International Journal for Rapid Research in Engineering Technology & Applied Science
ISSN (Online): 2455-4723
Volume II Issue XI December-2016

A Technique to Interpret the Software Reusability using Software Metrics

Amit Gupta[1], Dr. Pankaj Dashore[2.]

M-Tech Scholar, Sanghvi Innovative Academy,Indore[1]

Professor, Sanghvi Innovative Academy, Indore2

Amitgupta5997@gmail.com [1], Pankaj.dashore@sims-indore.com [2]

Abstract: **As in today's world, time is playing a crucial role while developing the applications. Reusing something will definitely increase the productivity of the application that is need to be developed. Reusing existing components not only save the time but also the efforts made by the development team and also the use of environment and other resources. It also helps us to reduce the cost of the product. In this paper, we are discussing the way through which we can able to identify whether the object oriented code of applications can be reuse in the future or not. We will also discuss the object oriented matrices that helps us to identify the reusable code.**

I. INTRODUCTION

Reusability is the best direction to increase developing productivity and maintainability of application. One must first search for good tested software component and reusable developed application software by one programmer can be shown useful for others components also. This is proving that code specifics to application requirements can also be reused in developing projects related with same requirements. The main aim of this paper proposed a way for reusable module. A process takes source code (Object Oriented Code) as input that will helped to take the decision approximately that the given code in reusable or not. This tool will help to identify the reusability of any object oriented code, which helps in various organizations and industries that they can choose the most reusable module from existing number of modules. The reusability is one of the most important factors to improve the productivity and quality of the product with a very less cost. This chapter includes the motivation, problem definition, approaches and scope of the report. It describes the basic theme of the report and provides overall idea about research.

II. LITERATURE SURVEY

The measurement of the reusability will help developers to control the current level of the reuse and providing the metrics to identify one of the important quality property reusability. Software complexity metrics reveal internal characteristics of a module, collection of modules, or object-oriented programs [21]. Studies indicate that complex modules cost the most to develop and have the highest rates of failure [10]. McCabe and Halstead developed the two most widely known complexity metrics:

- The McCabe Cyclomatic Complexity metric links the number of logical branches (decisions) in a module to the difficulty of programming [22].McCabe melds a graph theory approach with software engineering: if you represent the logic structure of a module using a flowchart (a graph) andcount the regions of the graph caused by program flow statements (do-while, if-then-else), the numberof regions in the graph corresponds to the complexity of the program. If the number of regions,V(G), exceeds 10, the module may have too many changes of control.
- Halstead's Software Science metrics link studies on human cognitive ability to software complexity [23]. Halstead's approach parses the program or problem statement into tokens and classifies the tokens into operators (verbs, functions, procedures)and operands (nouns, variables, files). Equations based on these tokens give program complexity interms of a variety of indicators including estimated effort, program volume, and size. Not surprisingly, basic software engineering principles address many of the aspects of software that might make software reusable. This particularly applies to those qualities that make software maintainable.

  Another factor affecting whether a programmer will choose to use an existing component in a new situation depends on how quickly the programmers can adapt what the component does and how to use it. Program understanding methods address this problem. These methods attempt to present the important information about a component to the user in a way the user can quickly assess [24]. For example, recognizing that expert programmers organize the important information about a component into mental templates, Lin and Clancy developed a visual template containing this same information. Their study shows that by using a standard layout, a potential reuse can quickly scan the important aspects of a component, such as text descriptions, pseudo code, illustrations, and implementation information [25]. Understanding how good reusable software works not only helps the programmer learn how to write good reusable software, it increases the chances the programmer will use more of what already exists. The discussion of what makes software reusable has taken place for a long time. In 1984 Matsumo to stressed qualities such as generality, definiteness (the degree of clarity or understandability), transferability (portability), and retrievability as the major characteristics leading to the reusability of a component [25].

One reason why we find it so hard to develop reusability metrics comes from the fact that no one completely understands "design for reuse" issues [27]. Given that humans often do not agree on what makes a component reusable, obtaining an equation that quantifies the concept offers a significant challenge. To put it simply, we need to define reusability before we can quantify it.

To illustrate this point, Woodfield, Embley, and Scottconducted an experiment where 51 developers had toassess the reusability of an Abstract Data Type (ADT)in 21 different situations [28]. They found developersuntrained in reuse did poorly; the developers based theirdecisions on unimportant factors such as

size of theADT and ignored important factors such as the effortneeded to modify the ADT. As a result, the studyrecommends developing tools and education that canhelp developers assess components for reuse and suggests reusability metric based on the effort needed tomodify a component as reflected by the number orpercent of operations to add or modify.

The following methods primarily use objective, quantifiable attributes of software as the basis for reusability metric. Most use module-oriented attributes, butthe methods to interpret the attributes vary greatly [10].

