Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Technology 6 (2012) 420 – 427

Procedia Technology

2nd International Conference on Communication, Computing & Security [ICCCS-2012]

Effectiveness of software metrics for object-oriented system

Yeresime Suresh*, Jayadeep Pati, Santanu Ku Rath

Department of Computer Science and Engineering
National Institute of Technology, Rourkela, India

Abstract

Measurement is a buzzword applied to each and every field of engineering for the development of hardware or firmware products, and software engineering is nowhere an exception. Measurement in software engineering can be best described in terms of metrics, as a quantitative measure of degree to which a system, a component or process possesses a given attribute. The urge for software metrics allows the analysts, designers, coders, testers and the managers to conceive the software development processes and assess the system. The objective of this paper is to evaluate software like ATM using available subset of metrics from traditional and object-oriented methodology. The traditional metrics such as cyclomatic complexity, size and comment percentage are used to compute the software complexity. This paper also analyses a widely used subset of object-oriented metrics such as the Chidamber and Kemerer metric suite to compute the system reliability. The metric values are evaluated for a real life application, which helps us to know the complexity and the reliability of the ATM software.

Keywords: Cyclomatic Complexity; Basis path; Object-Oriented; Chidamber and Kemerer metrics; Coupling;

1. Introduction

Software metrics are the decisive factors to measure the quality of the software product. Metrics are used to assess the system complexity, track the progress, and evaluate the effectiveness of the software. Metrics have great importance in designing appropriate ways to measure the system (Weyuker, 1988., Chidamber & Kemerer, 1994). There are several metrics available such as Fenton's (Fenton & Pfleeger, 1977), Shepperd's (Shepperd & Ince, 1993) to assess the software product and have been classified into Traditional and

* Corresponding author. Tel.: +91 8658828161
E-mail address: suresh.vec04@gmail.com

doi:10.1016/j.protcy.2012.10.050

Object-oriented metrics. This paper intends to explore the available subset of traditional metrics (cyclomatic complexity (McCabe, 1976), size (Sharble & Cohen, 1993) and comment percentage (Sharble & Cohen, 1993)) and object-oriented metrics (such as Chidamber and Kemerer metric suite (Chidamber & Kemerer, 1994), R. C. Martin's metric (Martin, 1994)). On the basis of theoretical and practical evaluation, the metrics are used to estimate system reliability, testing effort and complexity of the system.

The rest of the paper is organized as follows: Section 2 presents Introduction to various traditional and object-oriented metrics. Section 3 describes the approach used in mining and evaluating software metrics for a real life application. Section 4 represents interpretation of the software metrics based on the values obtained in Section 3. Section 5 represents application of metrics and Section 6 concludes the paper.

2. Metrics

2.1 *Traditional Metrics*

There are many traditional metrics that have been used since 1976 (McCabe, 1976). A subset of metrics such as the Cyclomatic Complexity, Size and Comment Percentage are the possible indicators to gauge the complexity. Cyclomatic complexity is one of the best indicators of system reliability.

Metric 1: McCabe's metric suite

Cyclomatic Complexity (McCabe, 1976) is used to evaluate the complexity of an algorithm in a method. A method which has a low value does not mean that the method is not complex but it only means that the decisions are delayed through message passing. Higher values indicate the complexity of the application or it highlights the presence of huge number of alternate flows in the application (Redin, et al., April 2008). It is an indication of number of test cases needed to test the method completely. This suite consists of additional set of traditional metrics such as Lines of Code (Kaur, et al., 2006), Method Lines of Code (Cheah, et al., 2008), Source Lines of Code (Allan J. Albrecht & Gaffney, 1983) and Nested Block Depth (Cheah, et al., 2008).

Metric 2: Size

Size of the class is used to estimate the ease with which the maintainers understand the code written by developers (Sharble & Cohen, 1993). There are alternate ways to find the size of the class viz., physical lines of code, the number of statements, the number of blank lines and the number of comment lines. The lines of code only estimate the volume of code; size can only be used to compare the software programs that are written in same language.

Metrics 3: Comment Percentage

The line count used to compute the size metrics can be expanded to include a count of the number of comments, both on-line (with code) and stand-alone (Sharble & Cohen, 1993). Comment percentage is given by the ratio of total number of comments to the total lines of code minus the number of blank lines. Hence this metric boosts the developers and maintainers to easily evaluate the attributes of understandability, reusability and maintainability.

