Software Metrics to Estimate Software Quality using Software Component Reusability

Prakriti Trivedi[1], Rajeev Kumar[2]

[1] Department of Computer Engineering, Government Engineering College, Ajmer, Rajasthan, India

[2] Department of Computer Engineering, Government Engineering College, Ajmer, Rajasthan, India

Abstract

Today most of the applications developed using some existing libraries, codes, open sources etc. As a code is accessed in a program, it is represented as the software component. Such as in java beans and in .net ActiveX controls are the software components. These components are ready to use programming code or controls that excel the code development. A component based software system defines the concept of software reusability. While using these components the main question arise is whether to use such components is beneficial or not. In this proposed work we are trying to present the answer for the same question. In this work we are presenting a set of software matrix that will check the interconnection between the software component and the application. How strong this relation defines the software quality after using this software component. The overall metrics will return the final result in terms of the boundless of the component with application.

Keywords: *Components, Reusability, Quality, Estimation, Metrics*

1. Introduction

Software engineering deals with the development of software systems. As the size of a software system increased, new approaches of software development come to the environment. These approaches include the object oriented programming, component based programming, aspect based programming etc. These approaches provide a better view to present and develop a software system. These approaches are very much inspired from the real world and provide a systematic and rapid development approach. In this proposed work we are basically dealing with the concept of reusability that exists in all kind of approaches. Here the object oriented and component based approaches are considered as the main concept of software reusability.

The code reusability means to use the existing code or the component in a software system. This existing code can exist in the form of some library, plug-in or software other software code developed by the user. There also exist different approaches to perform the reusability. Such as Coupling, composition etc. Each kind of interfacing has its own advantage as well as cost. In this proposed we are also presenting these approaches also as the cost estimator.

To understand the concept of component based reusability, we need to understand the concept of components, its type and accessibility. A software system is the collection of different software modules or the components that are integrated as the whole system. With the inclusion of the software components the complete life cycle of the software is changed. Now it needs to test and estimate each software components individually, and if these components are already in running mode in some other application we need to just perform the interfacing of the current application with these software components. Now there is need to test and estimate the interfacing between code and the software components. Here we are presenting more clear view of software components and the related reusability factor.

1.1 Software Components

Some basic properties of the software components are:

(i) A software component can be a code block, module, function, class, control or the project or software itself.

(ii) The software component can be language dependent or language independent.

(iii) A software component can be end product or it can be extendable.

(iv) A software component is the unit of interfacing that conceptually specifies it's internal and the external interfacing with main application.

(v) A software component can also be a deliverable software object.

(vi) A software component can be online or the offline product or code.

As we can see a software component is not an individual term it is the basic concept that gives the software reusability in some way. Any kind of internal or interfacing in software in the form of individual component is represented in the form of software components.

Each of the software language defines most of software components in different way.

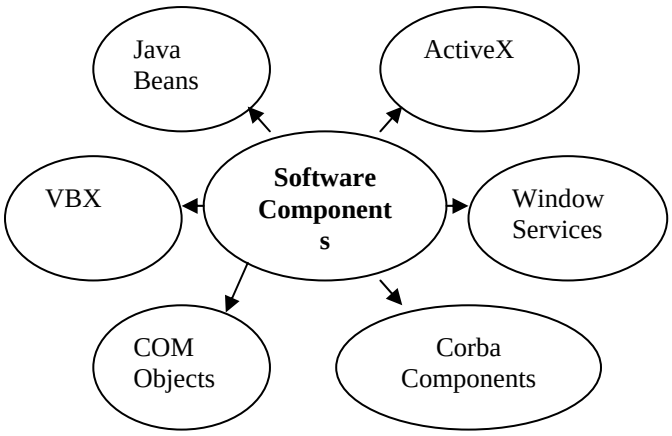


Fig. 1 Software Components.

1.2 Component Based Software Engineering

As the most of business software uses the concept component based development approach, because of this it require more scientific approach to estimate the software quality and the complexity. There was the requirement of some improvement in the software development process as well as in analysis of software system. It is one of the reason that the developers think in a new direction to estimate the software quality. There was requirement of such an approach that was structured and the rule based. Such an approach should be compatible to most of the available software and the software development processes. This gives the development of a new concept called CBSD i.e. component based software development [1].

