International Journal of Hybrid Information Technology
Vol. 10, No.11 (2017), pp. 31-40
http://dx.doi.org/10.14257/ijhit.2017.10.11.04

Conventional Program to Component-Based Software: A Critical Survey on Software Reuse

Umesh Kumar Tiwari[1] and Priya Matta[2]

[1,2]Graphic Era University, Dehradun
uneshtiwari22@gmail.com, mattapriya21@gmail.com

Abstract

Component-Based Software Engineering is an elite form of software engineering that offers the feature of reusability. Reuse of software artefacts and the process of reusability make CBSE a specialized paradigm of software development. CBSE stands on the philosophy of "the buy, don't build". The basic idea of CBSE is to reuse pre-constructed and available software constructs rather than developing them from the beginning. In this work we have performed a deep literature survey and a comparative analysis on the basis of reusability techniques offered by eminent practitioners considering three categories of development paradigms; conventional, object-oriented and component-based software. Literature review is done on three basic criteria: measures and metrics used, key finding of every paper and factors affecting the property of reusability.

***Keywords**: Component-based software, reusability, paradigm, conventional, object-oriented*

1. Introduction

Gaedke [1] defined Component-Based Software Development (CBSD) as "Component-Based Software Development (CBSD) aims at assembling large software systems from previously developed components (which in turn can be constructed from other components)". CBSD methodology promotes the development of software systems by picking suitable and apposite pre-existing, pre-built and reusable (off-the-shelf) software work-products called 'components' and integrating those components with a pre-defined architectural design. CBSD emphasizes on the thought of assembling heterogeneous, context-independent and pre-designed components to develop software applications. Component-Based Software applications are constructed by the assembling of reusable, pre-existing as well as new components which are integrated through error-free interfaces. The objectives of Component-Based Software Engineering are to develop extremely large and complex software systems by integrating 'Commercially Off-the-shelf Components (COTS)', third-party contractual components and newly developed components to minimize the development time, effort and the cost [2]. Component-Based Software Engineering offers an improved and enhanced reuse of software components with additional properties including flexibility, extendibility and better service quality to discharge the needs of the end users.

In CBSE the purpose is to develop a component (including classes, functions, methods or operations) only once and re-use them in various applications rather than being re-constructed every time. Reusing pre-developed and pre-tested components make the development life-cycle shorter, help to increase the reliability of the overall application, and reduce the time-to-market.

Received (July 10, 2017), Review Result (October 23, 2017), Accepted (October 30, 2017)

The CBSE development paradigm is used to develop generalized as well as specific components. The development processes of CBSE applications deploy four parallel processes, which are involved in the creation of more than one application concurrently, as described in Figure 1.2. Processes including new component development, selection of pre-existing components from the repository and integration of components take place concurrently in the generation of application in the Component-Based Development. With each process there must be a feedback method to address the problems and errors like, component selection problems, problems in developing new components, interaction and integration errors among the components, and their side effects.

2. Software Reuse

All Researchers and practitioners have given an array of definitions for the term 'software reuse'. Software reuse is the process of constructing software from existing software artefacts rather than developing it from the foundation. Krueger defined software reuse as, "the process of creating software systems from existing software rather than building them from scratch" [3]. Software reuse is the process of integrating predefined specifications, design architectures, tested code, or test plans with the proposed software. According to the Freeman, "reuse is the use of any information which a developer may need in the creation process", [4]. Basili and Rombach suggests reuse as the use of existing knowledge of any type or category mapped with the under development software project [5]. In the Tracz view, reuse means the use of software systems that were developed in past for reuse [6]. Braun considers the reuse in terms of use of existing components in the same context or in a context free environment [7].

Some eminent researchers have defined reuse in terms of adaptability or modifiability of the reused artefact. Lim proposes that the work-products and the delivers of the previous software are to be used in new software development without any modification [8]. On the other hand, Mcllroy argued the reuse as the use of off-the-shelf components with the ability to modify but in a managed and controlled manner [9]. Cooper described software reuses as the potential of use of a formerly designed software component repeatedly, with minor, major or no modifications [10]. Software reusability is the level of usefulness or the extent of reuse. Reusability is defined as the amount of the ease with which we can use pre-designed objects in the new context.

