Ultra Scientist Vol. 25(2)B, 317-323 (2013).

# Present development scenario for implementing software reuse

VIVEK BHATNAGAR[1] and ASHOK KUMAR[2]

**(Acceptance Date 7th August, 2013)**


## Abstract

For the development of new software software reuse plays very important role. The idea of software reuse was projected in late sixties. Software reusability is the use of engineering knowledge or artifacts from existing software components to build a new system. Reusability is the key paradigm for increasing software quality in the software development. History of software development describes that area of software reusability was not so innovative. But it is believed that software reuse is still in the development phase and has not achieved its full potential. This paper describes how far we are with software reuse research which can get success in the field of Software reusability.


*Key words:* software reuse, artifact, software component.

## Introduction

**S**oftware Reuse is defined as the process of creating software systems from predefined software components or previously existing software. Generally speaking, the benefits of software reuse far are more important than the costs. But in the chaotic world of real-world application development, the challenge can be complex. Many managers value rapid prototypes over "best practice engineers, not understanding that building on somewhat sloppy early results will typically and dramatically increase project time and cost and reduce quality. In larger organizations with shared interests, they wonder which project should bear the increased cost of building reusable software components. Who pays the cost of searching the (sometimes huge, distributed, and insufficiently documented) code-base to look for possible matches? Should a software project focus time and money on special packaging, documentation, and "marketing material to promote reuse of components it builds?

It has been associated with software engineering since its early days as the NATO Software Engineering Conference in 1968

marked the birth of the idea of systematic software reuse. Since then, reuse has become an important research area in software engineering. 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, from Raytheon's 50 increases in productivity due to a reuse rate of 60% [1], to GTE's saving of $1.5 Million from a reuse factor of 14% [2], to the Japanese software factories' claim of annual productivity increases of 20% by implementing a software reuse program[3]. It would be foolish to claim that software reuse is the solution to all the problems that have caused the current software crisis. Achieving software reuse on a level at which substantial benefits will be gained is a difficult task, and requires a great deal of commitment and effort. Tracz emphasizes that "there is no free lunch when it comes to software reuse."[4] There are, however, techniques that can help a company to maximize its resources and improve its productivity. It has been shown that reuse offers great benefits if used effectively in the right environment. This however raises the questions: how are software reuse techniques best employed and what is the right environment for software reuse to prosper? All the published examples quoted have been large, well structured companies, with top level management support for the reuse program. This suggests that software reuse tends to prosper in such an environment, 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.

*Product and Process of Software reuse :*

It is divided into product reuse and process reuse according to the reuse object. The product reuse means the reuse of software component, getting a new system from component integration and construction. The process reuse means the reuse of past software development process, automatically or half automatically producing the system using the reuse generator. The process reuse depends on the technical development in the software automation, only applicable to some special applied domain currently, but the product reuse is a realistic and essential path now.

*Methods of Software reuse :*

The software reuse can be divided into black box reuse and white box reuse according to the reuse mode of reuse information. The black box reuse means to use the component directly without modification while the white box reuse means that the component is not according to customer need hence can not be used until be properly modified according to the customer need. But in the mostly applied construction process, the adaptability modification of the component is essential[14].

*Process of Software reuse :*

i). Domain analysis phase: This phase is to come certain whether deserve to reuse the infrastructure for the domain development mainly through the definition and analysis of application domain.
ii). Property obtaining phase: This phase

includes development of reuse, may also include some exterior adopt of reuse property.
iii). Property categorizing phase: This mission is a database management mission actually, including categorizing and saving reuse property.
iv). Property maintaining phase: This mission is a maintenance mission actually.

*Tools of reuse :*

Software programming is a hard design task, mainly due to the complexity involved in the process. Reuse deals with the ability to combine independent software components to form a larger unit of software. To incorporate reusable components into a software system, programmers must be able to find and understand them. Thus Software reuse is software design, where previous components are the building blocks for the generation of new systems. These are the three or four specific tools by Reusability.

- White Box Reusability
- Black Box Reusability
- Glass Box Reusability

Reusability tools are based upon software testing development. Software reuse can apply to any life cycle product, not only to fragments of source code[18,19]. In White-box reusability is verification technique software engineers can use to examine if their code works as expected and a box can share its internal structure or implementation with another box through inheritance. One of the most common complaints of designers or print service providers when previewing and printing transparency from In Design is that a transparency effect like a drop shadow doesn't display or print correctly. Instead, a white box appears behind the transparency effect[20,21]. It discusses several benefits of component characterization, which includes improved cataloguing, improved usage, and improved retrieval and improved understanding eventually for better reuse[22,23]. In Black box reusability, the reuse sees the interface, not the implementation of the component.[24,25] If a programmer were to change the code of a black box component, compiling and linking the component would propagate the change to the applications that reuse the component. As the users of the component trust its interface, changes should not affect the logical behavior of the component. In Glass box reusability the inside of the box can be seen as well as the outside, but it is not possible to touch the inside to obtain the digital displays.

