TEXAS STATE UNIVERSITY

Regression Testing

A Survey of Existing Methods

Jesse J. Bruni
2/28/2012

Abstract
As programs are written the need to maintain the code from one version to the next, as well as the need to discover any faults introduced by the changed code is addressed by Regression testing. In this paper we will examine the various techniques of regression testing, state the pros and cons of each technique, as well as comparisons between them.

I. Introduction

Regression testing is defined as any type of testing which is done on a software program after changes have been made to the code [1]. This testing is performed in order to ensure that the changes made to the code have not inadvertently caused previously working parts of the code to malfunction. Regression tests are often performed to examine whether a change added in a new version of a program will cause other portions of the program to malfunction. The common practice is to re-run previously run tests which exposed bugs in order to determine whether fixed bugs have re-emerged, or the program has been adversely affected in any other way [5]. The problem emerges in the effort to keep costs low, while still providing a test suite capable of thoroughly testing the program. Regression test suites for larger and more complex programs may often examine only a small portion of the program behavior [2].

The main reason for this is that manually generating test cases that can cover the majority of a complex program is difficult and time consuming [1]. Developers often focus on the program's main functionality when writing test cases for regression testing and allow other methods, such as beta testing, to ensure the rest of the program is working [2].

In order to combat these costs and still provide a thorough test suite many researchers suggest that regression testing be automated [1], [2], [3], often using programs such as "JUnit" [1]. Whether or not testing is automated there tend to be four main techniques to approach regression testing [4]. These methods are:

1.) Retest All

This technique is considered a "Safe" method, and involves taking all the tests previously run for the older version of the program and re-running them on the new version. This method can be

quite expensive and wasteful as often not all tests from the previous test suite are applicable to the new software version [5].

2.) Regression Test Selection

Also called “Minimization Techniques” [5], this method attempts to cut down the test suite for the original program and only use those tests which are most likely to expose a bug or reveal a fault in the new program. There are many ways to approach test selection as this is one of the more popular methods used in creating test suites for regression testing [4].

3.) Test Case Prioritization

This method assigns priorities to the tests in the suite in order to cut down on costs and runs test in order from those with the most priority to those with the least. Those with the highest priority are those which are most likely to expose bugs quickly [4].

4) Hybrid Approach

This approach is a mix of regression test selection, and test prioritization, cutting out unnecessary test, and choosing only those which are most likely to expose bugs, and then prioritizing that new test suite [4].

Knowing these various methods which we have at our disposal the problem then becomes which method to choose. Throughout the rest of this paper we will explain these methods in more detail, and compare them to one another. The purpose will be to help the reader understand what testing options are available to a software developer, as well as which method may be the most prudent choice based on the size of the program being developed, it’s purpose, and complexity. We will also explain the difference between “safe” and “non-safe” [5] techniques.

II. Regression Testing Techniques

In this section we will give more detailed explanations of the four techniques outlined in section I. We will first, however, define what is meant by calling a technique “safe”.

Safe vs. Unsafe Techniques

A safe regression testing technique is one which will keep any and all test cases which can reveal a bug or fault in the program. Most regression testing techniques are not designed to be 100% safe [5].

Unsafe techniques are considered to be such because the algorithm used to implement them may choose to not include tests which would have exposed a fault in the program. This may be considered acceptable depending on how much time and money the unsafe technique saves compared to how few faults are missed [4],[5],[6].

A. Retest All

The retest all technique is perhaps the simplest and, depending on the size of the program can be the most cost-effective [5]. Because it is a safe technique, (meaning if a previously run test can expose a bug in the new version of the software the bug will be exposed) it is often used when test suites are not very large. In order for a different method to be more effective than the retest-all technique the cost of selecting or prioritizing tests to run, as well as the cost of running those tests must be less than the cost of simply re-running all tests from the original test suite [6].

B. Regression Test Selection

Perhaps more research and studies have been done on this technique than any other. The idea of keeping costs low when testing is of paramount importance to any company, and reducing the number of tests needed to maximize fault detection can certainly help cut down on costs. The

biggest issue with test selection is that by throwing out test cases you risk throwing out a test that, if run, may have detected a fault in the new version of the program [4],[5],[6]. Another concern is that the cost of selecting a test suite to run, as well as actually running the tests may end up costing more than just re-running all the original tests.

