An Extensive Conception of Reusability in Software Component Engineering

Devesh Kumar Srivastava and Priyanka Nair

Abstract In early 1960s, intricacy of software systems led to a call for the emergence of the concept of Software Reuse. Rather than building software applications from genesis, software reuse consents creating software systems from existing software. Efficient software reuse programs implemented by the firms may increase their productivity and value, thereby giving the organizations headway. Several reuse metric and models reign the software industry. Reuse assessment commit to high quality and economic system development. Despite its commencement as a potent vision, software reuse has botched to become a part of the typical software engineering practice. The paper is an attempt to articulate the notion of software reuse and the concerning issues. Reusability facet has been conferred analogous to OO paradigm and agile development. Here the concept of reuse has been addressed as a combination of artifacts as well as individual components.

Keywords Software reuse · Reusability · Reuse approaches · Software reuse metrics · Agile software development · Object oriented paradigm

1 Introduction

Reconstruction of new systems pertaining to changing requirements is not viable. Software components can be used time and again for creating new systems and applications. Components can be integrated into software systems. Everything associated with a software that can be reused is termed as software reuse. Software Reuse leverages the project structure and cost effective issues of software engineering. However, Reusability is difficult to maintain and its inclusion in new systems is even more severe [1]. The NATO Software Engineering Conference,

D.K. Srivastava (✉) · P. Nair
Department of CSE/IT, Manipal University, Jaipur, India
e-mail: devesh988@yahoo.com

P. Nair
e-mail: priyankanair@live.com

© Springer Nature Singapore Pte Ltd. 2017

N. Modi et al. (eds.), *Proceedings of International Conference on Communication and Networks*, Advances in Intelligent Systems and Computing 508,
DOI 10.1007/978-981-10-2750-5_47

1968, gave the prime valuable coverage to the bottlenecks of software engineering. From amongst various experts who attended the conference, McClory in his working paper proposed the notion of necessity and adequacy of reusable component factory. He contended the effectiveness of using component libraries for various system processing and computations [2, 3]. The code level reusability is coherent as compared to the conception of specification and design reusability which is challenging [1]. The problem of dealing with software reuse is the radical fixate with additional proposition of measurement of reusable components.

2 Approaches of Software Reuse

To realize software reuse work, conventional approaches are employed. On the frontier, the classification is primarily based on component level and process level. Reuse based on object of reuse or component is the *Compositional Reuse approach* whereas process reuse fall under *Generative Reuse approach*. In sync, these approaches serve as reuse aid to the system [3, 4].

Compositional Reuse appropriates the notion of reusable objects that are unaltered during reuse. It is a bottom up system development. Combinative accessions of simpler components frame obscure and complex objects [1]. Components that are compatible with reuse support features are archived in repositories. Retrieval is a key feature here. Components are dispersed segments which benefits the developers to achieve high productivity. *Generative Reuse* is reuse of process rather than product. Parsers and Lexical analyzers are based on generative approach. Reusable pattern generation is taken into account before assimilating objects of reuse into the program [1].

3 Types of Reuse

Reusability scrutinized over domain scope can be categorized in two forms: *Vertical Reuse* and *Horizontal Reuse. Vertical Reuse* is generative in nature. However it has not yet been widely accustomed in software business industry. In software development it has an impending and extensive connotation [5]. However, *White Box Reuse* is strenuous to maintain. It is an elemental form of *Vertical Reuse*. Code is modeled as the reused entity for white box reuse. The access to the source code and implementation is required herewith. Reuse is met with alteration and adaptation as the core [6, 7] *Horizontal Reuse* is widely accustomed across applications. It follows compositional reuse approach. *Black Box Reuse* forms the domain component of horizontal reuse. Component reuse is carried out without modifications. It employs Commercial Off the Shelf (COTS) which is a third party application. They are economical and reliable. COTS components are incorporated in already built software in order to provide additional services. However, they are employed for general applications [8]. It is well appropriated as Black box reuse as they are perceived only in terms of input and output without taking into account the functionality.

4 Reuse Assessment

In order to identify the effectiveness of various reuse methods, it is imperative to quantify and assess them. Various software pertinent metric can be employed as quantitative index to measure the reusability in terms of software assets: *product* and *process.* Some of the reuse metric models have been taken herewith.