2.2 *Object-Oriented Metrics*

The need for the present day software development is in the usage of object-oriented practices. The major benefit of using Object-oriented metrics is to improve quality as they are best indicator of system reliability,

work performed at the project level. The available metric suites are Chidamber and Kemerer metric suite (Chidamber & Kemerer, 1994), the MOOD metric suite (Abreu, 1995), Lorenz and Kidd metric suite (Lorenz & Kidd, 1994), R. C. Martins metric suite (Martin, 1994) etc.; but the most commonly used set of object-oriented metrics is the Chidamber and Kemerer metric suite. It is often called as CK metric suite.

1) Chidamber and Kemmerer metric suite

a) Weighted Method per Class (WMC)

WMC is included in the suite because it has been proven to be useful in predicting maintenance and testing effort. This method has its own pitfalls, but it can be combined with other metrics such as cyclomatic complexity to contribute vital information about a class complexity. Consider a Class C_1, with method $M_1,\ldots,M_n$ that is defined in a class. Let $c_1 \ldots c_n$ be the complexity of the methods, then the WMC is given as follows: WMC = $\sum c_i$ for i = 1 to n. Where c_i is the complexity of the methods associated in the i^{th} class. If all the method complexities are considered to be unity, then the value of WMC will be n, the number of methods.

b) Depth of Inheritance Tree (DIT)

DIT can be defined as the maximum length of a path from a class to the root class in the inheritance tree. The deeper a class will be in the hierarchy the greater the possibility to inherit more number of methods making it more complex to predict its behaviour.

c) Number of Children (NOC)

NOC is termed as the number of immediate sub-classes subordinate to a class in the class hierarchy. It is an indicator of the potential influence a class can have on the design or the system.

d) Coupling between Objects (CBO)

CBO for a class is a count of the number of other classes to which it is coupled. A measure of coupling is useful to determine how complex the testing of various parts in the design is likely to be. The higher the inter object class coupling, more rigorous testing needs to be done.

e) Response for a Class (RFC)

The response set of class is “a set of methods that can potentially be executed in response to a message received by an object of that class”. As RFC increases, the testing effort increases because the test sequence becomes abundant.

f) Lack of Cohesion in Methods (LCOM)

Consider a Class C_1 with n methods $M_1, M_2,\ldots..,M_n$. Let $\{ I_j \}$ = set of instance variables used by method M_i. There are ‘n’ such sets $\{ I_1 \},\ldots.\{ I_n \}$. Let $A = \{ (I_i, I_j) \mid I_i \cap I_j = \emptyset \}$ and $B = \{ (I_i, I_j) \mid I_i \cap I_j = \emptyset \}$, if all n sets $\{ I_1 \},\ldots\{ I_n \}$ are $\emptyset$ then let $A = \emptyset$. LCOM = $|A| - |B|$, if $|A| > |B| = 0$. If LCOM is high, methods may be coupled to other methods via attributes, which increases the complexity of class design. In a conventional way high values of LCOM imply that the class might be better designed by breaking it into two or more separate classes.

2) R. C. Martins metric suite

The most widely known package metrics was presented by R. C. Martin (Martin, 1994), this suite consists of

a) Efferent Coupling (Ce)

Efferent coupling between packages is the measure of total number of external classes coupled with classes of a package because of outgoing coupling. All classes are counted only once. If the classes in the same package do not use any external class linkage, then the value of Ce is zero.

b) Afferent Coupling (Ca)

Afferent coupling between packages (Ca) measures the total number of external classes coupled to classes of a package because of incoming coupling. All classes are counted only once. If the package does not contain any class or external classes do not use any of the classes of the package, then the value of Ca is zero.

c) Instability

Instability (I) is the ratio of efferent coupling (Ce) to total coupling (Ce + Ca) i.e., I = Ce / (Ce + Ca). This metric implies the package's adaptability to change. The range of this metric varies from 0 to 1, with I = 0 signifying an absolutely stable package and I = 1 signifying an absolutely instable package.

d) Abstractness

Abstractness is the comparison of the number of abstract classes (and interfaces) to the total number of classes in the evaluated package. The range for this metric is 0 to 1, with A = 0 signifies an absolute concrete package and A = 1 signifies an absolute abstract package.

e) Normalized Distance from Main Sequence (D_n)

Normalized Distance from Main Sequence is the perpendicular distance of a package from the idealized line A+ I = 1. This metric is a measure of abstractness and stability. Ideal packages are either absolutely abstract and stable (x = 0, y = 1) or absolutely concrete and unstable (x = 1, y = 0). The range for this metric is 0 to 1, with D = 0 represents a package that is coincident with the main sequence and D = 1 represents a package which is far away from the main sequence (Martin, 1994).

