ISSN No. 0976-5697

Volume 3, No. 5, Sept-Oct 2012

International Journal of Advanced Research in Computer Science

REVIEW ARTICLE

Available Online at www.ijarcs.info

A Survey of Regression Testing Techniques

Ahmed S. Ghiduk*
Department of computer science,
College of Computers and IT,
Taif University, Saudi Arabia
asaghiduk@{yahoo.com, tu.edu.sa}

Moheb R. Girgis
Department of computer science,
Faculty of Science,
Minia University, Egypt
moheb_girgis@eun.eg

Eman H. Abd-Elkawy
Department of Mathematics,
Faculty of Science,
Beni-Suef University, Egypt
eman_hassan2080@yahoo.com

Abstract: Regression testing is the process of validating the modified software to provide confidence that the changed parts of the software behave as intended and that the unchanged parts of the software have not been adversely affected by the modifications. Researchers have proposed many techniques for the different regression testing aspects. In this paper, we review the work which has been done so far in the field of regression testing. In addition, we will organize the surveyed techniques into categories according to the regression testing aspects.

I. INTRODUCTION

Regression testing is the process of validating modified software to provide confidence that the changed parts of the software behave as intended and that the unchanged parts of the software have not been adversely affected by the modifications. Both during development and after deployment, software is modified for several reasons, including bug fixing, functionality enhancement, and adaptation to changes in the software's operating environment. One of the most expensive activities that occur as the software is modified and enhanced is the (re)testing of the software after it has changed. This process is known as *regression testing*. Because regression testing is expensive, researchers have proposed techniques to reduce its cost. One approach reduces the cost of regression testing by reusing the test suite that was used to test the original version of the software. Rerunning all test cases in the test suite, however, may still require excessive time. An improvement is to reuse the existing test suite, but to apply a regression test selection technique to select an appropriate subset of the test suite to be run.

Regression testing has been used during the development and maintenance of a software product to assist software-testing activities and guarantee the attainment of adequate quality through various versions of the software product [1]. Regression testing permits to test modified software to provide confidence that no new errors are introduced into previously tested code [2].

In regression testing, a set of tests is executed whenever modification is done on a part of software. New output is compared with the old ones to prevent unwanted changes. Other parts of software are considered to be unaffected by the changes made on one part of the software if the new output matches with the old output.

Regression testing is the re-execution of a particular subset of tests that has been formerly performed. In regression testing, the number of regression tests increases with the progress of integration testing, and executing each test for every program function whenever changes occur is both impractical and inefficient. Regression test suites are often simply test that software engineers have previously developed, and that have been saved so that they can be used later to perform regression testing [3].

The rest of the paper is organized as follow: Section 2 surveys the regression test selection techniques. Section 3 discusses the test case prioritization techniques. Section 4 introduces the test suite augmentation techniques. Section 5 reviews change-identification techniques and computing program differences techniques. Section 6 surveys GA based regression testing techniques. Section 7 gives the conclusion.

II. REGRESSION TEST SELECTION TECHNIQUES

Regression test selection techniques attempt to reduce the cost of regression testing by selecting and running only a subset of the test cases in an existing test suite to ensure that changed parts behave as intended and the changes did not introduce unexpected faults. While this approach may lessen the cost of performing regression testing, it is quite difficult to find a balance between the time required to select and run test cases and the fault detection ability of the remaining test cases [4]. Although safe test selection techniques do exist [5], the amount of work required to prove that the subset of test cases exposes the same number of faults as the full test suite is difficult in some instances.

The main objective of selecting test cases that need to be rerun is to identify regression test cases that exercise modified parts of the system. This is referred to as safe regression testing as, it identifies all test cases in the original test set that can reveal one or more faults in the modified program [6]. Several safe regression-test-selection

techniques have been developed and proved to reduce the size of regression test suite effectively. One of them is for Java software [7], which uses a combination of static and dynamic (using a profiler) analysis to produce a model, Java Interclass Graph (JIG), which is then used in the safe regression-test selection (RTS) algorithm to select only those tests which are modification revealing. Since this method uses dynamic analysis, which requires the application to be run in a single Java Virtual Machine (JVM), this technique cannot be applied in any Web Service environment.

