Celal Bayar University Journal of Science
Volume 13, Issue 4, p 863-871

Determining the Tested Classes with Software Metrics

Fatih Yücalar[1]*, Emin Borandağ[1]

[1]Department of Software Engineering, Faculty of Technology, Celal Bayar University, Manisa, Turkey

*fatih.yucalar@cbu.edu.tr; emin.borandag@cbu.edu.tr
*Corresponding author

Recieved: 26 July 2017
Accepted: 31 October 2017
DOI: 10.18466/cbayarfbe.330995

Abstract

Early detection and correction of errors appearing in software projects reduces the risk of exceeding the estimated time and cost. An efficient and effective test plan should be implemented to detect potential errors as early as possible. In the earlier phases, codes can be analyzed by efficiently employing software metric and insight can be gained about error susceptibility and measures can be taken if necessary. It is possible to classify software metric according to the time of collecting data, information used in the measurement, type and interval of the data generated. Considering software metric depending on the type and interval of the data generated, object-oriented software metric is widely used in the literature. There are three main metric sets used for software projects that are developed as object-oriented. These are Chidamber & Kemerer, MOOD and QMOOD metric sets. In this study, an approach for identifying the classes that should primarily be tested has been developed by using the object-oriented software metric. Then, this approach is applied for selected versions of the project developed. According to the results obtained, the correct determination rate of sum of the metrics method, which was developed to identify the classes that should primarily be tested, is ranged between 55% and 68%. In the random selection method, which was used to make comparisons, the correct determination rate for identifying the classes that should primarily be tested is ranged between 9.23% and 11.05%. In the results obtained using sum of the metrics method, a significant rate of improvement is observed compared to the random selection method.

Keywords —Software Fault Prediction, Software Quality and Assuarance, Software Metrics, Software Testing.

1. Introduction

Today, in the software world, approaches to software quality assurance and testing and object-oriented software metrics have become the most studied subjects. This is mainly because the errors that occur in softwares and the increase in the cost of correcting these errors have increased with the growth of software projects. The software projects consists of requirements analysis, design, coding, testing and product creation. These software development steps follow each other in succession. Especially software testings should be considered at all stages of the software life-cycle. V process model is a software development model in which the testing process takes place at every stage of the software and corresponds to each stage in a test step [1]. This model applies the software testing process at each stage and from the beginning of the development of the project. With these planned testings, the quality of the software is checked. Potential errors that may occur in the source code or in design are determined at the earliest stage possible. It is very important to early detect the potential errors that may occur in software projects in order to be able to consider the risk of exceeding the estimated cost and duration. Examining software errors to minimize these risks is extremely crucial. The time that it takes to detect the errors in software directly affects the time and cost of maintenance. While the cost of correcting the errors found after software becomes a product is quite high, the tests made to detect errors in advance or during the development of the software considerably reduce these costs. Software maintenance costs constitute more than half of software development cost. In addition, softwares always undergo changes for different reasons during their development process such as adding new features, improving quality, and correcting errors. After each modification, software testing is required to protect the integrity and stability of the software. That's why the software testing is the most time-consuming and resource-intensive activity of the software life-cycle [2, 3].

Software developers, testers and managers need to identify critical parts of the software that must be tested first in order to be able to use the time and resources more effectively. Testing all the units and all the functionality in a software project is very costly; sometimes this is even impossible in terms of time and cost.

The main contribution of the study is clearly to identify the classes that should primarily be tested has been developed by using the object-oriented software metric. This study consists of six chapters in total. In the second chapter, the studies taking part in literature are mentioned. In the third chapter, software metrics and quality concepts are discussed and the measurements used for the improvement of software quality have been introduced. The fourth chapter tells about the steps taken in the approach, metrics and project information. The fifth chapter addresses experimental studies while the last chapter includes conclusions and evaluations obtained.

