An Overview of Regression Testing

Amir Ngah[1], Malcolm Munro[2] and Mohammad Abdallah[3]
[1]*School of Informatics and Applied Mathematics, University Malaysia Terengganu,*
21030 Kuala Nerus, Terengganu, Malaysia.
[2]*School of Engineering and Computing Sciences, Durham University, DL1 3LE, United Kingdom.*
[3]*Faculty of Science and IT, Az-Zaytoonah University of Jordan, Amman, Jordan.*
amirnma@umt.edu.my

***Abstract*—Regression testing is expensive but an essential activity in software maintenance. Regression testing validates modified software and ensure that the modified parts of the program do not introduce unexpected errors. This paper briefly describes an overview of regression testing specifically regression test selection techniques. Most regression test selection techniques are based on program slicing techniques.**

***Index Terms*—Regression Testing; Regression Test Selection; Test Case Minimization; Test Case Prioritization.**

I. Introduction

According to the IEEE Standard 1219-1998 [1], regression testing can be involved in different levels such as unit, integration or system level testing. Regression testing also described as one kind of testing that is applied at all these three levels. These three levels of testing are similar to the process of testing in development although they have to be focused on modifications that have occurred in the program. Most existing regression testing techniques concentrate on unit testing. Some of the techniques focused on all levels of testing [2; 3].

This paper discusses regression testing specifically regression test selection techniques. The paper is organized as follows. The second section presents an evaluation framework for regression test selection techniques. The third section presents regression testing strategies. Then, categories of regression testing techniques are discussed in the fourth section. The most significant topic in this chapter is about regression test selection techniques presented in the fifth section. Then, the sixth section discusses a regression testing in different environments.

II. An Evaluation Framework for Regression Test Selection Techniques

Rothermel and Harrold [4] proposed a framework for evaluating regression test selection techniques. This framework is used to evaluate the regression test selection techniques that will be explained in the later section. The framework is based on four categories which are inclusiveness, precision, efficiency and generality.

Inclusiveness measures the capabilities of techniques to select test cases that will cause the *modified* program to give a different output than the *certified* program. A regression test selection technique is *safe* if it selects all test cases that can give different output. *Precision* measures the ability of techniques to avoid select test cases that cannot give different output between the *certified* and the *modified* programs. A regression test selection technique is *precise* if the technique is capable of omitting test cases that cannot give different output. *Efficiency* measures the computational cost, thus the practicality of a regression test selection technique. The *generality* of a regression test selection technique is its ability to be used in a wide and practical range of situations.

III. Regression Testing Strategies

An important issue in regression testing is how to reuse the existing test suite for the *modified* program [2]. There are two main regression testing strategies; *retest all*, and *selective retest*. A *retest all* approach reruns all the existing test suite on the *modified* program. In theory, *retest all* approach is *safe* because it can exercise all modification parts in the *modified* program. However, it is not practical to use for large software systems because of the time and resources needed.

Selective retest techniques, in contrast, attempt to reduce the time required to retest a *modified* program by selecting a subset of the existing test suite and retesting only the relevant part of the *modified* program. Rothermel and Harrold [2] have identified two issues in the selective retest techniques: (1) the issue of how to select test cases from the existing test suite and (2) the issue of identifying where additional test cases may be required.

IV. Categories of Regression Testing Techniques

Rothermel et al. [5] consider three techniques for reducing the cost of regression testing. They are regression test selection, test suite minimization and test case prioritization techniques.

A. Regression Test Selection

Many papers concentrate on regression test selection techniques [2, 3, 6, 7, 8, 9, 10]. Those techniques attempt to reduce the cost of regression testing by selecting appropriate test cases using information from the certified program, the modified program and the existing test suite. A detailed explanation about this category will be given in the next section.

B. Test Suite Minimization

Test suite minimization techniques decrease cost by minimizing a test suite that still maintains the same coverage of the initial test suite with respect to a particular test coverage metric. Harrold et al. [11] propose a minimization technique that helps to manage a test suite by determining redundant and obsolete test cases. The technique introduced a mechanism that selects a set of test cases from the test suite, but still provides the desired testing coverage of the program. The technique requires an association between the test cases and the testing requirements of the program, but it is independent of the test selection criteria and can be applied if this association can be made. The minimization technique can also accommodate test suites that use more than one test selection criteria. The technique can be performed on the entire test suite or on a test suite consisting of those test cases that test the changed or affected parts of a program. This technique was incorporated into a data flow testing system called Combat. Hsu and Orso [12] have developed a general framework and tool for supporting test-suite minimization called MINTS. Their evaluation shows that MINTS can be used to instantiate a number of different test-suite minimization problems and efficiently find an optimal solution for such problems using different solvers [12].

