

A Novel Approach to Test Case Prioritization for Software Regression Tests

Tulin Boyar[1(✉)], Mert Oz[1], Ekin Oncu[2], and Mehmet S. Aktas[1]

[1] Yildiz Technical University, Istanbul, Turkey
{tulin.boyar,mert.oz}@std.yildiz.edu.tr,
aktas@yildiz.edu.tr
[2] Research and Development Center, Testinium, Istanbul, Turkey
ekin.oncu@testinium.com

Abstract. Regression testing of a comprehensive software system is very costly and time-consuming. Test prioritization techniques are used to reduce the time and cost spent in regression tests. Data such as various software quality metrics and past test run results are used when prioritizing test cases in the literature. In this study, a new test prioritization method is proposed based on the number and speed of detected errors. Applying the proposed technique aims to see the errors in the system as soon as possible. In our method, historical test data is analyzed to prioritize test cases. The extent to which the analysis focuses on the recent and distant past is controlled by the parameter. For each test case; prioritization is made based on the number of detected errors and the speed of error detection. We observe its effectiveness by applying our method on regression testing for a developer performance portal software. The experimental study shows that the proposed method is valid, and the results are promising. Our approach improves the error detection rate of test sets. This improvement can be applied even in the most complex situations. With the advancement in the error detection rate, the cost and effort spent on regression tests are also reduced.

Keywords: Test cases prioritization · Regression tests · Test prioritization techniques · Test cases · Average percentage of fault detected (APFD) metric

1 Introduction

Large-scale software constantly evolves to adapt to changing business requirements. A software development process of such software projects includes an endless loop of deleting, updating, or adding new elements. To ensure the robustness of the software systems, engineers should test them in every single change. Since testing complex software forms a considerable expense and time-consuming process via traditional methods, the importance of regression testing becomes

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12955, pp. 201–216, 2021.
https://doi.org/10.1007/978-3-030-87007-2_15

evident at this point. Regression testing aims to detect maximum errors in software systems with minimum time and effort. Literature introduces several great works for this purpose [6–13]. The regression testing for prioritizing test cases is used to determine the best order that saves money and time when they are run with their best order [30]. Software systems using regression tests are tested on all codes and modules again, even if a tiny change occurs in the codebase [1].

Regression testing of an extensive and comprehensive software system is also quite large. Regression test continues to grow as a result of all the studies and improvements made in the software. Regression tests are costly and time-consuming. In addition, in cases where there are limited resources, evaluating all test cases of the regression test may not be suitable.

There exits studies that take into account software metrics in order to improve the quality of software systems [28,29,34]. With the help of software quality metrics, some studies investigate the human factor in software system quality [35,36]. There are some studies that take into account collective behaviour of testers/users to be able to test the software systems [31,32]. There are also testing approaches that conduct testing based on the model-based approaches [33]. Different from these studies, this manuscript focuses on software regression tests and test case prioritization.

Approaches used to reduce the cost of regression testing: 1) Retest All: One technique to perform regression testing is to retest all test cases; it takes much time to run all test cases in an existing test suite. 2) Regression Test Selection: It selects certain test cases to carry out the regression testing. Ignoring some scenarios is the disadvantage of the Regression Test Selection technique. 3) Test Case Prioritization: It prioritizes tests in regression tests with too many test scenarios. Testing all available tests is carried out, but it brings a certain ranking according to the determined criteria. 4) Hybrid Approach: The Regression Test Selection and Test Case Prioritization techniques are used together. [1,6]

Motivation: One of the essential parts of a software system is software testing. One of the most critical points in comprehensive and extensive software is regression tests. Regression tests are crucial to software system reliability. In regression testing, prioritizing test cases and detecting errors in the minimum time provides excellent benefits to reducing effort and reducing costs. Furthermore, developing different methods in regression test prioritization techniques will contribute to software engineering. For these reasons, we study a test prioritization within the scope of this research.

Research Questions: In this research, we focus on the problem of prioritizing test cases by using metrics (working time, number of errors, past test results). In this study, we propose a method to prioritize test cases. The primary purpose of our research is to identify the errors as soon as possible by prioritizing the regression test cases. For this reason, the main research questions we focus on are as follows. 1) How should one evaluate error situations in test scenarios, which have an essential effect on ordering test cases? 2) How should the running times and fault detection times of test scenarios be evaluated? 3) How should previous test effectiveness be used in deciding on the new test sequences? 4) How

should the efficiency of the ordering of test cases be measured? 5) How does the prioritization of test cases affect resource utilization?

Contributions: In this paper, we propose a method based on past test case results for prioritizing test cases. The purpose of the method is to list the test cases to detect errors as soon as possible. Based on the test analysis made in the past, the amount of error detected by each test case and the error detection speed effectively take role in ordering test cases. When compared with other studies in the literature, this study has two main contributions.

