Research Article

Test Suite Reduction and Prioritization Framework in Regression Testing

Samaila Musa[1], Abu BakarMd Sultan[2]

[1]Federal University Gusau, Zamfara State, Nigeria
[2]Department of Software Engineering and Information System, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia

Article History: Received: 10 November 2020; Revised: 12 January 2021; Accepted: 27 January 2021; Published online: 05 April 2021

Abstract: Most of the test cases minimization reduced test cases during regression testing to generate new test suite to cover the same software requirements.The objective of this paper is to present new framework that integrate the idea of minimization and prioritization.Hence, reduction and prioritization able to reduce test cases based on the statements covered by the previous test cases to avoid redundancy.Beginning from the reduction of the test cases, followed by weighted prioritizationaccording to their usefulness.The framework was tested using sample test suite and the results obtained shown increases on the average percentage of faults detection (APFD). Future plan is to test on the larger size of test suite.

Keywords : Regression Testing, Test Suite Reduction, Prioritization, Framework

1. Introduction

According to Roger (2001) more than half of the total amount of the software develpment is spend on maintainingactivity of software. Rothermel and Harrold (1994) and Hussain et al., 2020assert that one of the most important phase in software maintenance is regression testing in order to be certain that amendments due to corrections or update did not change the initial functionality and the requirement. Test cases are use to evaluate the quality of the system by executing them. Sumalath and Raji (2014) state that measuring of the quality of the generated test cases needs to be conducted systematically.

The reduction and elimination of redundant test cases generated are what lead to test case minimization (TCM). TCM is the act of reducing the number of test cases using systematic method and procedure while maintaining the initial coverage criteria (Sumalath and Raji 2014). Yoo and Harman (2012) state that test suite reduction is aims at reducing the number of tests to test. However, the main objective of most of proposed algorithms is to reduce the test suite size

According to Musa et al. (2014a) after having the test case, the challenge in regression testing is the prioritization of the test cases by identifying and selecting of best ones from them, and prioritizing the test cases will result in less time of execution and increase the coverage of errors detection. Regression test case prioritization (RTCP) prioritized test cases for execution during regression testing efficiently. RTCP relied on the available resources to prioritize test cases since the position and nature of the errors are not given prio to the activity (Orso, Shi and Harrold 2004). Also Kim and Porter (2002) define RTCP as a method in which it can be structured with the aim of finding errors in good time in the test execution process. Since it become bigger in number as result of many changes and new versions of the existing one and only some part of it can be retested within given period.

RTCP technique is a process that allow the testers to arrange tests into certain order so that those with the most higher need are executed before the less need test case, and it can be utilized with tests selection when tests removal is satisfactory (Rothermel et al. 2001), also tests prioritization might increase the utilization of testing time more beneficial than non-prioritize when the process of re-executing the test cases is terminated without prior notice.

Rothermel et al. (2002) define the testsuite minimization problem as: Given a set of test cases T = t1, t2, … tn and set of requirements R = r1, r2, … rn which can be satisfied by the test cases in T, and a minimal subset of T that satisfies the same requirements as T itself.

Studies were conducted in regression test case minimization, one of them was conducted to generate a minimized number with the same coverage criteria as the original generated test cases (Ahmed 2016). Khan, Bora and Gupta (2017) developed a heuristic based testsuite minimization approach such that the size of the previous testsuite is reduced in order to generate new testsuite that will guarantee the same software requirements coverage that was achieved before the test suite minimization for an effective and efficient regression testing. But the approach minimized the test suite by throwing away unused testcases from the test suite according to different code coverage criteria's without prioritizing the test suite. Varadarajan et al. (2019) present a minimized test suite approach by eliminating unused test cases that tests same functionalities as other test case after generating the test cases based on the requirements. But approach eliminates test case by removing functionalities that were tested by the previous test cases and it doesn't used any criteria to order the testcases.

In this research we present a framework and an algorithm forRTC reduction and prioritization technique that reduce test cases based on the statements covered by the previous testcases to avoid redundancy of statements. After the reduction of the test cases, we prioritized the test cases based on their weight in order arrange them based on their usefulness.

2. Materials and Methods

This section describes the framework and algorithm for proposed RTC reduction and prioritization approach. Figure 2.1 shows the framework.

