Software Engineering - Survey of Reusability Based on Software Component

Sarbjeet Singh, Manjit Thapa, Sukhvinder singh and Gurpreet Singh
Department of Computer Science, Sri Sai College of Engg. & Tech. Badhani (Pathankot).

ABSTRACT

Survey of reusability based on software components that provide the assistance to the developer in the development of software. Reusability of software is an important prerequisite for cost and time-optimized software development Work in software reuse focuses on reusing artifacts. 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 and presents the special requirements on software in this domain and Reusability is about building a library of frequently used components, thus allowing new programs to be assembled quickly from existing components. Component-Based Systems (CBS) have now become more generalized approach for application development.

Keyword: Tools of reusability, Components of reuse, Reusability matrices.

1. 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. 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. It is an important area of software engineering research that promises significant improvements in software productivity and quality. The major advantages of CBSD are low cost, in-time and high quality solutions. Higher productivity, flexibility & quality through reusability, replace ability, efficient maintainability, and scalability are some additional benefits of CBSD. If there are a number of components available, it becomes necessary to devise some software metrics to qualify the various characteristics of components. It is necessary to measure the reusability of components in order to realize the reuse of components effectively. Reusability can also be measured indirectly. Complexity, adaptability and observability can be considered as a good measure of reusability indirectly.

The ability to reuse relies in an essential way on the ability to build larger things from smaller parts, and being able to identify commonalities among those parts[1,2,3]. Reusability is often a required characteristic of platform software. and implies some explicit management of build, packaging, distribution, installation, configuration, deployment, maintenance and upgrade issues. Reusability brings several aspects to software development that do not need to be considered when reusability is not required.

2. TOOLS OF REUSABILITY

Software programming is a hard design task, mainly due to the complexity involved in the process. Reuse deals with the ability to combine independent software components to form a

Larger unit of software.[4,5,6] To incorporate reusable components into a software system, programmers must be able to find and understand them. Thus Software reuse is software design, where previous components are the building blocks for the generation of new systems. These are the three or four specific tools by Reusability.

And shown by figure1.

- White Box Reusability
- Black Box Reusability
- Glass Box Reusability

Reusability tools are based upon software testing development. Software reuse can apply to any life cycle product, not only to fragments of source code[7,8].

In White-box reusability is verification technique software engineers can use to examine if their code works as expected and a box can share its internal structure or implementation with another box through inheritance or delegation.

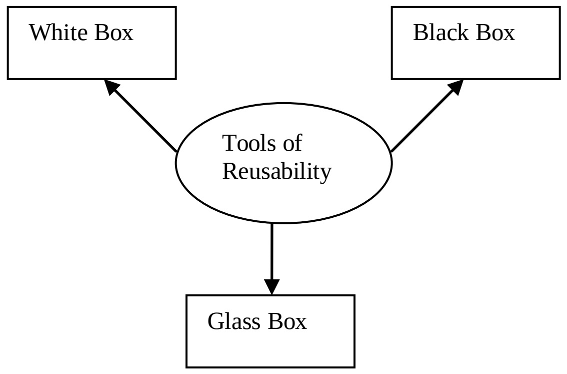

Figure1. Shown in tools of reuse.

One of the most common complaints of designers or print service providers when previewing and printing transparency from In Design is that a transparency effect like a drop shadow doesn't display or print correctly. Instead, a white box appears behind the transparency effect [9.10].

It discusses several benefits of component characterization, which includes improved cataloguing, improved usage, improved retrieval and improved understanding eventually for better reuse [11.12].

In Black box reusability, the reuse sees the interface, not the implementation of the component.[13,14,15] 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.

In **Glass box reusability** the inside of the box can be seen as well as the outside, but it is not possible to touch the inside to obtain the digital displays. [16,17]

• A good knowledge of PowerPoint (or other slideshow program) is necessary.

• Facility with Adobe Photoshop (or similar professional imaging software) can certainly help speed the process and enhance the content.

• A camera, scanner, screenshot software and a basic knowledge Of photography and digital
imagings are requisite[18.19].

• Some artistic talent helps.

• And keep in mind that the software, ppt files, and images demand a lot from a computer.

The computer I work on to create the display is new and has plenty Of oomph.

