Indonesian Journal of Electrical Engineering and Computer Science
Vol. 19, No. 2, August 2020, pp. 1085~1092
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v19.i2.pp1085-1092

Test suites effectiveness evolution in open source systems: empirical study

Mohammed Akour[1], Mamdouh Alenezi[2]
[1]Department of Computer Engineering, Al Yamamah University, Saudi Arabia
[1]Yarmouk University, Jordan
[2]Prince Sultan University, Saudi Arabia

Article Info

Article history:

Received Dec 24, 2019
Revised Feb 26, 2020
Accepted Mar 11, 2020

Keywords:

Empirical study
Open source evolution
Test suite effectiveness

ABSTRACT

Test suite code coverage is usually used to indicate the capability of a test suite in detecting faults. Earlier research studies, which explored the relationship among test suite effectiveness and code coverage, have not addressed this relationship evolutionally. Moreover, some of these works were studied small or identical domain systems, which make the result generalization process unclear for other systems. Finally, some of these studies were conducted with automatically generated test suites, which might not present the real situation for studied systems, so the results cannot be generalized to real test suites. In this paper, the authors empirically explore three open-source software systems along with their 11 versions. These versions are evolved over time and might have more sources of code and test suites. This work objective is to study the correlation between test suite effectiveness, the size of the test suite, and coverage for three Java programs during their evolution. In this work, the code coverage, test suite LOC and mutation testing coverage are measured to assess the correlation between the effectiveness of fault detection, code coverage, and test suite size. Based on the result we cannot generalize the assumption that test size is always revealing a positive correlation with its effectiveness, but still weak to the high correlation between test effectiveness, test size, and coverage.

Corresponding Author:

Mohammed Akour,
Department of Computer Engineering,
AL Yamamah University,
Riyadh, Saudi Arabia.
Email: m_akour@yu.edu.sa

1. INTRODUCTION

Having good effective test suites will result in better software quality. Fault-revealing ability evaluation test suites are very essential for comparing different test suites [1]. Testing is a vital stage in the software development process and product quality. When software systems grow in size and complexity, the need for thorough software testing. Since size and complexity are the main sources of software defects. However, comprehensive testing is usually impossible since the trade-off between a number of faults found and testing costs is always there. Testing effectiveness can be measured by how well a test suite can discover faults in a software system. The more test suites reveal bugs the more they are considered of high quality.

Testing activities can cost half the time and effort required to develop a software system and in some cases, it can go up easily to 75 percent of the development cost. Software testing is not as effective as it should be [2]. There is a lack of a commonly accepted standard for software testing that hinders development efforts to evaluate the test results. There is no generalized practice of software testing that is followed across the industries. There is a need for a well-defined underlying methodology that can guide testers. This is required to understand how an ideal test case should look like or what should be the focus of a test case in a given situation.

To identify an effective test suite, it is important to measure its quality. Software development starts with identifying a set of requirements that will guide the development efforts. Software testing verifies and validate that the built software system is according to these requirements. Software testing aims to prevent the occurrences of faults in the software system. Evaluating the software to catch the faults that are already occurred is also part of software testing. Moreover, testing can be utilized to examine the software fin terms of usability, security, compatibility, reliability, portability, integrity, efficiency, capability, etc. Software testing strives to accomplish the required goals and moralities that need to be fulfilled. In simple words, testing is the process of locating errors in the program. Test cases are considered as the heart of testing where software testers spent their effort and time in building these pieces and keep updating them to ensure high-quality testing is performed.

Lehman has educated software practitioners that software systems must evolve, or they will be less useful [3]. When software systems evolve, their source code is considered to be the main studied artifact. However, software systems are multi-dimensional, which involve different artifacts other than the source code. There are different artifacts like requirements, design elements, documentation, test suites, configuration scripts, etc. [4]. During the software development life cycle, the quality of all artifacts is very crucial, once the software systems are evolved, the associated artifacts must co-evolve to maintain the high-quality software from all dimensional artifacts not only the source code part. Artifacts evolution might cause new inconsistent situations and as mentioned earlier software tests are playing a significant role in accomplishing successful software, at the same time these tests need to be maintained as well during software evolution to make sure they are up-to-date. Evolving the tests brings a new burden to the software evolution process as well. Therefore, developing test-code that co-evolves simultaneously with the software source code is not an easy task, and having tools and techniques become mandatory for software developers to make sure the co-evolution process and its relationship are conducted gracefully and correctly.