Regression test selection techniques are either code-based or model-based. Code-based techniques use the information obtained from two different versions of the code to analyze the change impact and select the tests. In the case of model based techniques, change information is obtained through two versions of models constructed during the requirements analysis phase or system design phase. Code based techniques, [6-10], are very specific to the programming language used to develop the code. Chianti [11] and JDiff [9] are comprehensive techniques for managing changes in Java programs. Chianti selects regression tests after analyzing the change impact analysis, whereas JDiff performs only change impact analysis. As both these tools analyze the changes at statement level and are specific to Java programming language, hence, they are neither generic nor efficient. Model-based techniques [12-15] are based on UML design models used during the design phase of the system.

Few regression test selection techniques are software specifications-based [16-19]. These specification based techniques are generally meant for specific software such as API testing. Hence, there is a need to develop an approach for selection of regression test suite based on software requirements to validate software applications that are being modified.

Chen et al. [20] have developed a regression test selection technique based on the idea of detecting modified code entities such as functions, variables, types, and preprocessor macros. Test cases that have traversed modified code entities form the selected test suite. This technique has been implemented in a tool called "**Test Tube**", which has been developed around existing analysis tools, namely **app** (the Annotation Preprocessor for C [21]) and CIA (the C Information Abstractor [22]).

Rothermel and Harrold [23] define a framework for comparing different regression test selection methods, based on four characteristics: Inclusiveness (the ability to choose modification revealing tests), Precision (the ability to eliminate or exclude tests that will not reveal behavioral differences), Efficiency (the space and time requirements of the method) and Generality (the applicability of the method to different classes of languages, modifications, etc.)**.**

Rothermel and Harrold [5] have developed a regression test selection technique that is based on the idea of creating *control flow graphs* (CFGs) to represent, and compare, *P* and *P'*, where P is original program and P' is modified program. The nodes in the CFG contain actual program statements. During the execution of *P*, a list of all the edges traversed by each test case is maintained. The CFGs are compared by simultaneously traversing the nodes of each graph and looking for differences in either (i) the contents of a node, or (ii) the contents of succeeding nodes. When differences are detected, the test cases that have traversed the edges associated with these nodes are selected.

Rothermel and Harrold technique supports both intraprocedural and interprocedural analysis and is capable of detecting, with good precision, modification traversing test cases. Two different prototype tools, **DejaVu1** (for intraprocedural analysis) and **DejaVu2** (for interprocedural analysis) have been developed to analyze C programs. The authors have used these prototype tools on a large software system with encouraging results. However, as they point out, they were not able to instrument, or run their implementation, on about 15% of the procedures.

Fisher II et al. applied the data flow based regression test selection approach for test re-use in spreadsheet programs [24]. Fisher II et al. proposed an approach called What-You-See-Is-What-You-Test (WYSIWYT) to provide incremental, responsive and visual feedback about the *testedness* of cells in spreadsheets. The WYSIWYT framework collects and updates data flow information incrementally as the user of the spreadsheet makes modifications to cells, using Cell Relation Graph (CRG). Interestingly, the data flow analysis approach to re-test spreadsheets is largely free from the difficulties that the approach has used to test procedural programs, because spreadsheet programs are purely based on data flow and not on control flow information. This makes spreadsheet programs an ideal candidate for a data flow analysis approach.

Koju et al. [10] have proposed a technique for regression test selection based on the Microsoft Intermediate Language (MSIL). Their technique is based on the one developed by Harrold et al. [7] for Java. They present control flow graphs to handle .Net-specific features such as delegate and present a class hierarchy analysis technique to support the regression test selection.

Malhotra et al. [25] have proposed a technique, which is an extension of earlier regression test selection and prioritization techniques. They implemented this technique and validated it with the help of two case studies. Unlike other techniques, this technique identifies test cases that execute the modified lines of source code at least once and selects those test cases that execute the lines of source code after deletion of lines from the execution history of the test cases The results showed that the technique can significantly reduce the cost and resources for performing regression testing on modified programs.

