A COMPLETE METRICS BASED VIEW TO ESTIMATE SOFTWARE QUALITY IN COMPONENT BASED SYSTEM

Vanish

Mr. Manmohan Sharma

Abstract:

Most of the applications in today's IT industry are developed with the help of existing codes, libraries, design, open source etc. As the code is accessed in a program it is represented as the software component. Code is a ready to use component in programming. Developing software with the help of existing component or code is called software reusability. These components can be code, architecture, documents, designs etc. While using these components the main question arises whether to use such components is worth full or not which means reusing these components increases or decreases the quality of the software. In this proposed work i have made an attempt to answer this question. In this work i am presenting a set of software metrics that will check the interconnection between the software components and the application. How strong this relation defines the software quality after using this software component. For this to be happen work i have taken four components having interconnection between them. After applying software metrics on them i will be able to suggest which component will increase the quality of the software produced. The overall metrics will return the final result in terms of the dependencies of the component with application. No doubt many techniques have been developed to estimate the quality of the software but my approach will estimate the quality of software with the help of a concept called Software Reusability. As I earlier said software reuse is the process of developing software systems using existing software assets. Good software reuse always results in the increase of productivity, quality, reliability and the decrease of costs as well as implementation time. No doubt initial investment is definitely required to start some software reuse process but that investment will automatically recover itself in few reuses. The development of a software reuse process always improves the quality of software after every reuse, minimizing the amount of development work and time required for future projects and ultimately reducing the risk of new projects that are based on repository knowledge. Reuse eventually saves our time and money and will ultimately lead to a more stable and reliable product. The benefits from reusing abstract product of development process such as specifications and designs may be greater than those from reusing code components. On the other hand it will be more convenient for developers if they already know the complexity of reused components. This will not only reduce the efforts required but also we can use our available resources in some other tasks.

Keywords: Components, Reusability, Quality, Estimation, Metrics, Complexity.

Introduction: Software engineering is very vast term as we can imagine but software reusability has completely changed the view of creating the software. Reusability not only makes the software development easier but also makes development process transparent. For making reusability happen we need software components. These components may be code, whole module etc. With the addition of these components whole life cycle of software development has 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 then we need to just perform the interfacing of the current application with these software components. Software modularization deals with the interfacing between different modules. These modules can be integrated directly or indirectly. The numbers of levels between two modules also affect its quality. Here we introduce the concept of software metrics. This metrics firstly measures the interaction between these components. Then it will measure the quality of the software which further depends on the quality of the components used.

A) 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 impendent.

(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.

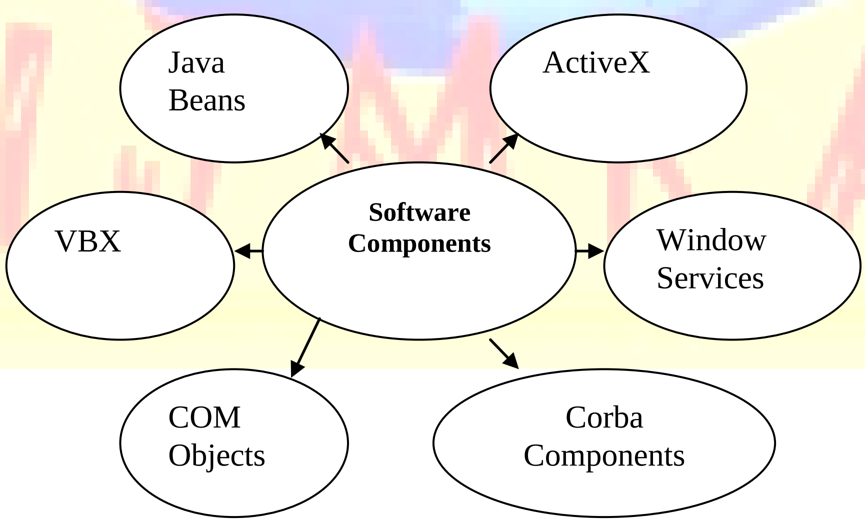


Figure 1: Software Components

b) Component Based Software Engineering:

With the increased use of business software there is requirement of more scientific ways to estimate the software quality and its complexity. Because of this there was the requirement of some improvement in the software development process. 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 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. 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.

C). Software metrics:

As the number of components available on the market increases it is becoming more important to utilize software metrics to quantify the various characteristics of components and their usage. Software metrics are used to measure the software quality as well as performance characteristics quantitatively which are encountered during the process of software development. These can serve as measures of software products for the purpose of comparison, cost estimation, fault prediction and forecasting. Metrics can be helpful in guiding decisions throughout the life cycle and determines whether the software quality improvement initiatives are financially worthwhile or not. 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.

