The Impact of Co-evolution of Code Production and Test Suites through Software Releases in Open Source Software Systems

Mamdouh Alenezi, Mohammed Akour, Hiba Al Sghaier

Abstract*: *The software system evolves and changes with the time, so the test suite must be maintained according to code changes. Maintaining test cases manually is an expensive and time-consuming activity, especially for large test suites, which has motivated the recent development of automated test-repair techniques. Several researchers indicate that software evolution shows a direct impact on test suites evolution, as they have strong relationships and they should be evolved concurrently. This article aims to provide statistical evidence of having this significant relationship between the code production and its associated test suites. Seven systems along with releases are collected and eight metrics were calculated to be used in this study. The result shows how the systems under study are evolving and have a high impact on their test suites, although two metrics provide a negative significant relationship.

Keywords: Software Testing, Software evolution, complexity, size.

I. INTRODUCTION

Software testing is an important and essential step to identify the correctness and quality of the software systems. In the software testing process, the tester should write one test case or more to check each function of the system. The test case is the smallest meaningful unit of the tests. The result of each test case is either pass or fail. .If test cases are passed (i.e., the actual results = the expected results), then the functionality of a software system corresponding to these passed test cases are working correctly. The test suite is a collection of test cases to test system functionalities. Any software system (S) is divided into two parts: program (P) and test suite (T). All system test cases (Tc) are stored in (T). These test cases are used to check the correctness of all parts of P. The new version of the software system (S') should have a different program (P') and test suite (T'). Software evolution is one of the essential and normal issues required for most existence software throughout their lifetime. The software requirements changes may come from business needs, competitions, changes in government rules, and environment. These changes in requirements are offset by a set of changes in the software code. Accordingly, the current test suite becomes obsolete for the new version of the software [1],[2].

Revised Manuscript Received on November 10, 2019.
***** Correspondence Author
Mamdouh Alenezi*, College of Computer and Information Sciences, Prince Sultan University, Riyadh, Saudi Arabia. E-mail: malenezi@psu.edu.sa
Mohammed Akour, Al Yamamah University, Riyadh, Saudi Arabia, Yarmouk University, Irbid, Jordan. E-mail: mohammed.akour@yu.edu.jo
Hiba Al Sghaier, Yarmouk University, Irbid, Jordan. E-mail: hibaal_sghaier@yahoo.com

Therefore, the tester must revise all changes on the code to repair the corresponding test cases in the test suite. While the code evolution may happen frequently, it is very hard for the tester to follow all these code evolutions and to make the correct decisions by creating, deleting, and updating test cases. Previous research indicates that software evolution shows a direct impact on test suites evolution, as they have strong relationships and they should be evolved concurrently. In this paper, authors try to provide statistical evidence of this impact in terms of size and complexity metric across several versions of diverse software systems.

II. RELATED WORK

Several researchers studied the nature of the co-evolution between code and test (i.e. synchronously or phased) [3], [4] and [5]. In [3], Lubsen et al. used two case studies: the open-source system and industrial software system, as they used association rule mining to study the natural of co-evolution. They concluded that within an open-source system the development and testing are separate activities, wherein the industrial software system, the developer used a test-driven development strategy. In [4] and [5], the researchers proposed three views which are: change history view, growth history view, and test coverage evolution view to study the nature of the co-evolution. In study number [4], the researchers used two open-source projects, while in [5], they used two open-source projects and one industrial software project. They concluded that the nature of co-evolution depends on the development style that is used to develop a project.

Several metrics are used to determine the size of production code or tests, such as the number of classes, Line of Code (LOC), number of methods, and number of packages. The software complexity focuses on how a piece of code interacts with other pieces. One of the most popular measurements of software complexity is the McCabe metric or Cyclomatic complexity metric. The Cyclomatic complexity per method metric is the maximum number of linearly independent paths within the method [6]. In [7], the authors experimented to find a good technique to evaluate the effectiveness of test cases for finding defects in open source systems.

III. RESEARCH METHODOLOGY

The main idea behind this research paper is to understand and identify how the test cases evolve during the code changes (releases) in terms of size and complexity and if the code evolution has a statistically

impact on the test suite evolution.