- To prioritize test scenarios, a methodology is proposed by using data on the number of errors detected, the number of errors controlled, and error detection time.
- By performing case analysis on an industrial system, the method's success is evaluated, and the effects of different parameter values are analyzed.

Organization: Section 2 presents a literature review on test case prioritization studies for regression test cases. Section 3 explains the essential parts of the method we propose. Section 4 details the prototype application of our proposed method. Section 5 evaluates the study results and explains future studies.

2 Literature Review

In a software development process, a lot of effort is spent on the testing process. For this reason, researchers are working to make test processes more efficient, and effective [14–16]. Prioritization of test cases is a technique used in a software development process to find the best order that gives maximum benefits. In many studies, the prioritization technique and algorithm of various test cases have been proposed and used [12,20–26].

Software TCP Survey, one of the pioneers of Rothermel et al., is an approach that recommends and comprehensively evaluates TCP. For prioritization, it examines the effectiveness of different prioritization techniques with seven programs written in C languages containing 1000–5500 test scenarios by defining a few technical definitions such as random approaches, context-based approaches. A significant performance difference was observed in the study, where they suggested applying the inclusion-based technique. To prioritize the test cases, the properties of the examined systems, the abstraction level of the developed techniques (source code line level, architectural component level, function level, etc.) were also evaluated [27].

S. Raju and G.V. Uma [10] has introduced a test case prioritization technique based on clustering. In this study, test cases are clustered according to dynamic runtime behavior. The researchers presented a test scenario prioritization approach that prioritizes system-level testing requirements by reducing the number of pairwise comparisons required. In this approach, the pre-qualification of test cases is based on four factors: speed of fault detection, variability of requirements, fault impact, and implementation complexity. The researchers evaluated

Table 1. Literature review summary

Reference	Finding
Aljawabrah, et al. (2021)	A study in which test cases are created automatically and prioritized based on software risk
Misra, S., et al. (2017)	A class complexity metric has been proposed in object-oriented programming
S. Singhal, et al. (2017)	A study in which the hybrid approach, in which the Genetic algorithm and Colony optimization are used together, is examined according to the time constraint
Jatana, N. et al. (2016)	A research on the applicability of PSO in test data generation with mutation testing
Sozer and Kurt (2014)	A based on history approach using defect detection rates to prioritize test cases
S. Raju and G.V. Uma (2012)	A test case prioritization technique based on clustering
Rothermel et al. (2001)	Examining different test prioritization approaches

the efficiency of test case prioritization techniques by performing loss-gain analysis for different abstraction levels. To prioritize test cases, many techniques and methods have been proposed, experimentally evaluated, and compared with each other [2,3]. Sozer and Kurt [5] presented a based on history method in their study. Test cases are prioritized according to the amount of error detection obtained from the analysis of past tests. In the study, how much to focus on the recent or distant past is controlled by a parameter.

Aljawabrah et al. Developed a method to automatically generate test cases based on software risk in prioritizing test cases. He confirmed the technique he developed by many software experts [15]. Jatana, N. et al. proposed an approach to generate optimized test data based on Particle Swarm Optimization combined with mutation testing [17]. S. Singhal et al. examined the results of the hybrid method (MHBG_TCS tool) that uses the Genetic algorithm and Colony optimization together in the Regression Test Selection method against the changing time constraint [18].

When the studies in the literature are examined, various metrics such as APFD, APFDc, f-measure, Relative Position, APSC have been used to measure the effectiveness of the proposed test prioritization technique. Among these metrics, the most used APFD metric [2,5,19,23,27] (Table 1).

3 Methodology

Several workflow steps need to be performed to validate the proposed study. Our research aims to detect the errors in the software with a regression test as soon as possible. For this purpose, the first step is to provide a regression test data set applied to a software system. It should contain test cases that include test scenarios grouped within the regression test and control the whole system.

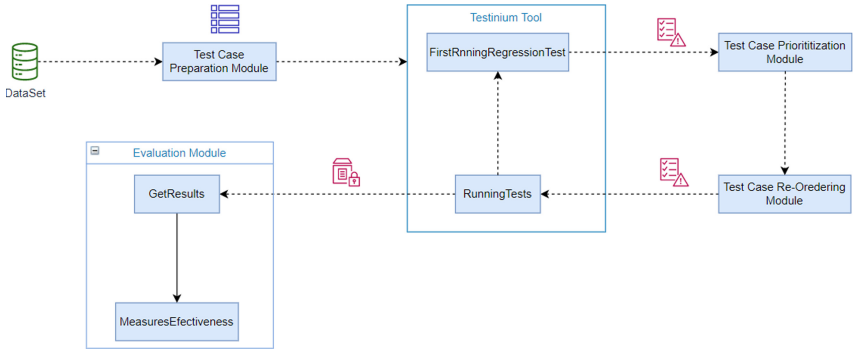

Fig. 1. The proposed system architecture

Within the regression test data set applied for an existing software system, initial data to be used to prioritize test cases are created. The regression test is then run without any prioritization work. Priority score is found for each test scenario by making test prioritization calculations from the data obtained through test runs. Test scenarios are ranked according to the test prioritization score. The test senates in the regression test are sorted according to the prioritization account and run again.