In this paper authors empirically explore three open-source software systems along with their 11 versions. These versions are evolved over time and might have more sources of code and test suites. The aim of this work is to determine whether code coverage has a strong correlation with the test suite's effectiveness through the evolution process or not. Several works are presented in the literature that addresses the correlation between code coverage and fault detection effectiveness such as Kochhar et al [5] and Inozemtseva et al. [6], Antinyan et al [7], and Papadakis et al. [8]. Although this work investigates the correlation between the code coverage and the test suit effectiveness using open-source projects, still their studies used the small and sometimes large system but none of them addressed this correlation empirically using several versions of the software under study.

Some of the previous studies employed mutants that are manually created or developed automatically using special tools such as PIT and then measure test suite effectiveness by calculating the capability of the test cases in killing the mutants. In this work, the mutants are developed using the PIT tool. After measuring the source code coverage and the effectiveness of the test suite we calculate the correlation between the test suites code coverage factor and test suites effectiveness factor by taking in our account the evolution in both the code production and test suites for the studied systems.
This paper answers the following two research questions:

a) *RQ1:* Is there an empirical correlation between a test suite’s code coverage and its effectiveness in terms of killing mutants?
b) *RQ2:* Is test suite size in term of LOC correlated with its effectiveness in terms of killing mutants?

The paper contribution can be summarized as follows: the experiments are conducted on three open-source systems from different domains, these systems are collected along with their 11 versions and their associated test suites. The essence behind this work is to understand to which extent the correlation is existing between the test suites code coverage and its effectiveness in detecting the fault through the evolution process of these systems. The structure of this paper is as follows. Section two briefly describes the related works, Section three presents the methodology of the work, Section four reveals the result and finding discussion, and Section five concludes the paper.

2. RELATED WORKS

Several research studies are found in the literature where test cases' effectiveness was studied. Gopinath et al. [9] investigated hundreds of GitHub projects. They studied test cases’ code coverage and conduct mutation testing on manual and automatic developed test cases by utilizing the Randoop tool. The result shows that statement coverage could be considered as an indicator of the effectiveness of the test suite. Inozemtseva et al. [10] investigated 5 OSS and developed 31,000 associated test cases and measured statement, modified condition, and decision, coverage. After conducting the mutation testing, the test suite's effectiveness is calculated to know the extent of correlation between the code coverage the test suite effectiveness, they found that there is no strong relationship between these two factors.

Other research studies focused on test cases' evolution. Zaidman et. al. [11] examined the correlation between source code and its associated test suites with regard to the synchronicity of co-evolution and the ability to detect faults, and the correlation between test code and test code coverage. They followed up their study with another study where they used association rules to address the co-evolution relation between test code and source code [12]. After that, they expanded their work to include industrial software [13] and observed that the same results are applicable to industrial systems.

Other research studies focused on visualizations of test cases' evolution. Marinescu et. al. [14] conducted a study on 6 open source projects where the introduce Covrig, a framework that allows engineers to analyze the source code, test code, and show how coverage is evolved. Their aim was to investigate the co-evolution correlation between source code and its associated tests code. They found that source code changes do not certainly mean increasing the testing effort and increasing the test code as well. Ens et. al. [15] presented a visualization technique to study test and source coevolution. They proposed a tool named ChronoTwigger, which uses co-evolve as the root to examine the relationship between source code and its associated tests and then presents this correlation visually.

