ISSN No. 0976-5697

Volume 4, No. 11, Nov-Dec 2013

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

A Review on Quality Attributes Based Software Metrics

Neha Gehlot
Cse Department, ITM University
Gurgaon, Haryana, India
neha12sep009@itmindia.edu

Jagdeep Kaur
Cse Department, ITM University
Gurgaon, Haryana, India
jagdeep@itmindia.edu

***Abstract*;** The process of software development, including documentation, design, program ,test and maintenance can be measured statistically. Therefore the quality of software can be monitored efficiently. Software metrics is very important in research of software engineering and it has developed gradually. Component-based systems(CBS) achieve flexibility by clearly separating the stable parts of systems (i.e. the components) from the specification of their composition. In order to realize the reuse of components effectively in component based system development (CBSD), it is required to measure the reusability of components. However, due to the black-box nature of components where the source code of these components are not available, it is difficult to use conventional metrics in CBSD as these metrics require analysis of source codes. In this paper, we survey few existing component-quality attribute based metrics with their limitations and how these metrics helps in computing the quality of the software and how their use can help achieve a high quality software system. These metrics give a border view of component's complexity, reusability, interface complexity and coupling among the components. As the CBS development is rising more and more quality attribute based metrics are being developed for the same.

Keywords: components; metrics; quality; attributes

I. INTRODUCTION

Component-based software engineering (CBSE) has been characterized by two development processes: the development of components for reuse and the development of component-based software systems (CBSS) with reuse by integrating components that have been deployed independently. CBSE proved to be the best practices development paradigm in terms of both time and Component-based software engineering (CBSE) has been characterized by two development processes: the development of components for reuse and the development of component-based cost. For continuous success of this developmental approach, the evaluation of CBSSs and individual components is an essential research area. Being able to isolate weaknesses over the entire software life cycle. The two different objectively measure the quality of CBSS attributes, helps us to better software systems (CBSS) with reuse by integrating components that have been deployed independently. CBSE proved to understand, evaluate, and control the quality of CBSSs and processes of CBSE led us to distinguish between metrics that are relevant to component producers and those that are relevant to component consumers. Component producers are concerned with the design, implementation and maintenance of individual components whereas component consumers search for specific components, evaluate them and integrate them to construct a CBSS[1].

One of the difficulties of applying the existing traditional metrics to a CBSS is the inadequacy of the measurement Unit. Procedural metrics focus on measures that are derived from code, for example, lines of code (LOC). Object-oriented metrics focus on measures that are derived from both the code level and the higher level units, such as methods, classes, packages or subsystems. Object-oriented metrics are also restricted in their application to CBSSs because CBSS interfaces are usually specified at the component level, not at the class level. Thus, several authors have described different techniques and guidelines and have proposed a wide-ranging set of metrics for assessing the quality of CBSS attributes. The limitations of existing CBSE metrics approaches not only are the lack of consistent approaches and measures that provide a reliable method to evaluate component quality, but also include the ambiguity in their definitions and the lack of an appropriate mathematical property that can fail quality metrics.

II. REUSABILITY

In CBD, applications are built from existing components, primarily by assembling and replacing interoperable parts. Thus a single component can be reused in many applications, giving a faster development of applications with reduced cost and high quality. The reason is that these components are tested under varieties of situations before being used in the application .Another characterization of software reuse is the way it is implemented. First, known as white-box Reuse, when reuse is attempted, developers usually have the access to the code that can be modified to cater the new demands of the application. This provides a flexible way to harvest software assets in development projects by fitting existing components to new requirements, thus maximizing the reuse opportunities.

III. REUSABILITY METRICS

According to [1] reusability can measure the degree of features that are reused in building new applications.

There are 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 entitysuch as methods and classes, and on the external attributes that measures the interaction among entities such as coupling & inheritance.

Poulin & Cho et al's reusability metrics

[1] presents a set of metrics to estimate the efforts saved by reuse. The study suggests the potential benefits against the expenditures of time and resources required to identify andintegrate reusable software into a product.

