Evaluating Software Complexity Based on Decision Coverage

Mustafa AL-HAJJAJI, Izzat ALSMADI, Samer SAMARAH
Department of Computer Information Systems, Yarmouk University, Irbed, Jordan
mustapha5_51@yahoo.com, ialsmadi@yu.edu.jo, samers@yu.edu.jo

It is becoming increasingly difficult to ignore the complexity of software products. Software metrics are proposed to help show indications for quality, size, complexity, etc. of software products. In this paper, software metrics related to complexity are developed and evaluated. A dataset of many open source projects is built to assess the value of the developed metrics. Comparisons and correlations are conducted among the different tested projects. A classification is proposed to classify software code into different levels of complexity. The results showed that measuring the complexity of software products based on decision coverage gives a significant indicator of degree of complexity of those software products. However, such indicator is not exclusive as there are many other complexity indicators that can be measured in software products. In addition, we conducted a comparison among several available metric tools that can collect software complexity metrics. Results among those different tools were not consistent. Such comparison shows the need to have a unified standard for measuring and collecting complexity attributes.

Keywords: *Complexity, Software Metrics, Decision Coverage, Software Quality, Testing*

1 Introduction

In recent years, the software products are getting more complex. Producing a software with all its functionalities while at the same time having high quality attribute is a serious challenge. Improving software testing and measurements can help in findings software bugs early and hence reduce their impact [1]. However, it is very difficult to test every aspect or attribute in the software, especially when the software application is very huge and has many branches. There are several metrics that have been developed to help developers and testers in their development process in order to guarantee the correctness of tasks and improving the maintainability of the software [2], [3], [4], [5]. Cyclomatic complexity is one of metrics that is used to measure the complexity of a program by measuring the number of linearly independent paths through the source code [6]. Cyclomatic complexity is computed using the Control Flow Graph (CFG). In CFG, there are nodes and directed edges. The nodes refer to commands or decisions in the program and each edge connects two nodes (i.e. commands) when the second command can be executed after the first one.

In this paper, measuring the complexity will be based on decision coverage. Decision coverage is a metric that measures the possible branches that are followed by a flow control structure [7]. A decision is a program point in which the control flow has two or more alternative branches [8]. Decision coverage is the percentage of the decision outcomes that have been tested or visited by test cases relative to the overall decisions [7]. The decision coverage metric will be added to the existing metrics in SWMetrics tool developed by one of the paper's authors [13]. SWMetrics computes many metrics such as: Line of Code (LOC), Statement Line of Code (SLOC), Cyclomatic complexity and math counts. The objective of decision coverage testing is to show all the decisions within a component that have been executed at least once. This is usually a software complexity indicator where more decisions in a program mean more complexity. The remainder of this paper is structured as follows: Section 2 presents a background of software metrics. Section 3 discusses some of the metrics that proposed to measure some features of software, especially complexity. Some tools that can calculate software metrics have also been discussed in this section. Section 4 presents the setup of our experiments. Section 5 describes the experimental results. Section 6 includes a conclusion or summary of the work presented in this paper.

2 Background

Software metrics provide a numerical data related to development, operation, and maintenance of the software product, project, process, etc. The metrics help the developers evaluate the attributes of software in an objective manner by giving them numerical figures to compare different software products with each other. Moreover, the metrics can help to get better management results. Generally, software metrics can be classified into three main classes: process, product, and project metrics [9]. Process metrics are related to enhancing the software development process. For example, the response time of fixing a problem. Product metrics are related to the characteristics of a product like complexity and size. Project metrics explain the project features and execution. This may include: the number of software developers, cost, and timetable [9]. In this research, the focus will be on the product metrics, particularity, complexity related metrics.
The community of software engineering has not consented upon a set of metrics. As a result, many developers have come up with diverse ways to measure the software attributes. One of the most important metrics is the complexity which is supposed to be an indicator of: correctness, clarity, and effectiveness of the software. In addition, it can provide a good estimation for the cost, efforts, the number of faults, cost of testing, etc. Several metrics have been proposed to measure the complexity of a program. Examples of software product complexity metrics include: Cyclomatic complexity, depth of inheritance, information flow (fan-in. fan-out], etc. The aim of this paper is to measure the complexity of the software based on Decision Coverage (DC). Furthermore, the relation between the complexity and other metrics will be studied. In addition, we can decide the best tool that can measure the complexity.