3. COMPONENTS OF REUSABILITY

Component Reusability is about building a library of frequently used components, thus allowing new programs to be assembled quickly from existing components.[20,21] Component Reusability has produced greater schedule and effort savings than any other practice. We have applied this concept not only to the code, but also to the design, data, documentation, test materials, specifications, and plan[22,23].

We have created a repository named 'Component Repository' to store the components that are identified as commonly used components. Currently this repository stores more than 100 reusable components. The repository has a search feature to look for similar components. The repository is managed by the Reuse Group, which has members from the design as well as development team. And shown by figure 2. 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

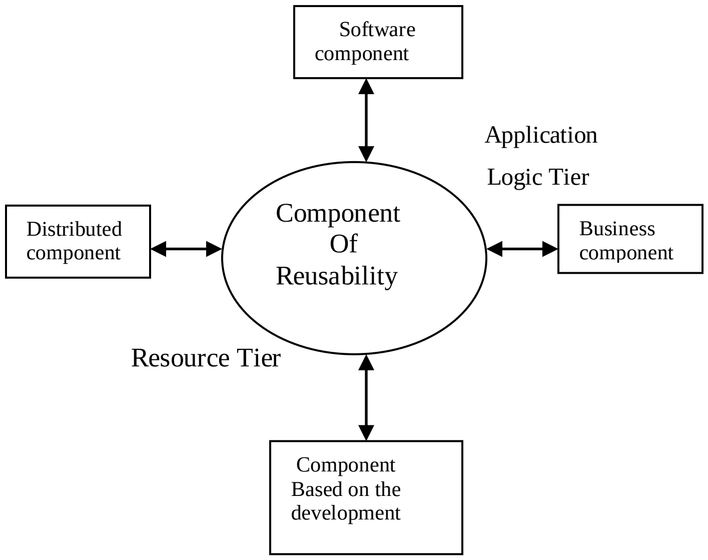

Figure 2.Shown as a tier on group of component.

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.

A **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. 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. 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. 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.

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).

4. 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
- Reusability
- 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 figure3.

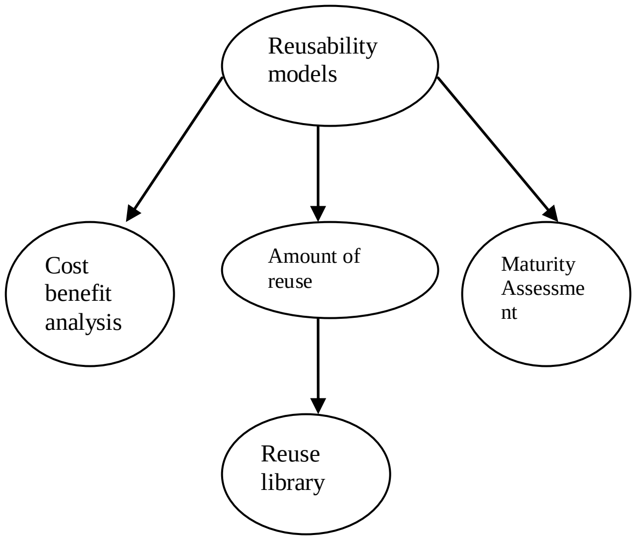

Figure 3.Shown by knowledge base upon Reusability models.

- Productivity model
- Quality of Investment
- Business Reuse Metrics

Cost and productivity models for software reuse.. The cost-of-development model builds upon the simple model by representing the cost of developing reusable components. Let E represent the cost of developing a reusable component relative to the cost of producing a component that is not reusable. E is expected to be .1 because creating a component for reuse generally requires extra effort. Let n be the number of uses over which the code development cost will be amortized. The new value for C (cost) incorporates these measures:

C = ~b 1 ~E/n! 2 1! R 1 1.

Quality of instrument based upon activities. Reuse activities are divided into producer activities and consumer activities. Producer activities are reuse investments, or costs incurred while making one or more work products easier to reuse by others. Consumer activities are reuse benefits or measures in dollars of how much the earlier reuse investment helped. Quality of an instrument based upon ratio as

$$Q = B/R$$

Q is less than B and R.

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. In general, the metric is given by

$$\frac{\text{Lines of reused code in system or module}}{\text{Total line of code system}}$$

These matrices are based upon the external lower level behavior and internal lower level behavior.

External lower level behavior = E/R

Internal lower level behavior = L/R

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

A coupling is based on references to variables and parameters (data bindings). Aliasing, or referencing, is not taken into account; only one level of data bindings is considered.