Chen et al. [26] have introduced a semi-supervised clustering method, namely semi-supervised Kmeans (SSKM) to improve cluster test selection. SSKM uses limited supervision in the form of pairwise constraints: Must-link and Cannot-link. These pairwise constraints are derived from previous test results to improve clustering results as well as test selection results. The experiment results illustrate the effectiveness of cluster test selection methods with SSKM. Two useful observations are made by analysis. (1) Cluster test selection with SSKM has a better effectiveness when the failed tests are in a medium proportion. (2) A strict definition of pairwise constraint can

improve the effectiveness of cluster test selection with SSKM.

III. TEST CASE PRIORITIZATION TECHNIQUES

Regression test prioritization techniques attempt to ascertain earlier release of faults in the test execution phase and execution of higher priority tests earlier than lower priority tests in the regression testing process by rearranging the execution of test suite and rearranging regression test suite, respectively [16]. General test case prioritization and version specific test case prioritization are two types of test case prioritization. Without any knowledge of the modified version of a given program P, the test cases in a given test suite T that will be useful over a series of subsequent modified versions of P are generally prioritized by test case prioritization. The knowledge of the changes that have been made in P when P is modified to P' are taken into account in version specific test case prioritization for prioritizing the test cases in T, [27]. A broad range of objectives can be addressed by test case prioritization. When the time required to run all test cases in the test suite is sufficiently long, the benefits offered by test case prioritization methods become more significant [28].

Several researchers have addressed the test case prioritization problem and presented techniques for handling it. Test case prioritization techniques reported in [29, 30] orders test cases such that the test cases with highest priority, according to some criterion, are executed first. Test case prioritization can address a wide variety of objectives [31]. For example, concerning coverage alone, testers might wish to schedule test cases in order to achieve code coverage at the fastest rate possible in the initial phase of regression testing, to reach 100% coverage soonest or to ensure that the maximum possible coverage is achieved by some pre–determined cut–off point. In the Microsoft Developer Network (MSDN) library, the achievement of adequate coverage without wasting time is a primary consideration when conducting regression tests [32]. Furthermore, several testing standards require branch adequate coverage, making the speedy achievement of coverage an important aspect of the regression testing process.

Srivastava and Thiagarajan [33], studied a prioritization technique that is based on the changes that have been made to the program and focused on the objective function of "impacted block coverage". Other non–coverage based techniques in the literature include fault–exposing–potential (FEP) prioritization [31], history–based test prioritization [34], and the incorporation of varying test costs and fault severities into test case prioritization [35, 36].

Saff and Ernst [37-39] considered test case prioritization for Java in the context of continuous testing, which used spare CPU resources to continuously run regression tests in the background as programmer codes. They combined the concepts of test frequency and test case prioritization, and reported that continuous prioritized testing can reduce waste of development time.

Do et al. [40] has discussed that the rate of fault detection of JUnit test suites could be substantially improved and differences in terms of earlier studies that could be associated with the language and testing paradigm could be revealed by test case prioritization. They have presented a set of cost-benefits models for test case prioritization to analyze the practical consequences of these results and demonstrated that the perceived effectiveness differences could lead to savings in practice that vary considerably with the cost factors related to the specific testing processes.

Test case prioritization has also been done based on the relevant slices. Jeffry and Gupta in [41] proposed a prioritization technique based on the coverage requirements present in the relevant slices of the outputs of test cases. However, these prioritization techniques are based on different sources of information, such as history of recent or frequent errors and test cost, code coverage information, and have not considered test suite time.

Li et al. [42] studied five search techniques: two meta–heuristic search techniques (Hill Climbing and Genetic Algorithms), together with three greedy algorithms (Basic Greedy, Additional Greedy and 2–Optimal Greedy) and proved that Genetic Algorithms performed well in test case prioritization.

Srivastava [3] has presented test case prioritization algorithm to compute average faults discovered per minute. Using an Average Percentage of Faults Detected (APFD) metric result demonstrating the effectiveness of the algorithm has been presented. Calculating the effectiveness of prioritized and non-prioritized cases by means of APFD has been its main objective.