3 Literature Review

A considerable amount of literature has been published on software metrics; and obviously measuring the complexity of the program is one of these metrics. In this section, first of all, the metrics which are related to the complexity metric will be reviewed. The tools that have been developed to measure the complexity of programs will then be discussed.

3.1 Software Metrics

This section provides a definition of metrics that can help in measuring the complexity of programs. They have been selected based on the least common denominator. The first one is the metric of: **Lines of Code (LOC)**. As the name indicates, LOC metric shows how many lines of source code are in the application, namespace, class or method. Four aspects have been considered to deal with LOC metric: blank lines, comment lines, data declarations, lines that include several instructions [12].Another accepted line of code metric is the one implemented by [13], which is called Non-Commented LOC (NCLOC). In this implementation, comment and blank lines are eliminated. Thus, the metric will give the right value of the
size of the program, because the blank and comment lines are not used by the software. LOC can be used practically as follows: check the size of code module, and estimate the effort in development and maintenance process.
Coupling Between Objects (CBO): CBO is the number of other classes that are coupled with a specific class [11]. According to [4], the CBO can be defined as the measure of the strength of the established by a connection from one unit to another.
Depth of Inheritance Tree (DIT): DIT is the maximum inheritance path from the class to the root class [11]. When a child class inherits from one parent, it’s called “single inheritance”. And when a child class inherits from more than one parent, it’s called “multiple inheritances” which is more complex than a single inheritance. Inheritance increases the efficiency by reducing the redundancy

[15]. In contrast, the deeper hierarchy inheritance, the harder it is to understand the code.

Number of Children (NOC): NOC is the number of immediate subclasses subordinated to a class hierarchy [11]. The number of children indicates the level of reuse in a system. Moreover, it indicates the testing level is required. If a class has a large number of subclasses, it is probably an improper abstraction of the parent class. A system has a lot of child classes, will be hard to understand.

Lack of Cohesion in Methods (LCOM): According to [14], cohesion of a class is defined by how closely the local method is related to the local variable. A high LCOM value could indicate that the design of the class is poor and it may be a good idea to split the class into two or more sub-classes [11]. The authors in [16] redefined the LCOM using a graph and they consider it as the number of connected components of a graph.

Response For a Class (RFC): RFC is the number of methods which can be executed in response to a message received by an object of a class [11].

Weight Methods per Class (WMC): WMC is the sum of weights for the methods of a class [11].

3.2 Tools

In this section, many software metric tools will be discussed. Each one of these tools calculates several possible metrics.

C and C++ Code Counter (CCCC): CCCC is an open source tool, developed as a testing ground to generate reports on various metrics such as LOC, McCabe's complexity and metrics proposed by Chidamber and Kemerer [11]. It deals with C++ and Java files[17].

Analyst4j: Analyst4j is a commercial tool works based on Eclipse platform as well as stand-alone [18]. It provides Java code search by using software metrics. Furthermore, it provides an environment to analyze code quality metrics and give visualization for metrics using graph/charts. [18]

Dependency Finder: Dependency finder is a free open source tool. This tool is used in analyzing compiled Java code. Basically, it is a dependency program that extracts dependency graphs and mines them for useful information [19].

Java Coding Standard Checker (JCSC): JCSC is a powerful tool used to examine a source code against a definable coding standard and potential bad code [20]. JCSC supports several metrics such as non-commenting source statement (NCSS) and Cyclomatic Complexity Number (CNN).

Chidamber and Kemerer Java Metrics (CKJM): CKJM is an open source command line tool that calculates also CK metrics from Java programs. The metrics proposed by CK are WMC, DIT, NOC, RFC, CBO, and LCOM[21].