D) Software Reusability

Software reuse is a step by step process of implementing and updating software systems using existing software components. A good software reuse process results in the increase of productivity, quality, reliability and the decrease of costs as well implementation time. An initial investment is required to start a software reuse process but that investment pays for itself in a few reuses. 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.

Why Reuse?

Reuse has proved to offer many rewards. When we reuse code, components and other artifacts our goal is to:

- 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.

Software development with reuse:

Reuse in software development is an attempt which tries to maximize the reuse of existing software components and benefit of this approach is reduction of overall development costs of the software [3]. Cost reduction is one of the potential benefits of the software reuse. Systematic reuse in the development offers further advantages:

- System reliability is increased.
- Reused components in the working systems should be more reliable than the new components. These components have been tested in the variety of the operational systems environment and have been exposed to many realistic operating conditions.

- Overall process risk is reduced.
- If we use a function which already exists there is less uncertainty in the cost of reusing that component than in the costs of development. In project management this is important factor as it decreases the uncertainty in the 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 embodied in reusable components.
- We can reuse some standards such as user interface standard which can be implemented as a set of standard components.

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 reuses based software development and evolution [1]. Software has been reused in applications development ever since programming started. The reuse practices used to be ad hoc and the potential benefits of reuse have never been fully realized. However, most of the available software development methodologies do not explicitly identify reuse activities. The applications of reusable software components of the software engineering institute is developing a reuse based software development methodology , the current direction and the progress of this methodology works are discussed in this paper. The methodology is based on the life cycle model of DoD-STD-2167A with refinement of each and every phase to identify reuse activities[2] The reuse activities which are common through the life cycle phases are identified as: 1- Study the problem and available solutions to that problem and develop a reuse plan or strategy 2- Identify the solution structure for the problem following the reuse strategy 3- Reconfigure the solution structure to enhance reuse at the next phases 4- Acquire instantiating and modifying existing reusable components 5- Integrate the reused as well as newly developed components into the products for the phases 6) Evaluate the products. These activities act as the base model for defining of the specific activities at every phase of the life cycle. This methodology has more focus 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 [2]. The component reuse and maintenance requires the development as well as utilization of specialized tools. In order to be correctly used any software components needs to be properly understood engineered and catalogued. Various kinds of information about components had to be organized, developed and retrieved during the whole process. In this paper we will discuss a methodology which is based on information retrieval techniques for automating existing software components. We describe an experiment for utilization of the system with prototype examples of reuse, maintenance and finally we evaluate the result of the experimental phase [3]. Software reuse enables the developers to cover past accomplishments and facilitates significant improvements in the software productivity and quality. Software reuse enhance the improvements in productivity by avoiding redevelopment of components and improvements in quality by incorporating those components whose reliability has already been checked and established. This study addresses a research issue that underlies software reuse:-what factors characterizes the successful software reuse in large-scale systems? The research attempt is to investigate, analyze and evaluate the software reuse empirically by analyzing software repositories from a NASA software development environment that actively reuses software components [5]. This software environment successfully follows the principles of reuse based software development to have an average reuse of 32% per project which is an average amount of software either reused or modified from the previous software. We identify two 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. In conclusion we outline the future research direction that build on these software reuse ideas as well as strategies [5]. Approaches to understand software development process and improving software productivity also include using and designing automated software development tools, study of human factors in software development, applying software productivity measurement as well as evaluation techniques. A meta-system environment that allows users to define functionalities, structures, and constraints of various software components is discussed. Information of these components is used by the knowledge based system to support the selection, configuration and distribution of reusable components [6]. One of the primary obstacle to the reuse of independently developed binary components on the industry level lies in the existing component technology that do not clearly separates the component assembly from the component development. To handle this problem a new system was proposed a component assembly method and a runtime framework which together amounts to what we call as Active Binding Technology. This component model mentions how to make software components as pure parts and the assembly method express the message flow between these components in a model while the runtime framework performs dependency injection to make the components interaction with each other observing type safety constraints. An empirical study of the methods for representing reusable software components is described. 35 subjects searched for reusable components in the database of UNIX tools using 4 different representation methods:1- attribute-value 2- enumerated 3- faceted 4- keyword. The study used Proteus, a reuse library system which supports multiple representations of methods. Searching effectiveness was measured with the help of recall, precision and overlap. Search time for the four methods was also compared. Subjects rated the methods in terms of preference and helpful in understanding the components. Some principles for constructing reuse libraries based on the results of this study have been discussed.

