AN ANALYSIS OF MATURITY MODEL FOR SOFTWARE REUSABILITY – A LITERATURE STUDY

O.Rajalakshmi alias Karthika
Ph.D Part Time Scholar
Department of Computer Science
Madurai Kamaraj University College
Madurai, Tamil Nadu, India

Dr.C.Rekha
Assistant Professor
Department of Computer Science
Government Arts College – Melur
(Affiliated to Madurai Kamaraj University)
Madurai, Tamil Nadu, India.

Abstract

Software reuse plays vital role in software engineering. It is a process of developing software from existing software components, instead of developing entire software from scratch. In other words, software reusability is developing brand new software from an existing one. Software reusability results into increased productivity and quality by minimizing risk of newly developed projects. It is not only depends on code. On the contrary, it entails all entities of software development life cycle like software components, test suites, documentations and designs. This paper presents a wide study of various software reusability models especially three reusability models that are component-based, software architecture and software product lines.

Keywords: software reusability, component-based, Software product lines, software engineering

1. Introduction

Developing cost-effective and quality products is an important and challenging aspect of software development. From the very beginning of software development, reusability has been considered as one of the most important characteristics of software quality.

From organizations' point of view, software reuse is an investment. Before committing resources to reuse, management must see its potential positive economic impacts on quality and productivity [1]. Organizations must be able to measure that impact and evaluate the effectiveness of different reuse techniques to identify and enforce the most successful ones. Therefore, reuse metrics and economical models have become an essential focus of the software reuse research.

Since the concept of systematic software reuse was proposed in 1968, several approaches have been suggested to achieve the promised potential of software reuse. Three of the major approaches are

- Component-based software reuse,
- Software architecture and design reuse, and
- Software product lines.

This paper is a study of some important aspects of software reuse research. As the major goal of reuse is increasing productivity and reducing cost, reuse economics are discussed first. Then three approaches to software reuse are summarized followed by reuse successful stories in industry.

2. Component-Based Software Reuse

The notion of building software from reused components the same way electronic circuits are built from prefabricated ICs was first published in the NATO conference in 1968. The idea emerged from object-oriented development failure to support systematic reuse, which needed more abstract components than detailed and specific object classes. Component-based development allows the reuse of large and highly abstract enterprise components so fewer components are needed to build the application. This reduces the assembling and integration efforts required when constructing large applications [3].

Zhang et al. [4] proposed a research for component-based software reuse (CBSR) based on local projects. The authors proposed a solution for CBSR and validated Trustee components system by practicing Cushion and Eclipse tools. The two strategies proposed by the authors include component-based software engineering and service oriented software development. Open Services Gateway Initiative (OSGI) provides an interface for CBSR and Software Oriented Architecture (SOA).

Chouambe et al. [5] focus on reverse engineering software models of component -based systems .The authors argue that a large number of software systems use component-based technologies such as Common Object Request Broker Architecture (CORBA), Enterprise JavaBeans (EJB) and Component Object Model (COM). However, there is a lack of efficient tools and service in order to reversing such techniques. Current approaches employed for such purposes mostly depend on components of reverse engineering, but these approaches do not resolve external dependencies. In addition, dependent and existing components description is also used to interchange classes and modules. Developers who want to use iterative reverse engineering strategies and approaches need to apply component -based software architectures and compare different tools, methodologies, approaches and techniques.

Crnkovic et al. [2] looked into different software component-based models which were developed by employing diverse themes, objectives, principles and technologies. All the models produced similar results but the principles they followed were different. While some models do not explain the concept clearly; yet they help identify, characterize and provide easy to understand concept of component -based model framework.

The majority of component-based model utilize high level programming development. In such cases, some component -based models execute a specific language for implementation that requires some specific predefined rules. Enterprise Java Beans component-based model uses Java with some additional requirements for implementation and some component-based model use translator for their specific language or multiple languages such as Corba component model.

The main advantage of CBSR is that it is independent for the development process of individual components. These processes work completely independent such as development of COTS (Commercial Off-the-Shelf) and COTS-based systems. Until other aspects of the component are integrated into a system, some specific stages are involved in the development of an individual component such as deployment, implementation, execution, requirements and design.

