Review of Software Reuse Processes

Ljupcho Antovski[1] and Florinda Imeri[2]

[1] Faculty of Computer Sciences and Engineering, University Ss. Cyril and Methodius Department
Skopje, 1000, Macedonia

[2] Department of Informatics, State University of Tetovo
Tetovo, 1200, Macedonia

Abstract
Clients expect that enterprise software systems, which are continually growing in complexity, are of high quality, with reasonable price and have a short time to market. To find ways to meet with the increasing demands of business, software engineers are forced to find ways to streamline development process. Software reuse is believed to be one such approach. Companies that have adopted its techniques to their development process have reported useful improvements in development productivity and quality. The practices of reuse have proven not to be simple and there are many misconceptions about how to implement and gain benefits out of it. Effective reuse is not a simple addition to existing software development processes; it puts strong demands on development methods in order to be successful. Our research, based on the literature and empirical results, presents basic principles of software reuse. Driving factors that facilitates reuse are also presented together with potential benefits and key issues to consider in order to successfully adapting this approach.
Keywords: *software reuse, software components, CBSE, reuse principles, reuse metrics*

1. Introduction

Software is the engine that makes run everyday life, such as in business, industry, administration, research, etc. As enterprise software systems are continually growing in complexity IT industry is forced to find ways to streamline development process[1]. Software reuse is believed to be one such approach as most effective way to significantly improve the software process, shorten time-to-market, improve software quality and application consistency, and reduce development and maintenance costs[2].

The concept of software reuse is the idea of building and using "software preferred parts"[3]. Building systems from pre-tested components, one will save the cost of designing, writing and testing new code. Companies that have adopted its techniques to their development process have reported useful improvements in development productivity and quality. The practice of reuse has not proven to be simple however, and there are many misconceptions about how to implement and gain benefit from software reuse[4]. There are many technical, economical and organizational issues to overcome.

In this paper we describe key characteristics of software reuse and/or component based development. It is a combination of literature survey and empirical results of software reuse processes.

The paper is organized as follows. Section 2 provides background on the concepts of software components and software reuse; section 3 describes reuse principles; section 4 describes the key factors that influence the success and/or failure of software reuse in software development and section 5 defines models and metrics which measure software reusability.

2. Software Reuse Scope

According to Pareto-Diaz[5] reusability is as old as humans are. To solve a problem, we try to apply the solution to similar new problems. If some elements of the solution apply than we try to adapt it to fit to the new problem. Solutions, used over and over to solve the same type of problem, become accepted, generalized, and standardized.

McIllroy[4], at a NATO Software Engineering Conference 1968, 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.

The availability of reusable software has increased dramatically. The open source movement has meant that there is a huge reusable code base available at low cost either in the form of program libraries or entire applica-

Copyright (c) 2013 International Journal of Computer Science Issues. All Rights Reserved.

tions. Some large companies provide a range of reusable components for their customers. Standards, such as webservice standards, have made it easier to develop general services and reuse them across a range of applications[6]. What can be reused? The most common form of reusable artifact is of course a source code in some programming language, but it is not the only one. The idea involves reusing experience, such as requirements specification, design, architecture, test data and documentation. 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[7]. According to Mili et al.[8], rates of actual and potential reuse range from 15% to 85%.

2.1. CBSE and Software Components

Component-based Software Engineering (CBSE) is an emerging discipline has the potential to bring the Software Engineering on a new level. Its aim is to deliver Software Engineering from a ‘cottage industry’ into an ‘industrial age for Information Technology’[9] . The main idea of CBSE is to build systems from pre-existing components developed by different people, at different times, possibly with different uses in mind when the system development process starts.

Software components are artifacts clearly identified in our software systems. They can be classes or frameworks; objects that can be dynamically plugged at run-time; high-level designs; specifications; patterns; extensions to existing components; or even project plans[10]; they have an interface, encapsulate their internal details and are documented separately; 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[11].

The literature defines two main categories of components, white box components and black box components known as COTS (components-of-the-shelf).

White box components are components with source code, directly changeable by the programmers that use them, while black box components are typically in compiled or in a binary form and cannot directly be changed[12]. All the programmer knows about them is the documentation that describes their functionality, and their published "publicly known" interfaces including properties (or attributes) that can be viewed. Even though in practice white box components are more preferred by programmers, the benefits of using black box components outweigh those of white box components, since black box components cannot directly be modified by a programmer, their original functionality stays intact so that upgrades, bug fixes, etc, made by the original developer can be implemented. By changing the source of a white box component you would have a new source stream, and old bugs would not be fixed and propagated to new instances of components.