According to my literature survey i came to know that no work has been done precisely on the topic which i am doing. But still lots of reviews has been given by different personalities on the topic (and related to) software reuse. I believe that in present scenario of software development there is huge need to calculate the complexity of components which are going to be reused so that developers will not face any problem while reusing any component. Since reuse components are already being tested so it will not only increase the quality of the software but also enhance as well encourages the software reusability which is the final goal of the proposed paper.

Methodology:

I will propose an interface complexity metric for software reused components which is based on their complexity involved. Based on properties of reused components, some values or degrees will be assigned to them and these values or degrees will be used to measure the complexity of target software. So, in the end these values or their sum will give the overall reusability interface complexity of the software. No precise criteria exists which measure the reusability with respect to quality characteristics of the components used like complexity, interfacing, maintainability etc. Components of the software are the main objects which performs reusability. Almost all applications are component based systems. For example, a car is made up of thousands of parts (or components) and these parts have come from a multitude of different places. The car engine might have been built in Germany, the tyres in France, the exhaust in the USA, the upholstery in Italy, etc... And all this assembled in a factory in the UK. Most of the work done while proposing the metrics of the characteristics are either theoretically done without any validated methodology or only considers source code as the main component. My work will find the degree of reusability on the basis of their components and their complexity. My study will use properties and methods of the components with its interfacing with the system. So, it will estimate the software quality in terms of software component reusability.

Implementation Design:

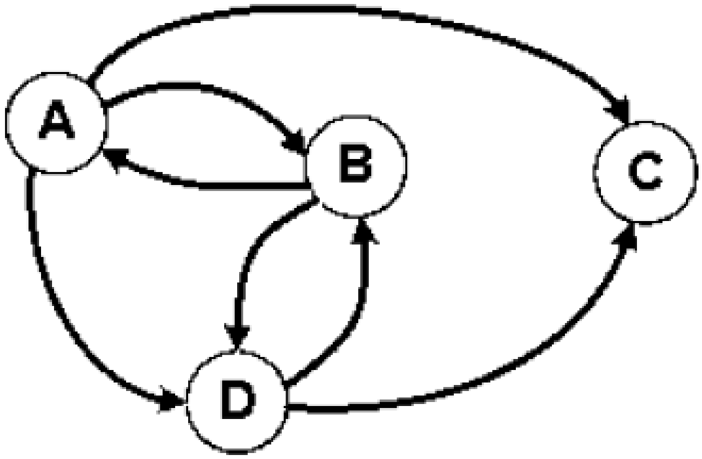
My proposed work is to estimate software quality with respect to software reusability. Now to estimate this in terms of software components we need to estimate some factors such as

1. Firstly we have to find all kinds of software components (as suggested in introduction) or the codes which are responsible for software reusability. We are representing all kinds of code and the components as a single unit called components.
2. Secondly we have to find that which kind of reusability it is, which means in what ways the component becomes the part of the main application. For example: - Is it being used as a parent class or is used as the composition etc.
3. Now we have to find the interface metrics of the component with the main program and assign some values to the components according to their complexity.

4. We have to estimate the relationship between different components. It is possible that different components may have some bonding between themselves. With the help of these relationships and the values of the components (as suggested in point no. 3) we will estimate the quality of the software with respect to software reusability.

Proposed work:

My emphasis is on evaluation of various functional and non-functional aspects of components of software development as well as the effect of components on the system. These aspects include complexity, reusability, maintainability, customizability etc. So my objective is to design a methodology to estimate the quality of software using the complexity of the reused components. I want to propose a system which estimates the quality of software with respect to the reusability of components. My focus is on present technologies but present methodologies are not enough so this is my small approach to improve software reusability by removing the drawbacks and making some enhancements. Software reusability is one of the important aspects of software engineering and software development. With the evolution of modular programming the software reuse is increased very fast. The component based software programming is the most promising technology for software reusability. The 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. As we all know that only that component will be beneficial for us if it has low coupling and high cohesion. This is the main concept behind implementation. For the representation of whole concept i am taking an example of four components.



This figure shows the bonding between four components: A, B, C and D. This also shows the interaction between four components A, B, C and D:

1. A to B [Both directions]

2. A to C [Single direction]

3. A to D [Single direction]

4. B to D [Both directions]

5. D to C [Single directions]

Another concept which is very useful in this implementation is coupling. Coupling means interconnections and interdependencies between two or more components. It means changes in one component leads to changes in other components which is not a good sign as it makes the software more complex and hence decreasing the quality of the software produced with the help of these components. So, the software is supposed to be of good quality if the coupling between its components is very low. This means low coupling leads to high quality.

