Comparative analysis of software metrics on the basis of complexity

Shweta
Department of Computer Science and Engineering
Chandigarh University
Gharuan(Mohali(Punjab)),India
sainishweta98@gmail.com

Abstract- Software metrics have been proposed to measure various attributes of the software like – complexity, cohesion,software quality and productivity. Among these "complexity" is considered to be most important attribute. It can be either based on the code of the software or its dependency on other projects. The aim of this paper is to perform comparative analysis of some of the complexity metrics. The paper provides a brief introduction to McCabe complexity metrics,Halstead's complexity metrics,object oriented metrics and Henry-Kafura Information flow metrics along with a comparison between them.

Keywords-complexity; measurements; coupling; cohesion.

I. INTRODUCTION

Software metrics are in use from decades to measure different properties of the software. The goal of using software metrics is to obtain objective and quantifiable measurements which can be beneficially used in budget and schedule planning,quality assurance activities[1],cost estimation and performance optimization.Complexity is an important aspect of a software which is used in predicting essential properties of the software like- reliability, extensibility, understandability, maintainability, portability etc. it measures the number of components of a software project and their interdependencies. Measuring software complexity helps in achieving more predictability in managing software.

Higher level of complexity in software increases the risk of interfering with interactions and so increases the chance of introducing defects while making changes managing software complexity thus helps in lowering the risk of defect-occurrence and in lowering the maintenance costs[14]. Static metrics are derived from measurements of static analysis of code.Object oriented metrics are derived from dynamic analysis of software code[2].Object oriented analysis and design of software provides an efficient way to evaluate and predict quality of the software by decomposing it into easily understandable objects. Object oriented metrics are used to measure and quantify the effectiveness of object oriented analysis techniques in the design of a software.

II. LITERATURE REVIEW

Many measures of software complexity have been proposed.

Thomas J.McCabe in 1976 developed cyclomatic complexity software metric to indicate complexity of a software program. It directly measures the number of independent paths in a program's source code. McCabe proposed a testing strategy called Basis path testing to test each linearly independent path through the software program.

In 1977 Mauric Howard Halstead introduced Halstead complexity metrics as a part of his treatise on establishing an empirical science of software development. Halstead observed that the software metrics should reflect implementation of algorithms in different languages. While being independent of their execution on a specific platform. Halstead's aim was to identify measureable properties of software and relations between them. Halstead's metrics are computed statically from the code.

S.Henry and D.Kafura in 1981 introduced software structure metrics based on information flow which measures software complexity as a function of fan-in and fan-out where fan-in of a procedure is the number of local flows into that procedure plus the number of data structures from which the procedure retrieves information. Fan-out is the number of local flows of the procedure plus the number of data structures it updates.

Two suits of metrics, Chidamber-Kemerer[8] and MOOD[15] are used when the code is analysed for object oriented properties. Shyam Chidamber and Chris Kemerer in 1994 introduced six metrics WMC,DIT, NOC,CBO,RFC and LCOM1. The original suit has later been amended by RFC',LCOM2,LCOM3 and LCOM4 by other authors. The MOOD metrics defined by Fernado Britoe Abreu, are designed to provide a summary of overall quality of an object oriented project. The original MOOD metrics consist of six metrics, the MOOD2 metrics were added later.

III. SOFTWARE COMPLEXITY METRICS

A. Cyclomatic Complexity Metric:

Cyclomatic complexity metric proposed by McCabe is the quantitative measure of logical strength of program.It measures the number of independent paths through a software module.

Mathematically, Cyclomatic Complexity of a software program is defined with reference to control flow graph of the software program which is a direct graph containing basic blocks of the program with directed edges between the basic blocks. Cyclomatic complexity (M) is defined as:

$M=E-N+2P$

Where,

E= number of edges in the graph

N= number of nodes in the graph

P= number of components in the graph

Other metrics used by Mc Cabe for calculating complexity of a software product are:

1. Actual Complexity Metric: It is the measure of number of independent paths traversed during testing
2. Module Design Complexity Metric: It is the complexity of design reduced module and reflects complexity of module's complexity patterns to its immediate subordinate.
3. Essential complexity Metrics: It is a measure of degree to which a module contains unstructured constructs.
4. Pathological complexity Metrics: It is the measure of degree to which a module contains extremely unstructured constructs.
5. Design complexity Metrics: It measures amount of interaction between modules in a system.
6. Integration Complexity Metric: It is the amount of integration testing needed to guard against errors.