Hunt and McGregor [6] argue that CBSR is a technique for developing as well as assembling system from recent components containing essential implications for assorted computer software engineering practices. CBSR approaches developed from time to time and this area of research is rapidly evolving to solve fundamental difficulties, problems and the convenience connected with CBSR. If CBSR course is included in the curriculum, then students would get the opportunity to gain practical knowledge of applying the CBSR techniques despite limitations of the existing software tools.

Abdellatief et al. [7] studied how to integrate CBSS with different components to deploy them independently. Though researchers used several proposed CBSR attributes, nevertheless, implementing the CBSR metrics practically is a difficult task because some metrics either overlap with other metrics or are not well defined. The authors described the interface complexity metrics for parameters and methods in the interface. The proposed collection of metrics is exclusively meant for reusability component of JavaBeans evaluation.

Software development depends on three major aspects which have a direct impact on software business such as time, cost and quality of product. Khan et al. [8] argue that information technology is facing enormous challenges such as customer demands to meet the product deadlines with minimum development time and cost.

Lubair and Moiz [9] highlight the main goal of CBSD by describing that the component-based technology changed from a simple component to the domain-specific components over a period of time. Though the impact of time and cost is still a challenging issue in the domain-specific components, the reusable design is among the main advantages of CBSR as the reusable components save time and cost investment. Component-based development models provide software reusability approach which is highly useful to the software engineers. QSM Associates report mentions that 84% decrease in project costs has been noticed due to the growing use of reusable components in the software development process.

The development of model-driven engineering prototypes could have many positive impacts on the overall performance of the projects. Particularly, the use of different traditional methodologies and approaches for embedded system could be useful in this regard. Reusable components are also used for quality improvement in CBSR. Bunse et al. [10] investigated valuation of CBSD approaches and model -driven prototypes in the construction of embedded systems. The authors used Marmot method for micro-controller subsystem of automotive. Development efforts and size of the model were compared and measured by applying two main methods, namely, agile software development and unified process.

Breivold and Larsson [11] describe that CBSR and service oriented software engineering (SOSE) are the most important paradigms adopted by the software development organizations. Both the approaches are used for similar methodologies in many senses, but their focus is different. The advantages of both the approaches are to improve quality of attributes in software engineering and develop the software projects rapidly. However, there could be escalating analysis required to comprehend and implement the combined possibilities of both the models. Essential areas of research within SOSE consist of service composition, service oriented engineering, monitoring and service foundation (which is support service oriented technologies to provide run time service oriented infrastructure), and perform connectivity to heterogeneous systems.

Koziolek [13] highlights the performance of evaluation measurement and prediction approaches for CBSS to evaluate the system component specification designed by component developers. Additionally, integrating conventional performance designs like stochastic process algebras, queuing networks or stochastic Petri nets techniques are used to benefit from the advantages of CBSR such as reusability. Even though several techniques and approaches have been proposed in this area during the last decade, yet they did not receive widespread business use. The functionality of reusable software components is complicated to specify because performance not only depends upon component implementation but also on the deployed context of the component. The performance of software component includes certain influence factors such as required services, usage profile, deployment platform, resource contention and component implementation.

Kahtan et al. [14] describe the future security challenges of the CBSD which concentrate on developing models by combining current software components. CBSD offers an enhanced capability to reuse existing components to develop high quality software at a lower cost and lesser time. Utilizing CBSD models in software engineering poses

several challenges. During the integration phase, the software components can result in producing many issues, therefore, security features of the components analyzed in the previous CBSD lifecycle such as maintainability, reliability, safety, integrity and dependability must be considered to check the software. Such an approach results in providing good customer services as well as enabling system modification from time to time. CBSR offers a large rang of functionality throughout the development of a software system.

Iqbal et al. [15] describe that CBSR focus on developing software system by reusing high quality of independent software components. With the help of OOT (Object Oriented Technology), reusable components have become key aspects of software development. The components modification is used in software development process which increases reusability approach at different levels such as at framework level, architecture level and modular design level. The development process of CBSR modifies the reusability approach into two different approaches, generation-based approach and composition-based approach, which are quite beneficial when programming components are reused.