C. *Test Case Prioritization*

Test case prioritization technique provides another method for assisting with regression testing. The prioritization technique let testers order their test cases, so that those test cases with the highest priority are executed earlier than those with lower priority according to some criterion. Elbaum et al. classify test case prioritization techniques into three groups. The groups are based on control, statements and function level of a program.

V. Regression Test Selection Techniques

The subject of selective regression testing has received considerable attention from the software testing and software maintenance research communities. Some of the regression test selection techniques are discussed below. These regression test selection techniques can be divided into few categories based on elements used in their techniques such as control-flow based [2], textual differencing based [3; 7], code entities based [10] and program slicing based [6; 8; 9].

A. *Control-flow Based*

Rothermel and Harrold [2] propose a *safe* and efficient regression test selection technique based on control-flow graphs (CFG). They have proposed two main algorithms; intraprocedural and interprocedural. The intraprocedural algorithm operates on individual procedures. The interprocedural algorithm operates on entire programs or subsystems. In this technique, both the *certified* and the *modified* programs will be transformed into a CFG in order to perform comparison. The comparison algorithm compares each node in both CFGs. If both nodes differ, the algorithm will select test cases from Test Suite (T) that execute the node in CFG of the certified program to test the modified program.

These two algorithms are implemented in two different tools. They are *DejaVu1* for intraprocedural algorithm and *DejaVu2* for interprocedural algorithm. Both tools have been developed to analyze C programs. By using both algorithms, this technique is suitable for a level of regression testing including unit, integration and system level.

Rothermel and Harrold claim that their technique can decrease the time required to carry out regression testing for the *modified* program, even when considering the cost of performing the analysis to select the test cases. Their interprocedural test selection algorithm can give huge savings than intraprocedural test selection algorithm in term of reducing the number of test cases. The technique can give significant savings when applied to large or complex programs. This result is based on their experiment of the application of their technique to the “Siemens programs” by Hutchins. The result show that DejaVu1which perform intraprocedural algorithm always selected 100% of test cases for the modified procedures. This means there is no significant reduction in the size of test suite for the modified procedures. In contrast to this, *DejaVu2* in average selects about 55.6% test cases for the modified program. This means *DejaVu2* can give saving about 44.4% of test cases size. This technique is considered as a *safe* regression test selection technique but not *precise* [2; 13].

B. *Textual Differencing Based*

Vokolos and Frankl [3] have developed a tool called *Pythia* that is used to reduce the cost of regression testing. The Unix-based tool implements an analysis technique that is called textual differencing because it works by comparing the source files from the *certified* and *modified* programs. The *Pythia* tool can be used to analyze software systems written in the C programming language. Vokolos and Frankl claimed that a novel characteristic of *Pythia* is that it has been implemented by using standard Unix tools. The characteristics of the *Pythia* tool are:

i. It selects a *safe* regression test suite.
ii. It supplies both intraprocedural and interprocedural analysis. So, it can be used for single *C* functions or software systems.
iii. It has been implemented using standard Unix tools.
iv. The comparison between the *certified* and the *modified* programs uses the Unix tool called *diff*. No abstract representation of the program is needed in the comparison.
v. Instrumentation, for determining the execution trace of the *certified* program, is done directly by the *C* compiler, during module compilation.
vi. In principle, it can be easily extended to support other popular programming languages, such as C++.

The *Pythia* tool has been integrated into a shell script to include *cc*, the C language compiler, *pretty*, a beautifier for C programs, and *diff*, the general purpose file comparison program. Pythia consists of a few stand-alone programs: *kform*, *instr*, *xqt*, and *txt*. The functionality of these programs and a description on how *Pythia* works is as follows:

i. The sources file for the *certified* program is converted using the program *kform*– into a canonical form. *Kform* is a script that uses the program *pretty*, the C program beautifier.
ii. The canonical files are instrumented and compiled using the program *instr*. Instrumentation is used to maintain a basic block execution trace for the *certified* program. *Instr* is a script that uses *cc*, the C compiler.
iii. The program being tested is executed via the program *xqt*, which maintains a history of test cases along with the basic blocks executed by each test case.
iv. The modified program are also converted into canonical files with the program *kform*.
v. The program *txt* compares the *certified* program with the *modified* program canonical files, by using *diff*, and analyses the differences, as reported by *diff*, to determine the set of all test cases that have exercised by the modified statements.