[6] suggests that in order to analyze regression test techniques we must create four categories or criteria for judging their effectiveness. These categories are, *Inclusiveness, Precision, Efficiency,* and *Generality*.

Inclusiveness measures how well a given technique can select appropriate tests from the original test suite that will expose bugs in the new version of the program. A technique which is 100% inclusive would be considered a safe technique.

Precision measures how well a technique determines appropriately which tests to cut from the original test suite as obsolete tests or tests which will not reveal faults in the new program version.

Efficiency measures the cost, in terms of system resources, of the technique.

Generality measures how well the technique actually functions in a wide range of situations.

Whether these four criteria are used for test selection or not the bottom line is that any method of test selection used must still cost less than retest-all in order to risk the chance of letting faults go undetected.

C. Test Case Prioritization

[4] divides Test Case Prioritization into 3 subcategories *Comparator Techniques, Statement Level Techniques ,* and *Function Level Techniques*.

Comparator Techniques include random ordering, where test cases are prioritized randomly, and optimal ordering, where test cases are prioritized based on their rate of fault detection.

Statement Level Techniques prioritize tests based on the number of program statements a test covers. The test which covers the most program statements gets the highest priority.

Function Level Techniques prioritizes tests based on the number of program functions a test includes. A test which tests multiple functions will receive a higher priority than a test which can only test a single function.

Test case prioritization requires good algorithms to be written in order to organize and prioritize tests effectively. Much research has been done on the subject of test case prioritization algorithms [4].

D. Hybrid Approach

The hybrid approach requires algorithms to be written which can both select tests from the original test suites as well as prioritize these tests. Various techniques have been proposed which incorporate a hybrid approach including the technique proposed by [7].

III. Technique Comparisons

A good comparison of regression testing techniques will compare the two main factors in regression testing; the total effectiveness of the technique, and the cost of the technique.

Since there are far too many techniques in each subcategory that could be compared to each other we will instead focus on comparing the main ideas from one category to another. Each of

these examples will be compared against the retest-all technique. Before we do this, however, we will give a brief description of the pros and cons of each technique.

A. Pros and Cons

Retest-All

Pros: Safe technique, low cost on small programs, easy to implement.

Cons: Cost increases dramatically as the size of the program increases.

Regression Test Selection

Pros: Costs less than retest-all on most large programs, if implemented properly can be a safe technique (such as [8]).

Cons: Implementation Algorithms can be quite complex and the cost of implementing along with executing may exceed the cost of retest-all.

Test Case Prioritization

Pros: Provides good program coverage in a short period of time; safe technique.

Cons: Not as effective at cutting costs as Regression Test Selection, or the Hybrid Approach. Complicated Algorithms.

Hybrid Approach

Pros: Costs tend to be lower than that of any other method as program sizes get very large. May or may not be safe depending on implementation.

Cons: Most complicated algorithms of any other method. Not often worth implementation costs on small programs.

B. Comparisons

[6] gives us many statistics and comparisons between various Regression Test Selection Techniques and the Retest-All method. These outcomes of these studies varied based on the test selection method used, as well as the program the tests were written for. In many cases there was no Regression Test Selection method was as cost effective and safe as retest-all. However there were other cases where an RTS method proved to be cheaper (in terms of system resources) and still safe.

As more research has been done and algorithms refined, RTS methods have been devised which are 100% inclusive and cheaper than retest-all such as the method discussed in [8].

Since the Test Case Prioritization technique does not cut down on the size of the test suite its main advantage over retest-all is the ability of prioritization techniques to discover faults quickly, as the sooner a fault is discovered the sooner it can be dealt with and corrected. Several search algorithms have been proposed and perhaps a good comparison of test case prioritization is in fact a comparison of those algorithms. However we do not have enough space in this paper to give an adequate explanation of the various search algorithms currently being employed as well as a comparison between them. For a detailed breakdown of these algorithms one should see [9].

According to most sources [4],[6],[9], test case prioritization, if used as the only method of improving regression testing, is most useful in longer programs where the need to find faults in the program quickly is of utmost importance.