4.1 Cost/Productivity Metric Model

There is an additional cost associated with software reuse. Reuse cost is viewed as an investment. Reuse incurs added cost to the traditional software development process. The cost model was based on cost benefit analysis [6]. The two models for cost and productivity commit to the cost of *reusing* software components and the cost of *developing* objects of reuse. The software reused is decisive and reliable thereby conforming to the black box properties. Apropos the properties, enough documentation related to the objects of reuse is available but the size remains non-existent. Negligible cost is associated with the reuse of components [9]. For estimating the relative size of reusable components, it is required to measure the size of object of reuse with the hypothesis that they are built from scratch. The relative size, R of reusable components is hereby articulated as:

$$R = \frac{S_R}{S_R + S_E} \tag{1}$$

where,

S_R estimated size of reusable components
S_E effective size of reusable components which is a regulated consolidation of altered and new source code.

The higher order cost model estimates the cost of developing objects of reuse. Let C_D be the relative cost of developing the software product corresponding to all current code and b is the cost relative to all new code, of using the reused code in the new product. C_D and b for all new code is assumed to be 1. The relative cost of software development is presented as follows:

$$C_D = 1\,(1 - R) + bR \; Or \; C_D = R\,(b - 1) + 1 \tag{2}$$

where,

R: proportion of reused code in the product $(1 - R)$: proportion of all new code.

According to Gaffney and Durek, when only source code is reused b = 0.85 whereas when requirements, design and code are reused b = 0.08. It is because in the former case all other phases are required to be endured [8].

The productivity, P is the inverse of *cost metric*

$$P = \frac{1}{C_D} \tag{3}$$

or

$$P = \frac{1}{R(b-1)+1} \tag{4}$$

Developing Software with reusable component incurs more cost as compared to developing software without reusable objects [6, 9].

Consider a small module of an ongoing Omnicare Healthcare Management Project. We will use the Gaffney and Durek Model to calculate the relative cost and productivity of the project module with reuse components.

Table 1 indicates the development cost corresponding to different stages of development of the module. Prototyping or design phase incurs the maximal cost with respect to other development stages (Fig. 1). Relative reuse cost can be calculated by taking into account the objects of reuse with additional activities. If we take source code as our object of reuse then requirement analysis and testing are to be performed as accompanying tasks. Here b = 0.33 (0.07 + 0.26) similarly, when

Table 1 Relative development cost corresponding to different phases of developing the module

Development phase	Relative development cost (phase wise)
Requirement analysis	0.07
Prototyping	0.42
Implementation	0.25
Validation and testing	0.26

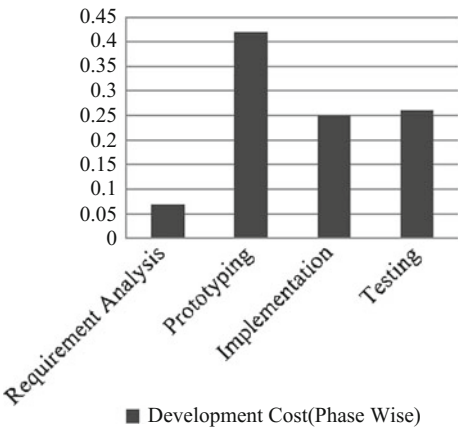

Fig. 1 Cost of developing the software corresponding to different phases of module development

Table 2 Integration cost of reusable component

Object of reuse	Accompanying tasks to be completed	Relative cost of component with reuse (*b*)
Requirement	Prototyping, Implementation, Testing	0.93
Prototype	Requirement Analysis, Implementation, Testing	0.58
Code	Requirement Analysis, Testing	0.33
Requirement, Prototype, Code	Testing	0.26

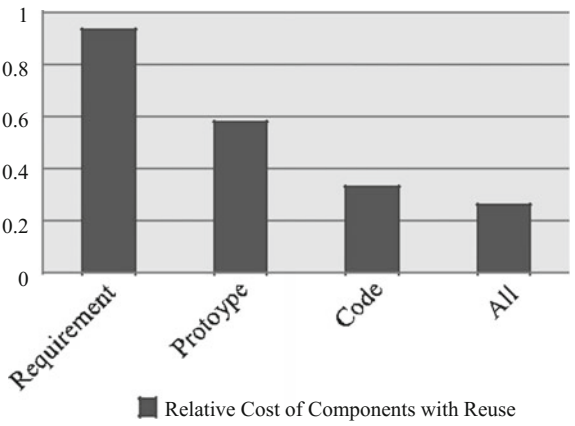

Fig. 2 Relative cost of components with reuse (cost of integrating reusable components)

requirement is reused then the relative reuse cost will be 0.42 + 0.25 + 0.26; i.e. b = 0.93. Table 2 shows the relative reuse cost of development. With requirement taken as the object of reuse holds the highest relative reuse cost (Fig. 2).

