An Analytical View on the Software Reuse

Florinda Imeri[1], Ljupcho Antovski[2]

[1]State University of Tetovo
Ilindenska bb, 1200 Tetovo, Macedonia
florindaimeri@yahoo.com

[2]University Ss. Cyril and Methodius
Rugjer Boshkovikj 16, 1000 Skopje, Macedonia
ljupcho.antovski@finki.ukim.mk

Abstract: High-quality software, delivered on time and budget, constitutes a critical part of most products and services. Today software engineers are faced with a demand for complex and powerful software systems. To be competitive in the market software engineers are forced to create software as quickly as possible. Software reuse or component-based development (CBD) is regarded as one of the most potent software technologies in order to reduce lead times, increase functionality, and reduce costs. In the region, CBD is still a process with lot of problems, not well defined either from theoretical or practical points of view. The lack of knowledge is probably the biggest problem. Our aim is to systemize the knowledge and understanding of CBD. This research is a literature review on the software reuse and the concepts behind it. It starts with an overview of software components and CBD, continues describing benefits and obstacles to software reuse. Nontechnical aspects like legal, economic and measurement issues are covered as well. Finally examples of successful software reuse and the state of practice are summarized.

Keywords: software components, components based software development, reuse benefits, obstacles and metrics.

1 Introduction

From the time that software development started to be discussed within the IT industry, researchers and practitioners have been searching for methods, techniques and tools that would allow for improvements in costs, time-to-market and quality. Thus, an envisioned scenario was that managers, analysts, architects, developers and testers would avoid performing the same activities over and over [1].

Software reuse or component-based software development is regarded as one of the most potent software technologies in order to reduce lead times, increase functionality, and reduce costs. Software reuse and component-based development is a promising way to promote the productivity of large systems development. It is proved that designing a system that supports this approach requires more efforts and the time to market might be longer, but in the long run, the reusable approach will prove profitable[2].

Component-based software development (CBSD) involves both, the development of software components and building of software systems through integration of preexisting software components, developed in-house or procured from the component market [3].

In this paper we describe the key characteristics of component based development - software reuse. This research is a literature survey on the software reuse.

The paper is organized as follows. Section 2 provides some background on the concepts of software components and on reuse; section 3 reviews the advantages and obstacles to reusability; section 4 describes the key factors that influence the success and/or failure of software reuse in software development. Section 5 defines some metrics to measure software reusability; sections 6 describe the industrial best practices of software reuse; section 7 describe the state of practice of reuse, and finally discuss future directions.

2 Component-based Software Development

Component-based Software Development (CBSD) is an emerging discipline that promises to take Software Engineering into a new era. The aim of CBSD is to deliver Software Engineering from a '*cottage industry'* into an '*industrial age for Information Technology*', whereby software can be assembled from components, in the same manner that hardware systems are constructed, from kits of parts [4].

2.1 Origins of Software Reuse

According to Pareto-Diaz[5] notion of reusability began when humans start finding solutions to problems. To solve a problem, we try to apply the solution to similar new problems. If only some elements of the solution apply, than we adapt it to fit to the new problem. Proven solutions, used over and over to solve the same type of problem, become accepted, generalized, and standardized.

Software reuse was first envisioned by McIllroy[4], at a NATO Software Engineering Conference 1968, where he predicted that mass-produced components would end the software crisis. He proposed an industry of off-the-shelf, standard source-code components and envisioned the construction of complex systems from small building blocks available through catalogs. The final objective was very clear: to make something once and to reuse it several times. Morisio et al. [6] define software reuse as: the systematic practice of developing software from a stock of building blocks, so that similarities in requirements and/or architecture between applications can be exploited to achieve substantial benefits in productivity, quality and business performance. CBSD advocates the use of prefabricated pieces, perhaps developed at different times, by different people, and possibly with different uses in mind [7]. The idea involves reusing experience, such as requirements specification, design, architecture, test data and documentation. A widely growing approach for the development of information systems is component-based engineering discipline which deals with both, developing components and developing with components [5].