Component-based software development (CBSD) is an approach in which systems are built from well -defined, independently produced pieces, known as 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 are executed within a well defined environment [1].

1.3 Software Metrics

As the number of components available on the market increases, it is becoming more important to devise software metrics to quantify the various characteristics of components and their usage. Software metrics are intended to measure the software quality and performance characteristics quantitatively, encountered during the planning and execution of software development. These can serve as measures of software products for the purpose of comparison, cost estimation, fault prediction and forecasting. Metrics can also be used in guiding decisions throughout the life cycle, determining whether software quality improvement initiatives are financially worthwhile (Sedigh *et al.,* 2001). A lot of research has been conducted on software metrics and their applications. Most of the metrics proposed in literature are based on the source code of the application. However, these metrics cannot be applied on components and component-based systems as the source code of the components is not available to application developers. Therefore, a different set of metrics is required to measure various aspects for component-based systems and their quality issues [2].

1.4 Software Reusability

Software reuse is the process of implementing or updating software systems using existing software components. A good software reuse process facilitates the increase of productivity, quality, and reliability, and the decrease of costs and implementation time. An initial investment is required to start a software reuse process, but that investment pays for itself in a few reuses [3].
In short, the development of a reuse process and repository produces a base of knowledge that improves in quality after every reuse, minimizing the amount of development work required for future projects, and ultimately reducing the risk of new projects that are based on repository knowledge [4].

Why Reuse?

Reuse has been proven to offer many rewards. When we reuse code, components and other artifacts, our goals are to [5]:

- Reduce time to market.
- Reduce the cost of developing the product.
- Improve the productivity of the development teams.
- Improve the predictability of the development process.
- Increase the quality and reliability of the product.

- When reuse is mentioned, we often think only of code reuse.

This is perhaps one of the lesser productive forms of reuse. That is not said it is of little or no value, just that there are rewards to be gained in considering inheritance, template, component, framework, pattern and domain component reuse. Additional worthy candidates for reuse are our previously created use cases, standards documents, models, procedures and guidelines.
Reuse eventually saves us time and money, and will ultimately lead to a more stable and reliable product.

Software development with reuse:
Software development with reuse is an approach which tries to maximize the reuse of existing software components. Benefit of this approach is that overall development costs of the software are decreased [6]. Cost reduction is only one potential benefit of software reuse. Systematic reuse in the development offers further advantages:

- System reliability is increased:
  Reused components in working systems should be more reliable than new components. These components have been tested in variety of operational systems environment and have therefore been exposed to realistic operating conditions.
- Overall process risk is reduced:
  If we use a function which is already exists, there is less uncertainty in the cost of reusing that in the cost of development. For project management this is important factor as it decreases uncertainty in project cost elimination. If relatively large components such as sub systems are reused then this becomes true.
- Effective use defined by specialists:
  Application specialists doing the same work on different project environment instead these specialists can develop reusable components which encapsulate their knowledge.
- Organizational standards can be embodies in reusable components:
  We can reuse some standards such as user interface standard which can be implemented as a set of standard components.

Software reuse is the process of implementing or updating software systems using existing software assets. A good software reuse process facilitates the increase of productivity, quality, and reliability, and the process of costs and implementation time. An initial investment is required to start a software reuse process, but that investment pays for itself in a few reuses [7].

In short, the development of a reuse process and repository produces a base of knowledge that improves in quality after every reuse, minimizing the amount of development work required for future projects, and ultimately reducing the risk of new projects that are based on repository knowledge.
Software component reuse does not just indicate the reuse of application code. It is possible to reuse specification and designs. The potential gains from reusing abstract product of development process such as specifications may be greater than those from reusing code components.

Software reuse enables developers to leverage past accomplishment and facilitates significant improvement in software Productivity and Quality. The Objective of this research is to estimate the software quality on the basis of Software reuse effectively.