Significance of Cyclomatic Complexity Metric:

1. It can be used as a ease of maintenance metric.
2. It is also used as a quality metric, it gives relative complexity of various designs.
3. It can be completed early in lifecycle than the Halstead's metrics.
4. It measures minimum effort and best area of concentration of testing.
5. It guides testing process by limiting program logic during development.
6. It is easy to apply.
7. Well-suited for measuring the number of test cases needed to test the model,

Drawbacks of Cyclomatic Complexity Metric:

1. Cyclomatic Complexity is the measure of program's control complexity and not the data complexity.
2. Same weight is placed on nested and non-nested loops. However, deeply nested conditional structures are difficult to understand than non-nested.
3. It may give a misleading figure with regards to a lot of simple comparisons and decision structure. Whereas fan-in and fan-out metric is more applicable as it can track data flow.

B. Halstead Complexity Metrics:

Halstead complexity Metrics are software metrics introduced by Maurice Halstead in 1977.

Halstead's goal was to identify measurable properties of software and relations between them.Halstead observed the following[8]:

1. Code complexity increases as volume increases.
2. Code complexity increases as program level decreases.

Unlike Mc Cabe complexity metrics, the Halstead metrics do not distinguish between conditional statements and straight line statements. All metrics are determined by mathematical relationships of 4 measures: distinct operators(n1),distinct operands(n2),total operators(N1),total operands(N1).

Table No.-1

Name of measure	**Notation**	**Formulae**	**Description**
Program-Vocabulary	N	N=n1+n2	It measures the breadth of operators and operands appearing in program.
Program-Length	N	N=N1+N2	Measures total usage of all operators and operands appearing in program.
Program- Volume	V	$V=N*\log_2(n)$	Measures the size of information used to specify the program.
Program-Difficulty	D	D=(n1/2)*(N2/n2)	Measures ease of reading the program.
Programming-Effort	E	$E=(n1*N2*N\log_2 n)/(2*n2)$	Measures the mental activity required to reduce a pre-considered algorithm to a program.
Programming-Time	T	T=E/S	S is the Stroud number 4,defined as number of elementary discriminations performed by human brain per second.
Intelligent-Content	I	I=(2*n2/n1*N2)*V	Measure the information content of the program.

Significance of Halstead Complexity metrics:

1. Don’t require in-depth analysis of programming structure.
2. It predicts rate of error.
3. It predicts maintenance effort.
4. It is useful in scheduling and reporting projects.
5. It can be used for any programming language.
6. Halstead complexity metrics is slightly stronger than McCabe’s metrics in time estimation ofsoftware development.

Drawbacks of Halstead Complexity metrics:

1. It depends on complete code.
2. It has little significance in estimation.
3. McCabe model is more suited at design level than it.

C. Fan-in Fan-out complexity metric:

Henry and Kafura identified a form of fan-in fan-out complexity metric which maintains a count of number of data flows from a component plus number of global data structures that the program updates. Data flow count includes updated procedure parameters and procedures called from within a module.

Complexity =Length * $(\text{Fan-in} * \text{Fan-out})^2$

Where,

Fan-in= local flows into a procedure + number of data structures from which procedures retrieve data

Fan-out=local flows from a procedure + number of data structures that procedure updates

Length=number of source statements in a procedure

Significance of Fan-in Fan-out Complexity Metrics:

1. It takes into account data-driven programs.
2. It can be driven prior to coding, during design stage.

Drawback of Fan-in Fan-out Complexity Metrics:

It can give complexity value of zero if a procedure has no external interactions.

D. Object Oriented Metrics:

Traditional metrics such as Cyclomatic Complexity failed to measure Object Oriented concepts such as: classes, encapsulation, inheritance and message passing. In object oriented program inheritance promotes reusability, coupling can be kept minimum to keep complexity of the software controlled[10].

