A Software Cohesion Metrics-Survey

S.Gayathri
Research Scholar
Department of Computer Science
St.Joseph College (Autonomous) Trichy
gayathriselvaraj306@gmail.com

Mr.K.Maheswaran
Assistant Professor
Department of Computer Science
St.Joseph College (Autonomous) Trichy.
mahes161@gmail.com

Abstract - With the evolution of the software development, the scale of the software is increasingly growing to the extent that we cannot hand it easily. Some metrics are proposed to measure the complexity of software in last a few years. Each system has its own complexity which should be measured to develop the quality of the system. Cohesion is an ordinal type of measurement and is usually described as “high cohesion” or “low cohesion”. Modules with high cohesion tend to be preferable, because high cohesion is associated with several desirable traits of software including robustness, reliability, reusability, and understandability. In contrast, low cohesion is associated with undesirable traits such as being difficult to maintain, test, reuse, or even understand. This paper describes the different types of cohesion metrics. After that the metrics are summarized on the basis of importance in decision the complexity and consequently help in better maintainability of the software code, retaining the quality and making it cost effective.

Index Terms — cohesion, McCabe’s Cyclomatic, Halstead’s Complexity, SLOC, SCOM, CCOM, CFCOM

INTRODUCTION

Software Metrics are used to enhance the quality of software since decades. For the better software growth, level plays a very important role for software engineering to make it a true engineering regulation. Hardware as well as software became complex day by day, so manageability is a main concern. Past were the days when only traditional metrics were used to develop the quality and technological decisions concerning software’s. Modern systems are impractical without OO design as object-oriented programming plays a very important role for efficient and efficient software development. Software engineers developed many ways to continue software quality and developed software’s using object-oriented programming to resolve the common problems. Object-oriented design contains all the properties and quality of software that is connected to any large or small project. It is a degree during which a system object can hold a particular quality or characteristics. Object-oriented is an order advance that is accomplished to classify the problem in terms of object and it can current many paybacks on adaptability, reliability, reusability and decomposition of problem into simply understood objects and provide some future alterations. While in principle a module can have perfect cohesion by only consisting of a single, atomic element having a single function, for examples in practice complex tasks are not expressible by a single, simple element. Thus a single-element module has an element that either is too complicated, in order to accomplish task, or is too narrow, and thus tightly coupled to other modules. Thus cohesion is balanced with both unit complexity and coupling.

II. LITERATURE SURVEY

Abreu et al. [1] provides a new classification framework for the TAPROOT. This framework was defined by two independent vectors i.e. category and granularity. Six categories of object oriented metrics are defined as design metrics, complexity metrics, size metrics, quality metrics, productivity metrics and reusability metrics. They also proposed three levels of granularity i.e. software level, class level and method level without providing the empirical/theoretical metrics base.

M. Alshayeb et al. [2] have given two iterative procedures for the pragmatic study of object oriented metrics. They include the short-cycled agile process and the long cycled framework evolution process. In short cycled agile process the prediction of design efforts and source lines of code may be added, modified and deleted by object oriented metrics where as in long cycled framework the same features may not be successfully predicted. It shows that design and implementation may change during development iterations and can predicted by object oriented metrics while it is not true in case of long term development of an established system.

R. D. Neal et al. [3] study the support of the object oriented software metrics and establish that some of the proposed metrics could not be measured as the valid measure for the dimension. They defined a model based on validation measurement theory and proposed ten new metrics as possible methods inheritance (PMI), density of methodological cohesiveness (DMC), proportion of methods inherited by a subclass (PMIS), messages and arguments (MAA), density of abstract classes (DAC), proportion of overriding methods in a subclass (POM), avoidable coupling through global usage (UCGU), degree of coupling between class objects (DCBO),
number of private instance methods (PIM) and strings of message links (SML).

R. Harrison et al. [4] optional a statistical model which is obtained from the logistic regression for identifying threshold values for the Chidamber and kemerer metrics. The process is empirically authenticated on eclipse project. They concluded that the chidamber and kemerer metrics have threshold things at different risk levels. On later releases, the authentication of these thresholds is exposed through the aid of decision trees.

L. H. Ethzkorn [5] shows that the chosen threshold values were more precise than chosen through their instinctive perspectives or data distribution parameters. Object oriented design metrics has also been assign the high level design quality attributes for the object oriented software with the help of hierarchical model. Rosenberg Linda perceive that the software quality also play an important role in the security and financial aspects.

M. Subramanyam et al. [6] proposed some metrics suites and concluded that designs metrics are very important to identify the design aspects of the software and to improve the quality of software for the developers.