Krishnamoorthi and Mary [43] have proposed a system level test case prioritization (TCP) model from software requirement specification to increase user satisfaction with increased quality software at decreased cost and increase rate of critical defect detection. The proposed prioritization technique has been credibly proved to improve rate of severe fault detection when validated using two diverse validation techniques and experimented in three stages with student projects and two sets of industrial projects Jyoti et al. [27] have proposed a model for version specific regression testing to achieve 100% code coverage optimally. Modified lines covered by the test case based priority value have been used for prioritization of test cases.

Kavitha and Sureshkumar [44] have proposed an algorithm that performs rate of fault detection and fault impact based prioritization of test cases. Experimental results using an Average Percentage of Faults Detected (APFD) metric have demonstrated that more effective severe fault identification at earlier stage of the testing process could be obtained by the proposed algorithm for prioritized test cases compared to unprioritized ones.

Kavitha and Sureshkumar [45] have proposed an algorithm to prioritize the regression testing test cases. In order to prioritize the test cases, some of the factors to be calculated. These factors are used in the prioritization algorithm. The factors are (1) customer assigned priority of requirements, (2) developer-perceived code implementation

complexity, (3) changes in requirements, (4) fault impact of requirements, (5) completeness and (6) traceability (7) Execution time etc,. Based on these factors, a weight age is assigned to each test case in the software. According to the weight age assigned, the test cases are prioritized. The prioritization is based on sorting the test case according to its weights. Focusing only on the particular test cases based on the prioritization will reduce the computation cost and time. From the implementation results and APFD metric, the performance of the proposed method is evaluated.

Raiyani and Pandya [46] have presented the various types of test case prioritization techniques and their classifications, explaining selective and prioritizing test cases and search algorithms for test case prioritization. At the end, they discussed the approaches which may be used to compare various regression testing techniques and challenges faced by these approaches.

IV. TEST SUITE AUGMENTATION TECHNIQUES

Test-suite augmentation (TSA) is an area that is closely related to the selective-retest techniques just presented, but, to date, has received less attention than those techniques. TSA techniques use information about the changes from a program P to a modified version P' to identify criteria for retesting the changes. These criteria can then be used to (1) assess the test suite used to test P', which consists of T' and any new test cases developed, and (2) guide the selection of new test cases that are needed to adequately test P' with respect to the changes. Like selective-retest techniques, TSA techniques create models of P and P', assess the differences between them, and use these differences to compute what we call change-test requirements according to some criterion. (A test suite that satisfies the computed change-test requirements is said to be change adequate.) Many TSA techniques use differences in entities between P and P' as the criterion. These entities include data-flow [47-49], control-flow [5, 50], and both data-flow and control-flow [51-53].

Yoo and Harman [54] presented a study of test data augmentation. They experiment with the quality of test cases generated from existing test suites using a heuristic search algorithm.

Four recent papers [55-58] specifically address test suite augmentation. Two of these [55, 57] present an approach that combines dependence analysis and symbolic execution to identify test requirements that are likely to exercise the effects of changes, using specific chains of data and control dependencies to point out changes to be exercised. A potential advantage of this approach is a fine-grained identification of coverage needs; however, the papers present no specific algorithms for generating test cases. A third paper [56] presents a more general approach to program differencing using symbolic execution, that can be used to identify requirements more precisely than [55, 57] and yields constraints that can be input to a solver to generate test cases for those requirements. However, this approach is not integrated with reuse of existing test cases.

The test suite augmentation approach presented in [58] integrates an RTS technique [5] with an adaptation of the concolic test case generation approach presented in [59]. This approach leverages test resources and data obtained from prior testing sessions to perform both the identification of coverage requirements and the generation of test cases to cover them.

V. CHANGE IDENTIFICATION TECHNIQUES

One of the major difficulties in software maintenance is to identify changes and their impact automatically, since it is very difficult to keep track of the changes when a software system is modified extensively by several persons. This capability becomes even more crucial when the modifications are performed by one group of persons and regression testing is performed by another group of persons.

A. Types of Code Changes:

Types of code changes are explained as follows:

a. Data change*:* Any datum (i.e., a global variable, a local variable, or a class data member) can be changed by updating its definition, declaration, access scope, access mode and initialization. In addition, adding new data and/or deleting existing data are also considered as data changes.

b. Method/function change: A function can be changed in various ways, which can be classified into three types: component changes, interface changes, and control structure changes.

