Managing Component-Based Systems With Reusable Components

Arun Sharma arunsharma@aiit.amity.edu
Amity Institute of Information Technology
Amity University, Noida, India.

Rajesh Kumar rajnagdev@yahoo.co.in
School of Maths and Computer Applications
Thapar University, Patiala, India.

P S Grover groverps@hotmail.com
Guru Tegh Bahadur Institute of Technology
GGSIP University, New Delhi. India.

Abstract

Component-Based Systems (CBS) have now become more generalized approach for application development. The main advantages of CBS are reduced development time, cost and efforts along with several others. These advantages are mainly contributed by the reuse of already built-in software components. In order to realize the reuse of components effectively in CBS, it is required to measure the reusability of components. However, due to the black-box nature of components where the source codes of these components are not available, it is difficult to use conventional metrics in Component-Based Development, as these metrics require analysis of source codes. The paper discusses the reusability concepts for Component based Systems and explores several existing metrics for both white-box and black box components to measure reusability directly or indirectly.

Keywords — Components, Customizability, Reusability, and Complexity.

1. INTRODUCTION

In spite of several strong features of Object-Oriented approach like objects, inheritance, reuse and others, this approach is not enough to cope with the rapidly changing requirements of present-day applications. Today's applications are large, complex and are not integrated. Although they come packaged with a wide range of features but most features can neither be removed, upgraded independently or replaced nor can be used in other applications. In particular, object-oriented methods do not typically lead to designs that make a clear separation between computational and compositional aspects [1].

Today Component Based Software Development (CBSD) is getting accepted in industry as a new effective development paradigm. It emphasizes the design & construction of software system using reusable components. CBSD is capable of reducing development costs and improving the reliability of an entire software system using components. The major advantages of CBSD are low cost, in-time and high quality solutions. Higher productivity, flexibility & quality through reusability, replaceability, efficient maintainability, and scalability are some additional benefits of CBSD. In a recent survey conducted on component based software development from 118 companies from around the world, it is found that around 53% of the organizations are using component-based approach in its development [2]. If there are a number of components available, it becomes necessary to devise some software metrics to qualify the various characteristics of components. Software metrics are intended to measure software quality characteristics quantitatively. Among several quality characteristics, the reusability is particularly important. It is necessary to measure the reusability of components in order to realize the reuse of components effectively.

2. REUSABILITY

In CBD, applications are built from existing components, primarily by assembling and replacing interoperable parts. Thus a single component can be reused in many applications, giving a faster development of applications with reduced cost and high quality. Also, as components are reused in various applications, they are likely to be more reliable than software developed ab initio. The reason is that these components are tested under varieties of situations before being used in the application(s) [3]. However, in spite of all such advantages of reuse, there are so many cases where reuse may lead to the software failure. These reasons may be due to the lack of experience for reuse, lack of documentation, tools and methodology for reuse along with several others. Therefore, it is very important to study all those concerns, which can improve the reusability so that the benefits of reuse can reach beyond the software development process.

Broadly, there are two types of component-based reuse: with no change to an existing component and with change. Reuse without change means simply selecting a component from a software component database, and dropping it into new software being developed. There are varieties of reasons that make reusing existing components without change a very difficult choice. The main reason may be due to the difference in functionality. Programming language may also be different, which needs a change in the component to be used in the target system. The difference in target environment, operating environment, industry standards etc. may also require some changes to be implemented in the component or in the system. On the other hand, the reuse with change to a component is also a very difficult task because it may take efforts to identify those parts of the components that require changes. Also, after changes, the modified components will have to be thoroughly tested before plugged into the system [4].

Another characterization of software reuse is the way it is implemented. First, known as white-box Reuse, when reuse is attempted, developers usually have the access to the code that can be modified to cater the new demands of the application. This provides a flexible way to harvest software assets in development projects by fitting existing components to new requirements, thus maximizing the reuse opportunities. But, with this advantage, this approach has one pitfall also. Code modification requires a high level familiarity with the implementation details. If this modification is to be done by assembler and not by the component developer, then it is very necessary to have the excellent documentation of the code. In contrast to this approach, another type of reuse, known as black-box reuse entails using software components "as is", does not allow to alter the component's internal logic and code. In this category, component must be flexible enough for better reusability. Black-box reuse based on component based development principles allows the component user to customize the artifacts using predefined parameters and switches [5].

Gill [6] discusses the importance of component characterization for better reusability. Component can be characterized on the basis of its informal description, external and internal specifications. Informal description consists of age, source, level of reuse, context, intent etc. External category defines its interactions with other application artifacts with the platform on which the component resides, which may consist of interoperability, portability, technology and other characteristics. Internal aspects of a component may include the nature of component like function, data package etc. Paper discusses several benefits of component characterization, which includes improved cataloguing, improved usage, improved retrieval and improved understanding eventually for better reuse.