2.1 Research hypotheses

To properly address the impact of the code evolution on test suite evolution, authors address the following hypotheses:

H0: There is no statistical relationship between the code and test evolution in terms of size and complexity

H1: There is a high statistical relationship between the code and test evolution in terms of size and complexity.

2.2 Experimental Setup

The main goal of this study is to understand how the test suite evolves over time. Thus, to achieve this goal, several versions of 8 open-source Java systems with their test suits were used to investigate different aspects of test-suite evolution. These systems were selected according to many criteria, which are popular, system size, each system has at least 5 versions, and each version has a JUnit test suite. The 7 open-source Java systems used in our empirical study are selected from GitHub (https://github.com/). Table 1 lists the systems and their versions.

Table 1: Software systems used in the empirical study

Program	Description	Number of versions
Gson	It is a library to serialize and deserialize Java objects to JSON	12
IzPack	It is a widely used library for packaging applications on the Java™ platform	12
JodaTime	It is the de facto standard date and time library for Java	12
PMD	It is a static source code analyzer that reports on issues found within the application code.	12
OGNL	It is an expression language for Java, which using the simpler expression than the Java language.	12
Biojava	It is an open-source project for manipulating and processing biological data in Java language.	12
Struts	It is a web application framework for developing Java EE web applications.	12

All the previous researches have studied the relationship between the code and the test suite generally and provided general information about the relationship between the code and the test suite. However, in the current investigation, the authors studied the relationship between code and test suite evolution impact in terms of size and complexity.

2.3 Software Metrics

In this study, several metrics were calculated for the software under study as input to the intended experiments. These metrics are associated with software size and complexity as follows:

- Number of Classes (NOC): This metric count the number of classes in a system.
- Number of Attributes (NOA): This metric count the total number of attributes defined in a class.
- Average Nested Block Depth per Method
- Line of code (LOC): This metric counts the lines of code in a system.
- Weighted Method per class (WMC): This metric counts the sum of complexities of all methods in a class.
- Number of Methods (NOM): This metric count the number of methods in a system.
- Number of Packages (NOP): This metric count the number of packages in a system.
- Cyclomatic Complexity per Method: This metric measures the cyclomatic complexity of methods. It measures whether individual methods are more or less complex.
- The main tools used in this study are
- Code Metrics: https://marketplace.eclipse.org/content/eclipse-metrics
- Code Coverage: http://www.eclemma.org/
- Mutation Testing: https://marketplace.eclipse.org/content/pitclipse

IV. RESULT AND DISCUSSION

The test suite is changing and evolving during its lifetime according to the code changes. Therefore, the relationship between the code and its test suite must be investigated. Accordingly, this paper studied the relationship between the code and test suite in terms of size and complexity. IBM SPSS [12] statistics version 22 tool is used to find the P-value based on linear regression, the above table shows the comparison between CODE and TEST based on seven projects and eight metrics. Table 2 presented that how different metrics expose different relationship impacts between code and test evolutions.

As mentioned previously, P-value is calculated for all releases between code and test suites in terms of size and complexity metrics. If P-value turns out to be less than the significance level (0.05), that means we reject the null hypothesis and accept the alternative hypothesis.

From the results that are presented in Table 2, we can conclude that Hypothesis 0 can be rejected for almost 6 metrics out of 8 as the P-value is less than 0.05. Although these metrics have one p-value that is higher than 0.05 still, in general, they have a significant evolution relationship for the software under study. In more detail, the null hypothesis is accepted for the Number of Packages,(NOP), Number of Methods (NOM), Number of Attributes (NOA) and Number of Classes (NOC) for 4 experiments out of 42 (i.e. IzPack, GSON, and OGNL respectively).

From the results that are presented in table 2, we can conclude that Hypothesis 1 can be rejected for Average Nested Block Depth per Method and Cyclomatic Complexity per Method metrics as the P-value is higher than 0.05. Still, there are 4 experiments out of 14 produce P-values less than 0.05 which means hypothesis 1 can be acted for this one but in general, these two metrics assure the null hypothesis.