To characterize the "object-orientedness" of a software design,some traditional metrics used are: SLOC, Cyclomatic Complexity,Comment Percentage, number of procedures and defects[18] and a suite of object oriented metrics proposed by Shyam Chidamber and Chris Kemerer[8] and MOOD (Metrics for Object Oriented Design)suit[6] are mainly used. Some of the object oriented metrics are described in the table that follows.

Table No.-2

Name of metric	**Measure**	**Calculation**	**Use**
Weighted Method per Class (WMC)	Measures some aspect of scope of methods making up class.	Summation of weighted methods of class.	Higher WMC values correlate with increased development,testing and maintenance efforts[16].
Depth of inheritance Tree (DIT)	Inheritance upon which a class was built[12].	Maximum of number of levels in each of class's inheritance path.	DIT count correspond with greater error density and lower quality.
Number of children(NOC)	How widely a class is reused to build other classes.	Count of classes that are directly derived from a specified class.	Larger NOC counts point to greater reuse of class. High NOCs may also flag a misuse of sub classing.
Response for a class (RFC)	Overall complexity of calling hierarchy of methods making up a class.	Counts of methods of a class and methods that they directly call.	Larger RFC count correlates with increased testing requirements.
Coupling between Object Classes(CBO)	How dependent a class is on other.	Count of external classes whose members are called, set, read or used as type by members of current class[16].	Excessive coupling limits availability of class for reuse and also leads to greater testing and maintenance efforts[17].
Lack of Cohesion in Method(LCOM)	How widely member variables are used for sharing member functions.	Counts of pairs of class members that do not access any of same class variables reduced by number of pairs that do[11].	A higher LMOC denotes low cohesion. This correlates with weaker encapsulation.Hierachical clustering can improve cohesion of classes [5].
Attributed Hiding Factor(AHF)	Measures invisibility of attributes in classes.	Percentage of total classes from which attribute isn't visible. AHF=sum of invisibilities of all attributes of all classes/total number of attributes in the project.	Higher value of AHF is desired for better encapsulation. High AHF correlates to high maintenance and independent code change ability
Method Hiding Factor(MHF)	Measures invisibility of methods in class.	MHF=sum of invisibilities of all methods defined in all classes/total number of methods defined in the project.	High MHF value depicts high encapsulation. It correlates to high maintenance of the software.

Significance of Object Oriented Metrics:

Used as an early indicator of some externally visible attributes such as reliability, maintainability and fault-proneness.

1. They appeared from the nature of the OO approach.
2. NOC and RFC metrics give some idea as to budgeting for testing that class.
3. Coupling metrics are good predictor of fault proneness.[3]
4. Careful use of inheritance can lead to better design.[3]

Drawbacks of Object Oriented Metrics:

1. Internal metrics are of a little value unless there is evidence that they are related to external values.
2. Measuring complexity of a class is subject to bias.
3. They cannot give a good size and effort estimation of software .
4. These metrics seem only to bring the design phase into play, and does not provide adequate coverage in terms of planning.
5. The OO metrics have no conversion rules to lines of code metrics[13].
6. The OO metrics have no conversion rules to function point metrics[13].
7. The OO metrics lack automation[13].
8. The OO metrics are difficult to enumerate[13].
9. CK metrics suit is not able to distinguish between a class with high cohesion and a class with medium cohesion[9].

Table No.-3 (Table of comparison between complexity metrics)

	Cyclomatic Complexity Metric	**Halstead Complexity Metric**	**Fan-in Fan-out Metric**	**Object Oriented Metric**
Complexity defined in relation to	Decision structure of organisation	Magnitude of computation	Information flow structure of program	Modular structure of program
Key Feature	Estimating code complexity, identifying most complex module.	Calculating program effort in man month	Locating modules that contribute the highest maintenance effort	Evaluate key features of Object Oriented design like encapsulation, polymorphism etc
Computation time in lifecycle	Can be computed early in lifecycle.	Depends on completed code.	Can be computed early in lifecycle.	Can be computed early in lifecycle.