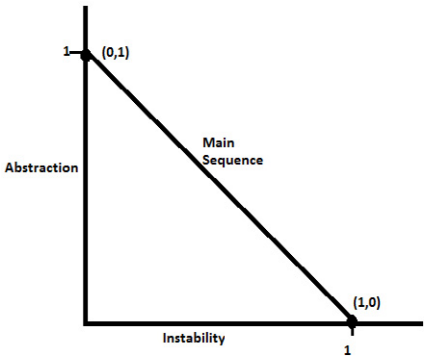

Figure 1: The A-I Graph.

The A-I Graph i.e., Figure-1, depicts the line drawn from (0, 1) to (1, 0), and it represents the categories with the fact that abstractness is balanced with stability. Since its similarity to a graph used in astronomy it is called as Main Sequence (Martin, 1994).

3. Mining and Evaluation of Metrics for a Real Life Application

The ATM example from Michael R Blaha, James R Rumbaugh (Blaha & Rumbaugh, 2005) is coded in java which consists of 1080 lines of code as a single package with eight classes in the system. The metric values were extracted using Eclipse. The table-1, 2, 3 tabulates the metric values for the three sets viz., , McCabe metric suite, Chidamber and Kemerer metric suite and R. C. Martins respectively briefed in section-2.

Table 1: McCabe Metrics for ATM example

McCabe's Metric	McCabe's Complexity: (Maximum)	Method Lines of Code	Total Lines of Code	Nested Block Depth
DB.java	17	101	163	4
DB2.java	15	92	216	4
Transfer.java	19	116	163	4
MainMenu.java	14	107	214	4
StandardCash.java	16	58	102	4
Information.java	3	27	47	2
Cash.java	2	94	113	3
CardTransaction.java	1	50	62	3

Table 2: CK Metric values for ATM example

CK Metric	WMC	LCOM	DIT	NOC
DB.java	21	0.939	6	0
DB2.java	20	0.789	6	0
Transfer.java	20	0.737	6	0
MainMenu.java	18	0.86	6	0
StandardCash.java	11	0.625	6	0
Information.java	4	0.5	6	0
Cash.java	3	0.929	6	0
CardTransaction.java	1	0	6	0

Table 3: R. C. Martrin metric values for ATM example

R.C. Martin's	Ca	Ce	I	A	Dn
ATM Project	0	1	1	0	0

4. Interpretation of Software Metrics for Real Life Application

Software metrics as mentioned in the earlier sections help us to assess the complexity, fault proneness or the system reliability of software.

Based on a set of metric evaluation guidelines i.e., the numerical thresholds specified on each metric as given by Rosenberg (Rosenberg & R.Stapko, 1999) is used to assess the software system. Rosenberg specified certain limitations on metrics in his interpretation of Object-Oriented metrics. For instance, it can be known that cyclomatic complexity value greater than ten, leads to difficulty in understanding the code and testing effort increases. Similarly a DIT values greater than six, indicates good re-use of classes but increased overhead in testing, and becomes difficult in assessing the behaviour of the class. Hence this subset of Object-oriented metrics can be used to predict the system reliability and traditional metrics to assess the complexity. Some of the inferences that can be drawn from the obtained metric values, which help in predicting the system stability, are listed below

i. If WMC value is high, it hinders the re-use of the class and to avoid this we need to reduce the number of methods or their complexity.
ii. If the DIT is more than six (Sinaalto, 2009), it increases the design complexity, and hence we think of reducing the inheritance usage while coding.
iii. If the NOC is high, dilution of abstraction of classes takes place, and hence the same reducing NOC as in case of DIT applies here.
iv. If the CBO is high, it complicates the testing process; hence we should reduce the number of collaborators in the class.
v. If the RFC of a class is high, it again results in more amount of testing effort i.e., increase in the test sequence and over design complexity of the system increases. Hence we should reduce the number of operations that will be executed in response to a message received.
vi. Similarly if the LCOM in methods is high then it increases the class design complexity. So we should reduce the LCOM values by breaking down the class into two or more separate classes i.e., writing the code in modular fashion.

