World Academy of Science, Engineering and Technology
International Journal of Industrial and Manufacturing Engineering
Vol:1, No:9, 2007

A Critical Survey of Reusability Aspects for Component-Based Systems

Arun Sharma, Rajesh Kumar, and P. S. Grover

***Abstract*—**The last decade has shown that object-oriented concept by itself is not that powerful to cope with the rapidly changing requirements of ongoing applications. Component-based systems achieve flexibility by clearly separating the stable parts of systems (i.e. the components) from the specification of their composition. In order to realize the reuse of components effectively in CBSD, it is required to measure the reusability of components. However, due to the black-box nature of components where the source code 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. In this paper, we survey few existing component-based reusability metrics. These metrics give a border view of component's understandability, adaptability, and portability. It also describes the analysis, in terms of quality factors related to reusability, contained in an approach that aids significantly in assessing existing components for reusability.

***Keywords*—**Components, Customizability, Reusability, and Observability.

I. Introduction

THE last decade has shown that object-oriented technology alone is not enough to cope with the rapidly changing requirements of present-day applications. One of the reasons is that, although object-oriented methods encourage one to develop rich models that reflect the objects of the problem domain, this does not necessarily yield software architectures that can be easily adapted to changing requirements. Moreover, 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].

Arun Sharma is with Department of Information Technology, Amity University, Noida, India. He can be contacted at 91-120-4392277, 09899202168 (M) e-mail: arunsharma@aiit.amity.edu

Rajesh Kumar is with Thapar Institute of Engineering and Technology, Patiala, India. e-mail: rajnagdev@yahoo.co.in

P S Grover was Dean and Head, Computer Science Department, Delhi University, India. Now he is Director General Guru Teg Bahadur Engg. College, New Delhi, India. E-mail: groverps@hotmail.com

Any application must have some additional characteristics like robustness, usability, flexibility, simple installation, maintainability, proper documentation, portable, interoperable, extensible etc. to fight with the advancement in the technology and rapidly changing requirements. To improve the business performance it is necessary to use the latest technologies available.

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 in-time and high quality solutions. Higher productivity, flexibility & quality through reusability, replaceability, efficient maintainability, and scalability are some additional benefits of CBSD.

II. Component based Software Development (CBSD)

CBSE is a paradigm that aims at constructing and designing systems using a pre-defined set of software components explicitly created for reuse. According to Clements [2], CBSE embodies the "the 'buy, don't build' philosophy". He also says about CBSE that "in the same way that early subroutines liberated the programmer from thinking about details, CBSE shifts the emphasis from programming to composing software systems". In Object-Oriented Programming (OOP) code is reused in the form of objects, and several mechanisms such as inheritance and polymorphism let the developer reuse these objects in several ways. The principle is the same with CBSE, but the focus is on reusing whole software components, not objects.

Component-based systems achieve flexibility by clearly separating the stable parts of the system (i.e. the components) from the specification of their composition. Components are black-box entities that encapsulate services behind well-defined interfaces. These interfaces tend to be very restricted in nature, reflecting a particular model of plug-compatibility supported by a component-framework, rather than being very rich and reflecting real-world entities of the application domain. Components are not used in isolation, but according to a software architecture that determines the interfaces that

components may have and the rules governing their composition [7].

Component

There are a number of definitions given related to the component, some of these are:

- A *software component* is a reusable piece of code or software in binary form, which can be plugged into components from other vendors with relatively little efforts.
- A *software component* is a unit of composition with contractually specified interface and explicit context dependencies only. A software component can be deployed independently and is subjected to composition by third parts.[4]
- A *software component* is a language neutral, independently implemented package of software services, delivered in an encapsulated and replaceable container, accessed via one or more published interface. A component is not platform-constrained nor is it application-bound [5].
- A *software component* is a unit of packaging, distribution or delivery that provides services within a data integrity or encapsulation boundary. [Microsoft Corp]

In summary, a *software component* is a reusable, self-contained piece of software in binary form with well-specified interface that is independent of any application. The important aspect, which has to be kept in mind while developing a component, is the usability of component, regardless of whether or not an organization can identify what the future requirements of the component will be. Components can be placed on any network node, depending on application needs and regardless on the type of particular network structure [6].