The method suggested in our study is also effective in test case prioritization. For this reason, the efficiency of the method will increase in successive test runs. Once the test results are obtained, efficiency is measured using the APFD metric for each new ranking. Figure 1 shows the workflow.

3.1 Test Case Preparation

Regression tests include test sets grouped by related test scenarios. There are one or more test scenarios in each test set. Each test case checks at least one error condition. Multiple sub-conditions are also checked according to the process steps in the test scenario. In our study, the number of situations controlled by each test scenario is considered the scenario's size. In the first step of our research, as shown in Figure 2, the error condition that it controls for each test scenario is determined first.

The number of cases checked by the test scenario and the number of errors detected are used to prioritize test cases. Suppose in a regression test shown in

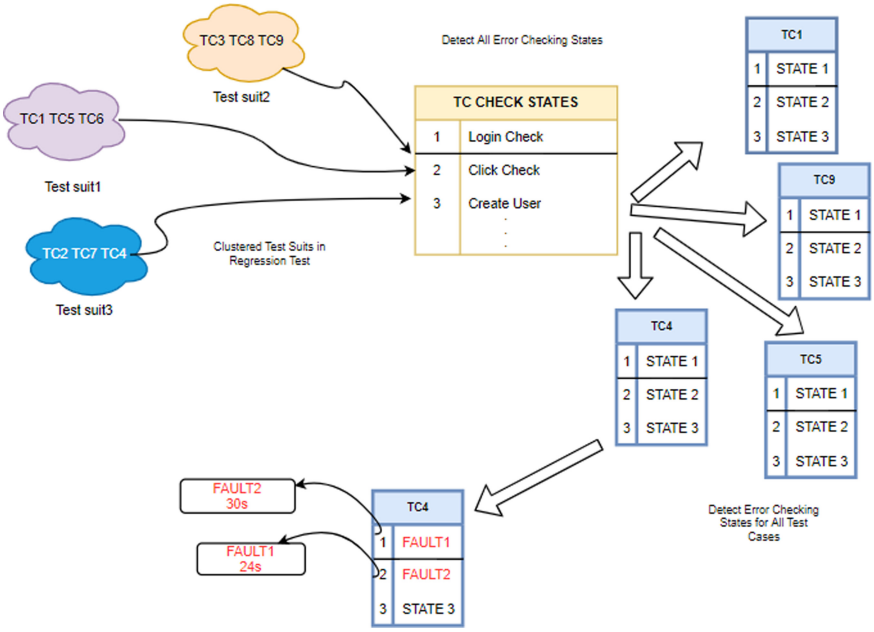

Fig. 2. Data processing

Table 2, 10 error conditions of 5 test scenarios are checked. Each test case can detect many errors, and various test cases can catch the same error. In our study, the number of matched cases will be referred to as the size of the test scenario.

Table 2. Test cases and states example

TC1	TC2	TC3	TC4	TC5
State1	State1	State1	State1	State1
State2	State3	State6	State2	State2
State3	State5		State4	State9
State4			State7	State10
			State8	
4	3	2	5	4

3.2 Test Case Re-ordering Module

First, the regression test consisting of randomly ordered test cases is run on its original form without prioritizing test cases. Second, data from test run results are recorded. Third, for each test scenario, the run time, the error detection time,

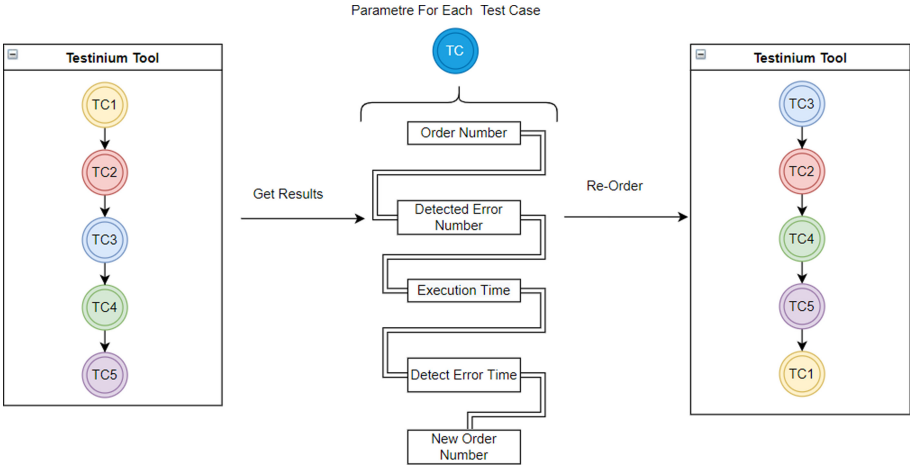

Fig. 3. Running regresion tests and getting data

the size of the scenario, the number of errors detected is used in the prioritization score calculation.