In literature, software reuse has been defined as a concept or a process whereas software reusability is defined as the outcome of that process. Reusability is the level of implementation of reuse in actual. Reusability acts as a measurement tool to quantify the proportion of reuse of a pre-designed, implemented and qualified artefact. This work defines a quantifiable approach to measure the reusability of a component in Component-Based Software.

3. Literature Review

The Reusability is the focal-point of Component-Based Software Development. Software reusability is such an approach that defines the effective reuse of pre-designed and tested parts of experienced software in new applications. In CBSE, we integrate components of all classes according to the design architecture and applications requirements. There are some components for which code is not available called Black-Box components; some components may be available with their code and documentation, which are known as White-Box components. In literature, various researchers have classified software reusability quantification methods into different categories.

Prieto *et al*. [11] defined some attributes of a program and related metrics to compute reusability in their work. They proposed that reuse depends on size, program structure, documentation, programming language and reuse experience. Further, they used lines of code to count the size, Cyclomatic complexity for structure, rating from 0 to 10 for documentation, inter-module language dependency to estimate difficulty of modification, and experience of using same module.

Caldiera and Basili [12] proposed one of the earliest methods to identify and qualify reusable components. They defined cost, usability, and quality as the three factors affecting the reusability. They characterized components reusability using four metrics: Volume like, operands and operators, using Halstead Software Science Indicators. Cyclomatic complexity using McCabe's method to compute the component's complexity, Regularity measures the component's implementation economy, and Reuse frequency that is the indirect measure of the functional usefulness.

Jeffrey [13] identified two basic categories of reusability models: empirical and qualitative. *Empirical models* use experimental data to estimate complexity, size, reliability and similar issues, which can be used by automated tools to estimate the reusability. The *qualitative models* stay with pre-defined assumptions and guidelines to address issues like quality and certification.

Chen *et al*. [14] presented their findings based on a large number of reused components. They computed size, program volume, program level, difficulty to develop, and effort for all these reused components. Their conclusion is that, to increase the productivity we should decrease the values of these metrics.

Gregory [15] defines the theory of function, form and similarity to compute the software reusability. *Function* defines the actions of a component, *form* characterizes the attributes like structure and size and the *similarity* identifies the common properties of components. Author uses the well-defined metrics like McCabe's complexity metric in his calculation.

Barnard [16] reused components developed in C++, Java and Eiffel to show his experimental findings. His work was based on estimations of component's attributes like simplicity, genericity and understandability through available properties, methods and defined interfaces.

Lee and Chang [17] suggested metrics including the complexity and modularity of components to predict the reusability and maintainability of object-oriented applications. They defined complexity metrics as Internal-External Class Complexity, and modularity metrics as Class Cohesion-Coupling.

Cho *et al*. [18] proposed component's reusability measures as the fraction of total interface methods and the number of interface methods in the component that have common functionality in their domain. Reusability is assumed higher as the value of ratio increases. They also defined Customizability metrics as the ratio between the customization methods and the count of methods present in the interface of that component.

Boxall and Araban [19] find in their study that the reuse level of a component is greatly affected by component's understandability. They derived the value of understandability by using the attributes of component's interfaces. In their work, Boxal and Arban taken interface size, counts of the argument, number of repetitions, scale of the repetitions and similar attributes to suggest some metrics for understandability.

Washizaki *et al*. [20] suggested a Reusability-Model for black-box components to reuse components efficiently. Authors identified some reusability affecting factors like functions offered, level of adaptation in new environments and varied requirements. They proposed metrics for better understandability of components.

Bhattacharya and Perry [21] focused on integration contexts of components rather than concentrating just on internal attributes of components. They proposed reusability estimations considering the integration architecture of the software. They have proposed Components characteristics and Architecture compliance metrics to measure the issues related to particular properties and integration issues of components.

Gui and Scott [22] suggested metrics to estimate the coupling properties among inter-components and cohesion properties of intra-components in their own terms. They used java components to propose their metrics.

