19]. Kobra has two main activities: initially, *framework engineering* creates and maintains a generic framework that embodies all product variant that make up the family, including information about their common and disjoint features. Next, *application engineering* uses the framework built during framework engineering to construct specific applications in the domain covered by the framework.

Even supporting the development *for* and *with* reuse, Kobra presents some drawbacks. For example, during the *framework engineering* activity, it does not present guidelines to perform tasks such as domain analysis and domain design. This question is a little strange, because PuLSE clearly supported activities for planning and scoping.

In the same time, another important and different point of view related to product line processes was proposed in [20]. According to America et al., software reuse processes are strongly related to non-technical aspects such as the business and the organizational constraints. America et al. mention the case of the Philips company, where several product families are developed, ranging from consumer electronics to professional systems. For each of these families, a specifically tuned family engineering method is necessary, because the business and organization constraints differ. Based on this motivation, the Component-Oriented Platform Architecting Method (CoPAM) Family, for Product Family Engineering, was proposed. Thus, for each product family, the CoPAM approach develops a specific family engineering method from the method family in a step called method engineering.

The CoPAM processes are based on several approaches that are described in the literature, such as PULSE, RSEB, RUP, and on authors' experience in industry. The processes have two main subprocesses: *platform engineering* and *product engineering*. *Platform engineering* process develops a platform, which consists of a number of reusable components. A *product engineering* process develops products using these platform components, adding new components where necessary. Both of these receive guidance from and provide feedback to the family engineering process.

In CoPAM, there is a single family engineering process per product family, but there can be more than one platform or product engineering subprocesses. The family engineering process has seven steps leading to the detailed component architecture for the family. The steps range from informal domain analysis to family architecting and support. However, details about how to achieve specific tasks, for example, domain analysis or family architecting, are not presented.

At the end, the last process related to product line may be seen in [21]. In this work, Kang et al. present an evolution of their previous work, FORM [16]. FORM was extended in order to support the development of software product lines.

FORM's product line consists of two processes: *asset development* and *product development*. The major contribution of this work, when compared to FORM is the concern with business aspects such as marketing and product plan (MPP). The MPP identifies the information to gather during the marketing and business analyses. It includes a market analysis, a marketing strategy, product features, and product feature delivery methods.

On the other hand, the process presents lack of details in specific activities such as conceptual architecture design and architecture refinement. In these activities, there are no information on how to identify the components and specify dependencies among them. Moreover, the process of product development and its activities are not discussed.

3 The Proposed Process

The argument for defining processes for specific software development tasks should be a familiar one. A well-defined process can be observed and measured, and thus improved. A process can be used to capture the best practices for dealing with a given problem. The adoption of processes also allows for dissemination of effective work practices to occur more quickly than the building up of personal experience. An emphasis on process helps software development to become more like engineering, with predictable time and effort constraints, and less like art [22].

A software reuse process presents issues related to non-technical aspects, but it must also describe two essential activities: the development *for* reuse and the development *with* reuse.

Our approach for it consists of the following activities: the development *for* reuse, with guidelines, reuse metrics and steps to perform domain analysis (planning, modeling and validation), domain design (to specify, design, validate, and documenting the domain architecture), and domain implementation. Next, the development *with* reuse, with steps to perform application development reusing the assets developed.

The research will consist of the following topics:

1. The analysis of the state-of-the-art in the software reuse processes area. The goal of this analysis is to identify the strengths and weakness of the current processes.
2. The investigation of a set of key requirements for an effective reuse process. These requirements will compose the core of the process.
3. The investigation of the domain analysis area in order to develop guidelines, metrics and identify the necessary steps for tasks of domain planning (scoping), domain modeling (feature modeling) and domain validation. The goal of this investigation is to develop a domain analysis process with its inputs, outputs, rules, and guidelines in a systematic way.
4. The investigation of the software architecture area, in order to identify effective ways to perform the design of the domain architecture. This task includes studies on architectural styles, architecture description languages, architecture documentation, architecture evaluation, and methods to identify, specify and design software components. The goal of this study is to develop a domain design process based on components using the results of the domain analysis process as input.
5. The investigation of techniques to perform the domain implementation. It includes languages and methods to implement and document components.
6. The investigation of techniques and methods to develop applications reusing the available information (requirements, models, components).
7. The definition of an experimental study [23] in order to identify the viability of the proposed process. At the end, through our partnership with the industry, we will perform an industrial case study, aiming to analyze the process applicability in real environments.

4 Work Status

The analysis of the state-of-the-art in the software reuse and software reuse processes area was conducted in conjunction with the definition of the key requirements. The results can be seen in [24, 25].

Currently, we are working in the domain analysis step. An initial version of the domain analysis process is being prepared to evaluation in a case study in the e-gov (electronic government) domain. Our main difficulties are the definition of an approach for domain scoping, and guidelines and systematic ways to perform domain modeling and validation. Moreover, the previous domain analysis works concentrated in specific points of the process, such as feature modeling, not in the all necessary steps (domain planning, domain modeling and domain validation). With the case study, we intend to refine the process and identify

the viability of developing a case tool to support the process. This process will be presented in future papers.