The test cases included in the regression tests prioritize the amount of error they have detected in the past and the speed of error detection.

As shown in Fig. 3, the result data to be used in the prioritization calculation is analyzed, and the new order of the regression test is obtained.

In our study, the regression test was run over Testinium Tool [4]. We took prioritization parameters through the tool.

3.3 Test Case Prioritization Calculation Module

Error Detection Rate. In the method we propose, error rates detected in the past are evaluated to prioritize the test scenarios. Our study focuses on the calculation of error detection rates, and a unique method is recommended. Error detection rate is calculated as follows.

i = Tested scenario
t = Time to be tested
EDC: Error detection capacity
EDS: Error detection speed
EDR: Error detection rate

$$EDC(i,t) = \frac{number\ of\ errors\ caught\ by\ scenario\ i\ at\ time\ t}{size\ of\ scenario\ i} \quad (1)$$

$$EDS(i,t) = 1 - \frac{time\ to\ catch\ the\ error\ at\ time\ t\ of\ scenario\ i}{time\ to\ complete\ scenario\ i} \quad (2)$$

$$EDR(i,t) = EDC(i,t) + EDS(i,t) \quad (3)$$

While calculating the error detection rate, it was aimed to further refine the EDR score by taking into account the error detection rate of test cases.

Priority Score

$$R(i,t) = \begin{cases} 0 & t = 0 \\ EDR(i,t-1) & t = 1 \\ \alpha\ xR(i,t-1) + (1-\alpha)xEDR(i,t-1)\ t > 2 \text{ and } 0 < \alpha < 1 \end{cases}$$

We tested no test cases during the first regression test ($t = 1$), so the error detection rates are unknown. Therefore, priority is determined as 0 for each test case i. During the second regression test ($t = 2$), the error detection rate measured during the first regression tests is set as a priority for the relevant test case. In later regression tests ($t > 2$), the error rate measured during the previous tests (i.e., EDR(i, t−1) and the predetermined priority value (i.e., R (i, t−1)) where the α parameter, which takes values between 0 and 1, assigns a weight for the last calculated fault detection rate with a predetermined priority value. The smaller the effect of −n. The higher α value allows more permanent memory of previous error rate calculations to be considered. When this value is 0, the priority of a test case is always equal to the error detection rate calculated in previous tests. At higher α values, the last and previously calculated error detection rates affect the priority calculation.

4 Evaluation

We conducted an experimental study on the OObeya software regression test data set, which we have obtained from Testinium, the method we propose within the scope of this study. Below, we explain the data set we use, the metric we use to evaluate the effectiveness of our research, and the results of the experimental study.

4.1 Dataset

For our study, the OObeya software regression test, which is under development, was used as a data set. The regression test consists of 14 test teams and 73 test scenarios within the test sets. A total of 90 error conditions are checked (Tables 3 and 4).

A regression test takes an hour or a few hours to run. The data set consists of the regression test results, which are performed consecutively over the weekend, lasting 60 days in total. The data about each test scenario, the number of errors it controls, the error time it detects, the test run time, and the moment of error detection are kept. In addition, we recorded error detection rate (EDR) and regression test score (t) data for each scenario obtained with test run results.

4.2 Effectiveness Measure

We need to use a measurement technique to evaluate the prioritization study effectiveness in our research questions. Our study used the APFD metric, which calculates the weighted average of the error percentage, to measure how quickly a

Table 3. Example of data set for test cases

Test cases	The number of errors it checks	The number of errors detected	Time to catch errors(s)	Execution time(s)
T1	5	2	20	50
T2	7	3	25	43
T3	4	1	10	52
T4	3	2	15	22
T5	2	1	12	30

Table 4. Example of data for running tests

Running the test	Date	Time	Total execution time(m)	Number of detected errors
Random order	05.04.2021	20:00	125	14
Order-1	10.04.2021	15:13	107	14
Order-2	15.04.2021	18:36	112	13
Order-3	20.04.2021	17:25	88	13
Order-4	25.04.2021	19:16	93	12
Order-5	30.04.2021	11:30	95	13

prioritized test package detects errors. [1] The weight value assigned to a test case increases in direct proportion to the rate of errors detected before this test case. APFD value is between 0 and 100; higher values indicate a high error detection rate. Below is an example calculated with the APFD metric. Suppose there are 10 errors in a software system, and there are only 5 test cases in the test set, and test cases can detect all errors together. The test cases are named TC1-TC2-TC3-TC4-TC5. The error-test case matrix generated is shown in Table 5.