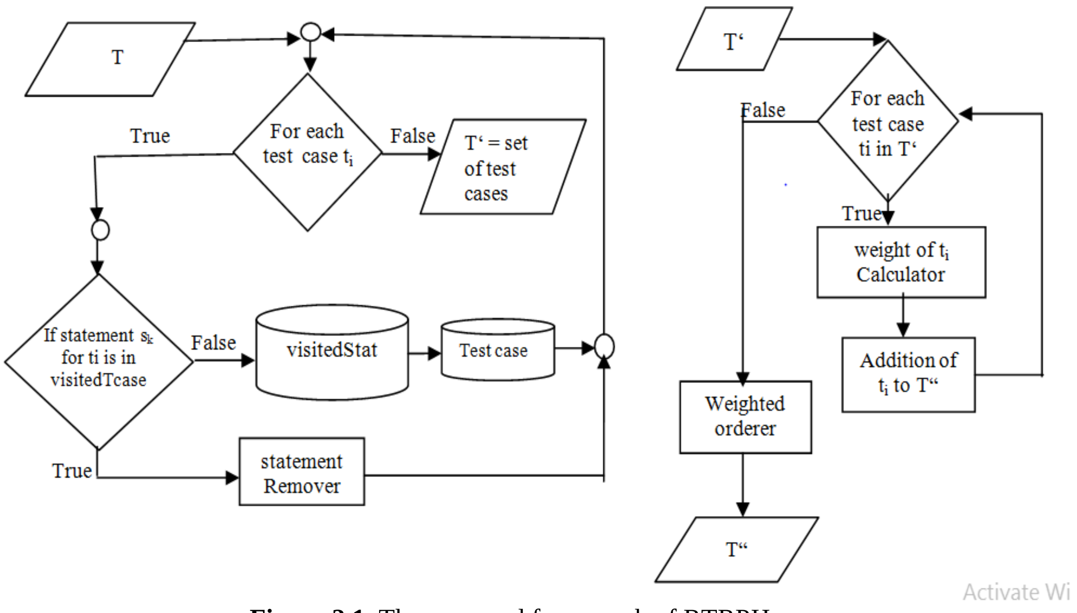

Figure 2.1. The proposed framework of RTRPHuce

Figure 2.1 presents the different components of the proposed framework. Algorithms 1 and 2 show the steps by steps to follow in implementing the proposed framework of Figure 2.1.

Algorithm 1 and 2 show the testsuite reduction and testsuite prioritization of the proposed approach. The generated test suite serves as input. For each testcase ti in T, If any of the previous test case contains the statement sk, delete the statement from the statements covered by ti. If the statement was not visited by any of the previous test cases, it should be added to the set of the visited test case. After executing all the test cases, we will have the remaining test cases those with statements not removed. This set of test cases T', serves as input to the prioritization part.

Algorithm 1 Test suite reduction

```
reducePriorHuce (T)
BEGIN
T = {set of generated test cases based on requirements}
T‘ = { }
visitedStat = { }
T“ = { }
// reduce the test suite T
For each test case $t_j$ in test suite T
Begin
    If statement $s_k$ is in visitedStat
        Remove the statement $s_k$
    Otherwise
        Add the statement $s_k$ to its test case
        Add the statement $s_k$ to the visitedStat
End
T‘ = {set of test cases with remaining statements}
priorReduceTest (T‘, T“)
END
```

Algorithm 2 Test suite prioritization

For each test case t_r, calculate it weight and add it to T" and finally order the test cases T".

```
priorReduceTest (T‘, T“)
BEGIN
T“
For each test case $t_r$ in T‘
Begin
    Get the total number of statements and assign as the weight of $t_r$
    Add the test case $t_r$ to T“
End
Arrange the test cases in T“ based on their weight
Return T“
END
```

3. Results and Discussion

The end goal is to guarantee that the regression test case reduction and prioritization algorithms work legitimately; of course, we give a preparatory acceptance of the algorithms in this section prior to their useful usage and observational/empirical assessment. This proof has been prepared by brief reduction and prioritizing tests for a program written in java program. The feasible paths presented for each test case and the information covered of triangle code based on the test suite is shown in Figure 3.1.