Mishra et al. [16] introduced a systematic review of test data generation and optimization for addressing path testing by utilizing Evolutionary Algorithms (EAs). Several EA algorithms are used for automatic test case generation and optimization to achieve maximum path coverage. Umar et al. [17] summarize different test case prioritization techniques that are related to object-oriented systems. Their review aimed to highlight the significance and Test Case Prioritization in relation to object-oriented software development lifecycle. Although they work has a good effort, our work will reduce the time and the effort to perform this prioritization if we can reach the best pattern in evolving the source code along with it is an associated test case in terms of effectiveness. Several research point out that software evolution reveals a noticeable impact on test suites evolution, as they have strong relationships and they should be evolved concurrently.

One of the main techniques to address the quality of software systems is the ability of test cases to detect and find defects (test suite effectiveness), Therefore, the effectiveness needs to be navigated. Mamdouh et al. [18] introduce a new technique to evaluate the effectiveness of test cases in order to find defects in open source systems. They perform several experiments on six open source software. The result shows how the effectiveness of test suite of could give an indication about studied systems quality. Moreover, Mamdouh et al. [19] work introduces a statistical evidence of having significant relationship between the code production and its associated test suites. In our work, the focus on the effectiveness of test cases while software systems evolve. Effectiveness is measured in two ways, namely code coverage, and mutation testing. We studied 10 versions of three open-source systems from different domains.

3. RESEARCH METHOD

This section reveals the methodology that is followed in this work as illustrated in Figure 1. The data set (programs understudy) is also described in this section along with correlation metrics that are used in this study.

3.1. Software systems releases collection

Three systems are used in this work where each system consists of versions as follows:

Commons Lang is Apache Java package that has utility classes in java.lang's hierarchy. It provides notable services such as string manipulation, numerical methods, concurrency, and system properties. OGNL is an abbreviation for Object-Graph Navigation Language which is an expression language to get and set Java objects properties. BioJava is a Java framework that processes biological data and provides analytical and statistical features.

Figures 2, 3 and 4 show how LOC for both the code production and test suites are evolved through the 11 versions. In the first system, there was a noticeable increment in the LOC for the code and test suites, while as shown in Figure 4 the OGLN code and test suites have almost the same LOC through the 11 versions of evolution. Biojava System has a larger LOC for code production in comparison with its associated LOC for test suites. Still, the LOC for both has increased through the evolution process from version 1 to version 11. It was obvious from Figure 2, 3 and 4 that the test suite's LOC of commonLang was larger in comparison with the Code's LOC, while was lower in OGNL and Biojava.