2. Literature Survey

Software reuse enables the developers to leverage past accomplishments and facilitates significant improvements in software productivity and quality. The contribution of this paper is a recommended process model for the implementation of software reuse effectively. A critical problem in today's practice of software reuse is the lack of a standard process model which describes the necessary details to support reuse based software development and evolution [8] Software has been reused in applications development ever since programming started. However, the reuse practices have mostly been ad hoc, and the potential benefits of reuse have never been fully realized. Most of the available software development methodologies do not explicitly identify reuse activities. The Application of Reusable Software Components Project of the Software Engineering Institute is developing a reuse-based software development methodology, and the current direction and the progress of the methodology work are discussed in this paper. The methodology is based on the life cycle model in DoD-STD-2167A with refinement of each phase to identify reuse activities [9]. The reuse activities that are common across the life cycle phases are identified as: 1) Studying the problem and available solutions to the problem and developing a reuse plan or strategy, 2) Identifying a solution structure for the problem following the reuse plan, 3) Reconfiguring the solution structure to improve reuse at the next phase, 4) Acquiring instantiating, and/or modifying existing reusable components, 5) Integrating the reused and any newly developed components into the products for the phase, and 6) Evaluating the products. These activities are used as the base model for defining the specific activities at each phase of the life cycle. This methodology focuses

more on identification and application of reusable resources than on construction of reusable resources, and some enhancements in the construction aspect might be necessary to make it more complete [9].

The component reuse and maintenance requires the development and utilization of specialized tools. In order to be correctly used, any software components need to be properly understood, engineered and catalogued. Different information about components had to be organized, developed and retrieved during the process. In this paper we discuss a methodology based on Information Retrieval techniques, for automating cataloguing existing software components. We describe with an experiment the utilization of the system with prototype examples of reuse and maintenance and finally we evaluate the results of the experimental phase [10]. Software reuse enables developers to leverage past accomplishments and facilitates significant improvements in software productivity and quality. Software reuse catalyzes improvements in productivity by avoiding redevelopment and improvements in quality by incorporating components whose reliability has already been established. This study addresses a pivotal research issue that underlies software reuse—what factors characterize successful software reuse in large-scale systems? The research approach is to investigate, analyze, and evaluate software reuse empirically by mining software repositories from a NASA software development environment that actively reuses software [11].

This software environment successfully follows principles of reuse-based software development in order to achieve an average reuse of 32 percent per project, which is the average amount of software either reused or modified from previous systems. We identify two categories of factors that characterize successful reuse-based software development of large-scale systems: module design factors and module implementation factors. The modules reused without revision had the minimum faults, per source line, and lowest fault correction effort. The modules reused with major revision had the highest fault correction effort and highest fault isolation effort as well as the most changes, most changes per source line, and highest change correction effort. In conclusion, we outline future research directions that build on these software reuse ideas and strategies [11].

A framework in which to study software productivity. Approaches to understanding software development processes and improving software productivity also discussed include using and designing automated software development tools, studying human factors in software development, and applying software productivity measurement and evaluation techniques. A meta-system environment that allows users to define functionalities, structures, and constraints of various software components is discussed. Information about these components is used by a knowledge-based system to support the selection, configuration, and distribution of reusable components [12].

One of the primary obstacles to the reuse of independently-developed binary components on the industrial level lies in that the existing component technologies do not clearly separate component assembly from component development. To tackle this problem [13]. A new system was proposed a component assembly method, and a runtime framework, which together amounts to what we call Active Binding Technology. This new component model suggests how to make software components as pure parts, and the assembly method expresses message flows between these components in a model, while the runtime framework performs dependency injection to make the components interact with each other observing type safety constraints [14].

An empirical study of methods for representing reusable software components is described. Thirty-five subjects searched for reusable components in a database of UNIX tools using four different representation methods: attribute-value, enumerated, faceted, and keyword [15]. The study used Proteus, a reuse library system that supports multiple representation methods. Searching effectiveness was measured with recall, precision, and overlap. Search time for the four methods was also compared. Subjects rated the methods in terms of preference and helpfulness in understanding components. Some principles for constructing reuse libraries, based on the results of this study, are discussed [16].