Now out of above chosen four components we will use only that component which has least coupling among all. The component with least coupling will increase the quality of the software to the most. For this, we need to have coupling metrics. I found out coupling metrics between these four components by using the formula:

$$MV_j = \frac{\left| \bigcup\limits_{1 \le i \le m \cap i \ne j} MV_{j,i} \right|}{\left| M_j \right| + \left| V_j \right|}$$

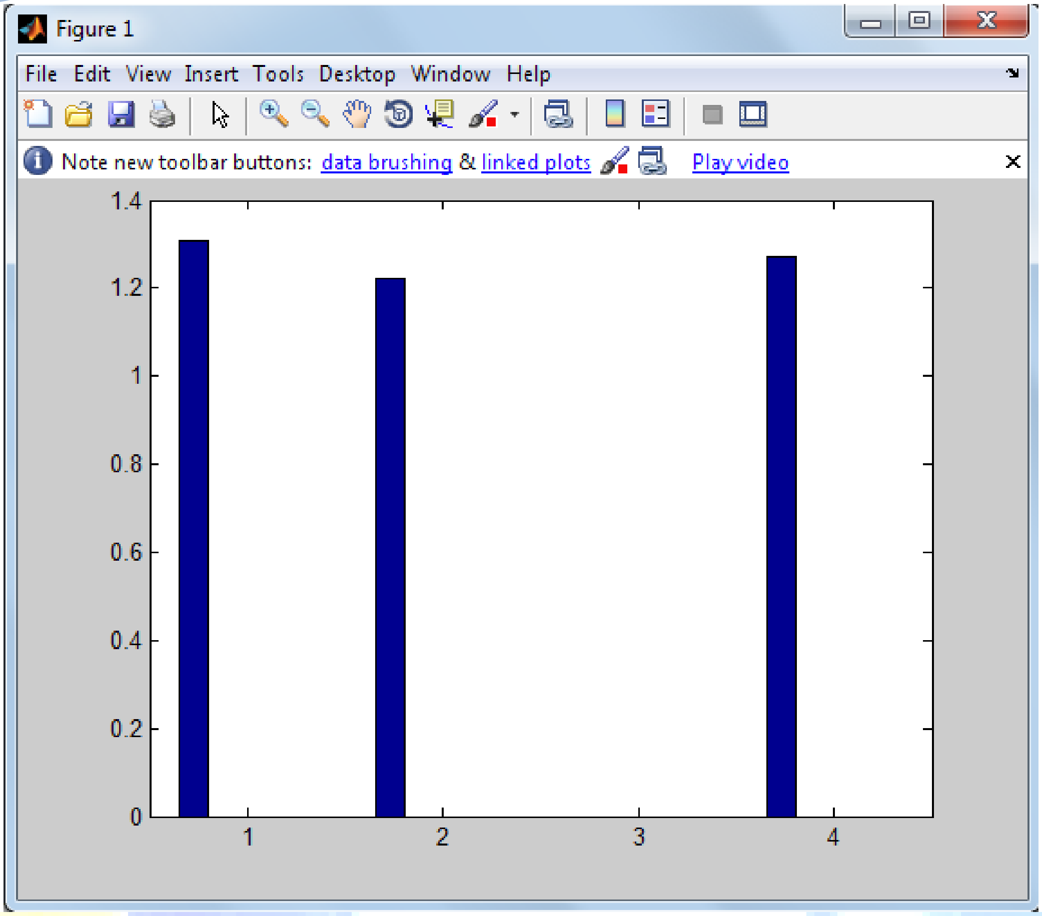
Here, M is method and V is variable.

Component	M1j	V1j	A		B		C		D		IImax	OImax	E C	II	OI
i	M1	V1	Mvij 1	Mvij 2	Mvij 1	Mvi 2	Mvij 1	Mvij 2	Mvij 1	Mvij 2					
A	7	6	0	0	3	2	3	3	4	2	10	9	20	7	17
B	4	5	2	4	0	0	0	0	3	2	8	7	5	5	12
C	5	6	0	0	0	0	0	0	0	0	6	9	10	13	0
D	6	5	3	4	0	0	4	3	0	0	8	9	25	11	14

In the above table MVij Represents the Method respective to the instance of some class. Here Mi represents the ith Method and Vj represents the jth Instance access in the program. II represents the available number of incoming interactions and OI represents the available number of outgoing interactions. The ratio of incoming and outgoing interactions is defined by II/OI

Result Achieved

When we execute all these values on the MATLAB we get a bar graph showing the coupling metrics of each component:



According of the graph, increasing order of coupling metrics values of all the components are as follows:

Component no. 3- 1st

Component no. 2- 2nd

Component no. 4- 3rd

Component no. 1- 4th

Component no. 3 has least value so it will be good choice if we recommend this component for reuse because low coupling and high cohesion is recommended for software development. Till now i have only done work till coupling, rest i will do after midterm.