Suppose there are 10,000 lines of code in the original software application. Assuming 4700 lines of code is reused. Here $C_D = 1$ (*relative cost of development of all new code is assumed to be* 1). The proportion of reused code R is 0.47 ($R \leq 1$). From Fig. 2, b = 0.33 for source code taken as the reused component. *b* is the integration cost of reusable component. Using the Gaffney and Durek model, we can calculate the cost and productivity of development of system with source code as the object of reuse.

$$C_D = 1\,(1 - 0.47) + 0.33\,(0.47) = 0.69 \tag{5}$$

Taking source code as the object of reuse, the relative cost and productivity of development of the module is estimated as 0.69 and 1.4 respectively. Cost of developing the module relative to all new code is assumed to be 1.

4.2 Maturity Metric Model

Maturity metric model is used to assess the implementation and effectiveness of systematic reuse activities. The advancement of reuse programs is measured on an ordinal scale [10]. Kolton and Hudsen Reuse Maturity Model is a five level model that directs an organization for effective reuse of activities in order to achieve maximal performance.. The levels are: *initial/chaotic*, *monitored*, *planned*, *coordinated* and *ingrained.* At the onset of any program, organizations are traceably between *initial/chaotic* and *monitored* level. Post *ingrained level*; reuse is viewed as a unified part of the system [1, 10].

4.3 Percent Reuse

To assess the reuse rate, calculating the percentage of reuse, is viewed as an essential metric. Substantially, amount of reuse is the ratio of the amount of object reused to the total size of the object considering the life cycle of the program or system [6]. Moreover, determining the amount of reuse on account of *lines of code* (*LOC*) in a program is trivial. Hence,

$$\%Reuse = \frac{Reused\ LOC\ in\ a\ Software}{Total\ Size\ of\ the\ Software\ (LOC)} * 100 \tag{6}$$

Consider a software application with 10,000 lines of code. Assuming 4,700 lines of code of the same application is reused to develop a new software product. The percentage of reuse is calculated as follows:

$$\%Reuse = \frac{4700}{10000} * 100 = 47\% \tag{7}$$

Higher percentage of reuse is indicative of better reuse rate.

5 Agile Development and Reuse

The extension of reusability concept in agile development is complex. The agile development is the software development methodology that focuses on continuous improvisation with effective communication between people. However, there is minimal documentation which makes reusability critical. The major limitation with reuse in agile environment is the difficulty in continuous redesign, due to paucity of application-specific artefacts. [11] Reusability can be employed with agile software development in three ways. The methodologies used for incorporating reusability in

agile development *Component Based Development (CBD)*, *Refactoring* and *Reusable Architectures. CBD* validates the component in conformance with the suitability for reuse. *Refactoring* reorganizes and remodels an existing code. It can be viewed as a template or design that can be employed in varied scenarios pertaining to requisite applications. Architectural patterns may be used to develop *reusable architectures* [11, 12].

6 OO and Reuse

In the 3rd International conference on software reuse it was substantiated that object oriented paradigm does not validate to be the necessary and sufficient condition to support reusability. Some of the credos of OO obstruct the reuse and hence there is need to be very cautious while incorporating its features [13]. Despite the restraints, OO approach complements features that support software reuse. The Object Oriented paradigm considerably enhances the productivity with reuse in an elemental role. The assortment and contrast of programming languages becomes a key consideration when incorporating reusability to improve the productivity [3, 9]. OO braces both types of reuse. *Inheritance* backs White Box reuse whereas *Client-Supplier relationship* supports black box reuse [10]. Much of the assistance of Object Oriented paradigm to reusability is on probation. However, some of the precepts of OO approach need to be scrutinized with respect to violations to the reuse support.