CBSE distinguishes the process of “component development” from that of “system development with components”. System development with components’ focus is on assembling software components that supply user services driven by specific business requirements. It introduces fundamental changes in the way systems are acquired, integrated, deployed and evolved. The process of system design involves the selection of components, together with an analysis of which components can be acquire from external sources, and which ones must be developed from scratch. In this case systems are designed by examining existing components to see how they meet the system requirements. The component development process focuses on acquire, wrap and build reusable components[13]. Figure 1 presents the basic processes of CBSE with and for reuse[6].

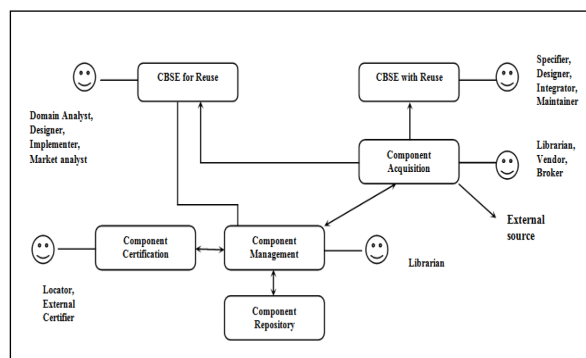

Fig. 1. CBSE processes [6]

3. Software Reuse Principles

Software reuse can have major, and possibly unforeseen, positive effects on the software development process. 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[9].

Copyright (c) 2013 International Journal of Computer Science Issues. All Rights Reserved.

Software reuse is categorized along six orthogonal axes[14]:

Transformational vs. compositional reuse. Transformational systems are obtained via transformations of high-level specification of the desired system whereas in the second approach systems are obtained from combining components by the choice of the developers.
Black box vs. white box reuse. In the first approach products are reused as-is whereas in the second approach products can be modified to the specific application.
Abstraction reuse. Reuse applied at the level of requirements, code, design, tests, etc.
Development of reusable assets vs. application reuse.
Vertical vs. horizontal reuse. The former takes place in the same domain for example, financial object models, algorithms, frameworks; the latter is related to the assets which are created for on domain but are reused in different one. Examples of them include GUI objects, database access libraries, authentication service, and network communication libraries.
Procedures reuse. It means reusing skills and know-how. This has received significant attention from the expert-systems community while project managers tend to reuse skills informally when they reassign personnel. To encapsulate knowledge funds are needed.

Software development is divided into stages such as requirements analysis and specification, design, coding, testing and maintenance. To manage difficulties of the development process different models are proposed. Reuse methods can be divided in two groups[15]:
Generative methods. The idea is very similar to automatic programming, however while automatic programming tries to automate the whole process of software development, the generative approach tries either to automate the sequences of transformations of the process development or narrows the application domain.
Compositional methods. It is the most common form of reuse and it is based on reusing components stored in libraries as potential assets for new software developments.

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 is the systematic application of software reuse. Software reuse can be opportunistic or ad-hoc and planned[5].

Most programmers use opportunistic reuse without even being aware of it. Techniques are very simple but usually require a lot of manual editing. In this case, reuse is conducted at the individual level, not the project level. Procedures do not exist and the libraries in use contain components which are not designed for reuse thus cataloging and classifying reusable components remains a time-consuming manual task[5].

Planned reuse techniques are based on some software system especially developed to support reuse[16]. In this case, reuse is systematic and formal practices, guidelines and procedures are defined. Planned reuse requires substantial up-front investment and commitment, a significant change in the current practice of software development, demands discipline and compromise from software practitioners and yet it is difficult to predict the return on investment[5]. Systematic software reuse means: understanding how reuse can contribute toward the goals of the whole business; defining a technical and managerial strategy to achieve maximum value from reuse; integrating reuse into the total software process, and into the software process improvement program; ensuring all software staff have the necessary competence and motivation; establishing appropriate organizational, technical budgetary support; and using appropriate measurements to control reuse performance[2].

4. Factors That Facilitate Reuse

Reuse principles place high demands on the reusable components. In order to cover different aspects of their use components had to be sufficiently general but at the same time they had to be concrete and simple enough to serve to particular requirements in an efficient way. According to de Almeida et al.[2], 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. In order to determine if systematic reuse is feasible, organizations must be able to work out a cost-benefit analysis. According to Poulin[17], 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 [18].