Vokolos and Frankl [3] have used the framework for evaluating selective regression testing techniques developed by Rothermel and Harrold [4]. They have claimed that textual differencing is a *safe* selective regression testing technique in terms of inclusiveness. For precision, textual differencing is not 100% *precise* due to the fact that they do not perform semantic analysis. In term of efficiency, the computational cost of textual differencing will be reasonable. In term of generality, textual differencing involves all forms of code modifications like insertions, deletions, and changes of statements. It can works on both in intraprocedural and interprocedural aspects of a program. They also claimed that their technique can easily be extended to programs written in languages that have a mechanism to perform basic block instrumentation and to transform the source code into canonical form.

Vokolos and Frankl [7] claimed that the *Pythia* tool can quickly analyze software systems written in C programs and be effective in reducing the set of regression test cases. The claim is based on the results from a case study involving a software system of approximately 11,000 lines of source code written for the European Space Agency. The system called *ORACOLO2* is written in C and was developed within the Microsoft Visual C++ 1.5 environment. There were 33 different faults discovered and recorded. Each fault was corrected and a new version of the program was created for each fault. The results of their case study shows that Pythia reduced the size of the regression test suite by at least 90% on average in almost 40% of the program versions (13/33). A reduction of at least 80% was reported in almost 50% of the program versions (16/33). This shows that the textual differencing based technique, Pythia, can give significant reduction in regression test suite size. Pythia is considered as a *safe* regression test selection technique but not *precise* [7].

C. Code Entities Based

Chen et al. [10] have proposed a regression test selection technique based on identifying modified code entities such as functions, variables, types, and macros. Test cases that have traversed modified code entities will be counted in the test suite for the modified program. The technique has been implemented in a tool called TestTube that combines static and dynamic analysis to perform selective retesting of programs or systems written in the C programming language. The tool has been developed with a combination of existing analysis tools. The collection of tools can be divided into three categories, including instrumentation tools, program database tools, and test selection tools. In the instrumentation tools, app (the Annotation Preprocessor C) instruments the source code automatically. The C Information Abstractor (CIA) is used to build a C program database in the program database tools category. The technique is considered as a safe regression test selection technique but less precise [4].

D. Slicing Based Techniques

There are a number of regression test selection techniques based on program slicing techniques. Binkley [6] conducted a survey about the application of program slicing to regression testing. He divided into three groups of program slicing that are used in regression testing. The first group uses dynamic slicing, the second group presents program slicing using program dependent graphs (PDG), and the third group is based on Weiser's data-flow definition of slicing.

Agrawal et al. [14] have proposed three algorithms to be used in their technique called an incremental regression testing. The algorithms are an execution slice, a dynamic slice, and a relevant slice. The execution slice of the program with respect to a test case is referred to as the set of statements executed under that test case. The dynamic program slice with respect to the output variables gives us the statements that are not only executed but also have an effect on the program output under that test case. The relevant slice with respect to the program output for a test case is referred to the set of statements that, if modified, may alter the program output for the given test case.

Agrawal et al. [14] have pointed out that the amount of regression testing effort saved using their technique obviously depends on the nature of test cases as well as the locations of the modifications made. If the number of test cases are large and each of them exercise small parts of the program's functionality then using these techniques should offer huge savings. The modification parts of the program may also have a major effect on the amount of savings implied by using these techniques. The incremental regression testing technique is considered as a *precise* regression test selection technique but less *safe* [15].

Gupta et al. [16] have developed a data flow based regression testing technique that uses slicing algorithms to explicitly determine the affected *definition-use* associations made by a program change. The technique uses two slicing algorithms to detect directly and indirectly affected *def-use* associations. The first algorithm works backward from the changed statement to its definitions. The second algorithm is a forward walk from the same point as the first algorithm. The forward algorithm detects uses, and subsequent *definitions* and *uses* that are affected by a definition that is changed at that point. Gupta et al. [16] claim that the slicing algorithms are efficient because they detect the *def-use* associations without considering either the data flow history or the complete recomputation of data flow for the *certified* program. They also claim that their technique could easily be modified from *all-uses* criterion to other data flow testing criteria. The technique can also be extended to interprocedural regression testing using interprocedural slicing. The technique is considered as a *safe* regression test selection technique but less *precise* [4].