2. Related Works

In his study, Xiaowei [4] used a metric system in order to prove the correctness of the metric used during maintenance stage. He compared the metric frequently used during maintenance stage. He used an equation to measure the workload and defined many variables in this equation. He has created a metric system for software maintenance. These metrics were adapted to software organizations for software maintenance and showed that they could be used to solve quality management problems.

In the study of Kaur et al. [5] it is argued that higher quality software are possible and and customer satisfaction can be increased by revealing high-error points before project coding and giving these points to experienced employees. In this study, they used the "Fuzzy C-Means Clustering Alghorithms" to predict the modules that are error-prone or not error-prone.

In their study, Raymond et al. [6] studied code readability and examined the relationship between code readability and software quality. In terms of readability, by collecting data from 120 people, they revealed the relationship between a simple set of regional code features and human ideas.

In their study, Ogasawara et al. [7] argue that many software quality metrics have been proposed to understand software products and processes over the last decade, and that these quality measurements are used to manage software quality in real projects.

In their study, Chaumun et al. [8] have introduced a change effect model, thinking that system design can change. It is important to find out which parts of the software are affected by the changes made in the software, to ensure that the software operates in a stable and correct way after the change. Therefore, the main focus of their work is finding out how the system will meet a change.

In the study carried out by Lee et al. [9], open-source "JFreeChart" software was analyzed to understand the "fan-in/out" dependency and compliance metric and software development behavior. They have developed a software called "JamTool" to extract software metric and quality features. They examined the relationship between the increase in the number of classes in the software analyzed in the experimental study and the dependency and compliance metric. In addition, they investigated the change in the addiction and compliance metric of the added and deleted classes. On the other hand, in their study, Kastro and Bener [10] have proposed an artificial neural network based methodology that takes into account changes in older versions to estimate the number of errors in the new version of the software. They stated that changes to the writing process could be an added, an algorithm modification or debugging. In addition, considering the volume changes in code size, they tried to correctly estimate the number of errors that could occur in the new version.

In the study of Li and Leung [11], they have developed an unchecked learning model in order to find error proneness. The main idea in their work is that the components in the same set of metrics have similar error proneness. The data set they use contains error records and source codes for 12 different projects [12] from NASA. The obtained data were pretreated and the data were normalized and the metric equals were calculated. They aimed to find error proneness patterns with the model they created by using the "Nearest Neighbor" algorithm.

When these studies are evaluated, the software metrics are used in the development process of the software projects. Especially, software metrics are taken into consideration to coding, integration and testing phases which will provide a significant advantage in completing the software project. A literature review shows a lack of studies focusing on software metrics related to sum of the metrics method.

3. Software Quality

Total quality management is ensuring service quality requirements for human, work, product/service which are used to meet customer needs through a systematic approach and with the contribution of all employees [13]. It has been seen that the quality of industrial products has increased with the use of total quality management on production lines. Total quality management aims to improve processes. A process is a sequence of interrelated events that starts with an input and that generates a specific output with the added value to the input. The basic principle of total quality management is to produce quality products from well-defined processes. It is aimed to obtain products with better quality by providing continuous improvement of existing processes. Quality standards such as ISO 12207 [14], ISO 15504 (SPICE) [15] and CMMI [16], which are based on total quality principles, are used by software companies.

The basis of continuous improvement is the process planning, supervision, output and performance measurement and process evaluation. Many byproducts can be created in a process. When we look at the software development process, the customer requirements analysis document, software project management plan, design document, code, test scenario and results can be considered as output. Measurements obtained during the process are used to evaluate and improve the process. The measurement, according to the defined rules, is expressing the results of the observations with numerical symbols by determining whether entities-objects have a certain attribute or not, and if they have; what are their degree of possession [17].

Software measurement is one of the most common ways of monitoring software quality [18]. The measurement of the software allows the determination of factors such as project size, effort, cost, time spent and quality. Numeric data related to the software can be obtained by using the metric in the software projects. This data provides project managers with critical information about the development of the software. Project managers can identify the risks that needs precaution and conduct studies on them. They can also update project plans using this data. In addition, they learn about the result table that will appear in software maintenance and testing. If necessary, they make improvement plans/studies. In brief, the measurement results obtained in software projects help project managers to manage projects more effectively and with less risk.