Reason for using only these parameters: Since the name of my paper is about software quality. The quality of a software depends on how and what kind of components we reuse in software development. It means if the components we are going to reuse are of high quality and less complex then ultimately the quality of the produced software will be of high quality. Now, in the components the things which are common between them are **methods, variables and functions** which mean the quality and complexity of a component is totally dependent on these parameters and their sharing as well as interdependency. So the best way to find out the quality of the components is to work on these parameters which i have done. With help of these parameters i am able to find the coupling between the components.

Reason for using MATLAB: No doubt there are lots of tools and techniques available for implementation but i prefer MATLAB. The need of a person for using a tool depends on his requirements and according to my requirements i find MATLAB beneficial for me. There are also other factors that compelled me to choose MATLAB:

1. It is very easy to use MATLAB for a beginner like me.
2. It simplifies the numerical calculations without writing the whole complicated and time consuming program.
3. It graphs the result easily without complex programming.
4. MATLAB is flexible and platform independent.

With the help of MATLAB i am trying to show a bar graph. This bar graph represents the coupling of all the four components which i have used. With the help of this bar graph it is very easy even for a new user to find which component has high coupling and which has low. So, due to this easy user interface i am using MATLAB for implementation.

Testing in implementation:

Testing is basically done to provide stakeholders with the information about the quality of the product. But in my thesis testing will be required to authenticate the values i will use while implementation. Testing also provides an independent view of the software to apply modifications and enhancing the quality, similarly in my thesis testing will provide me an independent view for understanding the risks of implementation. Testing will be helpful in validating as well as verifying the values of methods and variables used in the implementation. It will also prove to be helpful in the following way:

1. If the shows that the values used will meet the requirements of the design and development.

2. If it shows the working of the values as expected.

3. If it ensures that these values can be implemented with same characteristics without any changes.

=) Testing plays very important role while implementing the phases of thesis however it will be more beneficial if we do testing while implementing the values of methods and variables. Testing can never finds all the defects in the values but still it will be helpful in finding the wrong values and also those values whose existence is under doubt.

=) The persons who will get benefit from my thesis are users. It means our target is defined i.e. users; no other person will get direct benefit from our approach. So when our target is defined then testing should also be done by considering the users point of view. We will do testing in such a way that in future developer will not face problem which reusing software components. Now, when we have defined values and defined target here starts the main working of testing which not only helps in finding the defects but also helps in correcting them. Testing cannot confirm that the product will work correctly under all conditions but it defines the condition under which the product will not work properly. These conditions will be helpful for us in finding faulty values.

=) Since this is first phase of implementation that's why i have taken values from a case study. I have also chosen these values because i want to confirm what result it gives whether satisfactory or not. But in future if required i will perform testing on assumed values. This will help in defining the authentication of the values and also help in removing defected values.

Test Sequence- 1

1 Test Id: SR001

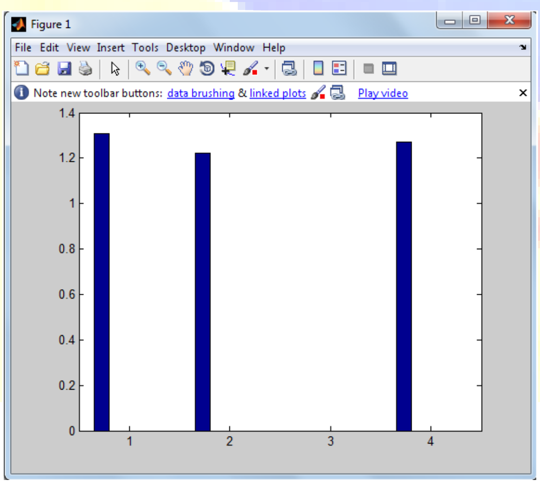
2 Test Name: To test whether values of method and variables are correct or not.