2.2 Software Components

A software component may be thought of as an independent module that provides information about what it does and how to use it through a public interface, while hiding its inner workings[8]. Components are seen as black-box and are binary unit of composition, whose internals cannot be viewed or accessed. Their quality characteristics can be evaluated through externally observable elements[9].

Software components may be any coherent unit of design effort that can be packaged, sold, kept in a library, assigned to one person or team to develop and maintain, and re-used. Components can be classes or frameworks; or objects that can be dynamically plugged at run-time; high-level designs; specifications; patterns; extensions to existing components; or even project plans [10].

Several studies into reuse have shown that 40% to 60% of code is reusable from one application to another, 60% of design and code are reusable in business applications, 75% of program functions are common to more than one program, and only 15% of the code found in most systems is unique and new to a specific application [11]. According to Mili et al.[3] rates of actual and potential reuse range from 15% to 85%.

According to Kim [12], there are two types of component-based reuse: with and without change to an existing component.

Reuse without change means simply selecting a component from a software component database, and dropping it into new software being developed. The cost of developing the component anew is zero! These components are called commercial components -COTS components.

Components-based reuse with change can be found in at least three types of use:

- The first and the most common is reuse of most of existing software when developing the next version of the software. Typically, some 60-80 percent of the existing software gets to be reused in this situation. However, developers do not go through the formality of "registering" components in a common software component database in this case.
- Reuse of thirty-party software, such as a sorting package, a database loader, etc. on the market or on the Internet as open source code. Again, such software is not "registered" in an organization's common software component database.
- The third type of reuse is the use of common functions available in programming language libraries, such as the math functions in the C Programming Library.

3 The Benefits and Obstacles of Software Reuse

The argument that software components will improve programmers' productivity is an old one with roots in the study of software reuse [13]. Thinking of effective software reuse as a problem-solving reuse provides a good general heuristic for judging a work product's reuse potential. For example, modules that solve difficult or complex problems (like hardware driver modules in an operating system) are excellent reuse candidates because they incorporate a high level of problem-solving expertise that is very expensive to replicate. Software reuse can have major, and possibly unforeseen, positive effects on the software development process. Components are standardized building blocks that can be used to assemble, rather that develop, information systems.

According to Bollinger [14] there are a number of benefits to software reuse:

- It increases the software productivity and decreases the time required for the development of software.
- By using the technique of software reuse, a company can improve software system interoperability and needs less people for software development; this provides a competitive advantage for the company and helps to produce better quality software and standardized software.
- Software reuse helps the company to reduce the costs involved in software development and maintenance; by using it the software developers can be moved from one project to the other project easily.
- Using well-tested components increase the reliability of a software system. Moreover, the use of a component in several systems increases the chance of errors being detected and strengthens confidence in that component.
- Software reuse reduces the risk involved in software development process.
- Since the documentation is very important for the maintenance of a system, reusing software components reduces the amount of documentation to be written

Even good components can corrupt a good product if they are managed in the wrong way. In some domains, such as industrial automation, this risk is unacceptable, and additional measures are required to minimize the risk [15]. There are some factors that directly or indirectly influence its adoption. These factors can be managerial, organizational, economical, conceptual and technical [14].

- Managerial and Organizational Obstacles; reuse is not just a technical problem that has to be solved by software engineers. The most common reuse obstacles are: lack of management support, project management, inadequate organizational structures and management incentives.
- Economic Obstacles**;** reuse can save money in the long run, but it is not for free. Cost associated with reuse can be: costs of making something reusable, costs of reusing it, and costs of defining and implementing a reuse process. Reuse requires up-front investments in infrastructure, methodology, training, tools and archives, with payoffs being realized only years later. Higher levels of quality, reliability, portability, maintainability, generality and more extensive documentation are necessary.
- Conceptual and Technical Obstacles; the technical obstacles for software reuse include issues related to search and recovery components, legacy components and aspects involving adaptation. In order to reuse software components there should exist efficient ways to search and recover them, this mean it is important to have a well-organized repository containing components with some means of accessing it.
- Lastly, the component must be integrated into the system under development, and thoroughly tested.