Table 2: P-value results

P-Value / Metrics	Cyclomatic Complexity per Method	Average Nested Block Depth per Method	Number of Packages (NOP)	Number of Methods (NOM)	Weighted Method per class (WMC)	Line of Code (LOC)	Number of Attributes (NOA)	Number of Classes (NOC)
GSON	0.004902	0.016856	0.001782	0.769104	0.000942	0.000222	0.307459	0.601492
IzPack	0.000009	0.000003	0.100170	0.000	0.000	0.000	0.002723	0.000002
JodaTime	0.087440	0.158083	0.000	0.000	0.000	0.000	0.012147	0.019981
PMD	0.119745	0.226645	0.000066	0.000	.000	0.000	0.000	0.000
OGNL	0.026443	0.166401	0.000	0.000	0.006880	0.000593	0.000	0.076325
Biojava	0.001186	0.054976	0.000	0.000	0.000	0.000	0.000	0.000
Struts	0.567880	0.091219	0.000	0.000	0.000	0.000	0.000	0.000

V. CONCLUSION AND FUTURE WORKS

In this study, the authors aim to provide statistical evidence of having a high impact of co-evolution between the code production and their associated test suites. The study calculates 8 size and complexity associated metrics for 7 open software systems along with 12 releases for each project. The result shows how to code production has a strong impact on co-evolution with their test suites for 44 experiments out of 56 were most of the low significant results are solely in two metrics out of 8 metrics under study. In the future, we are planning to study the effectiveness of test cases and how they evolve over time.

REFERENCES

1. Rwemalika, R., Kintis, M., Papadakis, M., Le Traon, Y. and Lorrach, P., 2019, April. On the Evolution of Keyword-Driven Test Suites. In 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST) (pp. 335-345). IEEE.
2. S. Levin and A. Yehudai, "The co-evolution of test maintenance and code maintenance through the lens of fine-grained semantic changes," in 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), Sept 2017, pp. 35–46.
3. Z. Lubsen, A. Zaidman, and M. Pinzger, 'Using association rules to study the co-evolution of production #x00026; test code', in 2009 6th IEEE International Working Conference on Mining Software Repositories, 2009, pp. 151–154.
4. A. Zaidman, B. V. Rompaey, S. Demeyer, and A. v Deursen, 'Mining Software Repositories to Study Co-Evolution of Production #x00026; Test Code', in and Validation 2008 1st International Conference on Software Testing, Verification, 2008, pp. 220–229.
5. A. Zaidman, B. Van Rompaey, A. van Deursen, and S. Demeyer, 'Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining', Empir. Softw. Eng., vol. 16, no. 3, pp. 325–364, Jun. 2011.
6. G. K. Gill and C. F. Kemerer, 'Cyclomatic complexity density and software maintenance productivity', IEEE Trans. Softw. Eng., vol. 17, no. 12, pp. 1284–1288, Dec. 1991.
7. Alenezi, M., Akour, M., Hussien, A. and Al-Saad, M.Z., 2016, December. Test suite effectiveness: an indicator for open source software quality. In 2016 2nd International Conference on Open Source Software Computing (OSSCOM) (pp. 1-5). IEEE.
8. D. Franke and C. Weise, 'Providing a Software Quality Framework for Testing of Mobile Applications', in Verification and Validation 2011 Fourth IEEE International Conference on Software Testing, 2011, pp. 431–434.
9. Shalini and S. I. Hassan, 'An empirical evaluation of the impact of aspectization of cross-cutting concerns in a Smart-phone based application', in 2014 International Conference on Computing for Sustainable Global Development (INDIACom), 2014, pp. 448–454.
10. J. Hernandez, A. Kubo, H. Washizaki, and F. Yoshiaki, 'Selection of metrics for predicting the appropriate application of design patterns', 2011, pp. 1–4.
11. T. Pessoa, F. Brito, M. P. Monteiro, and S. Bryton, 'An Eclipse Plugin to Support Code Smells Detection', p. 12.
12. IBM Corp. Released in 2013. IBM SPSS Statistics for Windows, Version 22.0. Armonk, NY: IBM Corp.

AUTHORS PROFILE

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. He has extensive experience in data mining and machine learning where he applied several data mining techniques to solve several Software Engineering problems. 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 new bug.

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.

Hiba Alsghaier got her Master's degree in Computer Information Systems from the Faculty of Information Technology and Computer Sciences, Yarmouk University. She worked at Jordan University of Science Technology as a teaching assistant for a few years, and now she is a web designer and a member in statistical consulting center advisory board at JUST, Alsghaier has many publications in the fields of software engineering, big data analytics, and software fault prediction and currently she is working on other fields