After that, we will concentrate in the domain design step.

5 Concluding Remarks

Developing reusable software is a complex task, which involves the systematic combination of methods, processes, and tools. On the other hand, the research community has continually discussed that a possible way of to obtain software reuse is with the adoption of a well-defined reuse process. However, the current processes present lack of details in important activities, plus some crucial gaps among its steps.

In this context, this work proposes the development of a reuse process in order to allow the development of components and applications in a problem domain. The process is based on the state-of-the-art in the area, and combines guidelines, reuse metrics, economic aspects, and a set of requirements for an effective software reuse process.

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.L. Griss, Making Software Reuse Work at Hewlett-Packard, IEEE Software, Vol. 12, No. 01, January, 1995, pp. 105-107.
4. R. Joos, Software Reuse at Motorola, IEEE Software, Vol. 11, No. 05, September, 1994, pp. 42-47.
5. W.B. Frakes, S. Isoda, Success Factors of Systematic Software Reuse, IEEE Software, Vol. 12, No. 01, January, 1995, pp. 14-19.
6. 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.
7. 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.
8. J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, J. DeBaud, PuLSE: A Methodology to Develop Software Product Lines, Symposium on Software Reusability (SSR), ACM Press, Los Angeles, USA, May, 1999, pp. 122-131.
9. E. S. Almeida, A. Alvaro, D. Lucrédio, V.C. Garcia, S.R.L. Meira, RiSE Project: Towards a Robust Framework for Software Reuse, IEEE International Conference on Information Reuse and Integration (IRI), Las Vegas, USA, November, 2004, pp. 48-53.
10. K. Czarnecki, U. W. Eisenecker, Generative Programming: Methods, Tools, and Applications, Addison-Wesley, 2000, p. 832.
11. J.M. Neighbors, Software Construction Using Components, PhD Thesis, University of California, Irvine, Department of Information and Computer Science, USA, 1980, p. 75.
12. Software Technology for Adaptable, Reliable Systems (STARS), The Reuse-Oriented Software Evolution (ROSE) Process Model, Technical Report, July, 1993, p. 143.
13. M. Simos, D. Creps, C. Klingler, L. Levine, D. Allemang, Organization Domain Modeling (ODM) Guidebook Version 2.0, Technical Report, June, 1996, p. 509.
14. I. Jacobson, M.L. Griss, P. Jonsson, Reuse-driven Software Engineering Business (RSEB), Addison-Wesley, 1997, p. 497.
15. M.L. Griss, J. Favaro, M. d' Alessandro, Integrating Feature Modeling with the RSEB, The Fifty International Conference on Software Reuse (ICSR), IEEE Computer Society, Victoria, Canada, June, 1998, pp. 76-85.
16. K.C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, M. Huh, FORM: A Feature-Oriented Reuse Method with domain-specific reference architectures, Annals of Software Engineering Notes, Vol. 05, 1998, pp. 143-168.
17. K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, A.S. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1990.
18. P. Clements, L. M. Northrop, Software Product Lines: Practices and Patterns, Addison Wesley, August, 2001, p. 608.

19. C. Atkinson, J. Bayer, D. Muthig, Component-Based Product Line Development: The Kobra Approach, The First Software Product Line Conference (SPLC), Kluwer International Series in Software Engineering and Computer Science, Denver, Colorado, USA, 2000, p.19.
20. P. America, H. Obbink, R. V. Ommering, F. V. D. Linden, CoPAM: A Component-Oriented Platform Architecting Method Family for Product Family Engineering, The First Software Product Line Conference (SPLC), Kluwer International Series in Software Engineering and Computer Science, Denver, Colorado, USA, 2000, p.15.
21. K. C. Kang, J. Lee, P. Donohoe, Feature-Oriented Product Line Engineering, IEEE Software, Vol. 19, No. 04, July/August, 2002, pp.58-65.
22. D. Rombach, Fraunhofer: The German Model for Applied Research and Technology Transfer, 22nd International Conference on Software Engineering (ICSE), ACM Press, Limerick, Ireland, May, 2000, pp. 25-34.
23. C. Wohlin, P. Runeson, M. Host, C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering: an Introduction, Kluwer Academic Publishers, Norwell, 2000.
24. E. S. Almeida, A. Alvaro, D. Lucrédio, V.C. Garcia, S.R.L. Meira, A Survey on Software Reuse Processes, IEEE International Conference on Information Reuse and Integration (IRI), Las Vegas, USA, August, 2005.
25. E. S. Almeida, A. Alvaro, S.R.L. Meira, Key Developments in the Field of Software Reuse, ACM Computing Survey, 2005, **in evaluation**.