Iqbal et al. [15, 16] proposed performance metrics for software design and software project management. Process improvement methodologies are elaborated in [17, 18] and Khan et al. [19] carried out quality assurance assessment. Amir et al. [20] discussed agile software development processes. Rehman et al. [21] and Khan et al. [22] analyzed issues pertaining to requirement engineering processes. Umar and Khan [23, 24] analyzed non-functional requirements for software maintainability.

Khan et al. [25, 26] proposed a machine learning approaches for post-event timeline reconstruction. Khan [27] suggests that Bayesian techniques are more promising than other conventional machine learning techniques for timeline reconstruction. Rafique and Khan [28] explored various methods, practices and tools being used for static and live digital forensics. In [29], Bashir and Khan discuss triaging methodologies being used for live digital forensic analysis.

3. Software Product Lines

The general process of product lines majorly hinges on reusability of requirements, architecture and components. The development processes is subdivided into two main phases - Domain Engineering and Application Engineering.

Domain engineering - captures the commonalities and variability in a set of software systems and uses them to build reusable assets. Most organizations work only in a small number of domains. For each domain they build a family of systems that are based on specific customer needs. Domain engineering deals with identifying the common features of existing systems within a particular domain and using these features as a common base to build a new system in the same domain [30]. This will result in higher efficiency and productivity.

Application Engineering - Focus on realization of the customized products, using the core assets and the specific variability's peculiar to individual systems. A differentiation of product is established by systematic binding of variation points with the predefined variants. This phase is composed of three processes; application requirements, application design and application coding.

Research in software product lines [31-33] shows that there are success stories, but there are no strong rules that can be derived from these success stories about how software is reused. Systematic reuse is one of the goals of software product lines, meaning that components must not only be copied but actually shared among several subprojects.

Frakes and Terry [34] defined a set of categories for reuse models in general, and classified reuse cost benefit analysis models into three categories: cost productivity models, quality of investment, and business reuse metrics. Nazareth and Rothenberger [35] characterized the models as metric-based and cost-based.

Mili et al. [36] based their classification on several different aspects of the reuse economic models such as investment cycle, economic functions, and cost factors, structure of reuse organization, scope, hypothesis, and viewpoint.

4. Architecture-Based Software Reuse

Effective reuse depends not only on finding and reusing components, but also on the ways those components are combined [37]. System software architecture is composed of its software components, their external properties, and their relationships with one another. Architecture-based reuse extends the definition of reusable assets to include these properties and relationships. Shaw classified software architecture into common architecture styles where every style has four major elements: components, connectors that mediate interactions among components, a control structure that governs execution and rules about other properties of the system, and a system model that captures the intuition about how the previous elements are integrated. Some of the popular architecture styles in [37] are pipeline, data abstraction, implicit invocation, repository, and layered.

Applying a combination of architecture styles create architectural patterns [38]. An architectural pattern is a high-level structure for software systems that contains a set of predefined sub-systems, defines the responsibilities of each sub-system, and details the relationships between sub-systems. Layers, Pipes and Filters, and Blackboard are some of the common patterns described by Buscmann et al in [39].

Software systems in the same domain have similar architectural patterns that can be describe with a generic architecture (domain architecture). Domain architecture is then refined to fit individual application in the domain creating application architecture [38].

The software architecture analysis method (SAAM) [40] is a scenario-based method for architectural assessment. A scenario, in this context, is a description of an expected use of a specific product line. SAAM also tests modifiability, e.g., by proposing specific changes to be made to the system. A related architectural analysis method is the architecture tradeoff analysis method (ATAM) [41]. This iterative method is based on identifying a set of quality attributes and associated analysis techniques that measure architecture along the dimensions of the attributes. Sensitive points in architecture are determined by assessing the degree to which an attribute analysis varies with variations in the architecture. In our approach, we focus on quality attributes that are specific to product line architectures. As such, the approach can be applied in either the SAAM or the ATAM context.