Component changes include:

a) Adding, deleting, or changing a predicate,
b) Adding, deleting a local data variable, and
c) Changing a sequential segment.

Control structure changes include:

a) Adding, deleting, or modifying a branch or a loop structure, and
b) Adding, or deleting a sequential segment.

Interface changes:

The interface of a function consists of its signature, access scope and mode, its interactions with other functions (for example, a function call). Any change on the interface is called an interface change of a function.

c. Class change: Direct modifications of a class can be classified into three types: Component changes, interface changes and relation changes.

Any change on a defined/ redefined member function or a defined data attribute is known as a component change.

A change is said to be an interface change if it adds, or deletes a defined/redefined attribute, or changes its access mode or scope.

A change is said to be a relation change if it adds, or deletes an inheritance, aggregation or association relationship between the class and another class.

d. Class library change: These include:

a) Changing the defined members of a class.
b) Adding, or deleting a class and its relationships with other classes.
c) Adding, or deleting a relationship between two existing classes.
d) Adding, or deleting an independent class.

B. Computing Program Differences Techniques:

Several techniques and tools for comparing source files textually (e.g., the UNIX diff utility) [60] have been proposed. However, these techniques have shortcomings. Textual differencing may report changes that have no effect on program semantics or syntax, such as the addition of a method that is never called and modifications in comments and white spaces, and do not consider changes in program semantics indirectly caused by textual modifications.

Horwitz's approach [61] computes both syntactic and semantic differences between two programs using a partitioning algorithm. Horwitz's technique is based on the program representation graph (PRG). Because PRGs are defined only for programs written in a language with scalar variables, assignment statements, conditional statements, while loops, and output statements only, the technique is limited and cannot be used in general. In particular, it cannot be applied to object-oriented programs.

Laski and Szermer [50] presented an algorithm that computes program differences by analyzing the control-flow graphs of the original and modified versions of a program. Their algorithm localizes program changes into clusters, which are single-entry, single-exit program fragments. Clusters are reduced to single nodes in the two graphs and are then recursively expanded and matched. Their control-flow graph representation does not model the object-oriented behaviors properly; thus, their algorithm may not compute accurate change information for object-oriented programs. For example, their algorithm cannot detect a difference at a call site that may invoke a method that has just been added in the modified version due to method overriding. Moreover, their algorithm for matching clusters has limited capability and may compute imprecise results. Their algorithm uses only the entry statements of two clusters to determine whether the two clusters are matched. If only the entry of one cluster in the modified version is changed, their algorithm may report that none of the statements in the cluster is matched. Their algorithm also does not allow matching of clusters at different nested levels. Thus, it may compute imprecise results.

Semantic diff [62], compares two versions of a program procedure-by-procedure, computes a set of input-output dependencies for each procedure and identifies the differences between two sets computed for the same procedure in the original and the modified programs. However, semantic diff is performed only at the procedure level and may miss changes that, although not affecting input-output dependencies, have inter-procedural side effects. Moreover, input-output dependencies are typically expensive to compute, so the approach is likely to have scalability issues when applied to medium and large programs.

Kung et al. [63, 64] introduced a class firewall concept to support program change and impact analysis based different class dependence relationships in object-oriented software. Using the class firewall concept, engineers can identify changed and affected classes based on class dependency, and perform class re-testing and re-integration using a strategy known as test orders. This method is useful only when a complete picture about class dependence relationships in object-oriented software is available.

BMAT (binary matching tool) [65] performs matching on both code and data blocks between two versions of a program in binary format. BMAT uses a number of heuristics to find matches for as many blocks as possible. Being designed for the purpose of program profile estimation, BMAT does not provide information about differences between matched entities. Moreover, BMAT does not compute information about changes related to object-oriented constructs, such as method overriding or changes in class hierarchy.

Maletic and Collard's approach [66] is a text-based program differencing technique. Their technique transforms C/C++ source files into a format called srcML that makes program structures more explicit than raw source code and leverages diff to compare the srcML representations for the original and modified versions of the source code. (srcML is an XML-based format that represents the source code annotated with syntactic information.) The results of the comparison are then post-processed to create a new XML document, also in srcML format, with the additional XML tags that indicate the common, inserted, and deleted XML elements. Their approach utilizes available XML tools to ease the process of extracting change-related information. However, the technique is limited by the fact that it still relies on line-based differencing information obtained from diff.