OOMeter: OOMeter is an experimental metric tool developed by the authors in [22]. It is used to measure the quality attributes of Java and C# source code and UML models. OOMeter supports many metrics such as WMC, DIT, NOC, CBO, RFC, RFC, and LOC.

Understand for Java: Understand tool for Java is a commercial tool used to calculate several code metrics [24].Many metrics are supported by this tool such as: Cyclomatic complexity, max inheritance, weighted methods per class, number of instance methods, and class coupling [25].

SWMetrics tool: SWMetrics is a tool used for a specific company as part of a master project. Besides LOC abdCyclomatic complexity, the tool collects metrics related to complexity that include: SLOC, Maximum nesting, Cyclomatic complexity, and Math counts [10].

4 Methodology

The methodology consists of three main parts: implementation of the proposed metric, classes and projects classification, and the comparison among tools. The first one includes the implementation of the complexity metric based on decision coverage using C# language. The implementation will be added to the SWMetrics tool. Later one, data mining classification methods will be used based on the output of SWMetrics tool, especially on the attributes of: the decision coverage, LOC and decision coverage with LOC.

1) Decision Coverage Metric Implementation Phase

SWMetrics can calculate Cyclomatic complexity, max nesting, and the number of operations. These metrics can indicate the degree of the complexity of given software. In our paper, we will add a new measure which can help further measuring the complexity. The parser of the tool, which is written in C# language, will then be modified. Furthermore, the new measure will appear in the interface of the SWMetrics tool. In this section, we explain the code modification of the tool.
Decision Coverage evaluation is based on the number of decisions in the code. There are certain keywords in programming languages that are indicators of decisions. These words include: if, for, while, switch, select case, do, try, catch, finally, etc.
In the implementation phase, the decision coverage is calculated with exception handling and without exception handling. We did it on purpose, because some opinions such as those mentioned in [23] assumed that a single level of exception will improve the software by making it more robust without affecting the degree of the complexity of that system. If the exception handling has more than one level or more than one level of exception in a class, the complexity of software will increase. In this study, we will assume that the iteration loop increases the complexity of the software by 3. Many authors proposed 3 as a weight for the loop [26]. They also increased the weight of the sequence statement by 1. However, this is ignored in our study because the focus in this paper is on code decisions only. The others keywords such as: if, else, switch, case, try, catch, finally, etc. increments complexity by 1.

2) Classification phase

In this phase, three columns will be added to SWMetrics. First, we will classify the classes, based on LOC, into three categories (low, medium, and high). Second, the classes will be classified, based on DC, into three categories (low, medium, and high). Finally, we will classify the classes, based on LOC and DC, into five categories (very low, low, medium, high, and very high). As we classified the classes, we will do the same for the software products. We will classify the set of software products based on LOC, DC, and LOC with DC. The classification process will be done automatically by implementing the classification process in SWMetrics.

3) Tools comparison and the case study

We will conduct a comparison study between SWMetrics and several selected software metric tools.
More than 70 open software projects are used to conduct the comparison and the analysis study. Most of metric-tools selected support Java based programs. SWMetrics tool is developed to deal with several types of code including Java, C++, and C#. However, since most of those tools used for comparison can only evaluate Java codes, the case study projects were selected from Java open source references (e.g. sourceforge).
The first criteria were to collect the software metrics tool that calculate metrics and can indicate the degree of the complexity of software. 14 different metric tools were selected. The majority of the software metric tools evaluated support metrics for Java programs.
As we mentioned in the literature review, we select the metrics based on the “least common denominator”. A large list of metrics was created. We had to refine these metrics; since there are many metrics that have different names in different tools but they are related to the same topic. Table 1 shows the metrics and the tools that are used in the evaluation study.

Table 1. Tools and metrics used in evaluation

Tools	Metrics							
Name	LOC	CBO	# math op	Max nesting	Cy. Complex.	DIT	DC	RFC
Analyst4j	x	x		x	x	x		x
SWMetrics	x		x	x	x		x	
CK Java Metrics	x	x			x	x		x
Understand for Java	x	x			x	x		
Eclipse Metrics	x	x			x	x		

Table 2 shows some of the general characteristics of the metric tools used in the evaluation.