3.1 The Concept of Software Quality

Software quality is described through a number of attributes and aims to identify their capability to achieve the software requirements [19]. In addition, Crosby defines software quality as meeting client requirements with zero error. However, quality software needs to be completed within anticipated budget and time. The software that meets customer requirements but exceeds its budget or is too long to be accepted is not considered to have good quality. Some techniques have been developed to measure the quality of the software. These techniques are based on the principle of forming an opinion about the software quality by measuring some features of the software by digitizing them and interpreting/evaluating the result.

3.2 Software Metrics

Software metrics aim to display the quality of source code and give understanding to it quantitatively [20]. It is possible to classify the software metric according to the collection time of the information, the information used in the metric, the type and range of the data they produce. According to the time of collection of information, the software measurements are divided into two: statically and dynamically. Static measurements use the information obtained without running the software. Dynamic measurements use the information obtained during software operation.

When the software metrics are classified according to the information used in the measurement, some metric only look at the parameter access, while others consider all data access of the methods. If we classify the software metric according to the type and range of the data they produce, the metric can be either real or integer values in the range of $[0, +\infty]$, real values in the range of bounds, real numbers in the bounded range. There are three sets of metrics commonly used within object-based software metric. These are Chidamber & Kemerer, MOOD and QMOOD sets of metrics [21].

3.2.1 Chidamber & Kemerer (CK) set of metric

There are 6 basic metric in the Chidamber & Kemer set of metric and the definitions are given below [21].

- ***Weighted Methods per Class (WMC):*** The sum of the complexities of all methods in a class. It helps to estimate how much time will be spent developing and maintaining the class.
- ***Depth of Inheritance Tree (DIT):*** The distance to the root of the hereditary tree. For underived classes, this metric measurement is 0. In the case of multiple inheritances, the metric measurement is the distance to the farthest root.
- ***Number of Children (NOC):*** The number of subclasses derived directly from a class. The classes that have multiple subclasses need more testing. This metrix can be used to determine the budget to be spent testing the relevant class.
- ***Coupling Between Object Classes (CBO):*** The number of classes that a class is dependent on. If methods or qualities within a class are used in another class, and there is no participation between classes, there is dependency between these two classes. High dependency means difficulty in care. It also reduces re-usability. Because high dependency would require more testing, it also increases testing costs.
- ***Response For a Class (RFC):*** The number of all methods that can be triggered when the methods of an object in the class are called. Calling a large number of methods in a message means increasing the cost of the test, making debugging more difficult.
- ***Lack of Cohesion in Methods (LCOM):*** If P is the cluster of method pairs which do not share any common quality variable and if Q is the cluster of method

pair which share common quality variable and if {|P|>|Q| then it is |P||Q| or else 0}. If the compatibility of the methods is low, it is necessary to separate the subcomponents of the class. Since low compatibility increases complexity, there is an increased risk of errors in the development phase. Errors in the design of classes can also be predicted by using these metric.

3.2.2 MOOD set of metric

MOOD (Metrics for Object Oriented Design) set of metric deals with mechanisms such as message transfer, encapsulation, inheritance, polymorphism of the method based on the object [22].

- ***Method Hiding Factor (MHF):*** The ratio of callable methods in all classes to all methods, regardless of the inherent methods. This metric measures the visibility of the class.
- ***Attribute Hiding Factor (AHF):*** The ratio of accessible qualities in all classes to all qualification, regardless of qualifications that come with inheritance. This metric measures the visibility of the class, too.
- ***Method Inheritance Factor (MIF):*** The ratio of the number of methods that come with inheritance in all classes to the number of all methods.
- ***Attribute Inheritance Factor (AI):*** The ratio of the number of qualifications that come with inheritance in all classes to all qualifications.
- ***Polymorphism Factor (PF):*** The ratio of different multiform situations of class C to the most likely multiform situations.
- ***Coupling Factor (CF):*** The ratio of the number of dependencies between classes to the number of dependencies that can occur, regardless of the use of dependencies that come with inheritance.