According to Prieto-Diaz and Freeman

In their landmark paper,Prieto-Diaz and Freeman identify five program attributesand associated metrics for evaluating reusability [29].Their process model encourages white-box reuse andconsists of finding candidate reusable modules, evaluating each, deciding which module the programmer canmodify the easiest, then adapting the module. In thismodel they identify four module-oriented metrics and afifth metric used to modify the first four. The followinglist shows the five metrics and gives a description ofeach:

1. **Program Size**. Reuse depends on a small modulesize, as indicated by lines of source code.
2. **Program Structure**. Reuse depends on a simpleprogram structure as indicated by fewer links toother modules (low coupling) and low Cyclomatic complexity [29].
3. **Program Documentation**. Reuse depends on excellent documentation as indicated by a subjectiveoverall rating on a scale of 1 to 10.
4. **Programming Language**. Reuse depends on programming language to the extent that it helps toreuse a module written in the same programming language [29]. If a reusable module in the same language does not exist, the degree of similaritybetween the target language and the one used in themodule affects the difficulty of modifying themodule to meet the new requirement.
5. **Reuse Experience**. The experience of the reuser inthe programming language and in the applicationdomain affects the previous metrics because everyprogrammer views a module from their own perspective. For example, programmers will have different views of what makes a "small" module,depending on their background. This fifth metricserves to modify the values of the other metrics [29].

According to Selby

To derive measures of reusability, we must look atinstances where reuse succeeded and try to determinewhy. Selby provides a statistical study of reusabilitycharacteristics of software using data from a NASAsoftware environment [16]. NASA used the productionenvironment to develop ground support software inFORTRAN for controlling unmanned spacecraft. Thestudy provides statistical evidence based on nonparametric analysis-of-variance on the contributions of awide range of code characteristics. The study validatedmost of the findings listed below at the .05 level of confidence, showing that most modules reused withoutmodification [16]:

- Have a smaller size, generally less than 140 sourcestatements.

- Have simple interfaces.
- Have few calls to other modules (low coupling).
- Have more calls to low-level system and utility functions.
- Have fewer input-output parameters.
- Have less human interaction (user interface).
- Have good documentation, as shown by the comment-to-source statement ratio.
- Have experienced few design changes during implementation.
- Took less effort to design and build.
- Have more assignment statements than logic statements per source statement.
- Do not necessarily have low code complexity.
- Do not depend on project size.

According to Chen and Lee

Although Selby's evidence did not find a statistically significant correlation between module complexity and reusability, other studies show such a link. In one example [30], Chen and Lee developed about 130 reusable C++ components and used these components in a controlled experiment to relate the level of reuse in a program to software productivity and quality [30]. In contrast to Selby, who worked with professional programmers, Chen and Lee's experiment involved 19 students who had to design and implement a small database system. The software metrics collected included the Halstead size, program volume, program level, estimated difficulty, and effort. They also included McCabe complexity and the Dunsmore live variable and variable span metrics [14]. They found that the lowerthe values for these complexity metrics, the higher the programmer productivity.

According to Caldiera and Basili

Caldiera and Basili [31] state that basic reusabilityattributes depend on the qualities of correctness, readability, testability, ease of modification, and performance, but they acknowledge we cannot directly measureor predict most of these attributes. Therefore, the paperproposes four candidate measures of reusability basedlargely on the McCabe and Halstead metrics. Thismodule-oriented approach has an advantage in that toolscan automatically calculate all of the four metrics and arrange of values for each [31]:

1. **Halstead's program volume**. A module mustcontain enough function to justify the costs ofretrieving and integrating it, but not so much function as to jeopardize quality.
2. **McCabe's Cyclomatic complexity**. Like Halstead'svolume, the acceptable values for the McCabemetric must balance cost and quality.
3. **Regularity**. Regularity measures the readability andthe non-redundancy of a module implementation bycomparing the actual versus estimated values ofHalstead's two length metrics. A clearly writtenmodule will have an actual Halstead length close toits theoretical Halstead length.
4. **Reuse frequency**. Reuse frequency indicates theproven usefulness of a module and comes from thenumber of static calls to the module.

The paper continues by calculating these four metricsfor software in nine example systems,

and noting thatthe four metrics show a high degree of statistical independence.

III. OVERVIEW OF OBJECT ORIENTED MATRICES

The metrics estimate the OO concepts such as: methods; classes; coupling; and inheritance etc [20]. The metrics focus on internal object structure that reflects the complexity of each individual entity and on external complexity that measures the interactions among entities. The metrics compute computational complexity more or less the efficiency of an algorithm and the use of machine resources, as well as psychosomatic complexity issues that influence the ability of a programmer to alter, build, and realize software and the end user to effectively use the software [20].