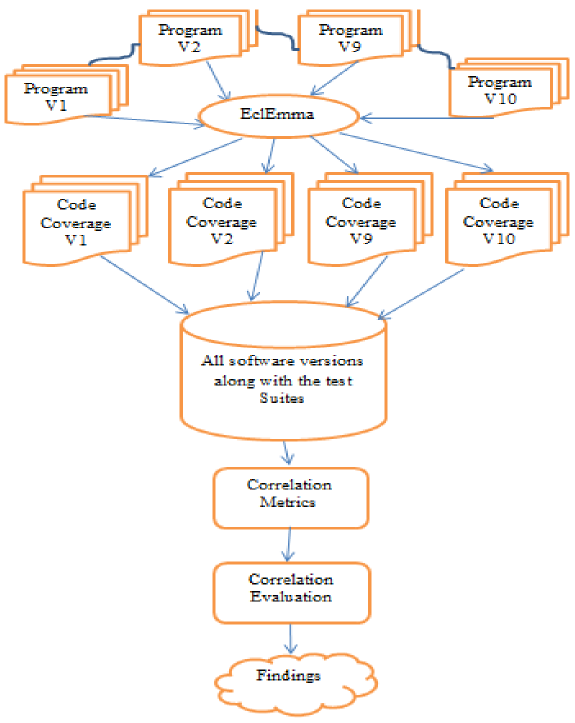

Figure.1 Methodology steps

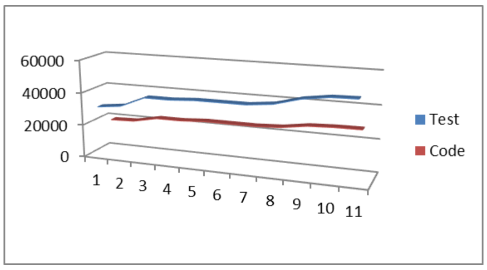

Figure 2. CommonLang system evolution

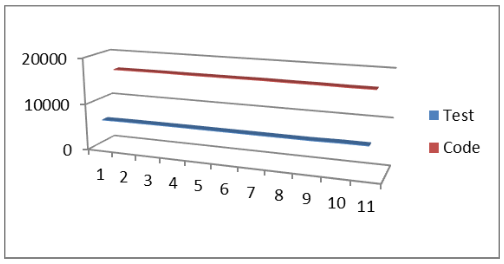

Figure 3. OGNL system evolution

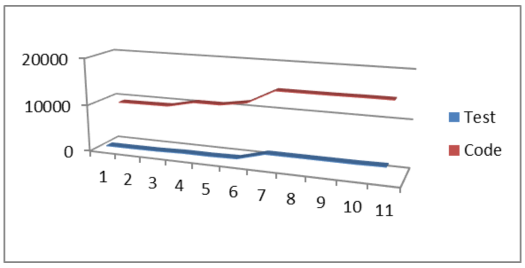

Figure 4. Biojava system evolution

3.2. Test suite code coverage and mutation measurements

Code coverage can be considered as an indicator of having good test suite quality, it presents to which extent the source code of software system is executed when the studied test suite is run [20]. Code coverage provides different levels of coverage for example branch coverage, function coverage, and statement coverage. As the first step and after collecting the source code and test suite data from the GitHub repository, EclEmma [21] is used to measure the code coverage for the systems under study and Eclipse Metrics [22] is used for measure the LOC metric.

The last step is to measure the effectiveness of the collected test suites in terms of killing the fault. For this purpose PIT tool [23] is used to generate faulty versions of the system under study. PIT allows creating mutants that present a different version of the original code that is developed by making a syntactic modification to the original system. PIT tools enable the mutation testing to generate a variant number of mutants and then run the code production's test suite against each one. If the test suite fails when it is run on the created mutant that means the studied suite kills that created mutant. Mutant coverage is calculated for test suites by dividing the number of killed mutants to the total number of studied mutants. PIT is commonly used to generate the mutants and measure the mutation coverage [24-26].

Figures 5, 6 and 7 show how to test suite size in terms of LOC might influence the effectiveness of the test suites in killing the mutants. We believe that increasing LOC might have some meaning of increasing the effectiveness of the test suites by killing more mutants. To have a clear cut conclusion on the relationship between the LOC of test suites and its effectiveness, the correlation is measured and the hypothesis is created and the result is obtained in the next section.

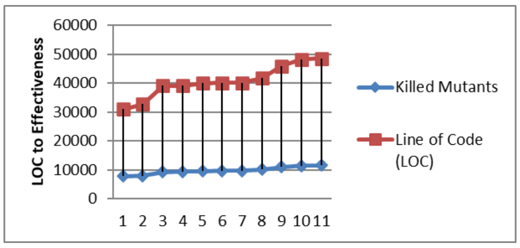

Figure 5. Commonlang LOC and test effectiveness comparison

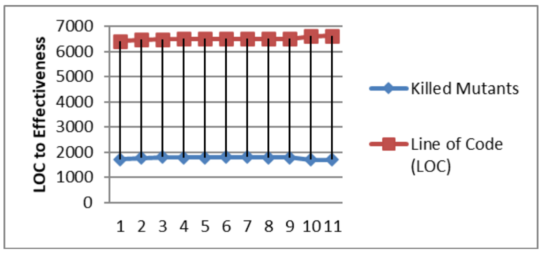

Figure 6. OGNL LOC and test effectiveness comparison

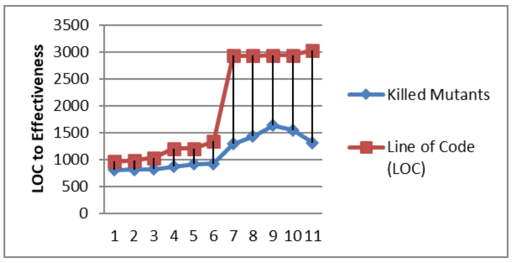

Figure 7. Biojava LOC and test effectiveness comparison

3.3. Correlation measurements

Four Hypotheses are evaluated to address the statistical significance of the results and to estimate the strength of the relationship between the factors mentioned in the RQ1 and RQ2.