4 Success and Failure Factors

Reuse principles place high demands on the reusable components. According to de Almeida et al. [1] developing a reusable component requires three to four times more resources than developing a component for particular use. The more reusable a component is, the more demands are placed upon from products using that component. According to Poulin[16] to recover development costs, software components-assets must be reused more than dozen times. A successful program of software reuse provides benefits in three areas: increased productivity and timeliness in the software development process, improved quality of the software product and an increase in the overall effectiveness of the software development process [17].

In general there are six factors that are critical for systematic software reuse: management, measurement, legal issues, economics, design for reuse and libraries [18][6].

As with any engineering activity, measurement is vital for systematic reuse. In general, reuse benefits (improved productivity and quality) are a function of the reuse level- the ratio of reused to total components- which, in turn, is a function of reuse factors, the set of issues that can be manipulated to increase reuse, either of managerial, legal, economic as technical background [18].

As regarding to legal issues, many of which are still to be resolved, are also important, like, what are the rights and responsibilities of providers and consumers of reusable assets? If a purchased component fails in a critical application should the provider of reusable assets be able to recover damages?

Once an organization acquires reusable assets, it must have a way to store, search, and retrieve them– a reuse library. Although libraries are a critical factor in systematic software reuse, they are not a necessary condition for success with reuse. An example to this is Agora, a software prototype being developed by the Commercial Off-the-Shelf (COTS)-Based Systems Initiative at the Software Engineering Institute (SEI)[19].

5 The Metrics of Software Reuse

What makes a software component reusable? The reusability can be seen as a combination of two attributes, (re)usefulness, which means that the component addresses a common need, or provides an often requested service, and usability, which means that the component is of good enough quality and easy enough to understand and use for new software developments.

Building a reusable asset represents a more or less major investment, depending on the reuse approach used. The concept such as reuse and reusability naturally has led to questions of how to measure them, and of how to run experiments to establish their impact on quality and productivity [20].

The universally accepted truth that what cannot be measured cannot be managed also holds for the software engineering field and particularly to the area of software reuse. These lead us to another important software engineering area closely related to software reuse: software metrics[21].

Existing software reuse metrics are divided into two main categories: Economics Oriented Reuse Metrics and Models (EORM), Software Structure Oriented Reuse Metrics (SORM) and Reuse Repository Metrics (RRM)[23][22][24].

Economics oriented reuse metrics are mainly concerned with the economical aspects related to reuse programs in organizations and are the basic instruments for organization-wide return on investment models (ROI) [24]. Models or software reuse economics try to help us answer the question as when is worth to incorporate reusable components into a software development process and when custom developments without reuse are preferable. The reuse benefit corresponds to how much was saved by reusing existing reusable components. The ratio of reuse benefits to reuse investments determines if the reuse effort resulted in profit or loss to the organization[14].

Software Structure Oriented Reuse Metrics are concerned on what is being reused and how it is being reused from a strictly technical standpoint. Such metrics can be used as a supporting tool to help organizations determine where should direct their effort regarding maintenance and evolution of reusable assets available in the repository and potential new reusable assets to be developed. Such metrics are concerned on how much was reused versus how much was developed from scratch, but fail to help on the analysis of what was reused and how it was reused..

Reuse Repository Metrics (RRM) is another software reuse metrics category related to reuse repositories whose target is the assessment of reuse repository. It covers the aspects such as availability of the repository, search engine performance, quality of available assets and number of times assets were successfully reused. The efficiency of reuse repositories in aspects such as availability and quality of search results may be a decisive factor for a better reuse activity and can have a greater positive impact on the quality and the cost of the produced software.