5. Conclusion

This study presented a literature review of the most up-to-date research work published on software reusability. This review of various software reusability concepts offers a good understanding of reusability for accelerating the adoption of reusability in software development.

The study found that few works examined barriers of reusability, which can motivate organizations to adapt software reusability approaches. Also the studies about maturity models of software reuse are limited, so exploring this domain for helping organizations to audit his maturity reuse levels, can be a subject of a future work.

Refereces

[1] Sajjan G. Shiva, Lubna Abou Shala, "Software Reuse: Research and Practice", Fourth International Conference on Information Technology, ITNG '07, 2007.

[2] I. Crnkovic, S. Sentilles, A. Vulgarakis and M. R. V. Chaudron, "A classification framework for Component models", IEEE (2011).

[3] C. McClure. Software Reuse: A Standards-Based Guide, Wiley-IEEE Computer Society Pr, New York, 2001

[4] X. Zhang, L. Zheng and C. Sun, "The research of the Component-based Software Engineering", Sixth International Conference on Information Technology: New Generations, (2009).

[5] L. Chouambe, B. Klatt and K. Krogmann, "Reverse Engineering Software-Models of Component-Based Systems", IEEE (2008).

[6] J. M. Hunt and J. D. McGregor, "Component Based Software Engineering across the Curriculum", 23rd IEEE Conference on Software Engineering Education and Training, (2010).

[7] M. Abdellatief, A. B. M. Sultan, A. Ghani and M. A. Jabar, "A mapping study to investigate component-based software system metrics", The Journal of Systems and Software, vol. 86, (2013), (2012), pp. 587-603.

[8] N. Mdlubair and S. A. Moiz, "Component Base Software Development: A State of Art", ICAESM (2012).

[9] C. Bunse, H. G. Gross and C. Peper, "Embedded System Construction–Evaluation of Model-Drive and Component-Based Development Approaches", Springer-Verlag Berlin Heidelberg (2009).

[10] K. Mahmood and B. Ahmad, "Testing strategies for stakeholders in Component-Based Software Development", The Computer Engineering and Intelligent Systems, vol. 3, no. 5, (2012).

[11] H. Koziolek, "Performance Evaluation of component-based Software systems: A survey", The Performance Evaluation, vol. 67, (2009), pp. 634-658.

[12] H. Kahtan, N. A. Bakar, R. Nordin, "A Reviewing the Challenges of Security Features in Component Based Software Development Models", IEEE (2012).

[13] F. McCarey, M. Eide and N. Kushmerick, "A Case Study on Recommending Reusable Software Components using Collaborative Filtering", (2004).

[14] K. Matsumura, A. Yamashiro, T. Tanaka and L. Takahashi, "Modeling of Software Reusable Component Approach and its Case Study", IEEE (1990).

[15] S. Iqbal, M. Khalid and M. N. A. Khan, "A Distinctive Suite of Performance Metrics for Software Design", International Journal of Software Engineering & Its Applications, vol. 7, no.5, (2013).

[16] S. Iqbal and M. N. A. Khan, "Yet another Set of Requirement Metrics for Software Projects", International Journal of Software Engineering & Its Applications, vol. 6, no. 1, (2012).

[17] M. Faizan, S. Ulhaq and M. N. A. Khan, "Defect Prevention and Process Improvement Methodology for Outsourced Software Projects", Middle-East Journal of Scientific Research, vol. 19, no. 5, (2014), pp. 674-682.

[18] M. Faizan, M. N. A. Khan and S. Ulhaq, "Contemporary Trends in Defect Prevention: A Survey Report", International Journal of Modern Education & Computer Science, vol. 4, no. 3, (2012).

[19] K. Khan, A. Khan, M. Aamir and M. N. A. Khan, "Quality Assurance Assessment in Global Software Development", World Applied Sciences Journal, vol. 24, no. 11, (2013).

[20] M. Amir, K. Khan, A. Khan and M. N. A. Khan, "An Appraisal of Agile Software Development Process", International Journal of Advanced Science & Technology, vol. 58, (2013).