Apiwattanapong et al. [67] presented a technique for comparing object-oriented programs that identifies both differences and correspondences between two versions of a program. The algorithm is based on a method-level representation that models the object-oriented features of the language. Given two programs, their algorithm identifies matching classes and methods, builds a representation for each pair of matching methods, and compares the representation to identify similarities and differences. Empirical results show the efficiency and effectiveness of the technique on a real program.

There is a tool that was written by Iqbal [68] called ClassDiff that takes as an input two versions of the class file, analyzes them to identify changed methods and fields and then outputs those entities to an XML file for that specific class file. Changes can be of the following types [68]:

i. Changes in a super class that the class inherits from;
ii. Changes in interfaces that the class implements;
iii. Changes in an access flag of the class;
iv. Changes in methods;
v. Changes in fields.

VI. USING GAs IN REGRESSION TESTING

In the test case prioritization using GAs, the prioritization criterion is based on fitness function of population and genetic operators [69].

Li et al. [42] have proved experimentally that genetic algorithms (GAs) perform well for test case prioritization. The benefits of code coverage based prioritization techniques are measured using a weighted average of the percentage of faults detected (APFD), average percentage

block coverage (APBC), average percentage decision coverage (APDC) and average percentage statement coverage (APSC).

Krishnamoorthi and Mary [70] have proposed a GA based test case prioritization method. A superior rate of fault detection when compared to rates of randomly prioritized test suites has been obtained when the new suite that consists of subsequences of the original test suite prioritized by the proposed technique is executed within a time-constrained execution environment. Test cases have been prioritized utilizing structurally based criterion by the experiment and the GA has been analyzed with regard to effectiveness and time overhead. The effectiveness of the new test case orderings have been calculated using an Average Percentage of Faults Detected (APFD) metric.

Xu et al. [71] discussed the use of GA in test suite augmentation, identified several factors that impact the effectiveness of this approach, and presented the results of an empirical study exploring the effects of one of these factors: the manner in which existing and newly generated test cases are utilized by the GA. this results reveal several ways in which this factor can influence augmentation results, and reveal open problems that researchers must address if they wish to create augmentation techniques that make use of genetic algorithms.

Sujata et al. [72] have proposed an approach useful in black box environment. This approach is based on requirements based test case prioritization using GA. The main idea of this approach is to find the most severe faults early in the testing process and hence to improve the quality of the system according to customer point of view.

Kaur and Goyal [73] have proposed a GA to prioritize the regression test cases on the basis of complete code coverage. The results representing the effectiveness of the proposed algorithm are presented with the help of an Average Percentage of Code Covered (APCC) metric**.**

VII. CONCLUSION

Regression testing is the process of validating the modified software to provide confidence that the changed parts of the software behave as intended and that the unchanged parts of the software have not been adversely affected by the modifications. Researchers have proposed many techniques for the different regression testing aspects. In this paper, we reviewed the work which has been done so far in the field of regression testing. In addition, we organized the surveyed techniques into categories according to the regression testing aspects.

VIII. REFERENCES

[1] Park, H., Ryu, H., and Baik, J., "Historical Value-Based Approach for Cost-cognizant Test Case Prioritization to Improve the Effectiveness of Regression Testing", The Second International Conference on Secure System Integration and Reliability Improvement, pp. 39-46, Yokohama, July 14-17, 2008.

[2] Muccini, H., Dias, M., and Richardson, D. J., "Software Architecture-based Regression Testing", Journal of Systems and Software, Vol. 79, No. 10, pp. 1379-1396, October 2006.

[3] Srivastava, P. R. "Test Case Prioritization", Journal of Theoretical and Applied Information Technology, pp. 178-181, 2008.

[4] Graves, T. L., Harrold, M. J., Kim, J., Porter, A., and Rothermel, G., "An empirical study of regression test selection techniques", ACM Transactions on Software Engineering and Methodology, Vol.10, No.2, pp. 184-208, 2001.