$$T_1 = \{s_1\ s_2\ s_3\ s_4\ s_5\}$$
$$T_2 = \{s_1\ s_2\ s_3\}$$
$$T_3 = \{s_1\ s_2\ s_6\ s_7\ s_8\ s_9\ s_{10}\}$$
$$T_4 = \{s_1\ s_2\ s_4\ s_9\ s_{10}\}$$
$$T_5 = \{s_1\ s_2\ s_{11}\ s_{12}\ s_{16}\ s_{17}\ s_{18}\ s_{19}\ s_{20}\}$$
$$T_6 = \{s_1\ s_2\ s_{21}\ s_{22}\ s_{23}\ s_{24}\ s_{25}\ s_{26}\}$$
$$T_7 = \{s_1\ s_2\ s_{21}\}$$
$$T_8 = \{s_1\ s_2\ s_{21}\ s_{27}s_{28}\ s_{29}s_{30}\ s_{31}\ s_{32}\ s_{33}\ s_{34}\ s_{35}\}$$
$$T_9 = \{s_1\ s_2\ s_{21}\ s_{27}\ s_{30}\ s_{33}\}$$
$$T_{10} = \{s_1\ s_2\ s_6\ s_{13}\ s_{14}\ s_{15}\ s_{36}\ s_{37}\ s_{38}\ s_{39}\}$$

Problem

Given T = $\{T_1\ T_2\ T_3\ T_4\ T_5\ T_6\}$, find T' as reduced test case and T" as the prioritized test cases.

Using Algorithm 1, for the first iteration:

$T_1 = \{\ s_1\ s_2\ s_3\ s_4\ s_5\}$

visitedStat = $\{\ s_1\ s_2\ s_3\ s_4\ s_5\}$

For the second iteration:

$T_1 = \{\ s_1\ s_2\ s_3\ s_4\ s_5\}$

$T_2 = \{\ \}$

visitedStat = $\{\ s_1\ s_2\ s_3\ s_4\ s_5\}$

For the third iteration:

$T_1 = \{\ s_1\ s_2\ s_3\ s_4\ s_5\}$

$T_2 = \{\ \}$

$T_3 = \{s_6\ s_7\ s_8\ s_9\ s_{10}\}$

visitedStat = $\{\ s_1\ s_2\ s_3\ s_4\ s_5\ s_6\ s_7\ s_8\ s_9\ s_{10}\}$

The iterations will continue up to 10th.

For the tenth iteration:

$T_1 = \{\ s_1 s_2\ s_3\ s_4\ s_5\}$

$T_3 = \{s_6\ s_7\ s_8\ s_9\ s_{10}\}$

$T_5 = \{s_{11}\ s_{12}\ s_{16}\ s_{17}\ s_{18}\ s_{19}\ s_{20}\}$

$T_6 = \{s_{21}\ s_{22}\ s_{23}\ s_{24}\ s_{25}\ s_{26}\}$

$T_8 = \{s_{27} s_{28}\ s_{29} s_{30}\ s_{31}\ s_{32}\ s_{33}\ s_{34}\ s_{35}\}$

$T_{10} = \{s_{13}\ s_{14}\ s_{15}\ s_{36}\ s_{37}\ s_{38}\ s_{39}\}$

visitedStat = $\{\ s_1\ s_2\ s_3\ s_4\ s_5\ s_6\ s_7\ s_8\ s_9\ s_{10}\ s_{11}\ s_{12}\ s_{13}\ s_{14}\ s_{15}\ s_{16}\ s_{17}\ s_{18}\ s_{19}\ s_{20}\ s_{21}\ s_{22}\ s_{23}\ s_{24}\ s_{25}\ s_{26}\ s_{27}\ s_{28}\ s_{29} s_{30}\ s_{31}\ s_{32}\ s_{33}\ s_{34}\ s_{35}\ s_{36}\ s_{37}\ s_{38}\ s_{39}\}$

The reduced test cases T'with remaining statements are:

T' = $\{T_1\ T_3\ T_5\ T_6\ T_8\ T_{10}\}$

The next action to perform is to calculate the weight of each test case in T'.

$T_1 = 5$, $T_3 = 5$, $T_5 = 7$, $T_6 = 6$, $T_8 = 9$, $T_{10} = 7$

The prioritize test cases T":

T" = $\{T_8\ T_5\ T_{10}\ T_6\ T_1\ T_3\}$

The results of our proposed approach (RTRPHuce)are evaluated using Eq. [1]APFD as in the previous researches (Musa etal. 2014a, 2014b, 2015a, 2015b) in order to effectively evaluate the performance of RTRPHuce, Non-Reduced (NR) without reduction of statements, and Non-Prioritized (NP).

i.e., T" = $\{T_8\ T_5 T_{10}\ T_6\ T_1\ T_3\}$,T = $\{T_1\ T_3\ T_5\ T_6\ T_8\ T_{10}\}$, T' = $\{T_1\ T_3\ T_5\ T_6\ T_8\ T_{10}\}$

$$APFD = 1 - \frac{Tf1 + Tf2 + \cdots + Tfm}{\#N * \#killed\ mutants} + \frac{1}{2 * \#N} \qquad [1]$$

For T" = $\{\ T_8\ T_5\ T_{10}\ T_6\ T_1\ T_3\}$

APFD =55.7%

For T' = $\{T_1\ T_3\ T_5\ T_6\ T_8\ T_{10}\}$

APFD = 45.5%

For T = $\{T_1\ T_3\ T_5\ T_6\ T_8\ T_{10}\}$

APFD = 22.0%

The results of APFD show that the retesting of all test cases without reduction and prioritization (T) is 22.0%, while that reduction of in statements (T') is 45.5%, and (RTRPHuce) the reduced and prioritized approach (T") is 55.7%.

