Test Suite Reduction and Prioritization Framework in Regression Testing

Samaila Musa¹, Abu BakarMd Sultan²

¹Federal University Gusau, Zamfara State, Nigeria
²Department 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 prioritization according 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 development is spend on maintaining activity of software. Rothermel and Harrold (1994) and Hussain et al., 2020 assert 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 prior 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 test suite minimization problem as: Given a set of test cases $T = t_1, t_2, \ldots t_n$ and set of requirements $R = r_1, r_2, \ldots r_m$ 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 test suite minimization approach such that the size of the previous test suite is reduced in order to generate new test suite 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 for RTC 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 $t_i$ in $T$, If any of the previous test case contains the statement $s_k$, delete the statement from the statements covered by $t_i$. 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.
Test Suite Reduction and Prioritization Framework in Regression Testing

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_i, 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_i in T'
Begin
Get the total number of statements and assign as the weight of t_i
Add the test case t_i 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_26 s_37 s_38 s_39}
```
Problem  
Given $T = \{T_1, T_2, T_3, T_4, T_5\}$, 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 = $\{s_1, s_2, s_4, s_5\}$

For the second iteration:
$T_1 = \{s_1, s_2, s_3, s_4, s_5\}$  
$T_2 = \{\}$  
visitedStat = $\{s_1, s_2, s_4\}$

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

The iterations will continue up to 10th.

For the tenth iteration:
$T_1 = \{s_3, s_2, s_4, s_5\}$  
$T_2 = \{\}$  
$T_3 = \{s_6, s_7, s_8, s_9, s_{10}\}$  
$T_4 = \{s_1, s_4, s_5, s_6, s_7, s_8, s_9\}$  
visitedStat = $\{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9\}$

The reduced test cases $T'$ with remaining statements are:
$T' = \{T_1, T_2, T_3, T_4, T_5, T_6, T_{10}\}$

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

$T_1 = 5, T_2 = 5, T_3 = 7, T_5 = 6, T_6 = 9, T_{10} = 7$

The prioritized test cases $T''$:
$T'' = \{T_8, T_3, T_10, T_6, T_1\}$

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

\[ APFD = 1 - \frac{\sum_{i=1}^{n} T_i f_i + T_{10} f_{10} + \cdots + T_{m} f_{m} + 1}{2N+} \]

For $T_{10}$  
$APFD = 55.7\%$

For $T'$  
$APFD = 45.5\%$

For $T''$  
$APFD = 22.0\%$

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\%.

1950
4. Conclusion

The results show that the proposed approach RTRPHuce yielded better results in terms of APFDas compared to reduced and retest-all approaches. The RTRPHuce framework was tested using small size test cases to show it workability. Besides, the order of the initial test suite may 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