3 Steps: First we take default values. Then run them on MATLAB to check they give Desired output or not.

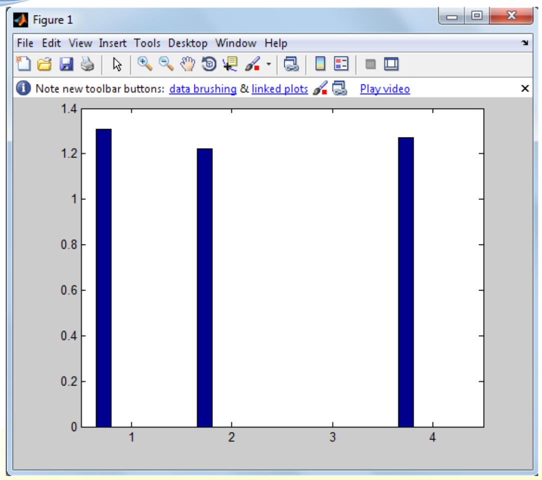
4 Input: Default values

Component	M1j	V1j	A		B		C		D		IImax	OImax	E C	II	OI
i	M1	V1	Mvij 1	Mvij 2	Mvij 1	Mvi 2	Mvij 1	Mvij 2	Mvij 1	Mvij 2					
A	7	6	0	0	3	2	3	3	4	2	10	9	20	7	17
B	4	5	2	4	0	0	0	0	3	2	8	7	5	5	12
C	5	6	0	0	0	0	0	0	0	0	6	9	10	13	0
D	6	5	3	4	0	0	4	3	0	0	8	9	25	11	14

5 Expected output:



6 Actual output:



7 Remarks: Pass

Test Sequence- 2

1 Test Id: SR002

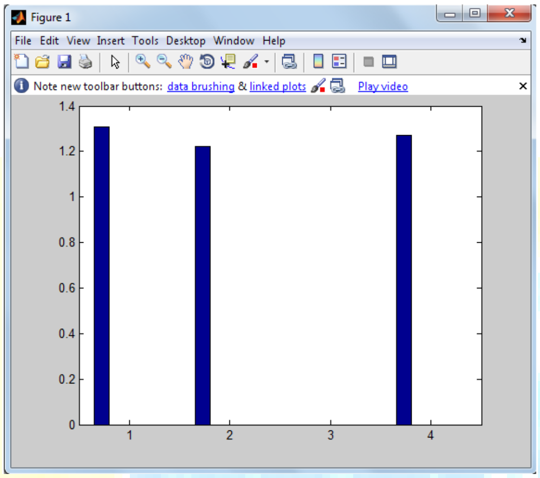
2 Test Name: To test whether values of method and variables are correct or not.

3 Steps: Firstly we change the values of component A. Then run them on MATLAB to Check whether they give desired output or not.

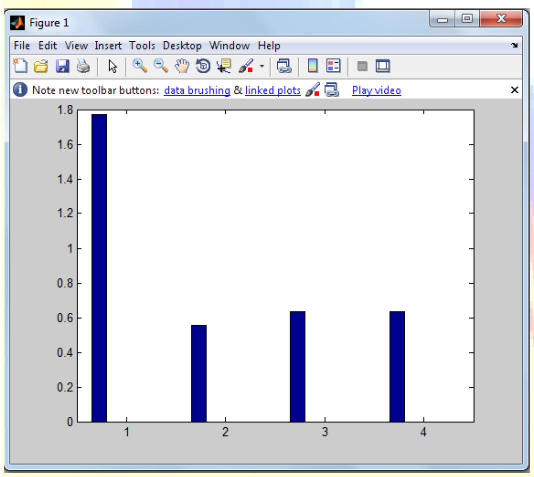
4 Input: Values with changes in component A.

Component	**M1j**	**V1j**	**A**		**B**		**C**		**D**		**IImax**	**OImax**	**E C**	**II**	**OI**
i	**M1**	**V1**	**Mvij 1**	**Mvij 2**	**Mvij 1**	**Mvi 2**	**Mvij 1**	**Mvij 2**	**Mvij 1**	**Mvij 2**					
A	7	6	2	4	3	2	3	3	4	2	10	9	20	7	17
B	4	5	0	0	0	0	0	0	3	2	8	7	5	5	12
C	5	6	3	4	0	0	0	0	0	0	6	9	10	13	0
D	6	5	0	0	0	0	4	3	0	0	8	9	25	11	14