5. Specific Application of Metrics

This section describes the applications of three metric suites in predicting the complexity, system reliability and design principles in Object-oriented approach.

a) Cyclomatic complexity

Cyclomatic complexity can be used to assess the system complexity and to estimate the number of test cases required to achieve maximum code coverage. Cyclomatic complexity can be calculated from the number of edges and number of nodes in a Control Flow Graph (CFG). It is represented as V (G) and is calculated by V (G) = E-N+2, where E and N represent the number of edges and number of nodes in a control flow graph respectively. Cyclomatic complexity is rooted in the program control structure and can be calculated based on the amount of conditional statements present in the source code. The V (G) value indicates all the possible paths of execution in the program and these set of paths are known as Basis path. Based on the number of paths achieved, test cases are generated manually or through automated process so as to cover all the executable paths which will ensure that maximum code coverage is achieved.

In the ATM example we have taken a sample class StandardCash.java. The control flow graph is extracted using Eclipse IDE. From McCabe's inference the cyclomatic complexity is limited to ten. So, maximum ten basis paths are possible. The basis paths are extracted from the generated DOT file of control flow graph. Now these act as the test paths for the module. Then appropriate test data is generated to test the module thoroughly. Figure-2 represents the CFG Block diagram of the actionPerformed() function for StandardCash.java. The basis paths for the figure-2 are shown in Table-4. Here A^* represents abnormal termination, which occurs due to an exception.

The necessity of this metric is important because it provides the tester a target to aim for or else testing

cannot cease until all faults have been removed because exhaustive testing is not possible. The main advantage of cyclomatic complexity is it does not suffer from redundancies. Hence cyclomatic complexity is one of the most suitable metric for generation of test cases for both feasible and non-feasible paths.

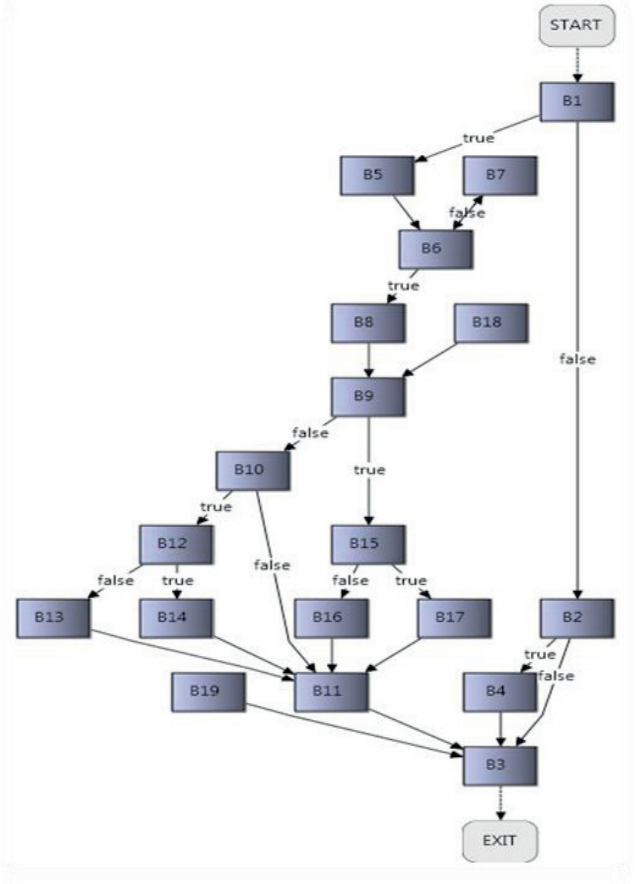

Figure 2 : Control Flow Graph(CFG) for actionPerformed() of class StandardCash.java

Table 4: Set of Basis paths for CFG in figure-2.

Sl. No	Basis Paths
1.	START-B1-B5-B6-B8-B9-B15-B17-B11-B3-EXIT.
2.	START-B1-B5-B6-B8-B9-B10-B12-B14-B11-B3- EXIT.
3.	START-B1-B5-B6-B8-B9-B15-B16-B11-B3-EXIT.
4.	START-B1-B5-B6-B8-B9 B10-B11-B3-EXIT.
5.	START-B1-B5-B6-B8-B9-B10-B12-B13-B11-B3-EXIT.
6.	START-B1-B2-B4-B3-EXIT.
7.	START-B1-B2-B3-EXIT.
8.	START-B1-B5-B6-B7-A*.
9.	START-B1-B5-B6-B8-B9-A*.
10.	B19-EXIT.

b) Chidamber and Kemerer metric suite

CK metrics are the best indicators of fault proneness. As from inference of the metrics mentioned in section 4, it will be convenient to predict the system reliability more accurately. This suite helps the coders and testers to take better design decision and also to estimate the testing effort. Except LOCM all the metrics are effective in predicting fault proneness of a system.