1.Study assumes the cost as the set of data elements like Shipped Source Instructions (SSI), Changed Source Instructions (CSI), Reused sourceInstructions (RSI) etc.

2. Reusability can also be measured indirectly. Complexity, adaptability and observability can be considered as a good measure of reusability indirectly.
3. Cho et al propose a set of metrics for measuring various aspects of software components like complexity, customizability and reusability. The work considers two approaches to measure the reusability of a component.
4. The first is a metric that measures how a component has reusability and may be used at design phase in a component development process.
5. This metric, Component Reusability (CR) is calculated by dividing sum of interface methods providing commonality functions in a domain to the sum of total interface methods.
6. The second approach is a metric called Component Reusability level (CRL) to measure particular component's reuse level per application in a component based software development. CRLLOC, which is measured by using lines of code, and is expressed as percentage as given as

CRL LOC (C) = (Reuse (C) / Size (C)) *100%

where:

Reuse(C): The lines of code reused component in an application,

Size(C): The total lines of code delivered in the application.

7 .The limitation of his metric was that this metric gives an indication of higher reusability if a large number of functions used in a component. However, the proposed metrics are based on lines of codes and can only be used only at design time for components.

Washizaki's reusability metrics for black-box components

1. Scope of Washizaki's reusability metrics is JavaBeans Interfaces.Intent is to propose a metrics set for assessing the reusability of JavaBeans.
2. The metrics set is defined in the scope of a quality model for black-box component reusability.
3. propose a Component Reusability Model for black-box components from the viewpoint of component users.

The proposed metrics suite considers understandability,adaptability and portability as relevant sub-characteristics of reusability. These metrics are:

1. Existence of Meta-Information (EMI) checks whether the BeanInfo class corresponding to the target component C is provided.
2. Rate of Component's Observability (RCO) is a percentage of readableb properties in all fields implemented within the Façade class of a component C. The metric indicates that high value of readability
3. Rate of Component's Customizability (RCC) is a percentage of writable properties in all fields implemented within Façade class of a component C. the high level of customizability of component as per the user's requirement and thus leading to high adaptability.
4. Self-completeness of Component's Return Value (SCCr) is the percentage of business methods without any return value in all business methods implemented within a component.

The limitation of this metric is that these metrics are applied on only for small Java Bean components and need to be validated for other component technologies like .NET, ActiveX and others also. It gives an insight view of the reusability metrics.

However, an independent experiment showed the metrics to be unreliable for components with a small number of features on their interface.. Further independent analysis is still required. Researchers for further study and empirical validation of these existing metrics can use the review done for CBS. Also, some new enhanced metrics can be proposed and empirically validated on the basis of the work already done by researchers in this area.

Dumke's metrics for reusability of JavaBeans

1 .According to [2] the scope of dumke's metrics are White-box Java Beans with an intent to present a metrics set for reusability of JavaBeans. using a technique of Informal definition of metrics, relying on access to the source code.
2. The metrics in this set are adapted from other contexts, such as OO design and structured programming.
3. The limitation of this metric is that the white-box view of components renders this approach inadequate for evaluation by independent component assemblers. The internal complexity of a component method should not be relevant for the understandability of its interface and the component's reusability.

IV. COMPLEXITY METRICS

Complexity can be defined as a measure of a how big or complex a system is to handle and work with. According to[6]

Size and complexity: direct metrics. We need a set of direct metrics (i.e., metrics computed directly from the source code) to describe a system in simple, absolute terms. They count the most significant modularity units of an object-oriented system, from the highest level (i.e., packages or namespaces), down to the there is one metric in the overview pyramid that measures it. The metrics are placed one per line in a top-down manner.

1.**NOP — Number of Packages**, i.e., the number of high-level packaging mechanisms, e.g., packages in Java, namespaces in C++.

2.NOC — Number of Classes, i.e., the number of classes defined in the system, not counting library classes.