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.

5. CONCLUSION

In this paper we survey different aspects of reusability for component-based, metrics and models of software reuse. A metric is a quantitative indicator of an attribute. A model specifies relationships between metrics. The work proposed here can be used by researchers for further study and empirical validation of these existing metrics for Component based upon the system.

6. REFERENCES

[1] J.J. Bunn. Floppy and flow user manual. 1997.

[2] B.W. Goodwin, T.H. Andres, D.C. Donahue, W.C. Hajas, S.B. Keeling, C.I. Kitson, D.M. LeNeveu, T.W. Melnyk, S.E. Oliver, J.G. Szekely, A.G. Wikjord, K. Witzke, and L. Wojciechowski. Radiological assessment. Technical ReportAECL-11494-5,COG-95-552-5, Atomic Energyof Canada Ltd, 1996.

[3] B.W. Goodwin, D.B. McConnell, T.H. Andres,waste: Postclosure assessment of a referencesystem. Technical Report AECL-10717, COG- 93-7, Atomic Energy of Canada Ltd, 1994.

[4] D.E. Knuth. Literate Programming. Center for the Study of Language and Information, 1992.

[5] D.M. LeNeveu. Analysis specifications for thecc3 vault model. Technical Report AECL- 10970,COG-94-100, Atomic Energy of Canada Ltd, 1994.

[6] Quality assurance of analytical, scientific, and design computer programs for nuclear power Plants. Technical Report N286.7-99, Canadian Standards Association, 178 Rexdale Blvd. Etobicoke, Ontario, Canada M9W 1R3, 1999.

[7] S. Oliver, K. Dougan, K. Kersch, C. Kitson, G. Sherman and L. Wojciechowski. Unit testing- a component of verification of scientific modeling software. In T.I. Oren and G.B. Birta, editors, 1995 Summer Computer Simulation Conference, pages 978–983. The SocietyFor Computer Simulation, 1995.

[8] N. Ramsey. Literate programming simplified. IEEE Software, September 1994.

[9] K. Rose. Very high level 2-dimensional graphics. In 1997 Tex User Group Conference. Textures Group, 1997.

[10] L. Wall, T. Christiansen, and R. Schwartz. Programming Perl. O'Reilly & Associates, 101 Morris Street, Sebastopol, CA 95472, second Edition, 1989.

[11] E. Yourdon. Modern Structured Analysis. Yourdon Press.

[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 Washita, Hirokazu Yamamoto and Yoshiaki Fukazawa," Software Component Metrics and its Experimental Evaluation," Proc. of the International Symposium on Empirical Software Engineering (ISESE 2002), October 2002. World Academy of Science, Engineering and Technology 33 200739.

[15] Gamma E., Helm R., Johnson R., Vlissides J.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesely, Professional Computing Series, Reading, Massachusetts, 1994.

[26] Meyer B.: *Object-Oriented Software Construction*, Second Edition, Prentice Hall PTR, New Jersey, 1997

[17] Johnson R.: *Documenting Frameworks Using Patterns*, Object-Oriented Programming Systems, Languages, and Applications conference pro-ceedings, pp. 63-76, Vancouver, British Columbia, Canada, ACM Press, October 1992.

[18] Pirklbauer K., Plösch R., Weinreich R.: *Object-Oriented and Conventional Process Automation Systems*, Proceedings of 39th International Scientific Colloqium at TU Ilmenau,Germany, September 27-30, 1994, pp. 566-571, Bd. 3, ISSN 0943-7207.

[5] Pomberger G., Blaschek G.: *Software Engineering*, Carl Hanser Verlag, 1996

[19] Pree W.: *Design Patterns for Object-Oriented Software Development*, Addison-Wesely, 1995

[20] REFORM: *A Reusable Framework for Rolling Mills*, online at http://www.ssw.uni-linz.ac.at/REFORM/home.html, accessed May 1998.

[21] Siegel J.: *CORBA Fundamentals and Programming*, John Wiley & Sons, Inc. 1996

[22] Stroustroup B.: *The C++ Programming Language*, Third Edition, Addison-Wesley 1997

[23] Taligent: *Building Object-Oriented Frameworks*, online http://www.ibm.com/java/education/oobuilding/index.html, accessed September 1998