Table 2. Characteristics of the evaluated software metric tools

Tool name / Characteristics	Analyst4j	CK Java Metrics	Understand	SWMetrics	Eclipse Metrics
GUI	x		x	x	x
Command line		X		x	
Support many language			x	x	
Stand-alone	x		x	x	
Plug-ins	x				x

5 Experiments and evaluation

In this paper, two experiments were conducted to evaluate the proposed metrics. The first experiment is designed to test and evaluate the modified SWMetrics tool. The second experiment will evaluate the SWMetrics in comparison with the tools mentioned in Table 1. The experiments were carried on a personal computer (PC) satisfying the minimum requirements for all tools.

5.1 Decision Coverage Metric Evaluation:

One of the most important limitations of software metrics is the absence of standards that define how to measure and evaluate the results. . Each tool has its own definition for the given metrics. For this reason the implementation of each metric will be also different; depending on the tool. To this end, the evaluation of SW Metrics tool will be manual. The evaluation will not be for all metrics, since, most of the metrics have been evaluated in [13]. We will focus on evaluating the two new metrics: decision coverage with Exceptional Handling (EH) and decision coverage without EH. As we described before, three columns will be added to the SWMetrics. These columns are used to give an indication for the degree of the complexity for all the classes. The first column metric measures the degree of complexity for each class in the software system based on LOC. These classes are classified into three categories: small, medium, and high. Table 3 shows the categories and their conditions.

Table 3. Metrics nominal classification based on LOC

Category	**Condition**
Small	When LOC less than 50
Medium	When LOC greater than or equal 50 and also LOC less than 100
High	When LOC greater than or equal 100

The same classification process is also applied to the other metrics. Table 4 shows DC nominal classification and Table 5 shows DC-LOC nominal classification.

Table 4. DC nominal classification

Category	Condition
Small	When DC less than 10
Medium	When DC greater than or equal 10 and less than 50
High	When DC greater than 50

Table 5 is divided into 5 classes rather than 3: very low, low, medium, high, and very high.

Table 5. DC-LOC nominal classification

Category	Condition
Very low	If the complexity based on LOC is “small” and the complexity based on DC is ”small”.
Low	If the complexity based on LOC is “small” and the complexity based on DC is “medium” or vice versa.
Medium	If the complexity based on LOC is “medium” and also the complexity based on DC is “medium”.
High	If the complexity based on LOC is “medium” and the complexity based on DC is “high” or vice versa.
Very high	If the complexity based on LOC is “high” and the complexity based on DC is “high”.

5.2 Evaluation of the Results

In the first stage, experiments are conducted to evaluate the accuracy of the developed algorithms. Accuracy is compared with the manual count of the decisions versus those collected from the tool automatically.

Table 6. Metric evaluation with EH for classes

Category Name	Project Name	Class Name	Language	Expected Output	Actual Output	Precision	Complexity based on DC
Communication	AsyncWcfLib-V2.1	Test1.TwoClients	C#	13	13	100%	Medium
Communication	AsyncWcfLib-V2.1	FrmClient	C#	35	35	100%	Medium
Communication	AsyncWcfLib-V2.1	Router	C#	53	53	100%	High
Communication	Commore-windows	List	C++	70	70	100%	high
Desktop environment	bsaf-1.9RC4	TaskTest	Java	5	5	100%	small
Education	pigale-1.3.12	free-glut_geometry	C++	140	140	100%	High
Enterprise	AMB New Generation Data Empowerment	BI	C#	132	132	100%	High
Financial	JKtoCheck_0.4	AccCheck	Java	8	8	100%	small
Game	ows_0.5_win	Tactics	C++	33	33	100%	Medium
Network	Euler	Datatype	Java	84	84	100%	High

Table 6 shows results that are related to the DC metrics with EH with additional information such as: project name, category name, programming language of software system, class name, expected output, actual output, and precision. The expected outputs have been calculated manually. The actual output column values represent the output from SWMetrics. Precision = actual output / expected output.Table7 contains the same type of information which is found in Table 6 while this information related to Decision Coverage without EH metric.