3.NOM — Number of Operations, 1 i.e., the total number of user defined operations within the system, including both methods and global functions.

4.LOC — Lines of Code, i.e., the lines of all user-defined operations. In the Overview Pyramid only the code lines containing functionality are counted.

5.CYCLO — Cyclomatic Number, i.e., the total number of possible program paths summed from all the operations in the system. It is the sum of McCabe's Cyclometric number for all operations.

These are the direct measure of the complexity of a system and can vary at different user level and are thus are not an appropiate choice for measuring complexity of large system with large number of operations and methods.

The Interface Method Complexity

According to [3]method for determining the complexity of interface methods has been defined. High interface methods complexity shows more complexity of component.

1.***The interface methods can be divided in the following categories:***

Interface methods having no return value and no parameter, having return value but no parameters ,no return value but having parameters, return value as well as parameters.

The complexity of the interface methods can be measured on the basis of data types of return value and parameters, and on the basis of number of parameters. On the basis of data type of

return value and parameters, and by considering the number of parameters in a method

2.Thus a Interface Method Complexity Metric for Black Box Component, IMCM(BB), has been defined as below:

IMCM(BB) = Wr + PCM(M) Where Wr represents the weight assigned to the category of return value's data type and PCM(M) is Parameters Complexity Metric for Method which calculate the complexity caused by parameters.

3.Parameters Complexity Metric for Method ,PCM(M), has been defined as below: PCM(M) = a*Wvs + b*Ws + c*Wm + d*Wc + e*WvcWhere a,b,c,d,e represent counts and Wvs,Ws,Wm,Wc,Wvc represent the assigned weights for very simple, simple, medium, complex and very complex data type categories for parameters of a method.***High value of IMCM(BB) shows decrease in understandability and increase in testing effort.***

Steps to Calculate CCCM(BB)

Step 1 : Calculate FICM(BB) Fan-in Complexity Metric for Black Box Component, FICM(BB) = fin * [Cn * .10 + (Count the different types of data type incompatibilities need to be handled to receive the data in the correct form and multiply the different counts with their respective weights

Step 2: Calculate FOCM(BB) Fan-out Complexity Metric for Black Box Component, FOCM(BB) = fout * [Cn *.10+ (Count the different types of data type incompatibilities need to be handled to provide the data in the correct form and multiply the different counts with their respective weights Cn represents the count of interactions causing no incompatibility problem.

Step 3: Calculate CCCM(BB) CCCM(BB) = FICM(BB) + FOCM(BB)

4.High coupling complexity shows that more integration andtesting effort is required. But it represents low maintainability.

Determine Component Complexity Metric for Black Box Component

Component Complexity Metric for Black Box Component, CCM(BB), has been defined as below

i=n CCM(BB) = CCCM(BB) + Σ IMCM(BB) i=1

5.Limitation of this metric is that it is based on component specfication, and component specification at an early stage are difficult to estimate,if there is an ambiguity in the specifications taken in the begining of the CBSD ,can create problems later. It has been proposed for the black box component only.

6.Thus measuring the component complexity during the component selection is a difficult task and can be misleading as an important component may not get selected.

Gill's interface complexity metrics

According to[2]scope of gills metric is Black-box component's interface, with the intent of providing the complexity aspects of interfaces' signature, with also constraints upon those interfaces, as well as their packaging, to account for different configurations that the interface may present, depending on the context of use.

1.Following technique that the overall complexity is defined as the weighted sum of the complexities related to signature, constraints and packaging of the interfaces.

2.For each of these aspects of interface complexity, a definition is also proposed, again using weighted sums of features (e.g. events and operations count).Thus has the merit of including constraints and packaging complexities on the assessment.

3. Demerits of Gill's proposal is that it still lacks any sort of empirical assessment. This hampers the ability of the authors to assign values to the coefficients on their definitions, and, more significantly, our ability to assess the extent to which this approach helps common practitioners to choose among alternative components.thus there is lack of maturity.

