Comparing Types of Regression Testing

Tyler Dishman
University of Kentucky
Lexington, KY, USA
tjdi226@g.uky.edu

ABSTRACT

Regression testing can be costly but it is undoubtedly needed for maintaining and modifying software. Regression tests show us that the changes are accurately correct without changing the other functionalities. As a software gets older, tests can become more expensive than the actual software needs, so it is important to know how to reduce the costs. Some would argue that testing is the most important and frustrating part of any software development, especially when talking about security vulnerabilities. If we can make speed up the time it takes to run testing, we can reduce the cost in essence. In my article I will present the different methods of regression testing while comparing them in the following areas: Cost, efficiency, and security. The types of regression testing we will be comparing are Test Suite Reduction and Regression Test Selection. Both Methods reduce the # of tests ran based on importance and both run less tests than the entire suite while still catching the same number of faults as the full suite would. But our goal is to find out which method is the better approach.

1. Introduction

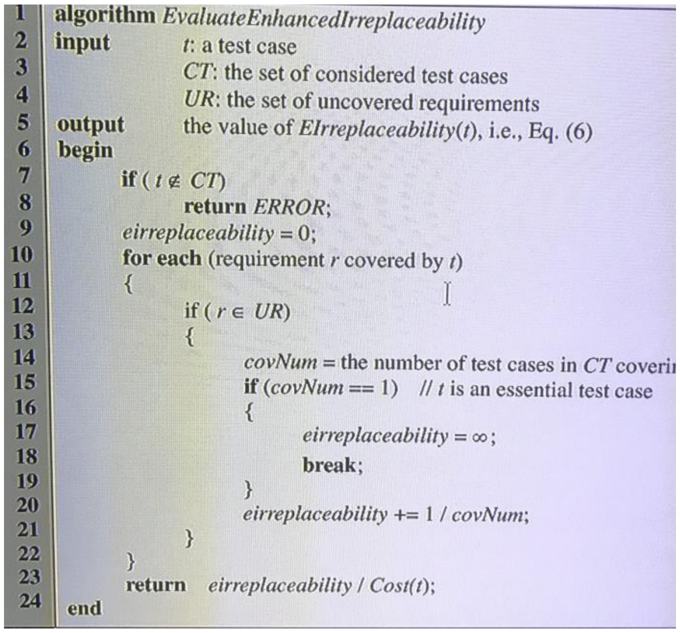
It seems today in 2020 that regression testing is becoming more important in Software development. Not only is the need for upgrades and changes at a higher demand than ever, but security is also becoming more difficult to maintain. Most test selection techniques—minimization and dataflow techniques among them—are not designed to be safe. Techniques that are not safe can fail to select a test case that would have revealed a fault in the modified program [3]. We see many methods for speeding up regression testing but I will compare; test-suite reduction and regression test selection [2]. Basically, these methods both run individual pieces of tests based on importance. In comparing these methods, I take the same approach proposed in [2] To compare test-suite reduction and regression test selection I measure how much time each method takes and the number of errors each find.

2. Test-Suite Reduction

Let's first take a look at the methods in Test-Suite Reduction and talk about the ones that can limit costs by identifying tests that are not needed. In [4] they propose a method that uses a cost-aware test case metric, called Irreplaceability and its enhanced version, called EIrreplaceability, to evaluate the possibility that each test case can be replaced by others during test suite reduction. They construct a cost-aware framework that incorporates the concept of test irreplaceability into some well-known test suite reduction algorithms [Figure 1]. Based on the results of their data the effectiveness of the cost-aware framework is evaluated by the subject programs and test suites found from the Software-artifact Infrastructure Repository. The empirical results reveal that the presented algorithms produce representative sets that normally incur a low cost to yield a high level of test coverage. The presented techniques did enhance the capability of the traditional reduction algorithms to reduce the execution cost of a test suite. Especially for the additional Greedy algorithm, the presented techniques decrease the costs of the representative sets by 8.10–46.57%.

3 Regression Test Selection

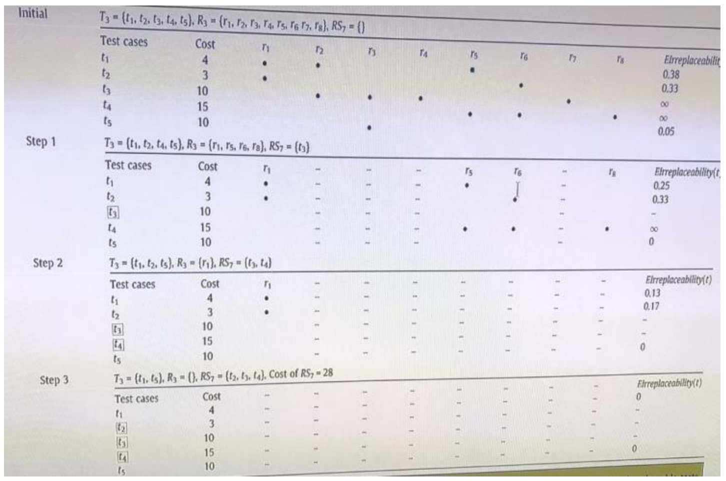
In Regression Test Selection, techniques operate on two software revisions. The goal of regression test selection is to run only the tests that may be affected by the changes made between the two revisions. Regression test selection techniques are often designed to be safe with respect to the changes, i.e., a test is not selected to be run only if the software changes cannot affect the outcome of the test [2]. Developers use regression test selection to get faster testing while focusing on detecting change-related faults. By identifying the most affected areas of a program we better our testing on cost and time. When an individual module is changed, we know that other areas of the program will be affected. Safe techniques will select the test cases that cause a modified program to produce different output other than the original program.[5] This is handy because it allows us to know if any breaches happen as we test new things. In article [2] they compare both Regression Test Selection and Test Suite Reduction methods on 17 source projects on Github [1] and they found that for Regression Testing



```
algorithm EvaluateEnhancedIrreplaceability
input    t: a test case
         CT: the set of considered test cases
         UR: the set of uncovered requirements
output   the value of EIrreplaceability(t), i.e., Eq. (6)
begin
    if ( $t \notin CT$ )
        return ERROR;
    eirreplaceability = 0;
    for each (requirement r covered by t)
    {
        if ( $r \in UR$ )
        {
            covNum = the number of test cases in CT coveri
            if (covNum == 1)    // t is an essential test case
            {
                eirreplaceability = $\infty$;
                break;
            }
            eirreplaceability += 1 / covNum;
        }
    }
    return   eirreplaceability / Cost(t);
end
```