Table 7. DC Metric evaluation without EH for Classes

Category Name	Project Name	Class Name	Language	Expected Output	Actual Output	Precision	Complexity based on DC without EH
Communication	AsyncWcfLib-V2.1	Test1.2Clients	C#	13	13	100%	Medium
Communication	AsyncWcfLib-V2.1	FrmClient	C#	31	31	100%	Medium
Communication	AsyncWcfLib-V2.1	Router	C#	51	51	100%	High
Communication	Commore-windows	List	C++	70	70	100%	High
Desktop environment	bsaf-1.9RC4	TaskTest	Java	3	3	100%	small
Education	pigale-1.3.12	free-glut_geometry	C++	140	140	100%	High
Enterprise	AMB N.Generation Data Emp.	BI	C#	89	89	100%	High
Financial	JKtoCheck_0.4	AccCheck	Java	8	8	100%	small
Game	ows_0.5_win	Tactics	C++	33	33	100%	Medium
Network	Euler	Datatype	Java	82	82	100%	high

While financial and system administrative projects showed an overall higher complexity as domains relative to other domains, however, this is not consistent across all projects of those domains. For example, in the database domain, we can observe MethodLib software with high complexity and IBMDatabaseProject with very low complexity. Financial applications fall largely in complex and high complex categories (based on decision complexity).

5.3 Tools comparison

The results of the tools comparison showed that there are differences in calculated metrics across tools although they have the same metric names (e.g. LOC, Cyclomatic complexity, etc.). The differences can be large or small depending on the size of the software product.

6 Conclusions

Software metrics have an important indirect role in increasing the quality of software systems. Through those measurements, they can ensure that the developed product is within regulations. In this paper, a software metric tool is extended to cover evaluating complexity metrics relation to decision coverage. It is expected to correlate the occurrence of many decisions in a particular code with increasing its complexity. In this paper, we also tried to see the combinational factor of the effect of the decision coverage (DC) with the traditional size metric (LOC).

We found that the DC can be used as a significant indicator for the software complexity. This of course does not mean that it is the only factor that can impact the software complexity. Through the evaluation and comparison of the developed tool with several software metric tools, it is noticed that there is a need to have a unified standard for: defining, developing and analyzing software metrics. Despite the fact that formulas of some metrics (LOC, e.g. CK and Halstead metrics) are widely known, nonetheless, the investigations showed that the actual implementation and results of those metrics can vary.

References

[1] A. Dominguez and R.J. Debouk. "Feature Interaction as a Source of Risk in Complex Software-intensive Systems". *The 25th International System Safety Conference*, Baltimore, 2007, pp: 13-17.

[2] W. Li and S. Henry. "Maintenance Metrics for the Object Oriented Paradigm". *In proc. Software Metrics Symposium*, 1993, pp. 52-60

[3] B. Henderson-Sellers, L. Constantine and I. Graham. "Coupling and Cohesion (Towards a Valid Metrics Suite for Object Oriented Analysis and Design)". *In proc. Object Oriented Systems*, Vol. 3, 1996, pp. 143-158

[4] L.C. Gray." A Coupling complexity metric suite for predicting software quality", *Master thesis*, California Polytechnic State University, 2008.

[5] D. Liu and S. Xu."New Quality Metrics for Object-Oriented Programs". *In proc. SNPD '07 of the Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, Vol. III, 2007, pp. 870 – 875.

[6]K. Kaur, K. Minhas, N. Mehan and N. Kakkar."Static and Dynamic Complexity Analysis of Software Metrics". *In proc.Empirical Software Engineering*, Volume: 56, Issue: V, 2009, pp: 159-161.

[7] A. Dupuyand A. Leveson, "An empirical evaluation of the MC/DC coverage criterion on the HETE-2 satellite software",*In proc. Digital Aviation Systems Conference*, Philadelphia, 2000.