For RQ1, The H0 hypothesis is stated that there is no empirical relationship between a test's code coverage and its effectiveness in terms of killing mutants, whereas the alternative hypothesis H1 states that there is a relationship between these two factors.

Moreover, for RQ2, The null hypothesis H0 is stated that there is no relationship between test suite size in terms of LOC and its effectiveness in terms of killing mutants?, whereas the alternative hypothesis H1 states that there is a relationship between these two factors. In this study, Pearson's correlation coefficient is measured using IBM SPSS [27], and the positive, negative or even no correlation is considered to reject or accept the null hypothesis after that we conclude if there is a correlation between the factors are addressed in this hypotheses or not.

4. RESULTS AND DISCUSSION

Tables 1 and 2 show the results of the Pearson Correlation coefficient for RQ1 and RQ2 hypotheses. The values in these tables tell which hypothesis we accept and which we have to reject.

Table 1 reports the answer for RQ2: Is test suite size in term of LOC correlated with its effectiveness in terms of killing mutants?

Table 1. Pearson’s correlation for test suite size in term of LOC and its effectiveness

Software Projects	Pearson’s correlation coefficient
CommonLang	0.340514
OGNL	-0.38856
Biojava	0.772127

As mentioned earlier, we might believe that increasing the LOC of the test suites could improve its associated effectiveness in killing the faults. Therefore, we asked this research question do we really have a relationship between the LOC of a test suite and if there is a relationship that does increase the LOC lead to increase the effectiveness. To provide the answer to this research question, we measured Pearson’s correlation coefficient between the test suite size (LOC) and test suite effectiveness.

When a positive correlation shows up that means there is a positive relationship between the two factors under study; as one-factor increases or decreases, the other will do the same. While When a negative correlation shows up that means if one of the factors increases, then another factor will decrease, and vice versa. Table 1 presents there is an empirical positive correlation for both systems CommonLang and Biojava while the negative correlation for OGNL. The H0 hypothesis states that there is no correlation between test effectiveness and size, while the H1 hypothesis states that there is a correlation between them. From Table 1 we observe that the null hypothesis is rejected for the three systems even though we got a negative correlation for OGNL.

Table 2. Pearson’s correlation for test code coverage and its effectiveness

Software Projects	Pearson’s correlation coefficient
CommonLang	0.227511
OGNL	0.288675
Biojava	0.794381

Table 2 reports the answer for RQ1: Is there an empirical correlation between a test suite’s code coverage and its effectiveness in terms of killing mutants?. In contrast to research question 2, it is obvious there was a positive correlation between a test suite’s code coverage and its effectiveness in terms of killing mutants for all systems under study. As shown in Table 2, we can conclude that the test suite’s code coverage is weakly to strongly correlate with its associated effectiveness. Moreover, the null hypothesis is rejected for the three systems under study. The deviation among test suite size in their correlations to its effectiveness is obvious, where some of the suite sizes are positively correlated with its effectiveness and others are negatively correlated with its effectiveness as shown in Table 1. Therefore, we cannot generalize the assumption that test size is always revealing a positive correlation with its effectiveness.

4.1. Threats to validity

In our study, we measured the size of the test suite in terms of LOC, the code coverage using Eclemma and the effectiveness of test suites in terms of killing the mutants that were created using PIT. The LOC and the code coverage are measured straightforward, but effectiveness is more blurry, as we aimed to expect the capability of the existing test suite that might not mature enough to detect the fault and achieve good effectiveness. Moreover, the PIT produces several mutants that might not present the real faults that should be presented in the actual practice of the systems. This means that the produced mutants might add faults that might not occur or even misses ones that might occur in the real life of the software, this leads to having high construct validity and raises a threat to the validity of this measurement.