[21] T. U. Rehman, M. N. A. Khan and N. Riaz, "Analysis of Requirement Engineering Processes", Tools/Techniques and Methodologies. International Journal of Information Technology & Computer Science, vol. 5, no. 3, (2013).

[22] M. N. A. Khan, M. Khalid and S. ulHaq, "Review of Requirements Management Issues in Software Development", International Journal of Modern Education & Computer Science, vol. 5, no. 1, (2013).

[23] M. Umar and M. N. A. Khan, "A Framework to Separate Non-Functional Requirements for System Maintainability", Kuwait Journal of Science & Engineering, vol. 39(1 B), (2012), pp. 211-231.

[24] M. Umar and M. N. A. Khan, "Analyzing Non-Functional Requirements (NFRs) for software development", In IEEE 2nd International Conference on Software Engineering and Service Science (ICSESS), 2011 pp. 675-678), (2011).

[25] M. N. A. Khan, C. R. Chatwin and R. C. Young, "A framework for post-event timeline reconstruction using neural networks", digital investigation, vol. 4, no. 3, (2007), pp. 146-157.

[26] M. N. A. Khan, C. R. Chatwin and R. C. Young, "Extracting Evidence from Filesystem Activity using Bayesian Networks", International journal of Forensic computer science, vol. 1, (2007), pp. 50-63.

[27] M. N. A. Khan, "Performance analysis of Bayesian networks and neural networks in classification of file system activities", Computers & Security, vol. 31, no. 4, (2012), pp. 391-401.

[28] M. Rafique and M. N. A. Khan, "Exploring Static and Live Digital Forensics: Methods, Practices and Tools", International Journal of Scientific & Engineering Research, vol. 4, no. 10, (2013), pp. 1048-1056.

[29] M. S. Bashir and M. N. A. Khan, "Triage in Live Digital Forensic Analysis", International journal of Forensic Computer Science, vol. 1, (2013), pp. 35-44.

[30] Sajjan G. Shiva, & Lubna Abou Shala, (2007), Software Reuse: Research and Practice,International Conference on Information and Technology, (ITNG 07), IEEE.

[31] Weiss D. M. and Lai C. R., Software Product Line Engineering: A FamilyBased Software Development Process. Addison-Wesley, 1999.

[32] Pohl K., Bockle G., and Linden F. v.d., Software Product Line Engineering: Foundations, Principles, and Techniques, 1st ed. New York, NY:Springer, 2005.

[33] Linden F. v.d, "Software Product Families in Europe: The ESAPS & CAFÉ Projects", IEEE Software, Vol. 13, No. 3, Pages: 41-49, 2002.

[34] W. Frakes and C. Terry, "Software Reuse: Metrics and Models," ACM Computing Surveys, vol. 28, pp. 415--435, June 1996.

[35] D. L. Nazareth and M. A. Rothenberger, "Assessing the CostEffectiveness of Software Reuse: A Model for Planned Reuse," Journal of Systems and Software, vol. 73, pp. 245-255, 2004.

[36] A. Mili, S. F. Chmiel, R. Gottumukkala, and L. Zhang, "An integrated cost model for software reuse," in ICSE '00: Proceedings.

[37] M. Shaw, Architectural issues in software reuse: It's Not Just the Functionality, It's the Packaging, Proc IEEE Symposium on Software Reusability, April 1995, pp 3-6.

[38] W. Frakes, and Kyo Kang, “Software Reuse Research: Status and Future”, IEEE Transactions on Software Engineering, vol. 31, no. 7, 2005, pp 529-536.

[39] F. Buschmann et al., Pattern-Oriented Software Architecture. Chichester, UK; New York: Wiley, 1996.

[40] Kazman, R., Abowd, G., Bass, L., Clements, P., 1996. Scenario-based analysis of software architecture. IEEE Software, pp. 47–55.

[41] Kazman, R., Klein, M., Barbacci, M., Lipson, H., Longstaff, T.,Carriere, S., 1998. The architecture tradeoff analysis method. In:Proceedings of ICECCS, pp. 68–78.