Table 5. Test cases with detected faults

TESTS		FAULTS									
		1	2	3	4	5	6	7	8	9	10
TC1	Create a new dash-board	X						X			
TC2	Update dash-board	X					X	X		X	
TC3	Delete the test dashboard	X	X		X	X	X	X			X
TC4	Add jira widget					X					
TC5	Add sonar widget			X						X	X

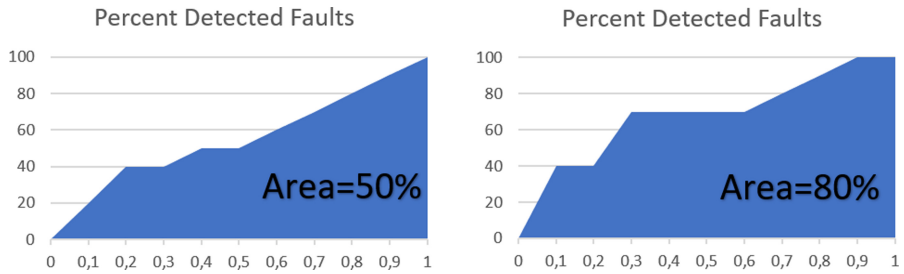

Fig. 4. Test case orders and APFD values

TC1 test case detects 1st and 7th errors, while TC2 test case detects 1st, 6th, 7th and 9th the errors. TC3 test case detects 7 errors at once. TC4 test case detects only 5th error, TC5 test case detects 3 errors. We created the ordering of test cases in 2 different ways, and APFD values were calculated. The x-axis shows the number of cases tested, the y-axis the ratio of the number of detected errors to the total number of errors. The area under the curve shows the weighted percentage of errors detected for all test cases. (APFD)

In Fig. 4, the APFD value calculated for TC1-TC2-TC3-TC4-TC5 is 50%. It is desired to reach the optimum APFD value by ordering the test cases in different ways. For the test cases ordered as TC3-TC5-TC2-TC1-TC4, the APFD value is seen as 80%. The fewer test cases are tested to detect all errors, and the APFD value will increase inversely. For this purpose, the best order in which a test can detect all errors by testing fewer test cases should be from the test case that detected the most error to the test case that caught the least.

4.3 Experimental Study Results

We used a selenium Gauge-based test automation system to run OObeya regression tests automatically. This system acts as a user in the web browser and compares the results obtained with reference images. Inconsistencies detected in the comparison are reported as errors. Test automation is run on the Testinium Tool.

The automatically tested test set contains hundreds of test steps. The test cases we used for case analysis were grouped under 14 groups and included 73 different scenarios. ($1 <= i <= 73$) It takes an average of one hour to perform the tests. Without applying the technique, we ran the regression test in the order of the original. ($t = 1$) The test completion time is 64 min and 21 s. The number of detected errors is 15. The calculated APFD value is 0.39. The ratio of the number of errors detected in Fig. 5 to the total number of errors on the y axis; error detection time is shown on the x-axis. We performed three consecutive tests using the prioritization technique we suggested in the regression tests. In the prioritization of the test scenarios, we used the error detection rates obtained from the test results data ($t = 1$) run in the original order. Data of TC1 and TC2 test cases are sampled in Table 6. In data processing, we obtained information

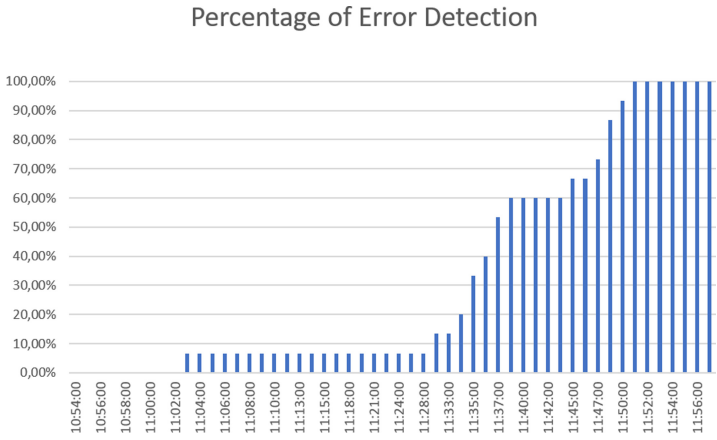

Fig. 5. Original order test results

about how many error conditions each test case controls. At the time $t = 1$, TC1 test case detected 1 out of 2 error conditions it controls as an error. The run time for TC1 is 46 s. The time to catch 1 error it detects is 21 s.