Gui and Scott [23] further defined a metric-suite to compute the reusability of Java components and ranked them according to their reusability. These metrics are used to estimate the indirect coupling among components as well as to assess the degree of coupling.

Wijayasiriwardhane and Lai [24] suggested a size measurements technique named Component point to estimate the size of overall component-based systems. These Component points can be reused as a metric to analyse components for future use. They defined three classes of components according to their usability.

Danail *et al*. [25] categorized reusability metrics into two broad classes: the White-Box metrics and the Black-Box metrics. *White-box reusability metrics* are based on the logic and coding structure of the software. *Black-box reusability metrics* are based on interfaces and other attributes, as code is not available for black-box components.

Reusability measures defined in literature are analysed by considering development paradigms like: Convention Software and Programs, Objet-Oriented Software, and Component-Based Software and summarized in Table 2.1.

Table 2.1. Summary of Reuse and Reusability Issues

Paradigm	**Measures and Metrics Used**	**Key Findings**	**Factors affecting reusability**	**Author(s)/ References**
Conventional Software and Program	• Lines of code, • Cyclomatic complexity, • Documentation Rating from 0 to 10, • Inter-module language dependency to estimate difficulty of modification, • Experience of using same module.	• Authors have defined attributes of a program and related metrics to compute reusability in their work. • They proposed that reuse depends on size, program structure, documentation, programming language and reuse experience.	• Size, • Program structure, • Documentation, • Programming language, • Reuse experience	Prieto-Diaz, Ruben and Peter Freeman [11]
Component-Based Software	• Halstead Software Science Indicator to find volume, • Cyclomatic complexity using McCabe's method.	• Regularity measures the component's implementation economy. • Reuse frequency that is the indirect measure of the functional usefulness.	• Cost, • Usability, • Quality	G. Caldiera and V.R. Basili [12]
Componen	• Automated tools	Two categories of	Cost, and	Jeffrey S.

t-Based Software	to estimate the reusability, • Predefined assumptions	reusability: • Empirical models use experimental data to estimate complexity, size, reliability and similar issues. • Qualitative models stay with pre-defined assumptions and guidelines to address issues like quality and certification.	Reliability	Poulin [13]
Object-Oriented Software	• Used reused components, • Modularity, • Complexity	• Presented their findings based on a large number of reused components. • Increase the productivity we should decrease the values of these metrics	• Size, • Volume, • Level, • Difficulty to develop, and • Effort	Chen, Deng-Jyi and P.J. Lee [14]
Componen t-Based Software	• Structure of components, • McCabe's complexity	Defines the reusability on the basis of: • Function defines the actions of a component, • Form characterizes the attributes like structure and size and the • Similarity identifies the common properties of components.	• Functions of the components, • Size, • Complexity of components	W. Gregory Hislop [15]
Componen t-Based Software	• Reusable components, • Automated tools	• Proposed the importance of reusability in his experimental findings. • His work was based on estimations of component's attributes like simplicity, genericity and understandability through available.	• Available properties, • Methods and • Interfaces	J. Barnard [16]
Object-Oriented Software	• Internal-External Class Complexity, and • Modularity metrics as Class Cohesion and, • Class Coupling.	• To estimate the reusability and maintainability of object-oriented software metrics considering the complexity and modularity of components.	• Complexity • Modularity	Y. Lee and K.H. Chang [17]