3.2.3 QMOOD set of metric

QMOOD (Quality Model for Object Oriented Design) set of metric is defined to calculate the total quality index of the software [23]. It is a four-level hierarchical model:

- ***Software Quality Attributes:*** QMOOD software quality attributes are functionality, efficiency, intelligibility, extensibility, re-usability and flexibility.
- ***Object-Oriented Software Properties:*** These metrics deal with inheritance, encapsulation, polymorphism, abstraction, dependency, messaging, hierarchy, software size, and complexity.
- ***Object-Oriented Software Metrics:*** There are eleven metrics in the QMOOD metric set.
- ***Object-Oriented Software Components:*** Object-oriented software components include attributes, methods, classes, relationships, and class hierarchy.

4. Approaches

This section explains the stages of the study done and the relation between the metrics used and the maladjustment of the classes. In addition, it discusses the approach adopted for classifying errors.

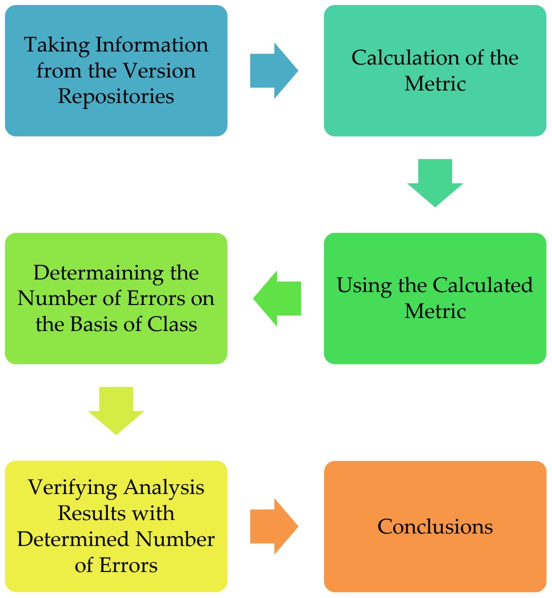
Taking Information from the Version Repositories
Calculation of the Metric
Using the Calculated Metric
Determaining the Number of Errors on the Basis of Class
Verifying Analysis Results with Determined Number of Errors
Conclusions

Figure 1. The stages of the study

As seen in Figure 1, the stages of the study generally include taking information from the version repositories, calculation of the metrics, making the necessary analyzes using the calculated metrics, determining the number of errors on the basis of class and verifying the analysis results with the determined number of errors.

- ***Taking information from the version repositories:*** At this stage, a mature version of the relevant project is taken from SVN repository.
- ***Calculation of metrics:*** At this stage, metric set for classes which belong to the project are calculated by using Understand tool.
- ***Making the necessary analyzes using the calculated metrics:*** At this stage, the sum of metric equals for each class is calculated using the metric equals of the classes and the formula developed. The calculated sum of metric equals is sorted starting from the highest value. Classes with the highest ranking sum of metric equals (as 10% of the total number of classes) is marked as the classes that should be tested as a priority.
- ***Determining the number of errors on the basis of class:*** At this stage, the errors that are defined on the basis of configuration on the bug tracking system of the project are analyzed to find total sum of errors for each class. At this stage, the aim is to make the verification results healthier regardless of errors that are not

related to any software class such as third party and equipment errors defined in bug tracking system. Moreover, it is also aimed at making verification results healthier by evaluating errors on bug tracking system due to testings of project team and customer.

- ***Verifying the analysis results with the determined number of errors:*** At this stage, the overlapping ratio of the classes that are determined to be tested primarily to the classes that have the most errors is calculated.