Harrison et al. [7] discussed six properties for object oriented design (MOOD) metrics and calculated the object oriented features like inheritance, coupling, encapsulation and polymorphism. In the result they showed that the metrics could be used to present an overall evaluation of the system.

A. Goldberg et al. [8] locate by comparing the object oriented function points with other predictors of lines of code (LOC) that, the object oriented function points can be extended to a significant amount with the aid of a bigger data set. Linear models are the independent variable to measure the conventional object oriented entity or an object oriented function using a cross validation approach.

Edson et al. [9] have presented two metrics namely complexity PLA and extensibility PLA for software Product Line Architecture (PLA) and concluded that the metrics are relevant indicators of complexity and extensibility of PLA by presenting correlation analysis.

Thamburaj et al. [10] have validated a software complexity metric named Cognitive Weighted Attribute Hiding Factor (CWAHF) through correlation analysis and claimed that the metric is robust and accurate in predicting the complexity of the software.

Aloysius et al. [11] have empirically explored the validation of three cognitive complexity metrics such as Attributed Weighted Class Complexity (AWCC), Cognitive Weighted Response for a Class (CWRFC) and Cognitive

Weighted Coupling between Object (CWCBO). The authors have also investigated that the effect of design complexity on maintenance time and found that the experiment is useful in predicting maintenance performance.

SALMAN et al. [12] have introduced three component based complexity metrics namely Total Number of Components (TNC), Average Number of Methods per Component (ANMC) and Total Number if Implemented Components (TNIC). The authors have conducted a case study to detect the power of complexity metrics in predicting integration and maintenance efforts and claimed the component complexity metrics are useful in predicting the integration and maintenance efforts.

Rajnish et al. [13] has presented a complexity metric named Attribute Method Complexity (AMC) to find whether the classes of software is less, moderate or high complex. The author has theoretically evaluated against Weyuker’s properties and experimentally validated against three open source system and the results indicates that the new complexity metric is correlated well with existing complexity metrics and used as predictors of complexity of class.

III. REVIEW OF SOFTWARE QUALITY METRICS

A. Static Code Metrics

Static metrics are resulting from the amount on static analysis of the software code. It is executed without executing any of the code. Static analysis is improved to understand the security problems within the program code and can simply recognize nearly 85% of the defects in the programming code.

1) Source lines of code (SLOC)

Source lines of code are software metric concern to evaluate the amount of a software line up by including the number of lines in the text of the program's source code statements. SLOC is usually to predict the amount of attempt that will be required to expand a program, as well as to approximate programming productivity or maintainability once the software is produced. There are some drawbacks in SLOC metrics that concern the quality of software because of SLOC metric output is used as an input in other Software Estimation techniques. Main types of SLOC measures are: physical SLOC (LOC) and logical SLOC (LLOC). Physical SLOC is the sum count up of lines in the program's source code as one with comment lines.
2) McCabe’s Cyclomatic Complexity Metric
McCabe’s Cyclomatic Complexity Metric is based on the program graph and is defined as [1, 21]:
\[ V(G) = e - n + 2p \]
Where e, n and p represent the number of edges, number of nodes in the graph and number of connected nodes respectively.
This metric gives calculate of independent algorithmic test paths. Extra self-governing paths mean more testing attempt.

3) Halstead’s Complexity Metric
Halstead’s Complexity Metric efforts to approximate the programming attempt. It events complexity by abbreviation the number of operators and operands enclosed in a program. The considerable and countable properties are:
n1 = number of unique or distinct operators appearing in that implementation
n2 = number of unique or distinct operands appearing in that implementation
N1 = total usage of all of the operators appearing in that implementation
N2 = total usage of all of the operands appearing in that implementation
Then the language, n of the program is defined as:
\[ n = n1 + n2 \]
The execution length, N of the program is defined as:
\[ N = N1 + N2 \]
From the length and language, the volume, V of the program is defined as:
\[ V = N \log_2(n) \]
The difficulty, D of the program is defined as:
\[ D = (n1 \times N2) / (2 \times n2) \]
And attempt, E is defined as:
\[ E = D \times V \]

B. Description of Cohesion Metrics
Co-incidental Functional Cohesion Metric (CFCOM) [14]. Sequential Cohesion Metric (SCOM) [15]. Communicational Cohesion Metric (CCOM) [16] are the three existing cohesion metrics that identifies the quotient of cohesion types that exists in the software. The description of each metric is given below.