Component-Based Software	• Metrics for reusability, • Metrics for Customizability.	Proposed metrics for Reusability and Customizability: • The ratio of total number of interface methods to the number of interface methods in the component that have common functionality in their domain. • Reusability is assumed higher as the value of ratio increases. • Ratio between the customization methods and total number of methods present in the interface of the component.	• Total number of Interface methods, • Total number of common function Interface methods, • Total number of Customization methods, • Total number of common function Customization methods.	E.S. Cho, M.S. Kim, and S.D. Kim [18]
Component-Based Software	• Size of interfaces, • Number of arguments, • Scale of repetition of arguments	• In their study they argued that the reuse level of a component is greatly affected by component's understandability. • Derived the value of understandability of a component by using the attributes of interfaces of that component.	• Understandability, • Interface property	M. Boxall, and S. Araban [19]
Component-Based Software	• Indicates these values as high or low, • Structured relationships between understandability, adaptability, and portability.	• Proposed Component Reusability Model, • Component's Observability: Readability of component/Total Properties. • Component's Customizability: Writability of component/Total Properties.	• Functionality of the component, • Adapting the changes in requirement, • Porting to the new environment	H. Washizaki, Y. Hirokazu and F. Yoshiaki [20]
Component-Based Software	• Interaction contexts, • Properties and attributes of components • Architecture of CBS	• Focused on integration contexts of components rather than concentrating just on internal attributes of components. • They proposed reusability estimations considering the	Integration architecture	S. Bhattacharya and D.E. Perry [21]

		integration architecture of the software. • They proposed Components characteristics and Architecture compliance metrics to measure the issues related to particular properties and integration issues of components.		
Object-Oriented Software	• Coupling and • Cohesion metrics	• Estimate the coupling properties among inter-components and cohesion properties of intra-components.	• Complexity of components	G. Gui and P.D. Scott [22]
Component-Based Software	• Coupling between objects, • Response for class • Coupling Factors	• Ranked components according to their reusability. • Defined metrics to estimate indirect coupling, degree of coupling, and functional complexity.	• Number of changes made to the code, • Time required to carry them	G. Gui and P.D. Scott [23]
Component-Based Software	• Interface complexity, • Interaction complexity, • Internal Logical files, • External Interface files, • Number of operations and interactions.	• Size measurement metric named Component point to estimate the size of the overall component-based system. • Three classes of components: User components, Service components, and Domain components.	• Size, • Weighting factor low, average and high • Unadjusted component point	Thareendhra Wijayasiriwardhane and Richard Lai [24]
Component-Based Software	• Reusability metrics for available code, • Interface properties	• White-box reusability metrics are based on the internal logic and coding structure of the software. • Black-box reusability metrics are based on interfaces and other attributes, as code is not available for black-box components.	• Internal code, • Interfaces, • Interaction, • Complexity	Danail Hristov, Oliver Hummel, Mahmudul Huq, Werner Janjic [25]

4. Conclusion

Effective and well-organized reuse of software deliverables, work-products and artefacts return the investment in terms of lower development cycle, effective cost, improved quality and increased productivity. Software reusability is such an approach that defines the effective reuse of pre-designed and tested parts of experienced software in new applications. In this paper, we have performed a review on various available reusability approaches for conventional, object-oriented and component based software. We considered some criteria on basis of which we examined the available approaches as measure and metrics used, key findings, and factors under consideration for reusability.

Some authors defined reusability as the inherent property of the code where as some has defined reusability in terms of black box and white box. Factors of reusability vary from code to documentation, complexity to adaptability, size to structure, cost to quality, and programming to language experience. Measures and metrics used by the available approaches also play a vital role in defining the property of reusability.

In this work, we have considered three categories of development paradigms:

a) Conventional software,

b) Object-oriented software, and

d) Component-based software.

These paradigms are selected as they support reusability in various forms, though the level of reusability is different in different paradigms. To assess the reusability, three factors are considered in this study:

a) Measures and metrics used by authors to propose their wok,

b) Key findings of their work,

c) Factors affecting the reusability.

References

[1] M. Gaedke and J. Rehse, "Supporting compositional reuse in component-based web engineering", In Proc. ACM symposium on Applied computing (SAC '00), ACM Press, New York, NY, USA, **(2000)**, pp. 927–933.

[2] B. W. Boehm, M. Pendo, A. Pyster, E. D. Stuckle and R. D. William, "An Environment for Improving Software Productivity", IEEE Computer, **(1984)**.

[3] C. W. Krueger, "Software Reuse", ACM Computing Surveys (CSUR), vol. 24, no. 2, **(1992)**, pp. 131 – 183.

[4] P. Freeman, "Reusable software engineering concepts and research directions", In Tutorial: Software Reusability, IEEE Computer Society Press, **(1987)**, pp. 10-23.