There are several kinds of components and the *granularity* of these components can vary [7]:

A *distributed component* is a possibly network addressable component which has the lowest granularity. It may be implemented as an Enterprise JavaBean, as a CORBA component, or as a DCOM component.

A *business component* implements a single autonomous business concept. *A business component system* is a group of business components that co-operate to deliver a cohesive set of functionality and properties required in a specific domain.

A *tier* is a group of components in the same layer. The classic three-tier architecture consists of the presentation tier (windows, reports,…), application logic tier (business rules of the application) and resource tier (persistent storage mechanism).

Component-based software development (CBSD) is an approach in which systems are built from well-defined, independently produced pieces by combining the pieces with self-made components. Some definitions emphasize that components are conceptually coherent packages of useful behavior, while some others state that components are physical, deployable units of software, which execute within a well-defined environment. The key to the success of CBSD is its ability to use software components that are often developed by and purchased from third parties. [8]

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 when reusing components. It is necessary to measure the reusability of components in order to realize the reuse of components effectively.

III. Reusability

Software programming is a hard design task, mainly due to the complexity involved in the process. Nowadays this complexity is increasing to levels in which reuse of previous software designs are very useful to short cut the development time. The main idea of software reuse is to use previous software components to create new software programs. Thus software reuse is software design, where previous components are the building blocks for the generation of new systems. In case of Component-based Development, software reuse refers to the utilization of a software component C within a product P, where the original motivation for constructing C was other than for use in P. In other words, *reuse* is the process of adapting a generalized component to various contexts of use. The idea of reusing software embodies several advantages. It improves productivity, maintainability, portability and quality of software systems. A reusable component can be seen as a box, which contains the code and the documentation [11]. These boxes are defines as:

A. Black Box Reuse

In ***black box*** *reuse, t*he reuser sees the interface, not the implementation of the component. The interface contains public methods, user documentation, requirements and restrictions of the component. 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. The clients will get what the contract promises only if the post condition is true after the changes to the internal implementation [9].

B. Glass Box Reuse

In ***glass box reuse*** the inside of the box can be seen as well as the outside, but it is not possible to touch the inside. This solution has an advantage when compared to black box reuse, as the reuser can understand the box and its use better. The disadvantage is that it is possible that the reuser will rely on a particular way of implementation or other factors that are not

in the contract. That can be hazardous when the implementation changes.

C. White Box Reuse

In ***white box reuse*** it is possible to see and change the inside of the box as well as its interface. A white box can share its internal structure or implementation with another box through inheritance or delegation. The new box can retain the reused box as such or it can change it. It is necessary to test anything new that is created or changed.

One of the essential problems in software reuse is the retrieval and selection of suitable software components from a large library of components. Gill [10] discusses the importance of component characterization for better reusability. It discusses several benefits of component characterization, which includes improved cataloguing, improved usage, improved retrieval and improved understanding eventually for better reuse.

IV. Metrics

The area of software measurement is one of the areas in software engineering where researchers are active from a long time. The area of software measurement is also known as software metrics. According to IEEE [IEEE 93],

"A software metric is a quantitative measure of the degree to which a system, component or process possess a given attribute".

Software metrics are intended to measure the software quality and performance characteristics quantitatively encountered during the planning and execution of software development resource and effort allocation, scheduling and product evaluation. These can serve as measures of software products for the purpose of comparison, cost estimation, fault prediction and forecasting.

Reusability Metrics

Reusability can measure the degree of features that are reused in building applications. There is 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. But there are some difficulties in applying existing object oriented metrics into the component development and CBSD. Object oriented metrics cannot be used to measure the component's quality. The reasons are:

i. Measurement unit is different. Object oriented metrics only focus on objects or classes. Component consists of one or more classes as well as one or more interfaces. Existing object-oriented metrics do not consider component itself or component's interfaces on measuring complexity, cohesion or coupling and so on. Therefore it is required new metrics that measure complexity of component itself.
ii. Measurement factor is insufficient. Because object oriented applications are developed with only classes, most of the object-oriented metrics measure the complexity or reusability by considering classes, methods and depth of class hierarchy. However, considering these factors are not sufficient to measure the complexity or reusability of component because components have more much information such as interfaces, interface methods and so on.
iii. Existing object oriented metrics do not consider customizability of classes or objects. Customizability of components is very important in CBSD because component's customizability effects on reusability of components in CBSD. Most of the traditional metrics are based on source code (LOC) or similar size counts, defects counts and effort figures. For Object Oriented Development also, reuse is assumed to be a very effective strategy to build high-quality software. All these approaches use the size factor to measure the empirical values of the quality attributes of the software.