V. INTERFACE COMPLEXITY

According to[4], a component is linked with other components and hence has interfaces with them. Two or more components are said to be interfaced if there is a link between them, where a link means that a component submits an event and other components receive it. The direction of the link indicates that which component requests the services or dependent on the other. Interface between two components can be through incoming and outgoing interactions.

1.These both types of interactions add complexity to a component-based software system. By taking only interface complexity into account, an interface complexity measure for a component-based system is suggested as

Average Incoming Interactions Complexity (AIIC) =sum of all incoming interaction/m

Average Outgoing Interactions Complexity (AOIC) = sum of all outgoing interactions/m

Average Interface Complexity of a Component Based System (AIC (CBS)) = AIIC+AOIC

Number of components in the Component Based System (CBS) = m

[4] also evaluated the metrics against Weyuker,who proposed an axiomatic framework in the form of several properties for evaluating complexity aspects of software systems.

2. The proposed interface complexity metric reported here is evaluated against these properties for compatibility. These properties are evaluated for the proposed interface metric.

3.Demerits of this metrics is that the experiment is based on directed graph of components, thus generating directed graphsfor lagrge system is an overhead, cannot deciding the overall complexity of a component-based system.

4..However, application of conclusions to real life situations needs further study and empirical support using data from industrial projects

to validate these findings and to derive more useful and generalized results.

VI. COUPLING METRIC

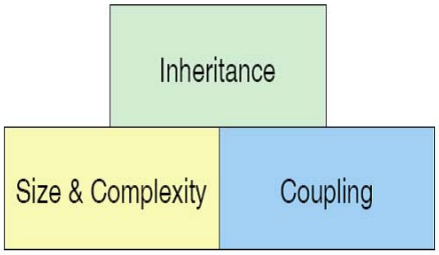

fig I:

The Right Part: System Coupling

According to suri,garg [6]the second part of the Overview Pyramid provides an overview with information about the level of coupling in the system,by means of operation invocations.

1. System coupling: direct metrics. The key questions when trying to characterize the level of coupling in a software system are: How intensive and how dispersed is coupling in the system? The two direct metrics that we use are:

2. CALLS — Number of Operation Calls*, i.*e., this metric counts the total number of distinct operation calls (invocations) in the project, by summing the number of operations called by all the user-defined operations. If an operation fo () is called three times by a method f1() it will be counted only once. If it is called by methods f1(), f2() and f3(), three calls will be counted for this metric.

*3. F****ANOUT — Number of Called Classes****,* this is computed as a sum of the FANOUT metric (i.e., classes from which operations call methods) for all user-defined operations. This metric provides raw information about how dispersed operation calls are in classes.

Dhama Coupling Metric

[6] Coupling is a measure of how closely tied are two or more modules or class. In particular, a coupling should indicate how likely would be that a change to another module would affect this module .

1. The basic form of coupling metric is to establish a list of items that cause one module to be tied to the internal working of another module.

2. One of the metric to measure coupling is Dhama‘s Module coupling Dhama proposed a coupling metric that measures the coupling of an individual component *C* ,which is equal to: 1/(i1 + q612+u1+q2u2+g1+q8g2+w+r) where q6,q7,q8 are constants assigned a value of 2 as a heuristic estimate, and

i1 is the number of in data parameters,

i2 is the number of in control parameters,

u1 is the number of out data parameters, and

u2 is the number of out control parameters.

3. For global coupling: *g1* is the number of global variables used as data, and *g2* is the number of global variables used for control.

4. For environment coupling: *w* is the number of other components called from component *C*, and *r* is the number of components calling component *C*;it has a minimum value of 1.

5. The limitation of Dhama metric considers the effect on coupling of a parameter to be the same as the effect of a global variable,which is a major deviation from the Myers classification [6]scheme. The Dhama metric is an example of an intrinsic coupling metric,which calculates the coupling value of each component individually.

Fenton and Melton Software Metric

[6**]** The Fenton and Melton metric is a direct quantification of the Myers coupling levels, considers all types of interconnections between components to have the same complexities and have the same effects on coupling.