[7] compares the difference between using a minimization technique and a prioritization technique on a test suite that has already been reduced using an RTS method. The main idea is that one method is not inherently better than the other; simply that one method may be preferable

to another depending on the circumstances of the situation. For example if one cannot afford to exercise many regression tests a prioritization method (in which the reduced test suite is ordered with those most likely to expose bugs having the highest priority) would be the best method. When compared with a retest-all method the cost of implementing multiple techniques (selection and prioritization), is clearly greater than that of selecting only one technique, unless we have a large program, so a hybrid approach should typically only be used on when the level of detail required in the modified test suite, and the ordering of the tests in the test suite is such that 8 maximum fault exposure is of paramount importance, or unless we have a program with sizes so large that minimization of the test suite is not enough and prioritization is also required.

V. Conclusion

Throughout this paper we have explored the methods and tools used by software developers to perform regression testing. We have seen the simplest method, retest-all, and examined its pros and cons. We have seen that due to its simplicity it will likely continue to be used on smaller programs.

We have also seen the Regression Test Selection method, and seen four criteria for evaluating the method, *Inclusiveness, Precision, Efficiency,* and *Generality*. We have seen that these methods may end up being more expensive than retest-all, and we have seen methods that are 100% inclusive and in most cases cost less than retest-all such as the methods of [8].

We have also discussed Test Case Prioritization and divided it into three subcategories, *Comparator Techniques, Statement Level Techniques,* and *Function Level Techniques.* We have seen that because this technique does not reduce the size of the test suite it is most effective in

situations where we cannot afford to run every test and so we would like to maximize the effectiveness of those we do run.

Finally we saw the Hybrid approach as well as situations where we may desire to use this approach including those situations where the program size was quite large and executing all of even an extremely reduced test suite would be quite expensive.

Whether or not the future of regression testing is simple rehashing of the same methods and finding better ways to implement them, or if the future turns out to be a different approach such as the use of intelligent agents, good computer scientists will always be working on ways to make improvements to the way we approach and solve the problem of regression testing.

VI. References

[1] Sommerville, Ian, “Software Engineering,” Software Testing, 9th ed., Pearson, Boston, 2011, pp. 223.

[2] Wei Jin; Orso, A.; Tao Xie; , "Automated Behavioral Regression Testing," *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on* , vol., no., pp. 137-146, 6-10 April 2010.

[3] Salima, T.M.S. Ummu; Askarunisha, A.; Ramaraj, N.; , "Enhancing the Efficiency of Regression Testing through Intelligent Agents," *Conference on Computational Intelligence and Multimedia Applications, 2007. International Conference on* , vol.1, no., pp.103-108, 13-15 Dec. 2007.

[4] Gaurav Duggal, Mrs. Bharti Suri. Understanding Regression Testing Techniques. Delhi, India: Guru Gobind Singh Indraprastha University, 2007.

[5] Graves, T.L.; Harrold, M.J.; Kim, J.; Porters, A.; Rothermel, G.; , "An empirical study of regression test selection techniques," *Software Engineering, 1998. Proceedings of the 1998 International Conference on* , vol., no., pp.188-197, 19-25 Apr 1998.

[6] Rothermel, G.; Harrold, M.J.; , "Empirical studies of a safe regression test selection technique," *Software Engineering, IEEE Transactions on* , vol.24, no.6, pp.401-419, Jun 1998 .

[7] Wong, W.E.; Horgan, J.R.; London, S.; Agrawal, H.; , "A study of effective regression testing in practice," *PROCEEDINGS The Eighth International Symposium On Software Reliability Engineering* , vol., no., pp.264-274, 2-5 Nov1997.

[8] Chittimalli, P.K.; Harrold, M.-J.; , "Recomputing Coverage Information to Assist Regression Testing," *Software Engineering, IEEE Transactions on* , vol.35, no.4, pp.452-469, July-Aug. 2009.

[9] Li, Z.; Harman, M.; Hierons, R.M.; , "Search Algorithms for Regression Test Case Prioritization," *Software Engineering,* 13 *IEEE Transactions on* , vol.33, no.4, pp.225-237, April 2007