Table 6. EDR calculation example

Test cases	Number of states	Number of detected faults	Fault detecting time(s)	Execution time(s)	EDR
TC1	2	1	21	46	1,043478
TC2	2	1	40	46	0,630431

Calculation for test case TC1:

$$EDR = \frac{1}{2} + (1 - \frac{21}{46}) \tag{4}$$

Calculation for test case TC2:

$$EDR = \frac{1}{2} + (1 - \frac{40}{46}) \tag{5}$$

All test cases are reordered according to the EDR scores sampled in Table 6. We reran the regression test with the new order. The total run time of the test is 58 min, and the number of errors detected is 19. The APFD value for the test run with the new order is 0.56 ($t = 2$).

All test cases are reordered according to the EDR scores sampled in Table 6. We reran the regression test with the new order. The total run time of the test is 58 min, and the number of errors detected is 19. The APFD value for the test

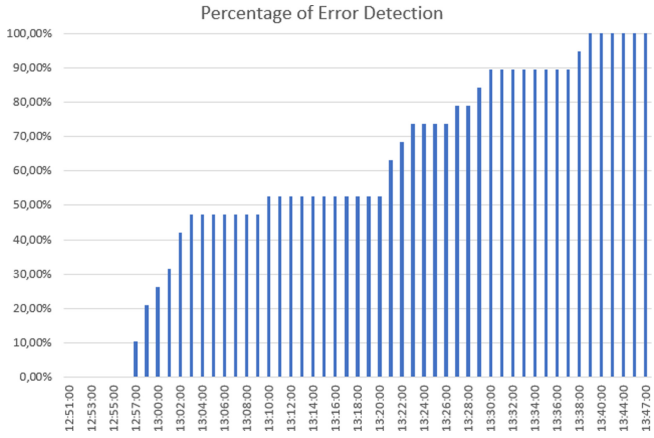

Fig. 6. Test results for t = 2

run with the new order is 0.56 ($t = 2$). According to the results obtained, the scores for each test scenario were calculated again. We created a new ordering for the regression test. The test run according to this order is 45 min and 23 s to complete. The number of errors detected is 13, and its APFD value is 0.64 ($t = 3$).

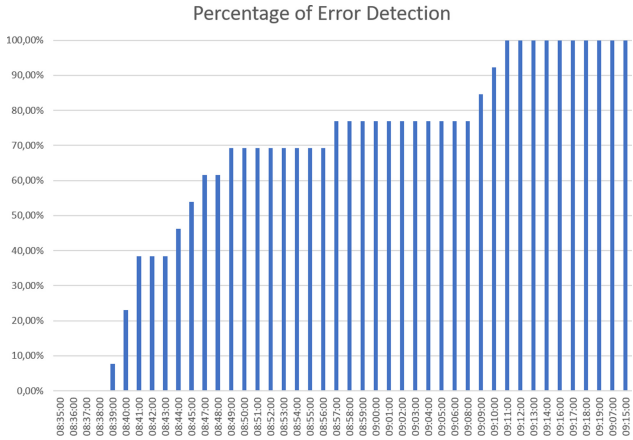

Fig. 7. Test results for t = 3

According to the last ordering results, the regression test completion time is 49 min and 30 s. The number of errors detected was 17, and the test APFD value was 0.74 ($t = 4$).

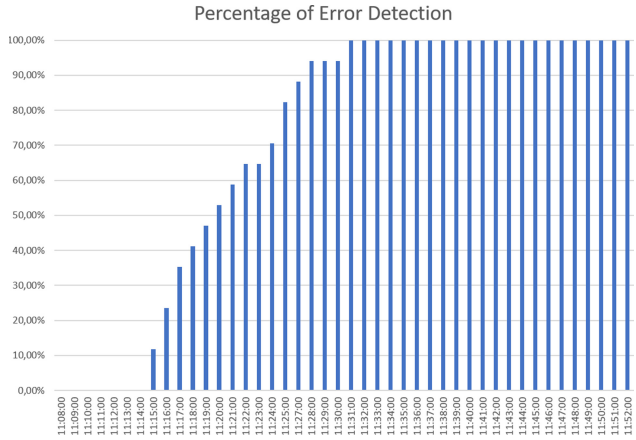

Fig. 8. Test results for t = 4

4.4 Discussion

This study aims to prioritize the test cases with the error detection rate obtained from the historical regression tests and detect the system's errors as soon as possible. It has been worked on OObeya software, where a real-time regression test is applied. The regression test cases are ordered with the method of prioritizing the test cases we recommend. We measured the results obtained with the APFD metric. When the regression test is tested with its original order (t = 1), as seen in Fig. 5, all test cases should be tested to detect all errors. APFD value calculated at time $t = 1$ is 39%. The regression test ($t = 2$) run with the first sequencing study in Fig. 6 lasted 59 min. 70% of the errors are detected at the 32nd minute of the test. At $t = 2$, it is close to the test period to detect all errors in the order in which our method is first applied. The exceptional situation encountered is that six more errors that were not in the system at $t = 1$ were detected. The fact that a few of these errors were at the bottom of the ordering extended this time. Increasing errors in the system reduce the effectiveness of the method. The method's efficiency increases when the number of errors is the same in the system or when the number of errors decreases. When looking at the results against this unfavorable situation, the APFD value calculated at $t = 2$ time is 56%. APFD value is increased compared to time $t = 1$.