5. CONCLUSION

The paper investigated the relationship between the test suite LOC, the suite’s code coverage and the suite’s mutation coverage effectiveness measurement. Three java systems along with 11 versions for each system are studied. Two main research questions were investigated in this study, RQ1: Is there an empirical correlation between a test suite’s code coverage and its effectiveness in terms of killing mutants?, RQ2: Is test suite size in term of LOC correlated with its effectiveness in terms of killing mutants?. The result shows that In general, the test suite’s code coverage is weakly to strongly correlate with its associated effectiveness. While in terms of test suite LOC, there was a deviation among test suite size in their correlations to its effectiveness as some of the suite sizes are positively correlated with its effectiveness and others are negatively correlated with its effectiveness as shown in Tables 1 and 2.

REFERENCES

[1] Zhang, Jie M., Lingming Zhang, Dan Hao, Meng Wang, and Lu Zhang, "Do pseudo test suites lead to inflated correlation in measuring test effectiveness?." In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST*), pp. 252-263, 2019.

[2] N. Juristo, A. M. Moreno, and W. Strigel, " Guest editors' introduction: software testing practices in industry," *IEEE Software*, vol. 23, no. 4, pp. 19–21, 2006.

[3] Lehman, Meir M, "On understanding laws, evolution, and conservation in the large-program life cycle," *Journal of Systems and Software*, vol. 1, pp. 213-221, 1979.

[4] Mens, Tom, Michel Wermelinger, Stéphane Ducasse, Serge Demeyer, Robert Hirschfeld, and Mehdi Jazayeri, "Challenges in software evolution," *In Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*, pp. 13-22, 2005.

[5] Kochhar, Pavneet Singh, Ferdian Thung, and David Lo., "Code coverage and test suite effectiveness: Empirical study with real bugs in large system," *2015 IEEE 22nd international conference on software analysis, evolution, and reengineering (SANER)*, pp. 560-564, 2015.

[6] Inozemtseva, Laura, and Reid Holmes, "Coverage is not strongly correlated with test suite effectiveness," *Proceedings of the 36th International Conference on Software Engineering*, pp. 435-445, 2014.

[7] Antinyan, Vard, *et al*, "Mythical unit test coverage," *IEEE Software*, vol. 35, no. 3, pp. 73-79, 2018.

[8] Papadakis, Mike, *et al*, "Are mutation scores correlated with real fault detection? a large scale empirical study on the relationship between mutants and real faults," *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp. 537-548, 2018.

[9] Rahul Gopinath, Carlos Jensen, and Groce Alex, "Code coverage for suite evaluation by developers," *Proceedings of the 36th International Conference on Software Engineering*, pp. 72–82, 2014.

[10] Laura Inozemtseva and Reid Holmes., "Coverage is not strongly correlated with test suite effectiveness," *Proceedings of the 36th international conference on software engineering*, pages 435–445, 2014.

[11] Zaidman, B. Van Rompaey, S. Demeyer, and A. Van Deursen, "Mining software repositories to study co-evolution of production & test code," *2008 1st International Conference on Software Testing, Verification, and Validation*, Lillehammer, pp. 220-229, 2008.

[12] Z. Lubsen, A. Zaidman, and M. Pinzger, "Using association rules to study the coevolution of production & test code," *2009 6th IEEE International Working Conference on Mining Software Repositories*, Vancouver, BC, pp. 151-154, 2009.

[13] Zaidman, B. Van Rompaey, A. van Deursen, and S. Demeyer, "Studying the coevolution of production and test code in open source and industrial developer test processes through repository mining," *Empirical Software Engineering*, vol. 16, no. 3, pp. 325– 364, 2011.

[14] P. Marinescu, *et al*, "Covrig: A framework for the analysis of code, test, and coverage evolution in real software," In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pp. 93–104, 2014.

