AN EFFECTIVE APPROACH TO REGRESSION TEST OPTIMIZATION TECHNIQUE

S. Narasimha Reddy[1]

Department of Computer Science and Engineering, KITS Engineering College, Opp Yerragattu Hillock, Hasanparthy, Warangal, Andhra Pradesh, 506 015, India
snreddy75@yahoo.co.uk

N.C. Santosh Kumar[2]

Department of Master of Computer Applications, KITS Engineering College, Opp Yerragattu Hillock, Hasanparthy, Warangal, Andhra Pradesh, 506 015, India
ncsantosh1980@yahoo.co.in

Abstract

Unswerving product quality is the main goal of any software engineering product. It involves rigorous product development and testing. Whenever new features are introduced to any existing product, the stress on quality is more, which will have more effect or impact on the 'regression' testing phase, in which all older functionality of the product are tested to make sure that the older functionality is intact. With an age, the product will have more features and will become more complex to choose regression test cases for such a complex products, if we don't have any regression optimization technique in place. Such situation will have direct impact on delivery and schedule of the product.

Hence, in this paper we propose a new technique by which we can reduce the regression test phase by using a variety of inputs to determine the appropriate smaller number of test cases out of entire large test suite.

***Keywords*:** Regression testing, cyclomatic complexity.

1. Introduction

Regression testing is an expensive testing process which attempts to validate not only modified software but also ensures that modified software does not impact the unmodified code. General approach to regression testing is to save the well prepared older test suite and re run the same suite on modified software along with the test cases to validate the modified code. This approach consumes lot of time out of regression cycle. Hence, we propose a new regression testing optimization technique, which reduces the overall cost of regression testing phase by selecting subset of test suite which was developed throughout the life of the software.

Over the past two decades, much effort was laid into the regression test selection techniques and many techniques have been proposed in the literature. Most of the proposed techniques are analytical and empirical based [1], [3], [4], [6]. But very few attempts were made to consider code changes as inputs [5].

In this paper, we propose a mechanism that takes the development changes as one of the inputs while selecting the subset of the regression suite and also to determine the test case execution order. It is a form of structured testing which uses cyclomatic complexity and the mathematical analysis of control flow graphs to guide the testing process. Our analysis shows a significant reduction in the number of test cases and a reduction in the effort while selecting the test cases.

2. Properties of a Test Case

A test case is developed to test a particular functionality of a product. The property of a test case is divided into two categories.

- Explicit property
- Implicit property

The explicit property of a test case can include the following items:

1. Functionality covered
2. Broader functional area
3. Product specific information such as feature number and release number
4. Category of test case such as basic test case or complex test case
5. Customer/product requirement numbers
6. Product configuration information

The implicit property of a test case is something that is not captured in the test case directly. They usually are the runtime information such as the methods/object code that gets invoked when the test case is executed.

3. Cyclomatic Complexity and Test Case Category

Cyclomatic complexity is software metric (measurement) developed by Thomas J. McCabe and is used to measure the complexity of a program. It directly measures the number of linearly independent paths through a program's source code. The concept, although not the method, is similar to that of general text complexity measured by the Flesch-Kincaid Readability Test. Cyclomatic complexity is computed using the control flow graph of the program: the nodes of the graph correspond to the commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command. The cyclomatic number V(G) of a graph G with n vertices, e edges, and p connected components is given by [2],

$$V(G) = e - n + p \qquad (1)$$

V(G) is the maximum number of linearly independent paths in G. V(G) is also known as the first Betti number. The cyclomatic complexity (M) of a structured program is defined as,

$$M = e - n + 2p \qquad (2)$$

Formally, cyclomatic complexity can be defined as a relative Betti number, the size of a relative homology group:

$$M := b_1(G,t) := \operatorname{rank} H_1(G,t) \qquad (3)$$

which is read as “the first homology of the graph G, relative to the terminal nodes t”. This is a technical way of saying “the number of linearly independent paths through the flow graph from an entry to an exit”, where:

- “linearly independent” corresponds to homology, means one does not double count backtracking;
- “paths” corresponds to first homology: a path is a 1-dimensional object;
- “relative” means the path must begin and end at an entry or exit point.

