Anatomizing Component based Software Reusability

O.Rajalakshmi@Karthika[1], C.Rekha[2]

[1]*Research Scholar, Madurai Kamaraj University, Madurai, India*
[2]*Department of Computer Science, Government Arts College, Melur, India*

Abstract

Over the last decade, the extensive uses of software have placed new demands and expectations on the software industry especially for improving quality and enhancing development productivity. In order to meet these challenges, software development must be able to cope with complexity and to adapt quickly to changes as the result of the increases in demand for the integration of different areas. Component-based development (CBD) is a procedure that accentuates the design and development of computer-based systems with the help of reusable software components. CBD techniques involve procedures for developing software systems by choosing ideal off-the-shelf components and then assembling them using well-defined software architecture. This paper focus on the overview of the Component based reusability and its characteristics.

Keywords: Component based development, Software reusability, Componentreusability.

I. Introduction

Reusability is the basic concept of software engineering .Software reuse has been a lofty goal for Software Engineering (SE) research and practice, as a means to reduced development costs, time, improved quality and component based development[1]. Reusability is about building a library of frequently used components, thus allowing new programs to be assembled quickly from existing components. 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. Software engineers have been of the view that software does not have to be developed from scratch all the times and have thought about assembling previously existing components into large software systems. Software engineers have reused abstractions and processes since the early days of computing, but the early approaches to reuse were ad hoc. As there is an ever-growing need for techniques that could improve the software development process, reduce the time to market and improve the quality of delivered software products, a more organized and focus approach to reuse called component-based software engineering (CBSE) has emerged [2].

Object-oriented modeling results in a plethora of fine-grained classes, objects and relationships. It is very hard to discover reusable parts among these smaller units. The idea behind CBD is to integrate the related parts and reuse them collectively. These integrated parts are known as components [3].

This paper organizes into x sections. Second section start with detailed literature review and the III chapter discuss about the basics of reusability and give the lead to the IV chapter. Section V highlights the metrics of reusability and its principles. The core concepts are discussed in section VI and give the conclusion.

II. Literature Review

McIlroy [4] first envisioned Software reuse, at a NATO Software Engineering Conference, where he predicted that mass-produced components would end the software crisis [4]. The final objective was very clear: to make something once and to reuse it several times. Frakes and Terry [5] – was first person to propose metric and models on software reuse. He suggested models based on cost benefits, assessing the maturity level, reuse library metrics [5]. Kim [6] takes – on the issue of component based software reuse. It discusses the difficulties in realizing the component based software reuse and to discuss the pre – conditions required to meet before practicing software reuse on a wide scale in a formalized manner [6]. Sharma et. al [7] discusses about managing component – based systems with reusable components. It discusses the reusability concepts for components based systems and to explore the reusability metrics to measure reusability directly or indirectly [7].

Anas al – badareen [8]proposed a framework that contains the extraction, adoption and storage of reusable software components [8]. Gupta and Kumar [9] conducted a study about reusable software component retrieval system. The study discusses the techniques for storage and retrieval of software components which can be reused [10].Crnkovic et al. [9] Looked into different software component-based models which were developed by employing diverse themes, objectives, principles and technologies. All the models produced similar results but the principles they followed were different. While some models do not explain the concept clearly; yet they help identify, characterize and provide easy to understand concept of component -based model framework.

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 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:

- Black Box Reuse
- White Box Reuse
- Glass Box Reuse

Black Box Reuse

In black box reuse, the reuser sees the interface, not the implementation of the component. The interface contains public methods, user documentation, requirements and restrictions of the component. As the users of the component trust its interface, changes should not affect the logical behavior of the component.

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.

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.

IV. Reusability Matrices and Model

Software reuse, the use of existing software artifacts or knowledge to create new software is a key method for significantly improving software quality and productivity. Reusability is the degree to which a thing can be reused. Software reuse reduces the amount of software that needs to be produced from scratch and thus allows a greater focus on quality. The reuse of well tested software should result in greater reliability and less testing time for new software. With reuse, software development becomes a capital investment. In this paper we survey metrics and models of software reuse and reusability. A metric is a quantitative indicator of an attribute of a thing. A model specifies relationships among metrics. In reuse models and metrics are categorized into types:

- Reuse cost benefits
- Maturity assessment
- Amount of reuse
- Reuse library

Cost benefit analysis models include economic cost-benefit models and quality and productivity payoff analyses. These are estimated by setting arbitrary values for cost and productivity measures of systems without reuse, and then estimating these parameters for systems with reuse and cost benefit analysis consist of several types of models. As shown by fig 1.

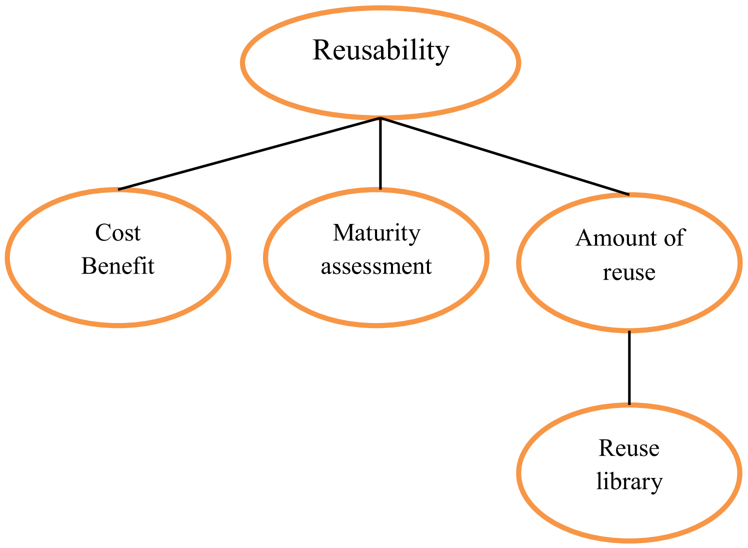

Fig 1. Overview of reusability structure

Maturity model is at the core of planned reuse, helping organizations understand their past, current, and future goals for reuse activities. Several reuse maturity models have been developed and used, though they have not been validated.