7 Issues with Reusability

Software reuse work can only be accomplished with the employment of any or both of the *reusable assets*: product and process. However, building software with the reusable assets raises certain methodological and technical concerns. Issues are often related to spotting of the reusable assets and identifying their conformance to the current requirements. For ensuring the adaptation of these reusable assets to the current needs automation of the reusable components is met employing OO features [3]. There is very little tool support for locating the reusable components and maintaining a component prospectus. However certain tools like CASE tools are viewed as a way to improve and promote reuse in software projects in organizations. Computer Aided Software Engineering (CASE) tool is used for retrieval of reusable components from a software catalogue [14]. Reusable Components are fragments that are stored in repositories. Another major issue that crops up is the reuse barrier. Efficient retrieval is necessary for development of reusable software. The repository keeps changing constantly which makes it difficult for the developer to foresee the occupancy of object of reuse [15]. Also, there is an additional cost associated with development of software with reusable components. It is more

challenging to reuse specification and design as compared to the code reuse. To reuse specification/design a reserve of solutions is required to be searched in a problem-oriented demeanor [16].

8 Conclusion

The above sections elucidate the imperative aspects of software reuse. Software reuse reinforces software productivity and quality. Software repository is essential for maintaining the catalogue of reusable software components. Reusable assets are conferred in terms of product and process. To carry out software reuse, component level and process level reuse approaches are prevalent in industry. From amongst the various software metrics employed for the measurement of different reuse techniques, cost/productivity metric model, maturity assessment and percent reuse estimation have been taken up. The cost is negligible when reusing the components. However an additional cost is incurred when developing software with reusable components. Despite the efficacy of reusability, there are issues raised while developing software with reusable components. With Agile development, reusability facet becomes complex with the limitation of documentation. However reusability can be incorporated with the agile environment employing various methodologies. Also, even though OO paradigm has been widely used for supporting conception of reuse, it has not been empirically proven to be the necessary and sufficient condition for reuse support. Reusability in Agile method is an open area for researchers for further improvement.

References

1. Sametinger, J *Software Engineering with Reusable Components.* Springer Science and Business Media. (2013).
2. Naur, P., Randell, B., & Committee, N. S. *Software Engineering*: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7–11 Oct. 1968. NATO Software Engineering Conference, (October 1968), 231. http://doi.org/10.1093/bib/bbp050 (1968).
3. Jalender, B., Govardhan, D. a., & Premchand, D. P. (2010). A Pragmatic Approach To Software Reuse. Journal of Theoretical and Applied Information Technology (JATIT) Vol, 14, 87–96. Retrieved from http://www.jatit.org/volumes/research-apers/Vol14No2/3Vol14No2.pdf.
4. Czarnecki, K. Overview of Generative Software Development. Unconventional Programming Paradigms, 3566, 326–341. http://doi.org/10.1007/11527800_25. 2005.
5. Jamwal, D. *Software Reuse : A Systematic review*. Proceedings of 4th National Conference; IndiaCom, 1–7, (2010).
6. Marshall, J. J., & Downs, R. Reuse readiness levels as a measure of software reusability. International Geoscience and Remote Sensing Symposium (IGARSS), 3(1), 1414–1417. http://doi.org/10.1109/IGARSS.2008.4779626, (2008).

7. Dosch, W., Lee, R.Y., Wu C. *Software Engineering Research & Applications*. Springer, 172 (2006).
8. Galorath, D. *Software Reuse and Commercial Off-the-Shelf Software*, Galorath Incorporation, El Segundo, CA. 1–22. Retrieved from http://www.compaid.com/caiinternet/ezine/galorath (2007).
9. Tripathy, P., Naik, K. (2014). *Software Evolution and Maintenance*. John Wiley & Sons.
10. Soora, S. K. *A Framework for Software Reuse and Research Challenges*, IJARCSSE, 4(10), 441–448, (2014).
11. Spoelstra, W. J. T. *Reusing software assets in agile development organizations—a management tool*. University of Twente, Hengelo, (2010).
12. Singh, S., & Chana, I. *Enabling Reusability in Agile Software Development*. International Journal of Computer Applications, 50(13), 33–40. http://doi.org/10.5120/7834-1132, (2012).
13. Patidar, R., & Singh, P. V. OPEN ACCESS A Survey of Software Reusability, 4(8), 96–101, (2014).
14. Sharma, A., Grover, P. S., & Kumar, R. Reusability assessment for software components. ACM SIGSOFT Software Engineering Notes, 34(2), 1. http://doi.org/10.1145/1507195.1507215. (2009).
15. Shiva, S. G., & Shala, L. A. S*oftware Reuse: Research and Practice.* Information Technology,. ITNG '07. Fourth International Conference on, 603–609. http://doi.org/10.1109/ITNG.2007.182, 2007.
16. Sharma, K., Agnihotri, N., & Hooda, M. Software Reusability: Possibilities From The Existing Software, 97–99. (2013).