The principles, methods, and skills required to develop reusable software cannot be learned effectively by generalities and platitudes. In order to succeed, reuse efforts must address both technical and non-technical issues. There is no agreement between authors which of these factors affects more significantly reusability. Non-technical factors include:
Economics. Investments in reuse are any of the costs in-

Copyright (c) 2013 International Journal of Computer Science Issues. All Rights Reserved.

tended to make one or more work products easier to reuse, for example, labor hours devoted specifically to classifying and placing code components in a reuse library are a reuse investment, since those hours are intended primarily to benefit subsequent activities[19].

Organizational issues. To distribute, search and sell/buy reusable assets requires a deep understanding of application developer needs and business requirements. As the number of developers and projects employing reusable assets increases, it becomes hard to structure an organization to provide effective feedback loops between these constituencies[20].

Management. It may require years of investment before it pays off; and it involves changes in the organizational funding and management structures. It can only be implemented with upper management support and guidance, without which none of the reuse activities is likely to be successful.

Educational issues. Different surveys have concluded that education is crucial to systematic reuse. To build reusable software can not only be taught in school but it requires appropriate training with developers.

Psychological issues. To make the best of reuse, developers must trust in reusable assets created from third parties. The most common psychological barrier for not accepting reuse is the syndrome "Not Invented Here".

Legal issues. 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?

Measurement. 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 [21].

Repositories. 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). The object is to create an automatically generated and indexed worldwide database of software products classified by component model. It combines introspection with Web search engines to reduce the costs of bringing software components to, and finding components in the software marketplace [22]. Technical factors for software reuse include issues related to search and recovery components, legacy components and aspects involving adaptation [11]:

Difficulty of finding reusable software. To reuse software components there should exist efficient ways to search and recovery them. It is very important to have a well-organized repository which will contain components with means to access it.

Non-reusability of found software. Easy access to existing software does not necessarily increase software reuse since reusable assets should be carefully specified, designed, implemented, and documented, thus, sometimes, modifying and adapting software can be more expensive than programming the needed functionality from scratch;

Legacy components not suitable for reuse. A known approach for software reuse is to use legacy software. However, simply recovering existing assets from legacy system and trying to reuse them for new developments is not sufficient for systematic reuse. Reengineering can help in extracting reusable components from legacy system, but the efforts needed for understanding and extraction should be considered; and

Modification. It is not always easy to find a component that works exactly as we want. Thus, modifications are necessary and for that ways to determine their effects on the component and its previous verification results should exist.

Table 1 presents a general summary of facilitators related to software reuse

Table 1. Software reuse facilitators [2]

Practice	Success
Business strategy	Yes
Organizational issues	Yes
Components	Yes
Economics	Yes
Education	Yes
Human factors	Yes
Legal issues	Need more research
Planning	Yes
Management	Yes
Measurement	Yes
Reuse process	Yes
Repository	Not if used alone
Training	Yes

5. Software Reuse Metrics

A very critical question, while trying hard to adopt reuse methods and technologies, is how much can be saved by using existing software components when developing new software systems? What is known is that a direct track of cost savings due to reuse is difficult. The easiest way to measure savings would be by analyzing the code for reuse of components. A metric is a quantitative indicator of an attribute of a thing while a model specifies relationships among metrics[23].

Copyright (c) 2013 International Journal of Computer Science Issues. All Rights Reserved.

According to Frakes[23], reuse models and metrics are categorized as following: (1) reuse cost benefits models, (2) maturity assessment, (3) amount of reuse, (4) failure modes, (5) reusability, and (6) reuse library metrics, fig. 2.

In order to justify time and cost involved in systematic reuse, organizations should be able to estimate costs and benefits. Cost-benefits models include economic cost/benefit analysis as well as quality and productivity payoff. The simplest model, the cost/productivity model, shows the cost of reusing software components, builds upon the simple model by representing the cost of developing reusable components. According to Matsumura[24] results of a reuse program implemented at Toshiba showed a 60 percent ratio of reused components and a decrease in errors by 20 to 30 percent. Managers felt that the reuse program would be profitable if a component were reused at least three times

Maturity assessment models categorize reuse programs by how advanced they are in implementing systematic reuse using an ordinal scale of reuse phases and is similar o the Capability Maturity Model. A maturity model is at the core of planned reuse, helping organizations understand their past, current, and future goals for reuse activities. Several reuse maturity models have been developed and used, though they have not been validated.