Amount of reuse metrics are used to assess and monitor a reuse improvement effort by tracking percentages of reuse of life cycle objects over time.

Reusability Assessment includes the concept of tools of reusability as black box, white box and glass box. And several modules are

- Fewer module calls per source line
- Fewer I/O parameters per source line
- Fewer read/write statements per line
- Higher comment to code ratios
- More utility function calls per source line
- Fewer source lines

Reuse library: Library assets can be obtained from existing systems through reengineering, designed and built from scratch, or purchased. Library efficiency deals with nonfunctional requirements such as memory usage, indexing file size, and retrieval speed.

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:

- 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.
- 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.
- Existing object oriented metrics do not consider customizability of classes or objects.

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.

V. COMPONENT BASED SOFTWARE ENGINEERING

In component based software engineering approach, the emphasis on components is more. A component can be treated or defined as an independent system that accomplishes a specific task. A component also can be composed of several other components. More specifically software components are prebuilt items that act as a building block in a software to perform specific functions. These components can communicate with each other using standard interface. After the strengthening of component based software engineering practices, the idea of software reuse has evolved more significantly. In fact component based software engineering is a process that emphasized the designing and construction of software using reusable software components [11].

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

- Software component
- Distributed component
- Business component
- Group of component
- Based on software development component

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 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. It is not platform constrained nor is it application-bound. It t 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.

A distributed component is a possibly network addressable component which has the lowest granularity. It may be implemented as an Enterprise JavaBeans, 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.
Reusability shifts focus from programming software to composing software system [11]. There lies many advantages of component reuse like reduced development time and cost, the quality and productivity will also be increased as the pre tested components are being used. But the developers could not yield much benefit from it. There may be several reasons for that but the lack of proper definition for the components which is to be reused can be the one reason. The non-availability of an established definition for a component can also create confusion and misconception among the developers. It can be explained with an example: you purchase a stereo system and bring it home. We can fit many components like speakers, earphones etc. into it. Each of these components has been designed to fit a specific architectural style, theconnections between components are standardized, and communication protocols have been pre-established. Assembly of such components is easy than to build a system from hundreds of discrete parts. This is what we need to achieve [11]. A component must be designed in such a way that it should be able to

- ➢ Integrate and communicate with other components in the same system easily.
- ➢ Integrate and communicate with other components outside the system but within same domain.
- ➢ Also it should integrate and communicate with applications outside the domain.

VI. CONCLUSION

Existing software or a software component can be reused to produce a new system without actually producing it. It can be a composition rather than the production. This paper highlights the issues which are primarily need to be addressed for the start of the re-use process. It also exhibits the importance of organizational issues which may not be given the due attention. A set of standards for components to be reused have been outlined, it will be beneficial in such a way that production, selection, adaptation, integration of components will become easier. Reduction in costs and time – to – market has always been an issue for the software industry. Industries desperately need a shift to software reuse. Thus software reuse will bring the improvements in productivity, quality and reliability of the software.

REFERENCES

[1] Arun Sharma, Rajesh Kumar, and P. S. Grover, "A Critical Survey of Reusability Aspects for Component-Based Systems", International Journal of Industrial and Manufacturing Engineering, Vol:1, No:9, 2007

[2] Manivasagama, G & Kumar, Dr. M. Anand&Balasubramanian, Bharathi. (2017). A Component-Based Model for Software Reusability.International Journal of Control Theory and Applications. 10. 113.

[3] Keswani, Raman & Joshi, Salil&Jatain, Aman. (2014). Software Reuse in Practice.159-162. 10.1109/ACCT.2014.57.

[4] Imeri F.; Antovski L., "An Analytical View on the Software Reuse", ICT Innovations 2012, Web Proceedings of the 4th ICT – ACT Conference, Ohrid - Macedonia, ISSN: 1857 – 7288, pp. 213 – 222, 2012.

[5] McIlroy M.D., "Mass Produced Software Components", Software Engineering: Report on a Conference by the NATO Science Committee, Brussels, pp. 138 – 155, 1968.

[6] Kim W., "On Issues with Component – Based Software Reuse", Journal of Object Technology, 4[7], pp. 45 – 50, 2005.

[7] Sharma A., Kumar R., Grover P., "Managing Component – Based Systems with Reusable Components", International Journal of Computer Science and Security, 1[2], pp. 60 – 65, 2007.

[8] Al – Badareen A., Selamat M.H., Jabar M.A., "Reusable Software Component Lifecycle", International Journal of Computers, 5(2), pp. 191 – 199, 2011.

[9] I. Crnkovic, S. Sentilles, A. Vulgarakis and M. R. V. Chaudron, "A classification framework for Component models", IEEE (2011).

[10] Gupta S., Kumar A., "Reusable Software Component Retrieval System", International Journal of Application or Innovation in Engineering and Management, 2[1], pp. 187 – 194, 2013.

[11] Pressman R.S., Software Engineering – A Practitioner's Approach, 8th ed. New York: McGraw-Hill Inc., 2010.