4.1 The Metrics Used in the Study

A total of seven metric were chosen to determine the classes to be tested primarily. Here are the reasons for selecting these metrics and how they are calculated in terms of error susceptibility of the classes:

- ***Lines of Code (LOC):*** The total number of lines that do not have any blank or comment in a class. The higher this value, the more complex and vulnerable this class will be.
- ***Average Complexity (AC):*** The average complexity value of all methods of a class. The higher this value, the more complex and vulnerable this class will be.
- ***Percent Lack of Cohesion (LCOM):*** It is the percentage of compatibility of methods with each other in a class. Low compatibility indicates that the class performs independent tasks and is therefore more vulnerable to failure.
- ***Max Inheritance Tree (DIT):*** It is the distance from the hereditary tree to its root. The fact that this value is high means complex hierarchy and error-proneness.
- ***Response For a Class (RFC):*** The sum of the number of methods in a class and the number of methods that is accessible by an object through inheritance. The fact that this value is high means that the class is bigger, which means that it is more vulnerable to failure.
- ***Number of Instance Variables (NIV):*** The number of instance variables in a class. The fact that the instance is high in number means that the class is more complex and more vulnerable to failure.
- ***Weighted Methods per Class (WMC)***: It is the sum of all the complexities of the methods in a class. The complexity of the methods and the number of methods of a class can give an idea of how much time will be spent developing and maintaining the class [24].

4.2 Information on Projects Used in the Study

This section provides information on a real-time simulation project and a real-time signaling project used to verify the proposed method.

Real-time simulation project (D project): This project is a real-time submarine tactical simulator project. It is carried out in a government institution with about 60 people in the organization. The project started in 2005 and still continues. It reached approximately 1,500,000 line codes. In this study, version 1.0.0 was used to verify with the error numbers belonging to the version delivered to the customer in 2011, and version 2.0.0 to verify with the error numbers of the version delivered in 2014. Basic information of the project is given in Table 1.

Table 1. D project version information

Number of Version	1.0.0	2.0.0
Date of Version	01.04.2007	04.05.2012
Number of Configuration Items	70	95
Number of Classes	1720	1908

Real time signalling project (U project): U Project is a real time signalling project. It is carried out in a government institution with about 10 people in the organization. The project started in 2008 and completed in 2011. In this study, version 1.0.0 was used to verify with the error numbers belonging to the version delivered to the customer in 2009 and version 2.0.0 to verify with the error numbers belonging to the version delivered in 2011. Basic information of the project is given in Table 2.

Table 2. U project version information

Number of Version	1.0.0	2.0.0
Date of Version	05.10.2008	05.11.201 0
Number of Configuration Items	32	45
Number of Classes	652	840

4.3 Methods to Identify the Classes to be Tested Primarily

Two different methods were used in this section.

4.3.1 Random marking method

In this method, 10% of the classes in the selected versions of the projects are marked as random priority classes to be tested. Then, a comparison was made between the classes that entered 10% in the class sequence and the success rate was found. Marking was done 100 times for each version of each project, with the minimum, maximum and average success values, which can be seen in Table 3. This method is used to compare with the method developed within the scope of the study and to give an idea of whether there is improvement within the scope of determining the classes to be tested primarily.

Table 3. Results of random marking method

Project	Version	Minimum	Maximum	Average
D	1.0.0	2	32	16
	2.0.0	2	45	21
U	1.0.0	0	15	6
	2.0.0	1	18	8

According to the findings, the rate of correctly determining the classes to be tested primarily with the random selection method was 9.3% for the D project version 1.0.0, 11.05% for the D project version 2.0.0, 9.23% for the U project version 1.0.0, and 9.52% for the version 2.0.0.

4.3.1 Metric sum method

In the metric sum method developed within the scope of the study, the Metric Equals List (MEL) belonging to the classes was created first. Sum of metric equals list (SMEL) belonging to the classes were also created by using these metric values lists. MEL, which belongs to the classes is created by calculating the metric equals for the concerning version of the project and sorting the resulting values in descending order. All metrics were subjected to normalization process to be on the same weight, meaning between 0-100. For the normalization process, the equation numbered 4.1 will be used.

$$V_n = \left(\frac{V-Min}{Max-Min}\right) * (NewMax - NewMin) + NewMin \tag{4.1}$$

The Min and Max values in Equation 4.1 refer to the smallest and the highest value of the corresponding metric equals calculated of the classes. The V value specifies the initial value of the relevant metric and the V_n value specifies the normalized value of the relevant metric. After the Metric Equals List is created, SMEL is created, the seven metric equals of each class are summed up, and the resulting values are sorted in descending order. The top 10% of the obtained MEL is marked as "the class to be tested primarily". The process of the metric sum method is summarized step-by-step in Table 4, in which the example data is generated.