There are many regression analysis methods which are widely used to predict an unknown variable based on one or more known variables. This regression analysis will help in discovering the relationships between the values of the metrics and the number of bugs found in the classes. The DIT metrics is untrustworthy (Ferenc, et al., 2005) and NOC cannot be used at all for fault-proneness. So the LCOM, LOC and WMC (which uses Cyclomatic Complexity) are the best indicators of system reliability.

c) R. C. Martin's metric suite

The metrics in this suite compares the project to the ideal models of dependency and abstraction. These measures depend on certain standards which might not be applicable for every system and as a result can only reduce the quality of the system. The examination of this metric suite is necessary, since the suite is widely used for agile development, which is becoming more and more popular. If applications believed to be well-made perform according to Martin's metric, the metric might capture some principles of good design. It also gives a clear description of stable software.

6. Conclusion

After investigation into the validity of these evaluated software metrics, it has led us to conclude that many Traditional and Object-oriented metrics exist and provide priceless information to the developers, coders and testers. This evaluation of the three selected metric suite i.e., McCabe's metric suite, Chidamber and Kemerer metric suite and Robert C. Martin's metric suite and the results obtained thus provide a basis for quality assessment to software professionals to fetch the requisite amount of information about those metric suites which can predict faults while developing the metric-quality software products, estimate the number of test cases required to achieve maximum code coverage and also to assess the complexity of the system using the Traditional and Object-Oriented approach.

This paper not only illustrates about a huge benefit in utilization of software metrics for Software Quality Assessment but also describes about evaluating the system effectively through these metrics.

This study can be further extended by applying the use of traditional- cyclomatic complexity which states about feasible and non-feasible paths to generate automated test cases using soft computing techniques such genetic algorithms, simulated annealing etc.

Another future prospect of study will be about applying neural network for predicting system reliability using Chidamber and Kemerer metric suite.

References

Abreu, F. B. e., 1995. The MOOD metric suite.

Albrecht, A. J., John, J. R. & Gaffney, E., 1983. Software function, source lines of code and development effort prediction: A software science validation. IEEE Transactions on Software Engineering, November, 9(6), pp. 639-648.

Blaha, M. R. & Rumbaugh, J. R., 2005. Object-oriented modeling and design with UML. Second ed. Pearson.

Cheah, W. P., Kim, B. K., Park, H. & Liu, Y., 2008. Predict software failure-prone by learning bayesian network. International Journal of Advance Science and Technology, 1(1), pp. 35-42.

Chidamber, S. R. & Kemerer, C. F., 1994. A metric suite for object-oriented design. IEEE Transactions on Software Engineering, 20(6), pp. 476-493.

Fenton, N. & Pfleeger, S. L., 1977. A Rigorous and Practical Approach. 2nd ed. International Thomson Computer Press.

Ferenc, R., Siket, I. & Gyimothy, T., 2005. Empirical validation of object-oriented metrics on open source software fault prediction. IEEE Transactions on Software Engineering, 31(10), pp. 897-910.

Kaur, K., Kaur, A. & Malhotra, R., 2006. Alternative methods to rank the impact of object-oriented metrics in fault prediction modeling using neural networks. Proceedings of the World Academy of Science, Engineering and Technology, May, 13(2), p. 99.

Lorenz, M. & Kidd, J., 1994. Object-oriented software metrics - A Practical Guide. Prentice Hall.

Martin, R. C., 1994. [Online]
Available at: http://www.objectmentor.com/rescources/articles/oodmetric.pdf

McCabe, T. J., 1976. A complexity measure. IEEE Transactions on Software Engineering, 2(4), pp. 308-320.

Redin, R. M. et al., April 2008. Software quality metrics and their impact on embedded software. Budapest, pp. 68-77.

Rosenberg, L. H. & R.Stapko, 1999. Applying object oriented metrics. Boca Raton, Florida, USA.

Sharble, R. C & Cohen, S. S., 1993. The object-oriented brewery-a comparison of two object-oriented development methods. IEEE Transactions on Software Engineering, 18(2), pp. 60-73.

Shepperd, M. J. & Ince, D., 1993. Derivation and Validation of Software Metrics. Clarendon Press, Oxford, UK.

Sinaalto, M., 2009. [Online]
Available at: http://www.agile-itea.org/public/deliverables/ITEA-AGILE-D5.2.10_v1.0.pdf

Weyuker, E. J., 1988. Evaluating software complexity measures. IEEE Transactions on Software Engineering, April, 14(9), pp. 1357-1365.