5 Expected output:



6 Actual output:



7 Remarks: Fail

Test Sequence- 3

1 Test Id: SR003

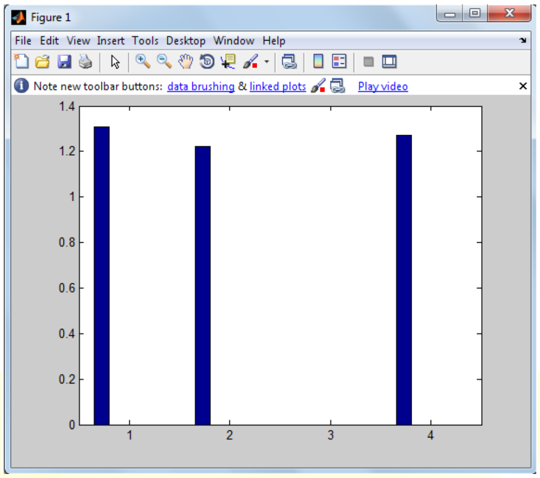
2 Test Name: To test whether values of method and variables are correct or not.

3 Steps: Firstly we change the values of component B. Then run them on MATLAB to Check whether they give desired output or not.

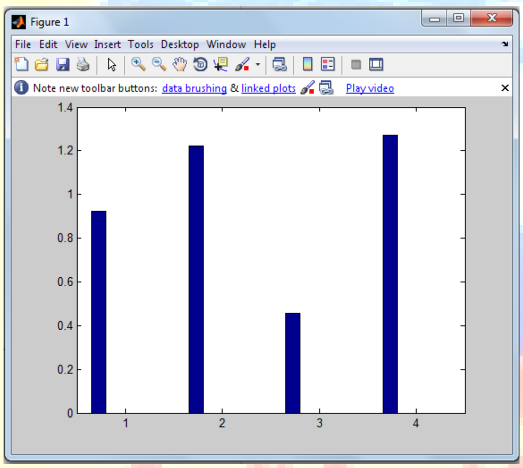
4 Input: Values with changes in component B.

Component	**M1j**	**V1j**	**A**		**B**		**C**		**D**		**IImax**	**OImax**	**E C**	**II**	**OI**
i	**M1**	**V1**	**Mvij 1**	**Mvij 2**	**Mvij 1**	**Mvi 2**	**Mvij 1**	**Mvij 2**	**Mvij 1**	**Mvij 2**					
A	7	6	2	4	0	0	3	3	4	2	10	9	20	7	17
B	4	5	0	0	0	0	0	0	3	2	8	7	5	5	12
C	5	6	3	4	3	2	0	0	0	0	6	9	10	13	0
D	6	5	0	0	0	0	4	3	0	0	8	9	25	11	14

5 Expected output:



6 Actual output:



7 Remarks: Fail

Test Sequence- 4

1 Test Id: SR004

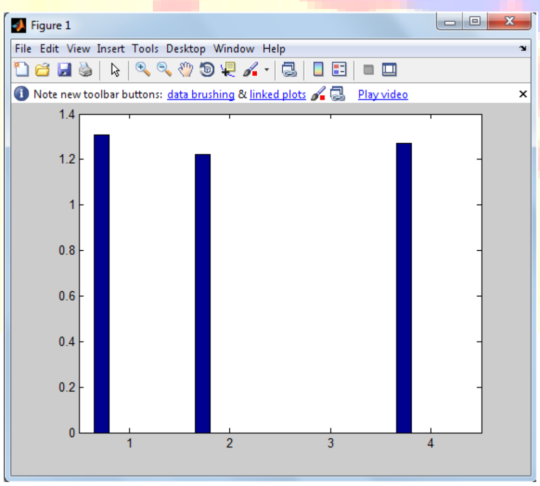
2 Test Name: To test whether values of method and variables are correct or not.

3 Steps: Firstly we change the values of component C. Then run them on MATLAB to Check whether they give desired output or not.

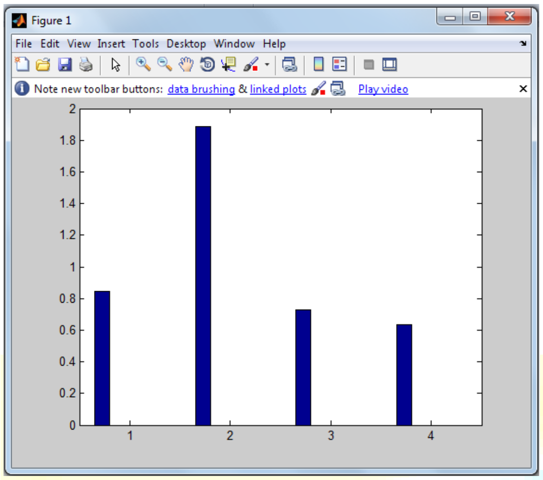
4 Input: Values with changes in component C.