6 Industry Projects Best Practices

"Software crisis" was first used to describe the frustration that software development and maintenance have placed on otherwise happy and productive organizations. The systematic application of software reuse to prototyping, development, and maintenance is one of the most effective ways to significantly improve the software process, shorten time-to-market, improve software quality and application consistency, and reduce development and maintenance costs. By building systems from pre-tested components, one will save the cost of designing, writing and testing new code. There are significant corporate reuse programs at AT&T, HP, IBM, GTE, NEC, Toshiba, Hitachi, Ltd, Motorola, Inc., National Aeronautics and Space Administration (NASA), and others [1][25].

The United States approach to software reuse is concentrated on tools, technology, and feature sets. There is significant work at research consortia such as MCC, SPC, SEI, and STARS program founded by DoD of US.

Manufacturers of computer systems and instruments, such as Hewlett-Packard Co. and IBM, whose businesses relied mostly on hardware and mechanical engineers, today find that over 70 percent of their research and development engineers are working in the areas of software and firmware. Maintenance and rework account for about 60 to 80 percent of the total software costs. It appears that product development costs, factoring in the cost of producing, supporting, and integrating reusable software components, can decrease by a sustainable 10 to 12 percent; defect rates in delivered products can drop drastically to 10 percent of their former levels; and long-term maintenance costs can drop to 20 to 50 percent of their former values when several products share the same, high-quality components [26][25]. IBM has a corporate program with several reuse support centers, a large library, and a multisite Corporate Reuse Council, key aspect of which is formal identification of reuse "champions" and agents [27].

Starting from 1988, AT&T developed a domain-specific, large-scale software-bus system for on-line transaction processing and network management. With a support staff of about 30, their reuse program has reduced development costs by about 12 percent and time-to-market from 18-24 months to 6-9 months[28] [29].

According to Joos, to increase quality and productivity, Motorola around the 90's had software reuse as one of its pillars [30]. To implement reuse approach several things were needed. As first the support from top management and reuse training for engineers is crucial; second, to provide incentives as support to the program until reuse becomes part of the culture; and third, provide tools to help engineers. Motorola started a 3-phase software reuse process. The first phase included creation of a reuse task which had two main activities: to educate how to spread the software reuse knowledge; and to emphasize metrics, rewards, career paths and job descriptions for producers and users of reusable components. At the second phase, senior management accepted the responsibility of initiating reuse by getting the upper management more involved with advancing the state-of-the-practice in software. And finally, at the last phase, groups of developers realized that software reuse promises more than internal savings since it provides new markets for reusable components and support tools.

Most of the European work consists of industrial or industrial-academic consortia. The ESPRIT initiative has funded several industrial-academic collaborations, such as KNOSOS, PRACTITIONER, ITHACA, SCALE, REDO, and REBOOT [31]. The objectives of the REBOOT project are to study, develop, evaluate and disseminate advanced methodologies and environment for reuse-driven and object-oriented software development with the emphasis on planned reuse.

According to Griss [26] the Japanese approach to reuse is concentrated on core functionality, design, productivity, and quality. They were able to produce much higher quality systems at a faster rate. Hitachi reduced the number of late projects from 72 percent in 1970 to 12 percent in 1984, and reduced the number of defects to 13 percent of the 1978 level. Toshiba increased productivity by a factor of 250 percent between 1976 and 1985, and by 1985 had reduced defect rates to 16 to 33 percent of the 1976 level. NEC improved productivity by 126 to 191 percent and reduced defect rates to 33 percent of prior levels. Nippon Telegraph & Telephone Corp. (NIT) has a comprehensive program including a reuse-specific organization, printed catalogs, guidelines, and certification for reuse, leading to reuse levels of 15 percent or more, with several hundred small components[32].

7 State of Practice of Software Reuse in the Cloud

To support scientific research, the development of technological infrastructure have include cloud computing, which offers capabilities for research communities to utilize scientific data and services in order to conduct analyses that otherwise would be more costly or prohibitive to complete [33]. Cloud providers install and operate software applications in the cloud and cloud users access it from clients. Cloud computing is a recent example of a technology that can benefit from software reuse. Employing software reuse techniques enables the adoption of new technological approaches to support new models for conducting scientific research. The software reuse methods and tools also can contribute to the evaluation of systems being developed to support scientific research. Software reuse instruments, such as the Reuse Readiness Levels (RRLs) [34], offer the potential to improve capabilities for software development and evaluation when developing and adopting software assets for scientific support systems [35].