Gallagher et al. [17] have proposed a novel approach for regression test selection based on *exclusion*. They claim that an exclusion-based technique is likely to be more effective that an inclusion-based technique in two ways. First, it will more confidently identify all non-modification revealing tests in terms of safety. Second, in terms of the impact of the approach, by reducing the size of regression tests by excluding tests that are not related to modification. Gallagher et al. proposed four steps in his exclusion technique as follows:

i. Decompose and Reduce System Version n. The decomposition slices are constructed for the considered system and reduced by equivalent slices.
ii. Match Tests with Code. The decomposition slices are match to the relevant test cases using Vokolos and Frankl technique [3].
iii. Decompose and Reduce System Version n + 1. The process is same as in step 1. Then, obtain the tests for decomposition slice clusters that remain unchanged.
iv. Use tests that remain after removing those obtained in step 3. Any tests for unchanged code are not needed.

These all slicing based RTS techniques are classified as inclusion techniques which select test cases from test suite that are needed in regression testing. The idea of the regression test selection by exclusion was proposed by Gallagher et al. [17]. Ngah et al. [18, 19] have developed a new regression test selection by exclusion using decomposition slicing called ReTSE. Exclusion technique omits test cases from test suite that are not needed in regression testing.

VI. Regression Testing in Different Environments

There are implementations of regression testing techniques in the literature. They can be divided into four groups: structured based programs, object-oriented based programs, web based applications and component-based systems.

A. Structured Based Programs

Structured based program are often composed of program flow structures such as sequence, selection and iteration compare to object-oriented program that are based on objects which have their attributes and methods. There are a number of techniques as well as tools that are proposed for regression testing for structured based programs, especially the C programming language. Examples are the Rothermel and Harrold technique with their tools DejaVu1 and DejaVu2 [2], TestTube tool by Chen et al. [10], and Pythia tool by Vokolos and Frankl [3]. The explanation of these techniques and tools have already been described in the previous section.

B. Object-oriented Based Programs

Orso et al. [20] have introduced a regression test selection technique for Java programs. The technique can handle the object-oriented features of the language, is *safe* and *precise*, and applicable to large systems. The technique consists of two parts: partitioning and selection. The partitioning part is executed first in order to build a high level graph representation of *certified* and *modified* programs and performs an analysis of the graphs. The goal of the analysis is to identify the parts of the *certified* and the *modified* programs that have changed based on information on changed classes and interfaces. Then, the selection part of the technique builds a more detailed graph representation of the identified parts of the *certified* and the *modified* programs, analyses the graph to identify differences between the programs, and selects a set of test cases in the test suite that traverse the changes. This technique is implemented in a tool called *DEJAVOO*. Orso et al. claim the results of the empirical study of their tool is encouraging in terms of efficiency and effectiveness. The technique reduces the time for regression testing as high as 62.5% for a largest system. The cost-effectiveness improves with the size of the program under test.

Wu et al. [21] have proposed a regression testing technique based on the analysis of the dependence relationship among functions in a system. They have defined that the object-oriented features, such as inheritance, dynamic binding, polymorphism and message passing are related to the function calls which are associated with certain objects. The technique performs in two phase analysis. The first phase is to analyze the affected variables, functions, function dependence relationships at the statement level after the modification. The technique is *safe* because it considers all possible effects of the modification on the system. This static phase is considerably more efficient. In the second phase, the technique dynamically select test cases that are needed to be retested by using the function calling graph (FCG) of each test case in order to precisely process object-oriented features and thus enhance the precision of the technique. The FCG can be constructed based on the record of the calling sequence of functions. So, the required overhead is proportional to the number of function calls.

Harrold et al. [22] have introduced a *safe* regression test selection technique for Java. The technique can efficiently handle the features of object-oriented language specifically the Java language, such as polymorphism, dynamic binding, and exception handling. The technique is an adaptation of Rothermel and Harrold technique [2], which is based on a control flow representation of the *certified* and *modified* programs to select test cases to be rerun. The technique performs three steps. First, it constructs a graph to represent the control flow and the type of information for the set of classes under analysis. Then, it traverses the graph to identify affected edges. Finally, based on the coverage matrix obtained through instrumentation, the technique selects the test cases that exercise the affected edges identified from the test suite for the *certified* program.