Component	**M1j**	**V1j**	**A**		**B**		**C**		**D**		**IImax**	**OImax**	**E C**	**II**	**OI**
i	**M1**	**V1**	**Mvij 1**	**Mvij 2**	**Mvij 1**	**Mvi 2**	**Mvij 1**	**Mvij 2**	**Mvij 1**	**Mvij 2**					
A	7	6	2	4	0	0	0	0	4	2	10	9	20	7	17
B	4	5	0	0	0	0	3	3	3	2	8	7	5	5	12
C	5	6	3	4	3	2	4	4	0	0	6	9	10	13	0
D	6	5	0	0	0	0	0	0	0	0	8	9	25	11	14

5 Expected output:



6 Actual output:



7 Remarks: Fail

Test Sequence- 5

1 Test Id: SR005

2 Test Name: To test whether values of method and variables are correct or not.

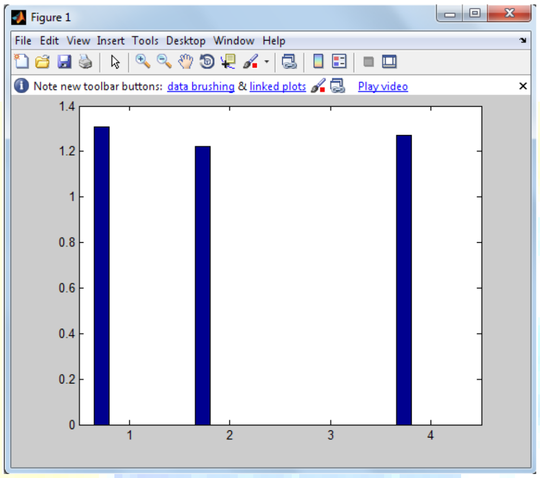
3 Steps: Firstly we change the values of component D. Then run them on MATLAB to Check whether they give desired output or not.

4 Input: Values with changes in component D.

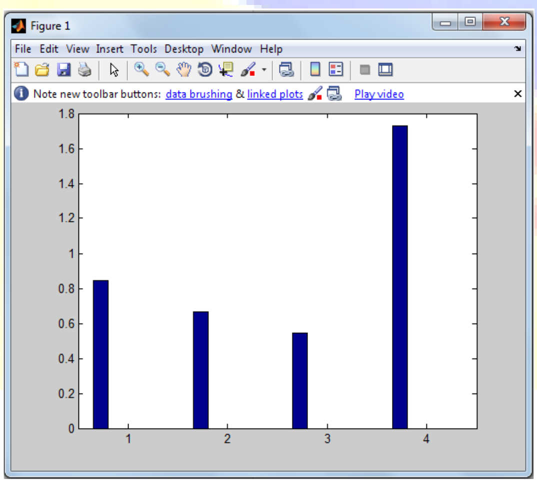
Component	M1j	V1j	A		B		C		D		IImax	OImax	E C	II	OI
i	M1	V1	Mvij 1	Mvij 2	Mvij 1	Mvi 2	Mvij 1	Mvij 2	Mvij 1	Mvij 2					
A	7	6	2	4	0	0	0	0	0	0	10	9	20	7	17
B	4	5	0	0	0	0	3	3	0	0	8	7	5	5	12

C	5	6	3	4	3	2	4	4	4	2	6	9	10	13	0
D	6	5	0	0	0	0	0	0	3	2	8	9	25	11	14

5 Expected output:



6 Actual output:



7 Remarks: Fail

Conclusion:

The proposed work is about to estimate the software quality using reusable software components. 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 which will recommend the quality of the software. The end result will show the bar graph having quality of each component based on which developer can use the component which will increase the quality of the software produced.

References:

[1] 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.

[2] 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.

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

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

[5]. 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.

[6] 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.

[7] 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.

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

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

[10] [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.

[11] [Feiler, 1993] P. H. Feiler, "Reengineering: an engineering problem," Software Engineering Institute, Carnegie Mellon University: Special Report CMU_SEI-93-SR-5, July, 1993

[12] Ivan Jacobson, Martin Griss and Patrik Jonsson, Software Reuse-Architecture, Process and Organization for Business Success, ACM Press,2000.

[13] Ian Somerville, Software Engineering, A practitioner's approach, 6th Edition, Pearson Education, 2001.