[8] http://shailajakiran-testing.blogspot.com/2007/11/statement-coverage-decision-coverage.html. (2010).

[9] S. Kan, *Metrics and Models in Software Quality Engineering*, 2nd edition, Addison-Wesley Professional, 2002.

[10] I. Alsmadi and K. Magel."Open source evolution analysis", *In proc. of the 22nd IEEE International Conference on Software Maintenance*, pp. 276 – 278, 2006.

[11] S. Chidamber and C. Kemerer. "A Metrics Suite for Object-Oriented Design",*In procIEEE Transactions on Software Engineering*, Vol. 20, No. 6, pp. 476-493, 1994.

[12] N. Fenton and S. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*,2nd edition, 1997.

[13] I. Alsmadi. "Software metrics: toward building proxy models", *Master's thesis*, North Dakota State University of Agriculture and Applied Science, 2006.

[14] M. Andersson and P. Vestergren. "Object-Oriented Design Quality Metrics". *Uppsala Master's Theses in Computer Science276*, 2004.

[15] G. Alkadi and I. Alkadi, " Application of a Revised DIT Metric to Redesign an OO Design", *In proc. Journal of Object Technology*, Vol. 2, No. 3, pp. 127-134, 2004.

[16] M. Hitzand B. Montazeri. "Chidamber and Kemerer's metric suite: A Measurement Theory Perspective," *In proc. IEEE Transactions on Software Engineering*, Vol. 4, pp. 267-271, 1996.

[17] http://cccc.sourceforge.net/. Accessed on 5/11/2010

[18] http://www.eclipse4you.com/?q=en/eclipse_plugins/analyst4j/. Accessed on 5/11/2010

[19] http://depfind.sourceforge.net/. Accessed on 6/11/2010

[20] http://jcsc.sourceforge.net/. Accessed on 10/11/2010

[21]http://www.spinellis.gr/sw/ckjm/. Accessed on 10/11/2010

[22] J. Alghamdi, R. Rufai and S. Khan "OOMeter: A Software Quality Assurance Tool",*In proc. Ninth European Conference on Software Maintenance and Reengineering (CSMR'05)*, pp.190-191, 2005.

[23] R. Chatterjee, B. Ryder,"Complexity of Points-To Analysis of Java in the Presence of Exceptions", *In proc. IEEE Transaction on Software Eng.*,Vol. 27, No. 6,pp. 481-512, 2001.

[24] R. Lincke, J. Lundberg and L. Welf. "Comparing software metrics tools",*In proc. International Symposium on Software Testing and Analysis*, pp. 131-142, 2008.

[25] http://www.scitools.com/ features/metrics.php. Accessed on 5/12/2010

[26] S. Misra and A.K. Misra. "Evaluation and Comparison of Cognitive Complexity Measure", *In proc. ACM SIGSOFT Software Engineering Notes*, Vol. 32, No. 2, pp.1–5, 2007.

Mustafa Zaid AL-HAJJAJI obtained his Master degree in computer information system from Yarmouk University (Jordan) in June 2011. He received his B.Sc. in computer information system from Mutah University (Jordan) in June 2008. His research interests include: software engineering, software metrics, and wireless sensor networks.

Izzat Mahmoud ALSMADI is an assistant professor in the department of computer information systems at Yarmouk University in Jordan. He obtained his master and Ph.D degree in software engineering from NDSU (USA). He had B.sc degree in telecommunication engineering from Mutah university in Jordan. Before joining Yarmouk University he worked for several years in several companies and institutions in Jordan, USA and UAE.

Samer SAMARAH received the Ph.D. degree in computer science from the University of Ottawa, Ottawa, ON, Canada, in 2008. He is currently an Assistant Professor with the Department of Computer Information System, Yarmouk University, Irbed, Jordan, and a Research Associate with the School of Information Technology and Engineering, University of Ottawa. His research interests are Software Engineering, wireless networks, wireless ad hoc and sensor networks, and data mining for both distributed systems and wireless sensor networks.