3. Proposed Work

Here we are performing an analysis of available software matrices in terms of software reusability .As these metrics are defined we will generate a system metric on combination of some independent matrices and derive the decision on the basis of it. As the basic concept the complete code will be represented as the software components, it can be a source code or some library or the components and then we will establish the relationship between these components and this whole relationship will be represented in the form of a directed graph. This whole part will be represented in the form of data acquisition. Now once all of the components related to the system are derived along with their relationships between them, we need to assign some weightage to each interfacing. We can

also represent the weightage on the bases of degree in the connected graph. As shown in following figure 2.

Fig. 2 Software Component Interfacing.

As the basic initialization done we need to define some metrics that will analyze these concepts respective to a separate relation type such as interface metric.

From the above figure we know that there are some interactions between component A to B (Single directions), from A to C (both direction), from A to D (single direction), between B to D (both directions) from D to C (single direction), D to F (both direction), and E to C (single direction), F to E (single direction).

Now to estimate the software reusability in terms of software components we estimate the following matrices.

3.1 System Coupling Metrics

The system Coupling Metrics (SCOUP) for CBSS will be

$$SCOUP = \sum_{j=1}^{m} \frac{MVj}{m} \quad (1)$$

Here MV_j is the Coupling metrics for component j and m is the number of the components in CBSS [16]. We use number of components is six, so m=6.
The output obtain in the above example is SCOUP = 2.7253.

3.2 System Cohesion Metrics

The system Cohesion Metrics (COM) for CBSS will be

$$SCOH = \sum_{j=1}^{m} \frac{COMj}{m} \quad (2)$$

Here COM_j is the Cohesion metrics for component j and m is the number of components in CBSS [16]. We use number of components is six, so m=6.
The output obtain in the above example is SCOUP =1.2226.

3.3 System Actual Interface Metrics

According to above table the measurement of actual interactions for a component j, System Actual Interface Metrics (SAIM) is the integration of the interface metrics of the total number of components [16].

$$SAIM = \sum_{j=1}^{m} \frac{AIMj}{m} \quad (3)$$

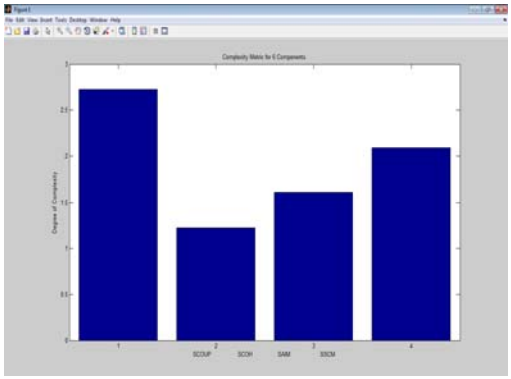
Here m is the number of components.
The output obtain in the above example is SAIM =1.6043.

3.4 Sole System Complexity Metrics

Similarly, we need to compute a sole system complexity metrics (SSCM), so we may combine above three system metrics with different weights for each metrics [16].

$$SSCM = \alpha' * SCOUP + \beta' * SCOH + \gamma' * SAIM \quad (4)$$

Here' α', β and γ are the weights for system coupling metrics, cohesion metrics, interface metrics with the condition as $\alpha' + \beta' + \gamma' = 1$.
The output obtain in the above example is SSCM



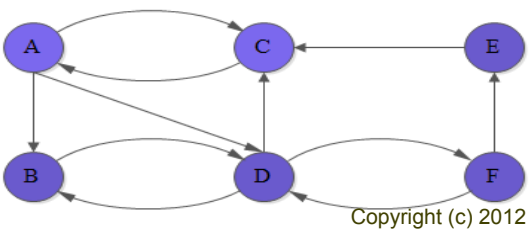
=2.0884.

Fig. 3 Results of Proposed System.

In the above figure all the matrices are collectively defined for the system. Complexity The above figure represents the complexity as 2.7253 (SCOUP), 1.2226 (SCOH), 1.6043 (SAIM) and 2.0884 (SSCM) have hardly any meaning for themselves. However, when such data are used to compare the complexity levels among several software systems, the developers will know which CBSS needs more people and more time during the coding and testing stages, or they may expect the vulnerabilities will happen in which component according to the complexity metrics.