[5] V.R. Basili and H. D. Rombach, "Towards a comprehensive framework for reuse: A reuse-enabling software evolution environment", Technical Report CS-TR-2158, University of Maryland, **(1988)**.

[6] W. Tracz, "Confessions of a Used Program Salesman: Institutionalizing Software Reuse", Addison-Wesley, **(1995)**.

[7] C. L. Braun, "Reuse", In Marciniak, **(1994)**, pp. 1055-1069.

[8] W. C. Lim, "Effects of reuse on quality, productivity, and economics", IEEE Software, vol.11, no.5, **(1994)**.

[9] M. D. McIlroy, "Mass produced software components", In J.M. Buxton, P. Naur and B. Randell, "Software Engineering Concepts and Techniques", NATO Conference on Software Engineering, **(1976)**, pp. 88-98, 1968.

[10] J. Cooper, "Reuse-the business implications", In Marciniak, **(1994)**, pp. 1071-077.

[11] R. Prieto-Diaz and P. Freeman, "Classifying Software for Reusability", IEEE Software, vol.4, no.1, **(1987)**, pp.6-16, January.

[12] G. Caldiera and V.R. Basili, "Identifying and qualifying reusable software components", IEEE Computer, vol. 24, **(1991)**.

[13] J. S. Poulin, "Measuring Software Reusability", In Proc. Third International Conference on Software, Rio de Janerio, Brazil, **(1994)**, pp. 126-138.

[14] D.-J. Chen and P.J. Lee, "On the Study of Software Reuse Using Reusable C++ Components", Journal of Systems Software, vol.20, no.1, **(1993)**, pp. 19-36.

[15] W. G. Hislop, "Using Existing Software in a Software Reuse Initiative", The Sixth Annual Workshop on Software Reuse (WISR'93), Owego, New York, **(1993)**.

[16] J. Barnard, "A new reusability metric for object-oriented software", Software Quality Journal, vol. 7, **(1998)**, pp. 35-50.

[17] Y. Lee and K.H. Chang, “Reusability and maintainability metrics for object-oriented software”, In Proc. 38th annual on Southeast regional con-ference (ACM-SE 38), ACM, New York, NY, USA, **(2000)**, pp. 88-94.
[18] E.S. Cho, M.S. Kim and S.D. Kim, “Component Metrics to Measure Component Quality”, Proceedings of the Eighth Asia-Pacific on Software Engineering Conference (APSEC '01), IEEE Computer Society, Washington, DC, USA, **(2001)**, pp.419-426.
[19] M. Boxall and S. Araban, “Interface Metrics for Reusability Analysis of Components”, In Proc, Australian Software Engineering Conference (ASWEC'2004), Melbourne, Australia, **(2004)**, pp. 40-46.
[20] H. Washizaki, Y. Hirokazu and F. Yoshiaki, “A Metrics Suite for Measuring Reusability of Software Components”, In Proc. 9th International Symposium on Software Metric, **(2003)**, pp. 211-223.
[21] S. Bhattacharya and D.E. Perry, “Contextual reusability metrics for event-based architectures”, International Symposium on Empirical Software Engineering, pp.459-468, **(2005)**.
[22] G. Gui and P.D. Scott, “New Coupling and Cohesion Metrics for Evaluation of Software Component Reusability”, Proceedings of the International Conference for Young Computer Scientists, **(2008)**, pp. 1181-1186.
[23] G. Gui and P.D. Scott, “Ranking reusability of software components using coupling metrics”, The Journal of Systems and Software, Elsevier, vol. 80, **(2007)**, pp. 1450–1459.
[24] T. Wijayasiriwardhane and R. Lai, “Component Point: A system-level size measure for Component-Based Software Systems”, The Journal of Systems and Software, Elsevier, vol. 83, **(2010)**, pp. 2456–2470.
[25] D. Hristov, O. Hummel, M. Huq and W. Janjic, “Structuring Software Reusability Metrics for Component-Based Software Development”, In Proc. The Seventh International Conference on Software Engineering Advances ICSEA, IARIA, **(2012)**.