1. Fenton and Melton [16] have proposed the following metric as a measure of coupling between two components x and y :

C(x,y)= i +n/(n+1) where,

n = number of interconnections between x and y , and

i = level of highest (worst) coupling type found between x and y .

2. Coupling Level Modified Definition between components x and y

Content- 5 Common- 4 Control- 3 Stamp -2 Data- 1

No Coupling -0

3. It is an example of an inter-modular coupling metric, which calculates the coupling between each pair of components in the system

Coupling Metric Proposed by Alghamdi S. jarallah

[7]. This metric involves breaking the calculation of coupling into two steps.

1. The first step is to generate a description matrix that captures the factors that affect coupling in a system.

2. The second step is to calculate the coupling between each two components of the system from the description matrix to produce a coupling matrix.

3. Each component of the software system is represented by a row of the description matrix. Components are classes in an object-oriented system, or functions, procedures, and subroutines in a procedural system.

4. Columns of the description matrix represent elements. Elements are methods and instance variables in an object-oriented system, or variables and parameters in a procedural system.

5.There are two limitations with these metrics.One is that an inverse means that the greater the number of situations that are counted , the greater the coupling that this module has with other modules and smaller will be the value of mc.

6. The other issue is that the parameters and calling counts offer potential for problems but do not guarantee that this module is linked to the inner working of the other modules.The use of global variables almost guarantees that this module is tied to the other modules that access the same global variables .

VII. ACKNOWLEDGMENT

I would like to gratefully and sincerely thank my guide Mrs. Jagdeep Kaur and my H.O.D Dr.Latika Singh for their guidance, understanding, patience and provided me with unending encouragement and support. Their mentorship was paramount in providing a well rounded experience for my long-term career goals. They encouraged me to not only grow as a researcher also as an independent thinker.

I would like to thank the Department of Computer science at ITM university, especially for their input, valuable discussions and accessibility.

VIII. CONCLUSION

This paper gives basic review about all the quality based attributes metrics for the software component and their limitations.Describing the evolving nature of metrics form the basic object oriented metrics towards a more complex and justifying metrices considering all quality attributes like reliabilty,complexity.Providing a more consistent approaches and measures that provides more reliable methods to evaluate component quality. However, application of conclusions to real life situations needs further study and empirical support using data from industrial projects to validate these findings and to derive more useful and generalized results. Using data from industry implemented projects will provide a basis to examine the relationship between metric values and several quality attributes of component-based systems.

IX. REFERENCES

[1] Arun Sharma, Rajesh Kumar, P. S. Grover "A Critical Survey of Reusability Aspects for Component-Based Systems" World Academy of Science, Engineering and Technology 2007

[2] Miguel Goulão, "Software Components Evaluation: an Overview Fernando International Journal of Computer Applications" (2829– 516 caparica) Volume 40– No.1, December2010

[3] Navneet Kaur, Ashima Singh"A Complexity Metric for Black Box Components International Journal of Software Computing Engineering" (IJSCE) Volume-3, Issue-2, May 2013

[4] Usha Kumari and Shuchita Upadhyaya "An Interface Complexity Measure for Component-based Software Systems" International Journal of Computer Applications (0975 – 8887) Volume 36– No.1, December 2011.

[5] Tu honglei, sun wei, zhang yanan"The Research on "software metrics and software complexity metrics "International Forum on Computer Science - Technology and Application,2009

[6] DrP.KSuri and Neeraj Garg "Software Reuse Metrics: Measuring Component Independence and its applicability in Software Reuse", IJCSNS International Journal of Computer Science and Network Security, VOL.9 No.5, May 2009

[7] Majdi Abdellatief, Abu Bakar Md Sultan, Abdul Azim Abdul Ghani1, Marzanah A.Jabar,"A mapping study to investigate component-based software system metrics "The Journal of Systems and Software 86 (2013) 587– 603 received 20 May 2011,available online 13 October 2012