However in CBD, the metrics are different than the conventional metrics. Components are termed as black box entities, for which size is not known so alternative measures have to be used to measure the quality of the software.

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. These concerns can be by 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.

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 are used in our system to rank candidate functions for reuse. Gill [11] discusses the various issues concerning component reusability and its benefits in terms of cost and time- savings. Papers also provide some guidelines to augment the level of software reusability in Component-Based development, which are summarized below:

i. 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.

ii. Performing 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.

iii. Adoption of standards for components to facilitate a better and faster understanding of a component and a faster integration into a system.

iv. Selecting pilot projects for wider development of reuse

v. Identifying reuse metrics.

Poulin [12] presents a set of metrics used by IBM 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.

Reuse Percentage measures how much of the product can be attributed to reuse and is given as

Product Reuse Percentage = (RSI / (RSI + SSI)) * 100%

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.

Cho et al [13] proposes 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 CRL_{LOC}, which is measured by using lines of code, and is expressed as percentage as given as

$$CRL_{LOC}(C) = (Reuse(C) / Size(C)) * 100\%$$

The second sub-metric is CRL_{Func}, 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 [14] discusses the importance of reusability of components in order to realize the reuse of components effectively and propose a Component Reusability Model for black-box components from the viewpoint of component users. The model defines a set of metrics to define quality factors that affect reusability. These metrics are:

i. 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 user to understand the component's usage.

ii. 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.

iii. 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 properties, it will loose the encapsulation and can be used wrongly.

iv. 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 JavaBean 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 JavaBean components and need to be validated for other component technologies like .NET, ActiveX and others also.

V. Conclusion

In this paper we survey different aspects of reusability for component-based systems. The paper gives an insight view of various reusability metrics for component-based systems. The work proposed here can be used by researchers for further study and empirical validation of these existing metrics 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.

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] Michael Sparling: "Lessons Learned through Six Years of Component Based Development" published in Castek (as published in "Communications of the ACM") date:04-09-03.
[3] Szyperski C: Component Software: beyond Object Oriented Programming, New York: ACM Press/ Addison Wesley 1998.
[4] 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.
[5] Miguel Goulão: "CBSE: a Quantitative Approach" PhD Workshop at ECOOP'2003, Darmstadt, Germany. July, 2003.
[6] Nasib S. Gill and P. S. Grover: "Necessary Guidelines for deriving Component Based Metrics". In ACM SIGPLAN SEN Vol 28, #6 Page:30, 2003.
[7] Pentti Virtanen "Measuring and Improving Component-Based Software Development by Pentti Virtanen "University of Turku, Department of Computer Science, FIN-20014 Turku Finland 2003.
[8] Arun Sharma, Rajesh Kumar, P S Grover, "Investigation of reusability, complexity and customizability for component-based systems", ICFAI Journal of IT, Vol.2 Iss. 1, June 2006.
[9] Goldberg A., Rubin K. S.: *Succeeding with Objects, Decision Frameworks for Project Management*, Addison-Wesley Publishing, 1995.
[10] Nasib Singh Gill, Importance of Software Component Characterization For Better Software Reusability", ACM SIGSOFT SEN Vol. 31 No. 1.
[11] Nasib Singh Gill, "Reusability Issues in Component-based Development", ACM SIGSOFT SEN Vol. 28 No. 6, pp. 30.
[12] J. Poulin, J Caruso and D Hancock, " The Business Case for Software Reuse, IBM Systems Journal, 32(40): 567-594, 1993.
[13] Eun Sook Cho et al., "Component Metrics to Measure Component Quality", Proceedings of the eighths Asia-Pacific Software Engineering Conference, 1530-1362/01.
[14] Hironori Washizaki, Hirokazu Yamamoto and Yoshiaki Fukazawa, "Software Component Metrics and It's Experimental Evaluation," Proc. of the International Symposium on Empirical Software Engineering (ISESE 2002), October 2002.