When the results obtained with the second ordering in Fig. 7 are examined, it takes about 50 min to test all test cases ($t = 3$). While the test detected 17 errors at time $t = 2$, the test detected 13 errors at time $t = 3$. 80% of the errors were detected in the 12th min of the test, and the test detected all the errors in the 35th minute. APFD value calculated at time $t = 3$ is 64%. APFD value has increased compared to previous tests. The results of the test run with the third-ordering are shown in Fig. 8 ($t = 4$). The test detected all errors at the 22nd minute of the total test period that lasted 49 min. In less than half of the

total test time, the test detected all errors in the system. At $t = 4$, the test reached the highest APFD value with 74%.

Since each regression test run result affects the following test sequence, the APFD value obtained from the study draws an increasing graph.

4.5 Threats to Validity

In this study, while applying our test prioritization method, consecutive running is necessary to see the technique's effectiveness, as each test result data will affect the following test sequence order. Our tests on the Oobeya software, which is still under development, were applied on weekends and nights. This way, we ensured the reliability of the results.

Not all cases in our test case set are independent. In other words, to run some test scenarios, we must run the necessary test scenarios first. For this reason, we checked whether there exits any prerequisites within the test sequences, at the beginning of each regression test. In case of the deterioration of the prerequisite status, we changed the ordering of the test cases regardless of the prioritization score. Since we applied this situation in every test sequence, it did not affect the test results.

5 Conclusion and Future Works

In this study, we propose a viable method for prioritizing test cases and empirically examine how quickly we can detect errors. Our approach improves the error detection rate of test sets. This improvement can be applied even in the most complex situations. With the advancement in the error detection rate, the cost and effort spent on regression tests are also reduced.

We used our proposed method to prioritize software development regression tests. When we applied the regression test ordered according to our method, we saw that the error detection rate was increased. Without using any test case prioritization, the APFD value was 39%. We were able to increase the APFD value to 74% by applying our test prioritization method.

Our results indicate that regression tests that run without any test case prioritization use resources inefficiently. We should note that there may not be enough resources to evaluate all test cases. Therefore, it is essential to use resources efficiently and detect errors as soon as possible. We plan on taking into account various other features/metrics of the software systems, such as code coverage, in the future work.

Acknowledgment. We thank Saha Bilgi Teknolojileri Egitim Danismanlık San. Tic. A.S. R&D Center for providing the computational facilities that made this study possible.

References

1. Patel, K.M., et al.: A study of regression testing for trade me website. In: CS & IT Conference Proceedings. CS & IT Conference Proceedings (2021)
2. Rothermel, G., et al.: Test case prioritization: an empirical study. In: Proceedings IEEE International Conference on Software Maintenance-1999 (ICSM 1999). Software Maintenance for Business Change (Cat. No. 99CB36360), pp. 179–188. IEEE (1999)
3. Elbaum, S., Malishevsky, A.G., Rothermel, G.: Test case prioritization: a family of empirical studies. IEEE Trans. Softw. Eng. **28**(2), 159–182 (2002)
4. Testinium Homepage. http://www.testinium.com. Accessed 17 May 2021
5. Kurt, D., Sözer, H.: Geçmişe dönük hata tespit oranlarının zamanla değişen etkisine göre regresyon testlerinin önceliklendirilmesi. In: CEUR Workshop Proceedings. CEUR-WS (2014)
6. Ansari, A., et al.: Optimized regression test using test case prioritization. Procedia Comput. Sci. **79**, 152–160 (2016)
7. Alazzam, I., Nahar, K.M.O.: Combined source code approach for test case prioritization. In: Proceedings of the 2018 International Conference on Information Science and System, pp. 12–15 (2018)
8. Muthusamy, T., Seetharaman, K.: Effectiveness of test case prioritization techniques based on regression testing. Int. J. Softw. Eng. Appl. **5**(6), 113 (2014)
9. Staats, M., Loyola, P., Rothermel, G.: Oracle-centric test case prioritization. In: IEEE 23rd International Symposium on Software Reliability Engineering, vol. 2012, pp. 311–320. IEEE (2012)
10. Raju, S., Uma, G.V.: An efficient method to achieve effective test case prioritization in regression testing using prioritization factors. Asian J. Inf. Technol. **11**(5), 169–180 (2012)
11. Shahid, M., Ibrahim, S.: A new code based test case prioritization technique. Int. J. Softw. Eng. Appl. **8**(6), 31–38 (2014)
12. Zhang, L., et al.: Bridging the gap between the total and additional test-case prioritization strategies. In: 2013 35th International Conference on Software Engineering (ICSE), pp. 192–201. IEEE (2013)
13. Yoon, M., et al.: A test case prioritization through correlation of requirement and risk. J. Softw. Eng. Appl. **5**(10), 823 (2012)
14. Misra, S., Adewumi, A., Maskeliūnas, R., Damaševičius, R., Cafer, F.: Unit testing in global software development environment. In: Panda, B., Sharma, S., Roy, N.R. (eds.) REDSET 2017. CCIS, vol. 799, pp. 309–317. Springer, Singapore (2018). https://doi.org/10.1007/978-981-10-8527-7_25
15. Aljawabrah, N., et al.: Automated recovery and visualization of test-to-code traceability (TCT) links: an evaluation. IEEE Access **9**, 40111–40123 (2021)
16. Jonathan, O., Omoregbe, N., Misra, S.: Empirical comparison of cross-validation and test data on internet traffic classification methods. In: Journal of Physics: Conference Series, p. 012044. IOP Publishing (2019)
17. Jatana, N., Suri, B., Misra, S., Kumar, P., Choudhury, A.R.: Particle swarm based evolution and generation of test data using mutation testing. In: Gervasi, O., et al. (eds.) ICCSA 2016. LNCS, vol. 9790, pp. 585–594. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42092-9_44
18. Singhal, S., Suri, B., Misra, S.: An empirical study of regression test suite reduction using MHBG_TCS tool. In: Proceedings International Conference Computation Network Information (ICCNI), pp. 1–5 (2017)