Unlike the Rothermel and Harrold technique [2], which is uses the CFG, the technique by Harrold et al. [22] introduces the Java Interclass Graph (JIG) as a representation of the program. A JIG accommodates the Java features and can be used by the graph-traversal algorithm to identify dangerous entities. Dangerous entity is an edge that affected by a change by comparing the *certified* and the *modified* programs. Empirical studies indicate that the technique can be effective in reducing the size of the test suite [22].

C. Web Based Applications

Tarhini et al. [23] have proposed a safe regression testing selection technique for web applications based on an Event Dependency Graphs (EDG). The EDG is used to model the *certified* and the *modified* web applications. Then both EDG's are compared in order to select the affected nodes and the potentially affected nodes. The affected nodes are used to select test suite for the *certified* web application. Empirical results show that the technique reduced the test set size [23]. About 44-90% of test cases were eliminated. The selected test cases still cover the modified and potentially modified components.

Lin et al. [24] have introduced a code transformation approach to regression test selection. The transformed code forms a local Java program which simulates the functionality and behavior of the Web service applications in an end-to-end manner. Safe regression test selection techniques can then be applied to the transformed code and safely reduce the test cases for the Web service applications. This approach is implemented on Web service applications written in Java and deployed in the Axis server only.

Ruth et al. [25; 26] have proposed a gray-box approach that support safe regression test selection technique for verification of Web service system in an end-to-end manner. A gray-box approach is a technique that does not involve code-based knowledge directly, in contrast to white box approach. Their approach is based on the safe regression test selection technique by Rothermel and Harrold [2] which is uses a CFG as a representation of the certified and modified programs. Each node represents a code entity and each edge represents the control flow from one code entity to another. The entities can be statements, methods, classes, or components [25]. Then, the technique identifies affected edges by comparing the CFGs of certified and modified programs. Finally, based on the set of affected edges, the technique selects test cases for T' from test suite T that need to be rerun.

D. Components Based System

Gao et al. [27] have proposed a systematic retest method for software components based on a component retest model. This method has been implemented in a component test tool called *COMPTest*. The *COMPTest* tool can automatically identify component-based API changes and impacts, as well as reusable test cases in a component test suite. They claimed that the tool has two major advantages:

i. Automatic identification and analysis of API-oriented component changes and impacts based on given API-based component test models and other meta-data, such as function and dependency information in a component.
ii. Automatic black-box test selection for reuse and test suit refreshment for a component.

VII. Conclusion

This paper discusses researches on regression testing specifically regression test selection techniques. Based on these studies, it is hard to identify the best techniques for regression test selection. This is because every proposed technique has their own focusses and purposes. Moreover, it is more difficult to compare because some proposed techniques are based on difference environment like structure based programs, object oriented programs, web based applications and component based systems as mentioned in previous section. The only current framework to evaluate the regression test selection technique has been proposed by Harrold et al. [28]. However this framework is quite old and may not suitable for current environment of the programs or systems. Therefore, a suitable and efficient framework or method is significantly needed in order to evaluate the regression test selection techniques.

Acknowledgment

This research is sponsor by Ministry of Education, Malaysia Government under Research Acculturation Collaborative Grant (RACE), Vot No. 56032.

References

[1] IEEE Standard for Software Maintenance. IEEE Std 1219-1998. Oct 1998.

[2] G. Rothermel and M. J. Harrold. "A safe, efficient regression test selection technique," *ACM Transactions on Software Engineering Methodology*, vol. 6, no. 2, pp. 173-210, 1997.

[3] F.I. Vokolos and P.G. Frankl. "Pythia: A regression test selection tool based on textual differencing," in *Proceedings of the International Conference on Reliability, Quality and Safety of Software-intensive Systems*, 1997, pp. 3-21.

[4] G. Rothermel and M. J. Harrold. "Analyzing regression test selection techniques," *IEEE Transactions on Software Engineering*, vol. 22, no. 8, pp. 529-551, 1996.

[5] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001.

[6] Da. Binkley. "The application of program slicing to regression testing,". *Information & Software Technology*, vol. 40, no. 11-12, pp. 583-594, 1998.