4. Conclusion

The proposed work is about to estimate the software reusability in a software program. Software components are one of the major factors that provide the software reusability. The system will check that the use of the



component based approach in the system is favorable to the system or not.

References

[1] Arun Sharma, Rajesh Kumar, P S Grover, "Managing Component-Based Systems with Reusable Components", International Journal of Computer Science and Security, Volume 1: Issue (2).

[2] Hironori Washizaki, Hirokazu Yamamotoand Yoshiaki Fukazawa, "A Metrics Suite for Measuring Reusability of Software Components", 2005.

[3] Jagdeep Kaur Saini, Amitabh Sharma, Dr. Parvinder S. Sandhu, "Software Reusability Prediction using Density Based Clustering", 2006.

[4] G.N.K. Suresh Babu, DR.S.K.Srivatsa, "Analysis and Measures of Software Reusability", International Journal of Reviews in Computing 2009.

[5] Jo Woodison, Managing Software Reuse with Perforce", Mandarin Consulting.

[6] Maurizio Pighin "A New Methodology for Component Reuse and Maintenance" University degli Studi di Udine, Italy.

[7] K. S. Jasmine, and R. Vasantha, "DRE - A Quality Metric for Component based Software Products", World Academy of Science, Engineering Technology 34, 2007.

[8] Jasmine K.S, Dr. R. Vasantha "A New Process Model for Reuse Based Software development Approach" Proceedings of the World Congress on Engineering 2008 Vol IWCE 2008, July 2 - 4, 2008, London, U.K.

[9] Kyo C. Kang, Sholom Cohen, Robert Holibaugh, James Perry, A. Spencer Peterson, "A Reuse-Based Software Development Methodology", Software Engineering Institute Carnegie Mellon University, Pittsburgh, Pennsylvania 15213.

[10] Richard W.Selby, "Enabling Reuse-Based Software Development of Large-Scale Systems", Northrop Grumman Space Technology, One Space Park, Redondo Beach, CA 90278.

[11] Jay F. Nunamaker, Jr.Minder Chen "Software Productivity: A Framework of Study and an Approach to Reusable Components", Department of Management Information Systems, the University of Arizona Tucson, Arizona 85721.

[12] Yoonsun Lim, Myung Kim, Seungnam Jeong and Anmo Jeong "A Reuse-Based Software Development Method" Dept. of Computer Science & Engineering, Ehwa Womans university,120-750 Seoul, Korea.

[13] William B. Frakes and Thomas P. Pole "An Empirical Study of Representation Methods for Reusable Software Components", IEEE transactions on software engineering, vol. 20, august 1994.

[14] McClure, Carma McClure, "Software Reuse Techniques", Prentic-Hall, Inc., 1997.

[15] [Yu, 1991] D. Yu, "A view on Three R's (3Rs): Reuse, Re-engineering, and Reverse Engineering," Software Engineering Notes, Vol. 16, No. 3, P. 69, July. 1991.

[16] Jianguo Chen, Hui Wang, Yongxia Zhou, Stefan D. Bruda "Complexity K. Metrics for Component-based Software Systems", International Journal of Digital Content Technology and its Applications, Volume 5, 2011.

Asst. Professor. Prakriti Trivedi received his BE (Computer Science & Engg.) from MBM Engineering College Jodhpur in 1994, ME (Computer Science & Engg.) from NITTTR, Punjab University, Chandigarh, India. She has teaching experience of more than 15 years (from Aug 1995) in the field of engineering. She is presently working as Head of Department (CS & IT) in Govt. Engg. College Ajmer, Rajasthan India. She has several papers published in international & national journals and presented papers in international and national conferences. She is also responsible person in different administrative department.

Rajeev Kumar has received his B.Tech degree in (Computer Science & Engg.) from BBIET&RC Bulandshahr (Uttar Pradesh Technical University, Lucknow) in 2007 and pursuing M.Tech from Govt. Engg.College Ajmer (Rajasthan Technical University, Kota). He has presenting several papers in international & national conferences. His research interests include Component based software Reuse, software Architecture, Verification Testing etc.