Table 4. Creating MEL

X Metric	
Class	Value
S4	8
S2	11
S1	1
S3	4

First, the metric values of the project classes are calculated and made a table. Seven metrics were selected to be uses in the study. Table 5 shows the table version of the metric equals calculated. Then, the calculated metric equals are sorted in descending order.

Table 5. MEL ranking

X Metric	
Class	Value
S2	11
S4	8
S3	4
S1	1

Table 6 shows the ranking of the calculated values of any metric.

Table 6. MEL normalization

X Metric	
Class	Value
S2	100
S4	70
S3	30
S1	0

Once the calculated metric equals are ranked, normalization is performed using Equation 4.1. The results of normalization performed on the ranked metric equals are given in Tables 7, 8 and 9.

Table 7. X metric normalization process results

X Metric	
Class	Value
S2	100
S4	70
S3	**30**
S1	0

Table 8. Y metric normalization process results

Y Metric	
Class	Value
S3	**100**
S4	90
S1	50
S3	0

Table 9. Z metric normalization process results

Z Metric	
Class	Value
S3	**100**
S2	75
S4	20
S1	0

Finally, for each class in the project, normalized values of seven metric are added. Thus, the sum of the metric values of the classes is obtained. These values are again sorted in descending order to obtain SMEL as seen in Table 10.

Table 10. Creating SMEL

Sums of the Metric	
Class	Value
S3	**230**
S4	180
S2	175
S1	50

5 Experimental Studies

For the selected versions of the projects that were worked on, MEL and SMEL were created, the classes to be tested primarily were determined, and the error numbers reported on a class basis were found.
As shown in Table 1 and Table 2;

- The number of classes for the Project D Version 1.0.0 is 1720,
- The number of classes for the Project D Version 2.0.0 is 1908,
- The number of classes for the Project U Version 1.0.0 is 652,
- The number of classes for the Project U Version 2.0.0 is 840.

The number of classes to be marked accordingly as 10%;

- The number of classes for the Project D Version 1.0.0 must be 172,
- The number of classes for the Project D Version 2.0.0 must be 190,
- The number of classes for the Project U Version 1.0.0 must be 65,
- The number of classes for the Project U Version 2.0.0 must be 84.

Table 11. Number of classes to be tested primarily according to MEL and SMEL methods

		D Project		U Project	
	Metrics	**1.0.0**	**2.0.0**	**1.0.0**	**2.0.0**
MEL	**LC**	122	95	32	60
	AC	103	108	27	29
	LCOM	60	47	28	39
	DIT	43	72	21	24
	RFC	107	104	38	66
	NIV	89	122	25	31
	WMC	124	120	30	38
SMEL	**TOTAL**	117	110	36	52

Table 11 shows how many of the marked classes using the MEL and SMEL methods actually occupy the 10% of the most frequently detected error. For each version of each project in the table, the correctly identified class numbers calculated by separately using the seven-metric used in the study and the correctly identified class numbers calculated by using all seven metrics are included.

Table 12 shows what percentage of the marked classes using the MEL and SMEL methods actually occupy the 10% of the most frequently detected error.