[Figure 1] Algorithm for evaluating the EIrreplaceability of a test case. [4]

EvaluateEnhancedIrreplaceability checks if the number of test cases satisfying r is equal to 1. If the value is equal to 1 then we let EIrreplaceability(t)=1 and get out of the for-each loop, thus indicating that t is an essential test case that cannot be replaced by any other test cases. The time complexity we get is O(m n min(m, n) k).



Initial $T_3 = \{t_1, t_2, t_3, t_4, t_5\}, R_3 = \{r_1, r_2, r_3, r_4, r_5, r_6 r_7, r_8\}, RS_7 = \{\}$

Test cases	Cost	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	EIrreplaceabilit
t_1	4	•	•			•				0.38
t_2	3	•					•			0.33
t_3	10		•	•	•			•		∞
t_4	15					•	•		•	∞
t_5	10			•						0.05

Step 1 $T_3 = \{t_1, t_2, t_4, t_5\}, R_3 = \{r_1, r_5, r_6, r_8\}, RS_7 = \{t_3\}$

Test cases	Cost	r_1	–	–	–	r_5	r_6	–	r_8	EIrreplaceability(t
t_1	4	•	–	–	–	•		–		0.25
t_2	3	•	–	–	–		•	–		0.33
$\boxed{t_3}$	10		–	–	–			–		–
t_4	15		–	–	–	•	•	–	•	∞
t_5	10		–	–	–			–		0

Step 2 $T_3 = \{t_1, t_2, t_5\}, R_3 = \{r_1\}, RS_7 = \{t_3, t_4\}$

Test cases	Cost	r_1	–	–	–	–	–	–	–	EIrreplaceability(t)
t_1	4	•	–	–	–	–	–	–	–	0.13
t_2	3	•	–	–	–	–	–	–	–	0.17
$\boxed{t_3}$	10		–	–	–	–	–	–	–	–
$\boxed{t_4}$	15		–	–	–	–	–	–	–	–
t_5	10		–	–	–	–	–	–	–	0

Step 3 $T_3 = \{t_1, t_5\}, R_3 = \{\}, RS_7 = \{t_2, t_3, t_4\}$, Cost of $RS_7 = 28$

Test cases	Cost	–	–	–	–	–	–	–	–	EIrreplaceability(t)
t_1	4	–	–	–	–	–	–	–	–	0
$\boxed{t_2}$	3	–	–	–	–	–	–	–	–	–
$\boxed{t_3}$	10	–	–	–	–	–	–	–	–	–
$\boxed{t_4}$	15	–	–	–	–	–	–	–	–	–
t_5	10	–	–	–	–	–	–	–	–	0

[Figure 2] Reduction Steps from using their algorithm [4]

4 Conclusion

Each method studied proved to be a good method but our goal was to find the best of the two. The conclusion made from [2] claimed Their results on 17 open-source projects show the following three conclusions. 1. Regression test selection on average selects less tests than the test-suite reduction. 2. Test suite reduction can lose change-related fault detection capability (of up to 5.93%), while Safe regression test selection has no loss. 3. Our proposed selection of reduction approach provides the greatest speed-up, though with the same loss in change-related fault-detection capability as the reduced test suite, and has about the same selection ratio from the reduced test suite as regression tests selection has from the full test suite. In summary, the results showed that if only one approach must be chosen, either test-suite reduction or regression tests election, to speed up regression testing, the test engineer should choose regression test selection, because it selects fewer tests and preserves change-related fault-detection capability, where the faults are most likely to appear. If there is a need to speed up testing even further, then combining both test suite reduction and regression test selection is a worthwhile approach that provides even better cost saving in the number of tests as long as one is willing to tolerate the possible losing fault-detection capability for some changes in software. I found their method to be an accurate representation of an area I had never looked into this in depth. I always assumed that testing was a one-track area and that when doing so there would be a stand-alone winner for choosing the right method. With better methods we can hopefully cut down the cost of the overall system. I am certainly no expert in the but I feel confident that with all the new algorithms people are coming up with, better methods will be available soon.

REFERENCES

[1] GitHub. https://github.com/.

[2] August Shi, Tiffany Yung, Alex Gyori, Darko Marinov. Comparing and Combining Test-Suite reduction and Regression Test Selection. In ESEC, 2015.

[3] Todd L. Graves, Mary Jean Harrold, Jung-Min Kim, Adam Porter, Greg Rothermel. An Empirical Study on Regression Test Selection Techniques. In *Proceedings of ACM* 2001.

[4] Chu-Ti Lin, Kai-Wei Tang, Gregory M. Kapfhammerc. Test Suite Reduction methods that Decrease Regression Testing Costs by Identifying Irreplaceable Tests. 2014.

[5] Regression Test Selection Techniques. 2019. https://www.professionalqa.com/regression-test-selection