1) Co-incidental Functional Cohesion Metric (CFCOM)
The proposed Coincidental- Functional Cohesion Metric is a novel metric intended to assess whether the given module or class is coincidentally cohesive or functionally cohesive. As it is discussed earlier a class or module should be functionally cohesive as it reduces the complexity in understanding, comprehending and maintaining the program. On the other hand, a coincident cohesive class or module rather increases the complexity as a whole. CFCOM can be calculated by using the equations as follows:
\[ CFCOM = \frac{\sum_{i=1}^{n} AM_i \cap TAC}{TAC} \]
Here, AM refers to the total number of Attributes defined in method i
TAC refers to the total number of attributes used in method i
SCOM denotes the percentage of attributes used in method i
CFCOM represents that the class is functionally cohesive, and Value 0 of CFCOM represents the class as coincidentally cohesive. Coincidentally cohesive class is an alarm for the programmers to redefine the class into an inseparable unit.

2) Sequential Cohesion Metric (SCOM)
Sequential cohesion refers to the communication between two methods where the output of one serves as an input to the other in a sequence of method calls. Software with sequential cohesion is accepted as it increases the possibility of integration of elements of within the methods of a module. So far, there is no such metric is proposed to calibrate the level of sequential cohesion presented in a module. Hence, as a novel attempt, we have proposed a sequential cohesion metric that evaluates the percentage of sequential cohesion involved in a module.
\[ SCOM = \frac{\sum_{i=1}^{n} m_i \cap m_i+1}{(n-1)\times TAC} \times 100\% \]
‘n’ denotes the total number of methods in the module, ‘mi’ denotes ith method whereas mi\(\cap m_i+1\) is the intersection of attributes of mi and mi+1, and TAC refers to the total number of attributes in a class. SCOM is the percentage of the summation of intersected variables of two consequent methods divided by the possible sequential cohesion. A software possessing 100% of SCOM denotes a strong sequential cohesion and 0% denotes weak sequential cohesion. The implementation of sequential cohesion in software enhances the modularity of software program.

3) Communicational Cohesion Metric (CCOM)
Communicational Cohesion is the grouping up of methods that operate on the same data within a class or module for measuring the integrity of methods. Software with high
quotient of communicational cohesion ensures a good representation of class design that proves the increased integrity of methods within a module or class. Software metric that evaluates the level of communicational cohesion in software modules is still being considered as a thrust area in research which is yet to be focused. Hence, in this paper an attempt is made to propose a communicational cohesion metric (CCOM) for assessing the percentage-wise communicational cohesion that the software modules are designed with. The low level communicational cohesion suggests developers for the modification of software code by increasing the sharing of attributes within the methods of class or modules. The CCOM value of a module is the percentage fraction of sum of intersecting variables between methods by both sums of intersecting and non-intersecting variables between the methods which is denoted using formula.

$$CCOM = \frac{CM}{CM + NIVBM} \times 100\%$$

CM is the communicational measure which is derived by multiplying the sum of intersecting variables between methods by two and can be represented using the formula shown in Equation.

$$CM = 2 \times IVBM$$

IVBM represents the sum of Intersection of Variables between Methods which is denoted using the formula for the computation of IVBM.

$$IVBM = \bigvee_{i=1}^{n} \sum_{j=i+1}^{n} m_i \cap m_j$$

where 'n' denotes the total number of methods in the module, ‘m_i’ and ‘m_j’ denotes ith and jth methods whereas m_i ∩ m_j is the intersection of attributes of m_i and m_j. Finally, NIVBM represents the sum Non-intersecting of Variables Between Methods which is depicted in Equation.

$$NIVBM = \bigvee_{i=1}^{n} \sum_{j=i+1}^{n} ! (m_i \cap m_j)$$

A software module with the CCOM 100% value denotes a strong communicational cohesion and 0% value denotes weak communicational cohesion. The implementation communicational cohesion in software enhances the modularity of software program.

IV. CONCLUSION AND FUTURE WORKS

This paper gives to an improved understanding of the position of the software metrics. This can also give some software quality metrics and the object-oriented metrics, which can describe that “how to evaluate the functionality of the software and How we can develop their characteristics. A method is provided for comparing all the object-oriented software metrics which describe all the methods, attributes are used in software engineering background. The enhance is software development means the capacity was also so high. The increasing meaning being placed software measurement which has to direct and increase amount of research on developing the new software measures. In this paper, we have presented all of the software metrics for object-oriented development. They offered a source for measuring all of the characteristics like size, complexity, performance and value. In rely of some ideas the quality may be increased by added some features like abstraction, polymorphism and inheritance which are inherent in object orientation. This paper provides some help for researchers and practitioners for better understanding and collection of software metrics for their reasons.

REFERENCES