An important issue in choosing the best component for reusability is deciding which components is more easily adapted. Generally, good guidelines for predicting reusability are: small size of code, simple structure and good documentation. Starting from the assumption that two functions have the same functionality, these three guidelines may be used in the system to rank candidate functions for reuse. Gill [7] discusses the various issues concerning component reusability and its benefits in terms of cost and timesavings. Paper also provides some guidelines to augment the level of software reusability in Component-Based development and emphasized on conducting thorough and detailed Software Reuse assessment to measure the potential for practicing reuse in an organization so that it can be ensured that the organization can get the maximum benefit from already practicing reuse. We should also perform Cost-Benefit Analysis to decide whether or not reuse is a worthwhile investment. This analysis can be performed by using well-established economic techniques like Net Present Value (NPV) and others. We should also adopt the standards, which are applicable for components and component-based systems.

3. REUSABILITY METRICS

Reusability can measure the degree of features that are reused in building new applications. There are a number of metrics available for measuring the reusability for Object-Oriented systems. These metrics focus on the object structure, which reflects on each individual entity such as methods and classes, and on the external attributes that measures the interaction among entities such as coupling & inheritance. However in CBD, the metrics are different than the conventional metrics. Components are termed as black box entities, for which size is not known. Also the measurement units and measurement factors are different in both the systems. Moreover, the performance and reliability of components also vary because only using the black box testing concepts can test these components and inherently biased vendor claims may be the only source of information [8]. These concerns can be overcome by using a separate set of metrics for CB systems, which keeps in mind the quality criteria to be measured, the methods to measure them along with their relative strength etc.

Devenbu et. al [9] considers the cost factor in assessing the benefits of reuse and proposes Reuse Benefit Metric Rb(S) of a system S, in terms of development cost. The direct measurement of the actual financial impact of reuse in system can be difficult. Therefore paper proposes an indirect measure and considers the source code of the system. To analytically evaluate the metric, paper proposes a set of desirable properties of reuse benefit measure and evaluated the metric in terms of its compliance with these properties. The proposed metric still needs an empirical evaluation and validation to be used in actual software organizations.

To measure the actual benefits of the reuse, one has to identify the various input parameters on which reuse can be based. Some of these parameters can be identified as the source code, cost and time involved in developing the code for reuse, relative cost of reuse and several others. Poulin [10] presents a set of metrics to estimate the efforts saved by reuse. The study suggests the potential benefits against the expenditures of time and resources required to identify and integrate reusable software into a product. Study assumes the cost as the set of data elements like Shipped Source Instructions (SSI), Changed Source Instructions (CSI), Reused source Instructions (RSI) etc.

Paper proposes several other reusability metrics in terms of cost and productivity like Reuse cost avoidance, Reuse value added and Additional development cost, which can be used significantly for business applications.

One more metric called Reuse Leverage for Productivity (RL) is proposed in the Literature, which is defined as

$$RL = \frac{\text{Productivity with Reuse}}{\text{Productivity without Reuse}} * 100$$

This notion of reuse leverage can only be measured after the introduction of reuse. It is desirable to be able to predict the effect of reuse, which can be obtained either to work initially on some pilot projects or to work with estimates or with industry benchmarks.

Dumke et. Al. [11] defines a metric set for reusability of Java Bean components. The proposed metrics are adapted from several contexts of Object Oriented design like percentage of public methods and others and structured programming like maximal McCabe Complexity number for a method in the Java Bean class. The metrics suite is based on white box concepts and relies on access to the source code, thus restricted the use only for the developers and not for he assemblers. One more indirect measure for reusability, which is also based on the access of source code is proposed in [12], which gives a complexity metric for software components by considering the internal constituents of a component like interfaces, methods and variable. The paper conducts an empirical evaluation of the metric on several Java Bean components and finally validates the proposed metric with another metric called Component Customizability. The result shows that higher complexity of a component needs higher maintenance and eventually will have a low reusability. The work proposes here considers the source code of the components while measuring the complexity and thus restricting the use of metrics only for component developers not with the application developer who is using the component.

Reusability can also be measured indirectly. Complexity, adaptability and observability can be considered as a good measure of reusability indirectly. Rotaru [13] et. al. measure the reusability by proposing some metrics for adaptability and interface complexity of the component. Adaptability refers to the accommodation of the changes required in its environment. Cho et al [14] propose a set of metrics for measuring various aspects of software components like complexity, customizability and reusability. The work considers two approaches to measure the reusability of a component. The first is a metric that measures how a component has reusability and may be used at design phase in a component development process. This metric, Component Reusability (CR) is calculated by dividing sum of interface methods providing commonality functions in a domain to the sum of total interface methods. The second approach is a metric called Component Reusability level (CRL) to measure particular component's reuse level per application in a component based software development. This metric is again divided into two sub-metrics. First is CRLLOC, which is measured by using lines of code, and is expressed as percentage as given as

$$CRL\ LOC(C) = (Reuse(C) / Size(C)) * 100\%$$

where:
Reuse(C): The lines of code reused component in an application,
Size(C): The total lines of code delivered in the application.

The second sub-metric is CRLFunc, which is measured by dividing functionality that a component supports into required functionality in an application. This metric gives an indication of higher reusability if a large number of functions used in a component. However, the proposed metrics are based on lines of codes and can only be used at design time for components.