The traditional metrics have been widely used, they are well understood by researchers and practitioners, and their relationships to software quality attributes have been validated. Table 2.2 presents an overview of the metrics proposed by the SATC for object-oriented systems. The SATC supports the continued use of traditional metrics, but within the structures and confines of object-oriented systems. The first three metrics in Table 2.2 are examples of how traditional metrics can be applied to the object-oriented structure of methods instead of functions or procedures.

The evaluation of the utility of a metric as a quantitative measure of software quality must relate to the SATC Software Quality Model.

Table 2.2 Metrics for Object-Oriented Systems [20]

METRIC	OBJECT-ORIENTED
CC (Cyclomatic complexity)	Mthod
SIZE (Lines of Code)	Method
COM (Comment percentage)	Method
WMC (Weighted methods per class)	Class/Method
RFC (Response for a class)	Class/Message
LCOM (Lack of cohesion of methods)	Class/Cohesion
CBO (Coupling between objects)	Coupling
DIT (Depth of inheritance tree)	Inheritance
NOC (Number of children)	Inheritance

The object-oriented metric criteria, therefore, are the evaluation of the following areas:

- Reusability/Application specific - Is the design application specific?
- Efficiency of the implementation of the design - Were the constructs efficiently designed?
- Testability/Maintenance - Does the structure enhance testing?
- Understandability/Usability - Does the
- design increase the psychological complexity?
- Complexity - Could the constructs be used more effectively to decrease the architectural complexity?

Traditional Metrics

In an OO system, traditional metrics are normally applied to the methods that comprise the operations of a class. A method is an element of an object that operates on data members in

response to messages and is defined as part of the declaration of a class. Two traditional metrics are discussed here: Cyclomatic complexity and line counts (size).

Metric 1: Cyclomatic Complexity (CC)

The Cyclomatic complexity (McCabe) is used to evaluate the application of an algorithm. A method with a low Cyclomatic complexity means that resolutions are deferred through message passing, not that the methods are not complex. Because of inheritance, CC cannot be used to evaluate the complexity of a class, but for individual methods can be combined with other measures to evaluate the complexity of the class [22].

Metric 2: Line Count - Size/Documentation

All physical lines of code, the number of statements and the number comment lines. However, since size limitations are based on ease of understanding by the developers and maintainers, routines of large size will always pose a higher risk in attributes such as Understandability, Reusability, and Maintainability [26]. This metric can be used to evaluate all the attributes, but most often is a measures Reusability, Understandability, and Maintainability.

Metric 3: Comment Percentage

The comment percentage is calculated by the total number of comments divided by the total lines of code less the number of blank lines. Since comments assist developers and maintainers, this metric is used to evaluate the attributes of Understandability, Reusability, and Maintainability.

Object-Oriented Specific Metrics

Many different metrics have been proposed for object-oriented systems. The object-oriented metrics that were chosen measure principle structures that, if they are improperly designed, negatively affect the design and code quality attributes [23].The selected object-oriented metrics are primarily applied to the concepts of classes, coupling, and inheritance. For some of the object-oriented metrics discussed here, multiple definitions are given. As with traditional metrics, researchers and practitioners have not reached a common definition or counting methodology. In some cases, the counting method for a metric is determined by the software analysis package being used to collect the metrics.

Class

A class is a collection of data members and member functions. A class is used to create an Object. Objects of a class share a common structure and a common behavior by the set of methods. Following class metrics measure the complexity of a class using the class's methods, messages and cohesion.

Methods

In an object-oriented system, traditional metrics are generally applied to the methods that contain the process of a class. A method is an element of a class that operates on data members of the class. Two traditional metrics are discussed here: Cyclomatic complexity and line counts (size).

Metric 4: Weighted Methods per Class (WMC)

The WMC is a count of the methods implemented within a class [22]. The second measurement is difficult to implement since due

to inheritance not all methods are assessable within the class hierarchy. Time and effort required to develop and maintain the class is predicted by the number of methods and the complexity of the methods involved. Classes with large numbers of methods limiting the possibility of reuse as these are more application specific, [21].

Cohesion

Cohesion is the degree to which methods within a class are related to one another and work together to provide well-bounded behavior. Effective object-oriented designs maximize cohesion since it promotes encapsulation [21]. The third class metric investigates cohesion.

Metric 5: Lack of Cohesion of Methods (LCOM)

LCOM calculates the degree of resemblance of methods by variables or attributes. Any assess of separateness of methods helps to determine flaws in the design of classes. There are following ways of measuring cohesion [31]:

1. Estimate for each element in a class to find the percentage of the methods use that data field. Lower the percentages will results greater cohesion of data and methods in the class.
2. If same attributes were operated then methods are more similar. Than count disjoint sets produced from the connection of the sets of attributes used by the methods. Greater the cohesion results good class subdivision. Complexity is increased when lack of cohesion or low cohesion increases complexity, thereby increasing the likelihood of errors during the development process [31]. This metric evaluates the design implementation as well as reusability.