*Making Reuse Attractive :*

There are two main challenges to effective reuse within a company: technological and organizational[7]. As technology has advanced, and the methods and tools to support reuse have become available, the technological challenges facing reuse have been surpassed by the economic and organizational issues that face a company intending to implement a reuse program[8]. One of the major challenges with software reuse is that of introducing a reuse framework and method into a company. There are several steps that we have used in this process. The first, and perhaps the most important, step is to gain the support of the top level management for the reuse program[9]. This is crucial, because the introduction of a reuse program affects all parts of the software production process in the company. Therefore, the support

of the high level management in charge of all aspects of development must be gained to allow changes to be supported and implemented and to allow changes to company policy to be made if needed[10].

Hence we conclude that the support of the high level management by giving a presentation on reuse, explaining how it could help in their company and how to best utilize it. This was a good opportunity to present the case for reuse, stressing the benefits that it could bring to the company, and the approach to introducing reuse into a small company that we were recommending. It was also a good point at which to get feedback from the management on what they expected from the reuse program, and how they wanted the company to change for the future. Suggestions were then made how to set up a reuse program into the company, along with the costs that would be associated with the setting up of this program, and a consideration of the risks involved.

*Implementation of Reuse :*

This research has recommended an incremental approach to introducing a reuse program within a small company. Although software reuse can offer major productivity gains when used in the right way in the right environment, it is not the solution to all problems. There will always be times when it will be more effective to write new code than to try to find and reuse old code. The key is to recognize which techniques will be most effective in different development environments, and utilize the most efficient development strategy in each case.

*Present Reuse scenario :*

The Standish Group has published a report to describe and analyze the projects in the software industry. Since this industry is so immature, the results presented in the report are so incredible. In that first report about 15% of the projects were classified as successful projects. Almost 15 year later, the successful projects are only a third of all projects in the software industry. Considering that the global economy tends to demand more and more software over the next years, emerges the necessity for a new way of develop software, a new modus operandi (mode of operation) in which we can move the software industry from the prehistoric age, from the craftsmanship to a new context based on manufacturing. The software factory approach leads us to put beyond all empiric methods that brought the software industry to this point. This approach address the economy of scope (completed by the economy of scale) in order to increase the Return on Investment (ROI). A software factory also allows the introduction of systematic methods developing software. Greenfield defines a software factory as “A software factory is a software product line that configures extensible tools, processes and content, using a software factory template based on a software factory schema, automate the development and maintenance of variants of an archetypical product, assembling and configuring framework-based components “ Thus, since we know that specific approaches are more efficient (and more limited) than generic approaches, emerges the concept of Domain Specific Languages (DSL). Deursen defines a DSL as a: “A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem

domain". Furthermore, in this context, the developers are focused in creating products based on the definition of the problem domain (not being worried about memory variables). So, it's believed that the productivity, reliability, maintainability and portability of the software will increase so much. Then, the developers can associate semantic transformers to generated code from the models expressed in a DSL form. Therefore, DSL can be an important tool to allow the software factoring become actually successful. However, we must be critic and introduces some questions:

- How efficient the DSL are to be used in the software factory context?
- Why using DSL instead executable models in UML + Semantics?
- How many successful cases exist using DSL in the industry?

The results gained so far have been varied. There have been some successes, but the challenges and difficulties encountered during the course of the project have also been interesting. Small companies are unique in their need to keep up with market trends, and succeed in every project that they undertake. Unlike larger companies, and even single project teams within a large organization, a small company cannot afford to fail in any project, because the livelihood of the company, and every employee, depends upon keeping their market share. In the company with which this project was associated, their business was dependent upon a single product. If that product failed, then the company would cease to exist. This does not compare with even isolated parts of a large company, because although the project may fail and cause difficulty within the company, this would not cause the collapse of the business.

It was found that with adoption of reuse 6 %to 40% increase in productivity was observed and 12 %to 42% less time was required for the development. In spite of this quality was improved by a factor 24% to 76%. It also helps the specialists to optimize software architectures and assets that may then be reused by others who are focusing on meeting product features and markets needs. Along with these factors reuse provides consistency and interoperability across products.

**Conclusion**

The above article describes how software reuse play important role in developing software in spite of having number of hurdles. Previous results shows that there have been some successes, but the challenges and difficulties encountered during the course of the project have also been interesting. As small companies exclusively try to complete their project and make use of little of reuse or sometime no use of reuse. Fear of failure in project for small companies cannot be afforded by the small companies because the livelihood of the company, and every employee, depends upon keeping their market share. All this lead to poor reuses activities in most software development organizations. One of the main reasons for the low reuse activity is that developers simply do not attempt to reuse because of a number of factors, including lack of knowledge about reusable assets and the notion that the cost for reusing is higher than the cost of developing new code. Reuse plays an important role in increasing efficiency of developer for producing large product within limited amount of time and with less risk, high quality. At this technique is

adopted by large companies. Though small organizations has also started to develop software by adopting reuse, but still most of these organizations afraid, because development with reuse is costly. As for this specialist persons and large repositories are required, which causes extra cost to the organizations. Various tools and techniques have been developed to achieve the reusability. Hence it is concluded that software reuse makes our development cheap and effective which is the main goal of software reuse. Hence in future reuse will play a vital role in development and will be the key term.