4. Conclusion

The results show that the proposed approach*RTRPHuce* yielded better results in term APFDas compared to reduced   and retest-all approaches. The RTRPHuceframework was tested using small size test cases to show it workability. Besides, the order of the initial test suite my also affect the results. In conclusion, this framework can be adopted for regression testing since it increases the rate of faults detections. Our future plan is to test on the larger size of test suite.

References

1. Ahmed, B. S. 2016. Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing. Int. J. Eng. Sci. Technol., vol. 19, no. 2, p. 737–753.
2. Hussain, A., Manikanthan, S.V., Padmapriya, T., Nagalingam, M. (2020). Genetic algorithm based adaptive offloading for improving IoT device communication efficiency. Wireless Networks, 26 (4), pp. 2329-2338.
3. Khan, F. A., Bora, D. J. and Gupta, A. K. 2017. An Efficient Heuristic Based Test Suite Minimization Approach Indian Journal of Science and Technology,*Vol 10(29), DOI: 10.17485/ijst/2017/v10i29/106374, August 2017 ISSN (Print): 0974-6846 ISSN (Online) : 0974-5645.*
4. Kim, J. M. and A. Porter, 2002. A history-based test prioritization technique for regression testing in resource constrained environments, In Proceedings of the 24th International Conference on Software Engineering, New york, NY, USA, 119-129.
5. Musa, S., A. M. d. Sultan, A. A. A. Ghani, S. Baharom. 2014a. Regression Test Case Selection & Prioritization Using Dependence Graph and Genetic Algorithm.   *IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 3, Ver. IV, PP 38-47.*
6. Musa, S., A. M. d. Sultan, A. A. A. Ghani, S. Baharom. 2014b. A Regression Test Case Selection and Prioritization for Object-Oriented Programs using Dependency Graph and Genetic Algorithm, Research Inventy: International Journal of Engineering And Science, 4(7), 54-64.
7. Musa, S., A. M. d. Sultan, A. A. A. Ghani, S. Baharom. 2015a. The Effects of Replacement Strategies of Genetic Algorithm in Regression Test Case Prioritization of Selected Test Cases. International Journal of Soft Computing, 10(6): 359-368. DOI: 10.3923/ijscomp.2015.359.368.
8. Musa, S., A. M. d. Sultan, A. A. A. Ghani, S. Baharom.2015b. SoftwareRegression Test Case Prioritization for Object-Oriented Programs using Genetic Algorithm with Reduced-Fitness Severity. Indian Journal of Science and Technology, 8(30), DOI: 10.17485/ijst/2015/v8i30/IPL0984
9. Orso, A., N. Shi, and M. Harrold. 2004. Scaling regression testing to large software systems,ACM SIGSOFT, Twelfth International Symposium on Foundations of Software Engineering, 29(6), 241–251.
10. Roger, S. P. 2001. Software engineering: A practitioner’s approach (Fifth Edition. McGraw-Hill Publisher, New York, America).
11. Rothermel, G. and M. Harrold. 1994. Selecting regression tests for object-oriented software, International Conference on Software Maintenance, Victoria, CA, 14–25.
12. Rothermel, G., Roland, H.U., Chu, C., & Harrold, M.J. 2001. Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering,* 27(10), 929-948
13. Sumalath, G. S. V. P. , and V. Raji, 2014. “Model Based Test Case Optimization of UML Activity Diagrams using Evolutionary Algorithms,” Int. J. Comput. Sci. Mob. Appl., vol. 2, no. 11, pp. 131–142.
14. Varadarajan, A., Jain, K., Kanakasabapathy, R. and Rajarathinam, m. 2019. Software Testing With Minimized Test Suite
15. Yoo, S. and M. Harman, 2012. Regression testing minimization, selection and prioritization: a survey, Software Testing, Verification and Reliability. pp.67–120, DOI: 10.1002/stvr.430