OOS development as a special instance of software reuse development, typically takes place in informal collaborations of globally distributed teams communicating over the internet. Software repositories on the Internet provide a tremendous amount of freely reusable source code, frameworks and libraries for many recurring problems. The Internet itself has become an interesting reuse repository for software projects [36]. Search engines like Google Code Search provide powerful search capabilities and direct access to millions of source code, written in a multitude of programming languages. It allows search of publicly available source code by language, package name, file name, and types of license. The extraordinary success of some of the resulting software products (such as GNU/Linux, Apache, Bind DNS server, OpenOffice, Mailman) has drawn attention from the public and both software creating and software-using organizations to this way of developing software.

8 Conclusion and Future Work

Software reuse has been practiced since programming began. Reuse as a distinct field of study in software engineering, however, is often traced to Doug Mcilroy's paper that proposed to base the software industry on reusable components.

Clearly it is seen that software reuse is an inevitable solution that has potential to improve time –to-market and man power/cost trends that have been ongoing. Currently seem to be one of the most active and creative research areas in Computer Science. Software reuse has a significant impact on software industry. It helps organize large-scale development and what is more important; it makes system building less expensive.

Software reuse has been a buzz word in large companies for some time now, with its potential for achieving good quality systems in short time scales by the reuse of currently available components. Many success stories have been quoted, but what about the small, less structured companies, whose livelihood depends on the ability to produce their product as quickly as possible, while trying to keep standards high enough to keep their customers happy and their maintenance costs low. To them, the benefits of software reuse could be invaluable. One important issue is how to make best use of reusable components at the companies of small size.

Our future work will focus on the small software development companies in Macedonia and the SEE region to reevaluate the issues surrounding software reuse from the perspective of developers involved in a software development. In particular, we want to explore their experience with software reuse and possibly try to increase the knowledge and understanding of CBD. We would like to look at the possible benefits, disadvantages and contributors towards successful reuse of software components

REFERENCES

[1] Eduardo Santana de Almeida, A. Alvaro, V. C. Garcia, J. C. C. P. Mascena, V. A. de A. Burégio, L. M. Nascimento, D. Lucrédio, and S. L. Meira, *Component Reuse in Software Engineering*. .

[2] I. Crnkovic and M. Larsson, "Challenges of Component-based Development Challenges of Component-based Development," *Computer Engineering*, vol. 61, no. 3, pp. 201–212, 2002.

[3] H. Mili, F. Mili, and A. Mili, "Reusing software- Issues and research directions," *IEEE Transactions on Software Engineering*, no. 6, pp. 528 – 562, 1995.

[4] *Component-Based Software Development-case studies*, vol. 1. World Scientific Publishing Co. Ptc. Ltd. 5 Toh Tuck Link, Singapore, 2004.

[5] R. Prieto-Diaz, W. Schafer, and M. Matsumoto, "Status report: Software reusability," *Software, IEEE*, vol. 10, no. 3, pp. 61–66, 1993.

[6] M. Morisio, M. Ezran, and C. Tully, "Success and failure factors in software reuse," *IEEE Transactions on Software Engineering*, vol. 28, no. 4, pp. 340–357, Apr. 2002.

[7] A. Cechich and M. Piattini, *Component-based software quality: methods and techniques*. 2003.

[8] T. N. Form, "Chapter 1. Introduction to IEEE Std. 1517— Software Reuse Processes 1.1," *Components*.

[9] M. F. Bertoa, J. M. Troya, and A. Vallecillo, "Measuring the usability of software components," *Journal of Systems and Software*, vol. 79, no. 3, pp. 427–439, Mar. 2005.

[10] A. C. Wills, D. D. Souza, and I. Computing, "Rigorous Component-Based Development," *Components*, pp. 1–28, 1997.