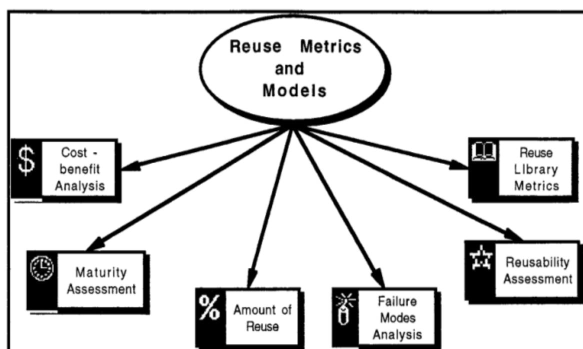

Figure 1. Reuse metrics and models[23]

Amount of reuse metrics are used to assess and monitor a reuse improvement effort by tracking percentages of reuse for life cycle objects. In general, the metric is:

$$\frac{\textbf{amount of life cycle object reused}}{\textbf{total size of life cycle object}}$$

A common form of this metric is based on lines of code as follows:

$$\frac{\textbf{lines of reused code in system or module}}{\textbf{total lines of code in system or module}}$$

To implement systematic reuse is by no doubt very difficult since it involves both technical and nontechnical factors. Failure modes analysis provides an approach to measure and improve a reuse process based on a model that shows ways a reuse process can fail. Thus failure modes analysis can be used to evaluate the quality of a systematic reuse program, to determine reuse impediments in an organization and to devise an improvement strategy for a systematic reuse program[25].

Each failure mode has failure causes associated with it. The failure modes are:

No Attempt to Reuse
Part Does Not Exist
Part Is Not Available
Part Is Not Found
Part Is Not Understood
Part Is Not Valid
Part Can Not Be Integrated

Reusability metrics indicate the likelihood that an artifact is reusable. These metrics are useful in two areas of reuse: reuse design and reengineering for reuse. We want to know whether there are any measurable attributes of a component that can indicate its potential reusability. If there are, will these attributes be goals for reuse design and reengineering. A difficulty in this area is that attributes of reusability are often specific to given types of reusable components, and to the languages in which they are implemented[26].
Library assets can be obtained from existing systems through reengineering, designed and built from scratch, or purchased. Reuse library metrics are used to manage and track usage of a reuse repository. Organizations often encounter the need for these metrics. To incorporate reusable components into systems, programmers must be able to find and understand them. If this process fails, then reuse cannot happen. Thus, how to index and represent these components so that they can be found and understood are two important issues in creating a reuse program. The evaluation criteria for indexing schemes of reuse libraries are: costs, searching effectiveness, support for understanding, and efficiency[27].

6. Conclusions

Software is starting to be noticed as the core of most of the industrial, economic and social situations in our everyday life. It is likely that, in the near future, all forms of organized human life we come across will be, somehow, mediated by software. Anyway, software has been facing

Copyright (c) 2013 International Journal of Computer Science Issues. All Rights Reserved.

a crisis since there is not enough educated human capital to produce all the software the economy and society need. Software reuse, in the form of principles, processes which are reuse centered, focused or influenced, component based development, metrics, certification, repositories, search and retrieval, as we presented and discussed over this research, is in some sense old hat. Almost fifty years have passed since the NATO 1968 conference where the problem of mass production of such complex knowledge artifacts as software was discussed at large. 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.

6. Reference

[1] Syed Raza Kirk Knoernschild, "Software reuse for business success," developerFusion. 2010.

[2] 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. 2006.

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

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

[5] I. Sommerville, Software Engineering 9e, vol. 9. Addison-Wesley, 2011, p. 790.

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

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

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

[9] J. Sametinger, Software Engineering with Reusable Components. 1997.

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

[11] A. W. Brown and K. C. Wallnau, "Engineering of Component-Based Systems," pp. 414–422, 1996.

[12] E. Dusink, "Reuse is not done in a Vacuum," WISR, 1992.

[13] M. Smolárová and P. Návrat, "Software reuse: Principles, patterns, prospects," CIT. Journal of computing and information …, 1997.

[14] M. Ramachandran, "Software reuse guidelines," IRI -2005 IEEE International Conference on Information Reuse and Integration, vol. 30, no. 3, pp. 1–8, 2005.

[15] J. S. Poulin, "The Business Case for Software Reuse: Reuse Metrics, Economic Models, Organizational Issues, and Case Studies," 9th International Conference on Software Reuse, pp. 471–516, 2006.

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

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

[18] D. Schmidt, "Why software reuse has failed and how to make it work for you," C++ Report, 1999.

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

[20] 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.

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

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

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

[24] J. Tirso, "Championing the cause: making reuse stick," … of the 5th Annual Workshop on Software Reuse, no. 914, pp. 1–6, 1992.

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

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

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

Copyright (c) 2013 International Journal of Computer Science Issues. All Rights Reserved.