[7] F. I. Vokolos and P. G. Frankl. "Empirical evaluation of the textual differencing regression testing technique," in *Proceedings of the International Conference on Software Maintenance*, 1998, pp. 44-53.

[8] I. Forgacs, A. Hajnal, and E. Takacs. "Regression slicing and its use in regression testing," in *Proceedings of the IEEE International Computer Software and Applications Conference*, 1998, pp. 464-469.

[9] K. B. Gallagher and J. R. Lyle. "Using program slicing in software maintenance," *IEEE Transactions on Software Engineering*, vol. 17, no. 8, pp. 751-761, 1991.

[10] Y. F. Chen, D. S. Rosenblum, and K. P. Vo. "Testtube: A system for selective regression testing," in *Proceeding of the International Conference on Software Engineering*, 1994, pp. 211-220.

[11] M. J. Harrold, R. Gupta, and M. L. Soffa. "A methodology for controlling the size of a test suite," *ACM Transactions on Software Engineering and Metholdology*, vol. 2, no. 3, pp. 270-285, 1993.

[12] H. Y. Hsu and A. Orso, "Mints: A general framework and tool for supporting test-suite minimization," in *Proceedings of the IEEE 31st International Conference on Software Engineering*, 2009, pp. 419-429.

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

[14] H. Agrawal, J. R. Horgan, E. W. Krauser, and S. London. "Incremental regression testing," in *Proceedings of the International Conference on Software Maintenance*, 1993, pp. 348-357.

[15] G. Baradhi and N. Mansour. "A comparative study of five regression testing algorithms," in *Proceedings of the Australian Software Engineering Conference* , 1997, pp. 174-180.

[16] R. Gupta, M. J. Harrold, and M. L. Soffa. "An approach to regression testing using slicing," in *Proceedings of the International Conference on Software maintenance*, 1992, pp. 299-308.

[17] K. Gallagher, T. Hall, and S. Black. "Reducing regression test size by exclusion," in *Proceedings of the International Conference on Software Maintenance*, 2007, pp. 154-163.

[18] A. Ngah, M. Munro and K. Gallagher. "Regression test selection model using decomposition slicing," in the *Proceedings of the IASTED International Conference on Software Engineering*, 2012, pp. 23-24.

[19] A. Ngah, M. Y. M. Saman and M. Munro. "ReTSE: Slicing based regression testing," *WIT Transaction Engineering Sciences*, vol. 86, page 457-470, 2014.

[20] A. Orso, N. Shi, and M. J. Harrold. "Scaling regression testing to large software systems," in *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2004, pp. 241-251.

[21] Y. Wu, M. H. Chen, and H. M. Kao. "Regression testing on object-oriented programs," in *Proceedings of the 10th International Symposium on Software Reliability Engineering*, 1999, pp. 270–279.

[22] M. J. Harrold, J. A. Jones, T. Li, D. Liang, A. Orso, M. Pennings, S. Snha, S. A. Spoon, and A. Gujarathi. "Regression test selection for java software," in *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Language, and Applications*, 2001, pp. 312-326.

[23] A. Tarhini, Z. Ismail, and N. Mansour. "Regression testing web applications," in *Proceedings of the International Conference on Advanced Computer Theory and Engineering*, 2008, pp. 902-906.

[24] F. Lin, M. Ruth, and S. Tu. "Applying safe regression test selection techniques to java web services," in *Proceedings of the International Conference on Next Generation Web Services Practices*, 2006, pp. 133-142.

[25] M. Ruth and S. Tu. "A safe regression test selection technique for web services," in *Proceedings of the International Conference on Internet and Web Applications and Services*, 2007, pg. 47.

[26] M. Ruth, S. Oh, A. Loup, B. Horton, O. G., M. Mata, and S. Tu. "Towards automatic regression test selection for web services," in *Proceedings of the 31st Annual International Computer Software and Applications Conference,* 2007, pp. 729-736.

[27] J. Gao, D. Gopinathan, Q. Mai, and J. He. "A systematic regression testing method and tool for software components," in *Proceedings of the IEEE International Computer Software and Application Conference*, 2006, pp. 455-466.

[28] M. J. Harrold, D. Rosenblum, G. Rothermel, and E. Weyuker. Empirical studies of a prediction model for regression test selection. *IEEE Transactions on Software Engineering*, vol. 27, no. 3, pp. 248-263, 2001.