This corresponds to the intuitive notion of cyclomatic complexity, and can be calculated as above. Alternatively, one can compute this via absolute Betti number (absolute homology – not relative) by identifying (gluing together) all terminal nodes on a given component (or equivalently, draw paths connecting the exits to the entrance), in which case (calling the new, augmented graph $\tilde{G}$, which is), one obtains:

$$M = b_1(\tilde{G}) = \operatorname{rank} H_1(\tilde{G}) \qquad (4)$$

This corresponds to the characterization of cyclomatic complexity as “number of loops plus number of components”. The cyclomatic complexity numbers can be obtained for the various developmental changes that are incorporated into the product. This also acts as a measure of the complexity of the test case and its relative priority with respect to other test cases.

A test case belongs to any of the following categories depending on its objective (more categories can exist depending on the nature of product):

1. Simple Path - Classic method of exercising the code through a single path as in unit test or simple host testing.
2. Complex Path - Exercising the code using combinations of code paths and multiple functions; host testing with multiple legs.
3. Coverage - Normal usage of the method or function as in basic feature testing.
4. Variation - Changing parameters for the method or function; running the same test multiple times, but using different values to try to produce different or the same results; recent change testing of a parameter with different values.
5. Sequencing - Revealing a defect while exercising multiple function calls in a specific sequence.
6. Interaction - Causing a failure while exercising two or more functions simultaneously.

With the above details, it is possible to form a model which would use the various characteristics of a test case and the associated complexity of the code it is testing, and create a database which would be used later during the regression cycle to retrieve and execute accurately only the relevant test cases.

4. Test Case to Implementation Mapping

To implement a comprehensive regression testing strategy, taking into account the development changes and incorporating the implicit and explicit properties of a test case in order to best map it with the development changes, an extensive database of test cases has to be created. This database needs to have several fields associated with a single test case so that only the relevant test cases can be selected, based on various parameters, during the regression cycle.

The mapping of test cases with its explicit properties gives a birds-eye-view of all the cases associated with broad functional areas, feature IDs, release numbers, associated requirement IDs and so on.

Further, each test case has to be also associated with its implicit properties such as the code areas/functions that they parse through, within the development code, and the complexity of the tested code. To capture these details, it is necessary to have an extensive debugging facility within all the functions of the development code and turning them on during the initial test execution so as to capture all the debugging information which yield the complexity of the code and the various areas/paths it traverses. With these details, the test case can now be associated with its implicit properties also.

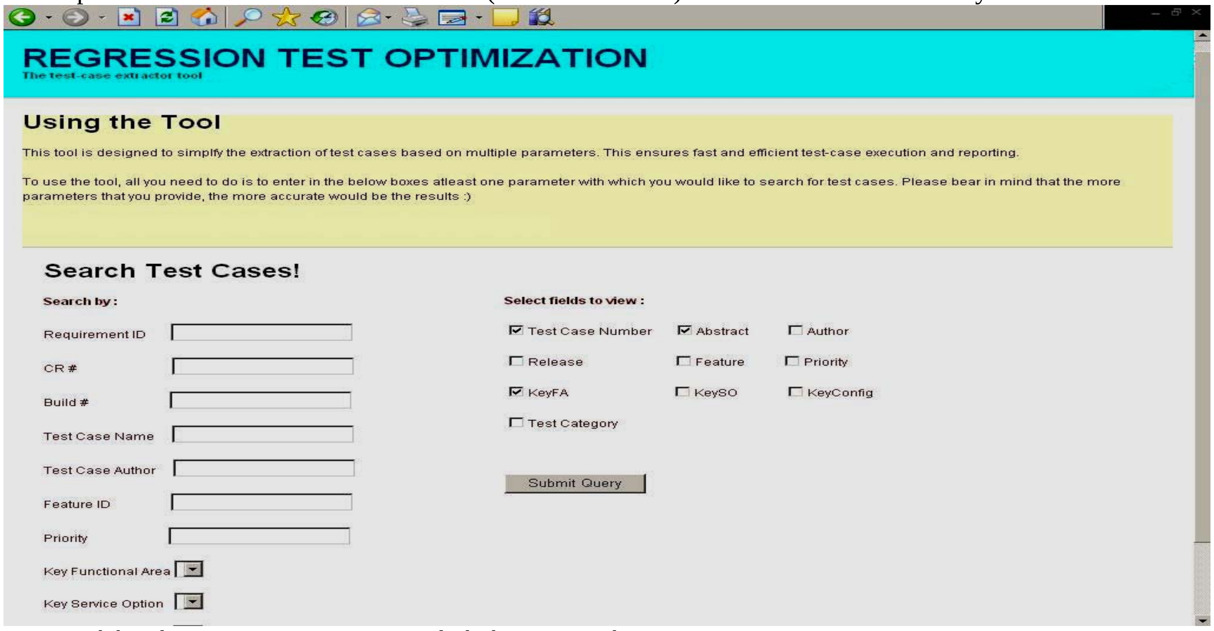
At this point we would like to stress on the importance of the development and the test teams to network together so as to develop this kind of a comprehensive test strategy which is useful down the product development cycle when the complexity increases.