Table 12. Number of classes to be tested primarily according to MEL and SMEL methods

		D Project		U Project	
	Metrics	**1.0.0**	**2.0.0**	**1.0.0**	**2.0.0**
MEL	**LC**	%70.93	%50.00	%49.23	%71.42
	AC	%59.88	%56.84	%41.53	%34.52
	LCOM	%34.88	%24.73	%43.07	%46.42
	DIT	%25.00	%37.89	%32.30	%28.57
	RFC	%62.20	%54.73	%58.46	%78.57
	NIV	%51.74	%64.21	%38.46	%36.90
	WMC	%72.09	%63.15	%46.15	%45.23
SMEL	**TO-TAL**	%68.02	%57.89	%55.38	%61.90

For each version of each project in the Table 12, the correctly identified class percentages calculated by separately using the seven-metric used in the study and the correctly identified class percentages calculated by using all seven metrics are included.

The percentage values in Table 12 are calculated by dividing the error numbers in Table 11 by 1/10 of the project class numbers given in Tables 1 and 2.

When Table 12 is examined,

- For project D version 1.0.0, the metric that make better estimations than sum of metric equals list (SMEL) are respectively WMC and LC.
- For project D version 2.0.0, the metric that make better estimations than sum of metric equals list (SMEL) are respectively NIV and WMC.
- For project U version 1.0.0, the metric that make better estimations than sum of metric equals list (SMEL) are respectively RFC.
- For project U version 2.0.0, the metric that make better estimations than sum of metric equals list (SMEL) are respectively RFC and LC.

It can be deduced from this analysis that there is not a common metric that makes better estimates than the metric sum method alone. When Table 12 is examined, the metric sum method for all versions appears to give better results than the average estimations of the seven metrics separately. Also, for all versions it seems that there is no common metric that gives the best or the worst result.

6. Conclusions and Reviews

The current software measurement trends are focusing on software metrics, we will propose a sum of the metrics method to identify the classes that should primarily be tested has been developed by using the object-oriented software metric.

According to the results obtained, it is seen that the percentage of metric sum method developed within the scope of the study to determine the classes to be tested primarily is between 55% and 68%. The percentage of random marking method, which was used for comparing, developed within the scope of this study to determine the classes to be tested primarily is between 9.23% and 11.05%. It is observed that the results obtained in the metric sum method show a significant improvement compared to the random marking method. In addition, for each selected version of each project, the average of the rates of accurately determining the classes to be tested primarily and separately for each metric was found to be lower than the rate found by using the metric sum method. The average success rates of the metric were 53.82% for the project D version 1.0.0, 50.22% for project D version 2.0.0; 44.17% for the project U version 1.0.0 and 48.80% for the project U version 2.0.0. In the metric sum method, these values are respectively 68.02%, 57.89%, 55.38% and 61.90%.

When the result Tables are examined, it is observed that some of the metric are more accurate than the metric sum method alone. However, this is not the case for all the versions examined. In addition, when the four versions used are considered, it is also seen that there is not a common metric that makes more accurate estimations alone. It is, therefore, understood that the metric sum method is more reliable. Further studies are planned to create a more comprehensive study by calculating the success rates of all the potential metric combinations to determine the classes to be tested along with all the metric selected.

References

1. Tiftik, N, Öztarak, H, Ercek, G, Özgün, S, Sistem/Yazılım Geliştirme Sürecinde Doğrulama Faaliyetleri, *3. Ulusal Yazilim Mühendisliği Sempozyumu (UYMS'07)*, Ankara, **2007**.

2. Song, O, Sheppard, M, Cartwright, M, and Mair, C, Software Defect Association Mining and Defect Correction Effort Prediction, *IEEE Transactions on Software Engineering*, **2006**, 32(2), 69-82.

3. Fenton, N, Ohlsson, N, Quantitative Analysis of Faults and Failures in a Complex Software System, *IEEE Transactions on Software Engineering*, **2000**, 26(8), 797-814.

4. Xiaowei, W, The Metric System about Software Maintenance, *2011 International Conference of Information Technology, Computer Engineering and Management Sciences*, Wuhan, **2011**.

5. Kaur, A, Sandhu, P.S, Brar, A.S, An Empirical Approach for Software Fault Prediction, *5th International Conference on Industrial and Information Systems*, Mangalore, India, **2010**, pp 261–265.