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

[6] Orso, A., Shi, N., and Harrold, M.J., "Scaling regression testing to large software systems", Proceeding of the 12th ACMSIGSOFT International Symposium on Foundation of Software Engineering, pp 241-251, Newport Beach, CA, USA, 2004.

[7] Harrold, M.J., Jones, J.A., Li, T., Liang, D., Orso, A., Pennings, M., Sinha, S., and Spoon, S.A., "Regression test selection for java software", in: Proceedings of ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'01), Vol. 36, No. 11, pp. 312-326, 2001.

[8] Pasala, A., Yannick, L.H., Fady, A., Appala R. G., and Ravi P. G., "Selection of regression test suite to validate software applications upon deployment of upgrades", 19th Australian Software Engineering conference, pp 130-138, Aswec, March 26-28, 2008.

[9] Apiwattanapong, T., Orso, A., and Harrold, M.J., "JDiff: A Differencing Technique and Tool for Object-Oriented Programs", Journal of Automated Software Engineering, Vol.14, No.1, pp. 3-36, March 2007.

[10] Koju, T., Takada, S., and Doi, N. "Regression test selection based on intermediate code for virtual machines", Proceeding of International Conference on Software Maintenance (ICSM 03), pp. 420-429, Amsterdam, the Netherlands, September 22-26, 2003.

[11] Xiaoxia, R., Barbara, G. R., Maximilian, S., and Frank, T., "Chianti: A prototype change impact analysis tool for Java", Proceedings of 27th international conference on Software engineering (ICSE), St. Louis,USA , pp. 664-665, May 15-21, 2005.

[12] Ravi P. G., Anjaneyulu Pasala, Kailash KP and Benny Leong, "Specification-based approach to select regression test suite to validate change software", 15th Asia-Pacific Software Engineering conference, pp. 153-160, Beijing, December 3-5, 2008.

[13] Briand, L.C., Labiche, Y., and He, S., "Automating regression test selection based on UML designs" Original Research Article Information and Software Technology, Vol 51, No 1, pp 16-30, January 2009.

[14] Orest, P., Hunay, U., and Andrews, A., "Regression Testing UML Designs", Proceedings of 22nd IEEE International Conference on Software Maintenance (ICSM), Philadelphia, Pennsylvania, , pp. 254-264 ,September 24-27, 2006.

[15] Ali, A., Nadeem, A., Iqbal, M.Z., and Usman, M., "Regression testing based on UML design models", 13th IEEE International Symposium on Pacific Rim Dependable Computing, pp 85-88, 2007.

[16] Paul, R., Yu, L., Tsai, W-T., and Ba, X., "Scenario-Based Functional Regression Testing", COMPSAC '01 Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development, pp. 496,Washington, DC, 2001.

[17] Chakrabarti, S. K., and Srikanth, Y. N., "Specification based regression testing using explicit state space enumeration", International conference on software engineering advances, Tahiti, October 29 to November 3, 2006.

[18] Chen, Y., Robert L. P., and Sims, D. P., "Specification-based Regression Test Selection with Risk Analysis", CASCON'02 Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research, pp. 175-182, IBM Canada, September 2002.

[19] Chen, Y., Robert L. P., and Ural, H., "Model-Based regression test suite generation using dependency analysis", 3rd International workshop on advances in model-based testing, pp 54-62, London, July 9-12, 2007.

[20] Chen, Y. F., Rosenblum, D. S., and Vo, K. P., " TestTube: A system for selective regression testing". In ICSE '94: Proceedings of the 16th international conference on Software engineering, pp. 211-222, Sorrento, Italy, May 16-21, 1994.

[21] Rosenblum, D.S. "Towards a Method of Programming With Assertions", Proceedings. 14th International. Conference on Software Engineering, proceeding.New York: Association for computing Machinery, pp. 92-104, May 1992.

[22] Chen, Y-F., Nishimoto, M., and Ramamoorthy, C.V., "The C Information Abstraction System", IEEE Transactions on Software Engineering, Vol.16, No.3, pp. 325-334, March 1990.

[23] Rothermel, G., and Harrold, M. J., "Analyzing regression test selection Techniques". IEEE Transactions on Software Engineering, Vol. 22, No. 8, pp. 529–551, August 1996.