With these mappings, the selection of test cases during a regression cycle becomes easier and faster, as the test cases can be retrieved from the repository based on the new features that were introduced as part of a particular build, which now needs to undergo regression. The relevant cases can be picked more comprehensively and accurately as the choice is now made not just on the features introduced but going down to a more granular level and also retrieving those functions that could have been associated with previous features, and whose functionality could have been modified by development of new code within existing functions.

With this method we observed almost a 50% reduction in the regression cycle time as only a subset of the entire test suite actually needed to be re-tested. This technique of selective regression improved cycle time and lessened the impact on product delivery.

5. Web-Based Approach

In order to make the availability of this feature, we took a web-based approach (shown in below Figure) to interface with the entire database. The test suite can contain very large number of test cases with many explicit and implicit attributes associated in its fields (database schema) and made available to every test member.



REGRESSION TEST OPTIMIZATION

The test-case extractor tool

Using the Tool

This tool is designed to simplfy the extraction of test cases based on multiple parameters. This ensures fast and efficient test-case execution and reporting.

To use the tool, all you need to do is to enter in the below boxes atleast one parameter with which you would like to search for test cases. Please bear in mind that the more parameters that you provide, the more accurate would be the results :)

Search Test Cases!

Search by:

Requirement ID
CR #
Build #
Test Case Name
Test Case Author
Feature ID
Priority
Key Functional Area
Key Service Option

Select fields to view:

[✓ Test Case Number] [✓ Abstract] Author
Release Feature Priority
[✓ KeyFA] KeySO KeyConfig
Test Category

Submit Query

Fig. 1. Web-based regression test optimization tool which interacts with Data Base.

There are a few challenges which still needs to be overcome. One of the major challenges is to map the test cases with its attributes after enabling debugging and measuring the complexity of the code for prioritizing purposes. This requires extensive debugging features within the code. It is relatively easier to adopt this technique right from the inception of the product rather than introducing it later, which would require more time and effort.

6. Conclusion

In this paper we have developed a new regression optimization technique, which reduces the regression cycle time and improves the quality of testing. We have introduced some concepts to include right from the development of code and tying of various attributes into the test cases. Due to the general nature of the problem of regression that is found in every product lifecycle, we see a large scope of the application of our technique to tackle regression cycle reduction in many projects, which in turn will result in delivery of the product within the project/feature deadline with good quality.

References

[1] Harrold, M.J., Rosenblum, D., Rothermel, G.,Weyuker, E. 2001. “Empirical studies of a prediction model for regression test selection”. IEEE Transactions on Software Engineering, Vol.27, Issue 3.

[2] McCabe, Thomas J. 1976. “A Complexity Measure.” IEEE Transactions on Software Engineering, Vol.se-2, No.4.

[3] Watson, Arthur H. and McCabe, Thomas J. 1996. “Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric.” Computer Systems Laboratory, National Institute of Standards and Technology (NIST).

[4] Abdullah, K., Kimble, J., and White, L., "Correcting for Unreliable Regression Integration Testing," International Conference on Software Maintenance, Nice, France, 1995, pp. 232-241

[5] Bible, J., Rothermel, G., and Rosenblum, D., "A Comparative Study of Course- and Fine-Grained Safe Regression Test-Selection Techniques," ACM Transactionson Software Engineering and Methodology, Vol. 10, No. 2, Apr. 2001, pp. 149-183

[6] Vokolos, F. and Frankl, P., "Empirical evaluation of the textual differencing regression testing technique," International Conference on Software Maintenance, Nov.1998, pp. 44-53.