Coupling

Coupling is a computation of the strength of relationship established by a connection from one entity to another. Classes / objects are coupled in three ways:

1. The objects are said to be coupled, then a message is passed between objects.
2. Classes are coupled when methods declared in one class use methods or attributes of the other classes.
3. Inheritance introduces significant tight coupling between parent class and their child class.

The next object-oriented metric measures coupling strength.

Metric 6: Coupling Between Object Classes (CBO)

CBO is a count of the number of other classes to which a class is coupled. It is measured by counting the number of distinct non-inheritance related class hierarchies on which a class depends [20]. The larger the number of couples, the higher the sensitivity to changes in other parts of the design and therefore maintenance is more difficult. Strong coupling complicates a system since a module is harder to understand change or correct by itself if it is inter-related with other modules [21]. Complexity can be reduced by designing systems with the weakest possible coupling between modules.

Inheritance

Using Inheritance we can reuse or derive the properties of base class in one or more than one derived classes. This permits programmers to use again previously defined objects including variables, functions and operators. By reducing

the number of operations and operators, inheritance decreases complexity, but this abstraction of objects can make maintenance and design difficult. Following metrics used to measure the amount of inheritance are the depth and breadth of the inheritance hierarchy.

Metric 7: Depth of Inheritance Tree (DIT)

The depth of a class within the inheritance hierarchy is the maximum length from the class node to the root of the tree and is measured by the number of ancestor classes. The deeper a class is within the hierarchy, the greater the number methods it is likely to inherit making it more complex to predict its behavior. Deeper trees constitute greater design complexity, since more methods and classes are involved, but the greater the potential for reuse of inherited methods. A support metric for DIT is the number of methods inherited (NMI). This metric primarily evaluates reuse but also relates to understandability and testability.

Metric 8: Number of Children (NOC)

The number of children is the number of immediate subclasses subordinate to a class in the hierarchy. It is an indicator of the potential influence a class can have on the design and on the system [21]. The greater the number of children, the greater the likelihood of improper abstraction of the parent and may be a case of misuse of sub-classing. But the greater the number of children, the greater the reuse since inheritance is a form of reuse. If a class has a large number of children, it may require more testing of the methods of that class, thus increase the testing time. NOC, therefore, primarily evaluates testability and design.

IV. CONCLUSION

The purpose of this thesis is to finding the approach and way to calculate reusability of object oriented programs. Reusability is one of the quality attribute and it is of prime important in object oriented software development. As developer's productivity leads to be increased by reusability, it reduces development cost as well as reduces time to market too. The work presented in this thesis can be effectively used to calculate the reusability of any object oriented software module.

V. REFERENCES

[1] Software Reuse Plans BringPaybacks," Computeworld, Vol. 27, KO. 49, pp.73-76. Anthes, Gary I I.,

[2] J.W. Bailey and V.R. Basili. "A s meta-model for oftware development resource expenditures". Proc. Fifth Int. Conf. Software Engineering.Pages107-116. 1981

[3] Norman Fenton. "Software Metrics A Rigorous Approach" .Chapman & Hall, London, 1991

[4] Software Reusability Vol II Applications and Experiences, Addison Wesley, 1989.

[5] http://www.indiawebdevelopers.com/articles/reusability.asp

[6] James M. Bieman "Deriving Measures of Software Reuse in Object Oriented Systems" Springer-Verlag 1992 pp 79-82.

[7] Chris Luer, "Assessing Module Reusability", First International Workshop on Assessment

of Contemporary Modularization techniques (ACoM'07).

[8] Dandashi F., "A Method for Assessing the Reusability of Object-oriented Code Using a Validated Set of Automated Measurements", ACM 2002 pp 997-1003.

[9] Young Lee and Kai H. Chang, "Reusability and Maintainability Metrics for object oriented software", ACM 2002 pp 88 – 94.

[10] Jeffrey S. Poulin "Measuring Software Reusability", IEEE 1994 pp 126- 138.

[11] http://en.wikipedia.org/wiki/Coupling_(computer science)

[12] B. W. Boehm. "Software Engineering Economics" .Prenntice Hall, Englewood Cliffs, NJ, 1981.

[13] Shyam R. Chidamber, Chris F. Kemerer, "A metrics suit for object oriented design",1993

[14] S.D. Conte, H.E. Dunsmore, and V.Y. Shen, "Software Engineering Metrics and Models". Benjamin"Cummings, Menlo Park, California 1986.

[15] M. Burgin. H. K. Lee. N. Debnath, "Software Technological Roles, Usability, and Reusability, Dept. of Math". California Univ., Los Angeles, 2004.

[16] Richard W. Selby. "Quantitative studies of software reuse". In Ted J. Biggersta and Alan J. Perlis, editors,