**References**


1. Lanergan, R.G., Grasso, C.A., 'Software Engineering with Reusable Design and Code'; *IEEE Transactions on Software Engineering;* Sept; Vol. 10 No.5 P 498-501 (1984).
2. Prieto-Diaz, R., 'Implementing Faceted Classification for Software Reuse'; In: Proc.of 12th International Conference on *Softwware Engineering; IEEE, Nice, France;* Mar; P300-304 (1990).
3. Matsumoto, Y., 'Some Experiences in Promoting Reusable Software: Presentation in Higher Abstract Levels'; In: Software Reusability. Concepts and Models, Vol. II; Biggerstaff, T.J., Perlis, A.J. (ed.); ACM Press, Addison-Wesley, Reading, Mass.; (1989).
4. Tracz, W., 'Software Reuse Maxims'; ACM Software Engineering Notes; Oct., Vol. 13 No. 4 P28-31 (1988).
5. M. Friedewald *et al.*, "Status of the Software Development Industry in ermany," *In formatik Spektrum,* Vol. *24*, no. 2, pp. 81–90 (2001).
6. P. Louridas, "Using Wikis in Software Development," *IEEE Software,* Vol. *23*, no. 2, pp. 88–91 (2006).
7. B. Leuf and W. Cunningham, The Wiki Way: Quick Collaboration on the Web, Addison-Wesley (2001).
8. S. Lauesen, Software Requirements: Styles and Tech-niques, Addison-Wesley, (2002).
9. Boehm, B., "Managing Software Productivity and Reuse", *Computer,* Vol. *32*, issue *9,* Sept., pp. 111-113 (1999).
10. Crnkovic, I., Larsson, S., Stafford, J., "Component- Based Software Engineering: *Building systems from components",* ACM SIGSOFT, Vol. *27*, no. 3, May, pp. 47-50 (2002).
11. Bacili, K., Oliveira, M., "DigitalAssets Manager: sharing and managing software development assets", OOPSLA'06 Demo Session, ACM, NY, pp. 700-701 (2006).
12. Oliveira, M., Garcia, I., Nunes, A. (2005). "RCCS: uma Rede de Compartilhamento de Componentes de Software", Brazilian Symposium of Computers Networks (SBRC), Fortaleza, Brazil (2005).
13. Zhang Qiu-yu, Zhang Dong-dong *etc.* The research andapplication of special domain software reuse technology [J]. *Computer Engineering and Application, 40(14),* 213-215 (2004).
14. Li Ji-hong.The software-reuse technology based on reuse component [J]. *Shanxi Coal Management Cadre Institute Transaction, 17(3),* 109-110 (2004).
15. Hafedh Mili, Ali Mili *etc.*. Reused-Based Software Engineering—Techniques, Organization, and Controls [M]. Beijing : Electronic Industries Publishing Company, (2004).
16. Arango G, Prieto—Diaz R. Domain

analysis :Concepts and re-search directions [M]. IEEE Computer Society Press (1989).
17. Microsoft Corporation; 'OLE 2 Classes for the Microsoft Foundation Class Library'; Microsoft Corp. (1993).
18. S. Oliver, K. Dougan, K. Kersch, C. Kitson, G. Sherman and L. Wojciechowski. Unit testing a component of verification of scientific modeling software. In T.I. Oren and G.B. Birta, editors, 1995 Summer Computer Simulation Conference, pages 978– 983. The Society For Computer Simulation (1995).
19. N. Ramsey. *Literate programming simplified. IEEE Software,* September (1994).
20. L. Wall, T. Christiansen, and R. Schwartz. Programming Perl. O Reilly & Associates,
21. Morris Street, Sebastopol, CA 95472, second Edition (1989).
22. E. Yourdon. Modern Structured Analysis. Yourdon Press.
23. J. Poulin, J Caruso and D Hancock, "The Business Case for Software Reuse, IBM *Systems Journal, 32(40),* 567-594 (1993).
24. Eun Sook Cho *et al*., "Component Metrics to Measure Component Quality", Proceedings of the eighths Asia-Pacific *Software Engineering Conference,* 1530-1362/01.
25. Hironori Washita, Hirokazu Yamamoto and Yoshiaki Fukazawa," Software Component Metrics and its Experimental Evaluation," Proc. of the International Symposium on Empirical Software Engineering (ISESE 2002), October World Academy of Science, Engineering and Technology 33 200739 (2002).
26. Gamma E., Helm R., Johnson R., Vlissides J., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesely, Professional Computing Series, Reading, Massachusetts, (1994).