19. Nawar, M.N., Ragheb, M.M.: Multi-heuristic based algorithm for test case prioritization. In: Murgante, B., et al. (eds.) ICCSA 2014. LNCS, vol. 8583, pp. 449–460. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09156-3_32
20. Lin, X., et al.: Test case minimization for regression testing of composite service based on modification impact analysis. In: Wang, G., Lin, X., Hendler, J., Song, W., Xu, Z., Liu, G. (eds.) WISA 2020. LNCS, vol. 12432, pp. 15–26. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-60029-7_2
21. Arora, P.K., Bhatia, R.: A systematic review of agent-based test case generation for regression testing. Arabian J. Sci. Eng. **43**(2), 447–470 (2018)
22. Onoma, A.K., et al.: Regression testing in an industrial environment. Commun. ACM **41**(5), 81–86 (1998)
23. Malishevsky, A.G., et al.: Cost-cognizant test case prioritization. Technical Report TR-UNL-CSE-2006-0004, University of Nebraska-Lincoln (2006)
24. Hao, D., et al.: A unified test case prioritization approach. ACM Trans. Softw. Eng. Methodol. (TOSEM) **24**(2), 1–31 (2014)
25. Jeffrey, D., Gupta, N.: Test case prioritization using relevant slices. In: 30th Annual International Computer Software and Applications Conference (COMPSAC 2006), pp. 411–420. IEEE (2006)
26. Jiang, B., et al.: Adaptive random test case prioritization. In: 2009 IEEE/ACM International Conference on Automated Software Engineering, pp. 233–244. IEEE (2009)
27. Rothermel, G., et al.: Prioritizing test cases for regression testing. IEEE Trans. Softw. Eng. **27**(10), 929–948 (2001)
28. Aktas, M.S., Kapdan, M.: Structural code clone detection methodology using software metrics. Int. J. Softw. Eng. Knowl. Eng. **26**(2), 307–332 (2016)
29. Aktas, M.S., Kapdan, M.: Implementation of analytical hierarchy process in detecting structural code clones. In: The 17th International Conference on Computational Science and its Applications (ICCSA 2017) (2017)
30. Ozturk, S., et al.: Test case prioritization related to code quality. In: The 2013 International Conference on Software Engineering Research and Practice (SERP-13) (2013)
31. Uygun, Y., et al.: On the large-scale graph data processing for user interface testing in big data science projects. In: 2020 IEEE Big Data, 6th International Workshop to Improve Big Data Science Project Team Processes (2020)
32. Tekin, C., et al.: Opinion mining on usability testing data. In: IEEE SIU Conference, Gaziantep, Turkey (2020)
33. Akpinar, P., et al.: Web application testing with model based testing method: case study. In: 2nd International Conference on Electrical, Communication and Computer Engineering, Turkey (2020)
34. Kapdan, M., Aktas, M., Yigit, M.: On the structural code clone detection problem: a survey and software metric based approach. In: ICCSA 2014: Computational Science and its Applications - ICCSA 2014, pp. 492–507 (2014)
35. Guveyi, E., Aktas, M., Kalipsiz, O.: Human factor on software quality: a systematic literature review. In: ICCSA 2020: Computational Science and its Applications - ICCSA 2020 (2020)
36. Guveyi, E., Aktas, M., Kalipsiz, O.: Impact of personal factors on software quality. In: UYMS 2020 (2020)