CONCLUSION

A comparative analysis of different complexity metrics is done to depict their significance and drawbacks in different contexts. It has been concluded that Mc Cabe Cyclomatic Complexity can be completed early in lifecycle than Halstead’s metric as such more beneficial for predicting software quality. Fan-in Fan-out metric is more applicable for simple comparison and decision structures than Mc Cabe Cyclomatic Complexity Metrics due to its data tracking capability. Time prediction by Halstead’s metrics is slightly more stronger than that by Mc Cabe Complexity Metric. For measuring object oriented features like localisation, encapsulation, inheritance, polymorphism, specialisation of classes etc, the Object Oriented metrics are used. Various external software attributes like reusability and maintenance etc can be predicted by using object oriented metrics.

REFERENCES

[1] Mathur, Kirti, and Amber Jain."A Comparative Survey of Software Quality Metrics."International Journal of Research(2013).

[2] Chawla, Sonal, and Gagandeep Kaur."Comparative Study of the Software Metrics for the complexity and Maintainability of Software Development."International Journal of Advanced Computer Science & Applications 4.9 (2013).

[3] Aggarwal, K. K., et al. "Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: a replicated case study." Software Process: Improvement and Practice 14.1 (2009): 39-62.

[4] Kaur, Kiranjit, and Sami Anand."A Maintainability Estimation Model and Metrics for Object-Oriented Design (MOOD)."International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) 2.5 (2013): pp-1841.

[5] Sadaoui, Lazhar, MouradBadri, and Linda Badri."Improving Class Cohesion Measurement: Towards a Novel Approach Using Hierarchical Clustering."Journal of Software Engineering & Applications 5.7 (2012).

[6] Sharma, Aman Kumar, ArvindKalia, and Hardeep Singh. "Metrics Identification for Measuring Object Oriented Software Quality."

[7] Malathi, S., and S. Sridhar."ANALYSIS OF SIZE METRICS AND EFFORT PERFORMANCE CRITERION IN SOFTWARE COST ESTIMATION."Indian Journal of Computer Science and Engineering 3.1 (2012): 24-31.

[8] Chidamber, Shyam R., David P. Darcy, and Chris F. Kemerer. "Managerial use of metrics for object-oriented software: An exploratory analysis." Software Engineering, IEEE Transactions on 24.8 (1998): 629-639.

[9] Salem, Ahmed M., and Abrar A. Qureshi. "Analysis of Inconsistencies in Object Oriented Metrics." Journal of Software Engineering & Applications 4.2 (2011).

[10] Chawla, Sonia, and RajenderNath."Evaluating Inheritance and Coupling Metrics."

[11] Aggarwal, K. K., et al. "Empirical Study of Object-Oriented Metrics." Journal of Object Technology 5.8 (2006): 149-173.

[12] Rajnish, Kumar, Arbind Kumar Choudhary, and Anand Mohan Agrawal."INHERITANCE METRICS FOR OBJECT-ORIENTED DESIGN."International Journal of Computer Science & Information Technology 2.6 (2010).

[13] Rawat, Mrinal Singh, Arpita Mittal, and Sanjay Kumar Dubey. "Survey on Impact of Software Metrics on Software Quality."International Journal of Advanced Computer Science & Applications 3.1 (2012).

[14] Debbarma, MrinalKanti, et al. "A Review and Analysis of Software Complexity Metrics in Structural Testing." International Journal of Computer and Communication Engineering 2 (2013): 129-133.

[15] MansiAggarwal,Vinit Kumar Verma,HarshVardhan Mishra "An Analytical Study of Object Oriented Metrics " International Journal of Engineering Trends and Technology (IJETT) 6.2 (2013)

[16] Prabhjot Kaur "Study of Various Class Oriented Metrics"International Journal of Computers and Distributed Systems.2.1(2012)

[17] Gulia, Preeti, and Rajender Singh Chillar. "Design based Object-Oriented Metrics to Measure Coupling and Cohesion." International Journal of Engineering Science & Technology 3.12 (2011).

[18] VasudhaDixit,RajeevVishwkarma "Static and Dynamic Coupling and Cohesion Measures in Object Oriented Programming" International Journal of Engineering Research (IJER) 2.7 (2013):472-477