[11] M. Ezran, M. Morisio, and C. Tully, *Practical software reuse*. 2002.

[12] W. Kim, "On Issues with Component-Based Software Reuse," *Journal of object technology*, vol. 4, no. 7, pp. 45–50, 2005.

[13] C. Buhman and F. Long, "Volume I : Market Assessment of Component-Based," *Internal Research and Development*, vol. I, 2000.

[14] B. Bollinger and B. Barnes, "Making Reuse Cost -Effective," *IEEE Software*, 1991.

[15] I. Crnkovic and M. Larsson, "Overview Component-based Software Engineering State of the Art Report," *Computer*.

[16] J. S. Poulin, "The Business Case for Software Reuse: Reuse Metrics, Economic Models, Organizational Issues, and Case Studies," *9th International Conference on Software Reuse*, vol. 4039/2006, p. 439, 2006.

[17] Y. Kim, "Software reuse: survey and research directions," *Journal of Management Information Systems*, 1998.

[18] W. B. Frakes and S. Isoda, "Success factors of systematic reuse," *IEEE Software*, vol. 11, no. 5, pp. 14–19, Sep. 1994.

[19] R. C. Seacord, S. A. Hissam, and K. C. Wallnau, "Agora : A Search Engine for Software Components Agora : A Search Engine for Software Components," no. August, 1998.

[20] W. Frakes and C. Terry, "Software reuse: metrics and models," *ACM Computing Surveys*, vol. 28, no. 2, pp. 415–435, Jun. 1996.

[21] K. Jasmine, "Cost estimation model for reuse based software products," *IMECS 2008: International Multiconference of*, vol. I, pp. 19–21, 2008.

[22] J. Mascena and E. de Almeida, "A comparative study on software reuse metrics and economic models from a traceability perspective," *Information Reuse and Integration*, pp. 72–77, 2005.

[23] W. Lim, "Effects of reuse on quality, productivity, and economics," *Software, IEEE*, 1994.

[24] J. Poulin, "An agenda for software Reuse Economics," *International Conference on Software Reuse*, no. April, 2002.

[25] K. Wentzel, "Software reuse-facts and myths," *Software Engineering, 1994. Proceedings. ICSE-*, 1994.

[26] M. L. Griss, "Software reuse: From library to factory," *IBM Systems Journal*, vol. 32, no. 4, pp. 548–566, 1993.

[27] J. Tirso, "The IBM reuse program," *of the 4th Annual Workshop on Software Reuse*, 1991.

[28] R. Martin and G. Jackoway, "Software Reuse Across Continents," *Workshop in Reuse*, pp. 1–6, 1991.

[29] J. Tirso, "Championing the cause: making reuse stick," *... of the 5th Annual Workshop on Software Reuse*, 1992.

[30] R. Joos, "Software reuse at Motorola," *IEEE Software*, vol. 11, no. 5, pp. 42–47, Sep. 1994.

[31] J. Faget and J. Morel, "The REBOOT approach to the concept of a reusable component," *5th Workshop Institutionalizing*, 1992.

[32] M. Aoyama, "CBSE in Japan and Asia," Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2001, pp. 213–225.

[33] J. Marshall, R. R. Downs, and C. A. Mattmann, "Software reuse methods to improve technological infrastructure for e-Science," *Information Reuse and Integration (IRI), 2011 IEEE International Conference*, pp. 528–532, 2011.

[34] J. Marshall, S. Berrick, and A. Bertolli, "Reuse Readiness Levels (RRLs)," *sciencedatasystems.org*, 2010.

[35] P. Vitharana, J. King, and H. S. Chapman, "Impact of Internal Open Source Development on Reuse: Participatory Reuse in Action," *Journal of Management Information Systems*, vol. 27, no. 2, pp. 277–304, Oct. 2010.

[36] W. Frakes, "Software reuse research: Status and future," *Software Engineering, IEEE Transactions*, vol. 31, no. 7, pp. 529–536, 2005.