[15] B. Ens, *et al*, "Chronotwigger: A visual analytics tool for understanding source and test co-evolution," *2014 Second IEEE Working Conference on Software Visualization, Victoria*, BC, pp. 117-126, 2014.

[16] Mishra, D. B., Acharya, A. A., & Mishra, R., "Evolutionary algorithms for path coverage test data generation and optimization: a review," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 15, no. 1, pp. 504-510, 2019.

[17] Umar Farooq, Hannani Aman, Aida Mustapha, Zainuri Saringat, "A review of object-oriented approach for test case prioritization," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 16, no. 1, pp. 429-434, 2019.

[18] Mamdouh Alenezi, Mohammed Akour, Alaa Hussien, and Mohammad Z. Al-Saad, "Test suite effectiveness: an indicator for open source software quality," *2016 2nd International Conference on Open Source Software Computing (OSSCOM)*, Beirut, pp. 1-5, 2016.

[19] Mamdouh Alenezi, Mohammed Akour, and Hiba Alsghaier, "The impact of co-evolution of code production and test suites through software releases in open source software systems," *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, vol. 9, no. 1, pp. 2737-2739, 2019.

[20] Paul Ammann and Jeff Offutt, *Introduction to software testing*, Cambridge University Press, 2008.

[21] EclEmma: https://www.eclemma.org/, Access 2019

[22] Eclipse Metrics: https://marketplace.eclipse.org/content/eclipse-metrics, Access 2019

[23] PIT. http://pitest.org/, Access 2019

[24] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser, "Are mutants a valid substitute for real faults in software testing? Technical Report UW-CSE-14-02-02," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 654-665, 2014

[25] J. H. Andrews, L. C. Briand, and Y. Labiche., "Is mutation an appropriate tool for testing experiments?," *Proceedings of the 27th international conference on Software engineering*, pp. 402-411, 2005.

[26] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, "Using mutation analysis for assessing and comparing testing coverage criteria," *IEEE Transactions on Soft. Eng.*, vol. 32, no. 8, pp. 608-624, 2006.

[27] IBM Corp. Released in 2013. *IBM SPSS Statistics for Windows*, Version 22.0. Armonk, NY: IBM Corp

BIOGRAPHIES OF AUTHORS

Dr. Mohammed Akour is an associate Professor of Software Engineering at Al Yamamah University (YU). He got his Bachelor's (2006) and Master's (2008) degree from Yarmouk University in Computer Information Systems with Honor. He joined Yarmouk University as a Lecturer in August 2008 after graduating with his master's in Computer Information Systems. In August 2009, He left Yarmouk University to pursue his Ph.D. in Software Engineering at North Dakota State University (NDSU). He joined Yarmouk University again in April 2013 after graduating with his Ph.D. in Software Engineering from NDSU with Honor. He serves as Keynote Speaker, Organizer, a Co-chair and publicity Chair for several IEEE conferences, and as ERB for more than 10 ISI indexed prestigious journals. He is a member of the International Association of Engineers (IAENG). Dr. Akour at Yarmouk University served as Head of accreditation and Quality assurance and then was hired as director of computer and Information Center. In 2018, Dr. Akour has been hired as Vice Dean of Student Affairs at Yarmouk University. In 2019, Dr. Akour joins Al Yamamah University -Riyadh Saudi Arabia- as an associate professor in Software Engineering.

Dr. Mamdouh Alenezi is currently the Dean of Educational Services at Prince Sultan University. Dr. Alenezi received his MS and Ph.D. degrees from DePaul University and North Dakota State University in 2011 and 2014, respectively. Dr. Alenezi is an associate professor in software engineering with the teaching emphasis on software engineering and software security. He participates in organizing several international scientific conferences and editorial boards of the well-reputed journals. He has extensive experience in applying data mining and machine learning techniques to solve software engineering problems. He published more than 80 papers. He conducted several research areas and development of predictive models using machine learning to predict fault-prone classes, comprehend source code, and predict the appropriate developer to be assigned to a newly reported bug. His research focuses on Software Engineering, Software Security, Machine Learning, and Open Source Software Systems.