Washizaki et al [15] propose a Component Reusability Model for black-box components from the viewpoint of component users. The proposed metrics suite considers understandability, adaptability and portability as relevant sub-characteristics of reusability. These metrics are:

Existence of Meta-Information (EMI) checks whether the BeanInfo class corresponding to the target component C is provided. The metric can be used by the users to understand the component's usage. Rate of Component's Observability (RCO) is a percentage of readable properties in all fields implemented within the Façade class of a component C. The metric indicates that high value of readability would help user to understand the behavior of a component from outside the component. Rate of Component's Customizability (RCC) is a percentage of writable properties in all fields implemented within Façade class of a component C. High value of the metric indicates the high level of customizability of component as per the user's requirement and thus leading to high adaptability. But if a component has too much writable property, it will loose the encapsulation and can be used wrongly. Self-completeness of Component's Return Value (SCCr) is the percentage of business methods without any return value in all business methods implemented within a component C, while Self-completeness of Component's Parameter (SCCp) is the percentage of business methods without any parameters in all business methods implemented within a component C. The business methods without return value/parameter will lead to self-completeness of a component and thus lead to high portability of the component. The paper also conducts an empirical evaluation of these metrics on various Java Bean components and set confidence intervals for these metrics. It also establishes a relationship among these proposed metrics. These metrics are applied on only for small Java Bean components and need to be validated for other component technologies like .NET, ActiveX and others also. The proposed metrics suite was validated with a case study where the reusability of over 120 components was assessed, both with this metric set and by a panel of experts. Results show a high correlation between both assessments, indicating that the metrics defined in this set can be used to assess the reusability of the component.

4. CONCLUSION

Building software systems with reusable components bring many advantages to Organizations. Reusability may have several direct or indirect factors like cost, efforts, and time. It may also have the issues like whether reusability is for the entire component or only for a selected service provided by that component. Paper discusses various aspects of reusability for Component-Based systems. It gives an insight view of various reusability metrics for Component-Based systems. Researchers for further study and empirical validation of these existing metrics can use the review done in the paper for CBS. Also, some new enhanced metrics can be proposed and empirically validated on the basis of the work already done by researchers in this area.

5. REFERENCES

[1] Jean-Guy Schneider: "Component Scripts and Glue: A Conceptual framework for software composition" Ph.D. thesis, Institute für Informatik (IAM), Universität Bern, Berne, Switzerland, 2003.

[2] Paul Allen, "CBD Survey: The State of the Practice", a white paper by Cutter Consortium. Web: http://www.cutter.com/research/2002/edge020305.html

[3] Arun Sharma, Rajesh Kumar, P S Grover, "Few Useful Considerations for Maintaining Software Components and Component-Based Systems", ACM SIGSOFT Software Engineering Notes, Vol. 32, Issue 4, September 2007 (to be published).

[4] Won Kim: "On Issues with Component-Based Software Reuse", in Journal of Object Technology, vol. 4, no. 7, September-October 2005, pp. 45-50.

[5] T. Ravichandran, Rothenberger M A, ”Software Reuse Strategies and Component Market”, Communications of the ACM, August 2003, Vol. 46, No. 8, pp. 109 –114.

[6] Nasib Singh Gill, Importance of Software Component Characterization For Better Software Reusability”, ACM SIGSOFT SEN Vol. 31 No. 1.

[7] Arun Sharma, Rajesh Kumar, P S Grover,” Critical Survey of Reusability Aspects for Software Components”, in the proceedings of International Conference on Computer, Information and Systems Engineering (2007), Bangkok, Thailand, pp: 419-424.

[8] Nasib Singh Gill, “Reusability Issues in Component-based Development”, ACM SIGSOFT SEN Vol. 28 No. 6, pp. 30-33.

[9] P Devenbu , S Karstu, W Melo, W Thomas, “Analytical evaluation of Software Reuse Metrics", Proceedings of the 18th International Conference on Software Engineering (ICSE’96), IEEE, pp 189-199

[10] J Poulin, J Caruso and D Hancock, “ The Business Case for Software Reuse, IBM Systems Journal, 32(40): 567-594, 1993.

[11] Dumke R., Schmietendorf A., “Possibilities of the Description and Evaluation of Software Components”, Metrics News 5(1) (2000).

[12] Arun Sharma, Rajesh Kumar, P S Grover, “Complexity Measures for Software Components” in “WSEAS Transactions on Computers”, Vol. 6, Issue 7, July 2007, pp: 1005-1012.

[13]Rotaru O P, Dobre M, “Reusability Metrics for Software Components”, in the Proceedings of the ACS/IEEE 2005 International Conference on Computer Systems and Applications, pp: 24-I

[14] Eun Sook Cho et al., “Component Metrics to Measure Component Quality”, Proceedings of the eighths Asia-Pacific Software Engineering Conference, 1530-1362/01.

[15] Hironori Washizaki, Hirokazu Yamamoto and Yoshiaki Fukazawa: "A Metrics Suite for Measuring Reusability of Software Components", **Proceedings of the 9th International Symposium on Software Metrics** September 2003.