6. Raymond, P.L, Weimer, B, Weimer, W.R, Learning a Metric for Code Readability, *IEEE Transactions of Software Engineering*, **2010**, 36(4), 546-558.

7. Ogasawara, H, Yamada, A, Kojo, M, Experiences of Software Quality Management Using Metrics through the Life-Cycle, *18th International Conference on Software Engineering*, Berlin, **1996**, pp 179–188.

8. Chaumun, M, Kabaili, H, Keller, R, Lustman, F, Change Impact Model for Changeability Assessment in Object-Oriented Software Systems, *Science of Computer Programming*, Elsevier, **2002**, 45(2-3), 155-174.

9. Lee, Y, Yang, J, Chang, K.H, Metrics and Evolution in Open Source Software, *Seventh International Conference on Quality Software (QSIC 2007)*, IEEE: Portland, OR, **2007**, pp 191-197.

10. Kastro, Y, Bener, A.B, A defect prediction method for software versioning, *Software Quality Journal*, Springer, **2008**, 16(4), 543-562.

11. Li, L, Leung, H, Mining Static Code Metrics for a Robust Prediction of Software Defect Proneness, *International Symposium on Empirical Software Engineering and Measurement*, IEEE: Banff, AB, **2011**, pp 207-214

12. NASA Datasets. (accessed 22.08.2014) http://promise.site.uottawa.ca/SERepository/datasets-page.html

13. Efil, İ, Toplam Kalite Yönetimi ve Toplam Kaliteye Ulaşmada Önemli Bir Araç: ISO 9000 Kalite Güvence Sistemi, *Bursa: Uludağ Üniversitesi Basımevi*, s.29, **1995**.

14. Galin, D, Software Quality Assurance: From Theory to Implementation, *Addison Wesley*, 2004; pp 510-514.

15. Loon, H.V, Process Assessment and ISO/IEC 15504: A Reference Book, *Springer*, 2nd Edition, **2007**.

16. Hofmann, H, Yedlin, D.K, Mishler, J, Kushner, S, CMMI for Outsourcing: Guidelines for Software, Systems, and IT Acquisition, *Addison-Wesley Professional*, 1st Edition, **2007**, pp. 2-4.

17. Yücalar, F, Yazılım Ölçümüne Giriş, *Maltepe Üniversitesi*, Yazılım Mühendisliği Bölümü, Yazılım Ölçütleri Ders Notları, **2013**.

18. Arvanitoua, E.M, Ampatzoglou, A, Chatzigeorgiou, A, Avgeriou, P, Software metrics fluctuation: a property for assisting the metric selection process, *Information and Software Technology*, **2016**, 72, 110-124.

19. Amara, D, Ben Arfa Rabai, L, Towards a New Framework of Software Reliability Measurement Based on Software Metrics, *8th International Conference on Ambient Systems, Networks and Technologies (ANT 2017)*, Procedia Computer Science, **2017**, 109, pp 725-730.

20. Arar, Ö.F, Ayan, K, Deriving thresholds of software metrics to predict faults on open source software: Replicated case studies, *Expert Systems with Applications*, **2016**, 61, 106-121.

21. Chidamber, S, Kemerer, C, A Metrics Suite for Object-Oriented Design, *IEEE Transactions on Software Engineering*, **1994**, 20(6), pp 476-493.

22. Brito e Abreu, F, Pereira, G, Soursa, P, Coupling-Guided Cluster Analysis Approach to Reengineer the Modularity of Object-Oriented Systems, *Conference on Software Maintenance and Reengineering*, IEEE: Washington, DC, USA, **2000,** pp 13-22.

23. Bansiya, J, Davis, C, A Hierarchical Model for Object-Oriented Design Quality Assessment, *IEEE Transactions on Software Engineering*, **2002**, 28(1), pp 4-17.

24. Erdemir, U, Tekin, U, Buzluca, F, Nesneye Dayalı Yazılım Metrikleri ve Yazılım Kalitesi, *Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu (YKGS'2008)*, İstanbul, **2008**.