[24] Fisher II M, Jin, D., Rothermel, G., and Burnett, M., "Test reuse in the spreadsheet paradigm." ISSRE '02 Proceedings of the 13th International Symposium on Software Reliability Engineering. pp. 257-268, Annapolis, MD, USA, November 12-15, 2002.

[25] Malhotra, R., Kaur, A., and Singh, Y., "A Regression Test Selection and Prioritization Technique". Journal of Information Processing Systems, Vol.6, No.2, Pp. 235-252 June 2010.

[26] Songyu Chen, S., Chen1, Z., Zhao, Z., Xu, B., and Feng, Y., "Using Semi-Supervised Clustering to Improve Regression Test Selection Techniques". ICST '11 Proceedings of the 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation pp. 1-10, Berline, Germany, March 21-25, 2011.

[27] Jyoti, A., Sharma, Y. K., Bagla, A., and Pandey, D., "Recent Priority Algorithm In Regression Testing", International Journal of Information Technology and Knowledge Management, Vol.2, No.2, pp. 391-394, July-December 2010.

[28] Roongruangsuwan, S., Daengdej, J., "Test case prioritization techniques", Journal of Theoretical and Applied Information Technology, Vol. 18, No. 2, pp 45-60, 2010.

[29] Rothermel, G., Untch, R., Chu, C., and Harrold, M. J., "Test case prioritization: An empirical study", In Proceedings ICSM 1999, pp. 179–188. Oxford, England, UK, August 30 to September 3, 1999.

[30] Wong, W. E., Horgan, J. R., London, S., and Agrawal, H., "A study of effective regression testing in practice", ISSRE '97 Proceedings of the Eighth International Symposium on Software Reliability Engineering, pp. 264–274, Albuquerque, NM, YSA, Nov 2-5, 1997.

[31] Rothermel, G., Roland H. U., Chu, ch., and Harrold, M. J., "Prioritizing Test Cases For Regression Testing", IEEE Transactions on Software Engineering, Vol.27, No.10,pp. 929-948, October 2001.

[32] http://msdn.microsoft.com/library/default.asp?url=/library/enus/vsent7/html/vxconregressiontesting.asp. Last accessed 26/1/2011.

[33] Srivastava, A., and Thiagarajan, J., "Effectively prioritizing tests in development environment", In ISSTA '02, Proceedings of the 2002 ACM SIGSOFT International Symposium on Software testing and analysis, New York, NY, USA, ACM Press, pp. 97–106, 2002.

[34] Kim, J-M., and Porter, A., "A history-based test prioritization technique for regression testing in resource constrained environments", In ICSE '02: Proceedings of the 24th International Conference on Software Engineering, New York, NY, USA, ACM Press, pp. 119–129, 2002.

[35] Elbaum, S., Malishevsky, A., and Rothermel, G., "Incorporating varying test costs and fault severities into test case prioritization", In ICSE '01, Proceedings of the 23rd International Conference on Software Engineering, Washington, DC, USA, IEEE Computer Society, pp. 329–338, 2001.

[36] Elbaum, S., Malishevsky, A.G., and Rothermel, G., "Test Case Prioritization: A Family of Empirical Studies," IEEE Trans. Software Eng., Vol. 28, No. 2, pp. 159-182, Feb. 2002.

[37] Saff, D., and Ernst, M. D., "Reducing wasted development time via continuous testing", ISSRE '03 Proceedings of the 14th International Symposium on Software Reliability Engineering, pp. 281–292, Denver, Colorado November 17-20, 2003.

[38] Saff, D., and Ernst, M. D., "An experimental evaluation of continuous testing during development", ISSRE '04 Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis, Vol. 29, No.4, pp. 76–85, July 2004.

[39] Saff, D., and Ernst, M. D., "Continuous testing in Eclipse", In Proceedings of the 2nd Eclipse Technology Exchange Workshop, Barcelona, Spain, March 30, 2004.

[40] Do, H., Rotherme, G., and Kinneer, A., "Prioritizing JUnit Test Cases: An Empirical Assessment and Cost-Benefits Analysis", Journal Empirical Software Engineering", Vol.11, No.1, pp. 33-70, March 2006.

[41] Jeffrey, D., and Gupta, N., "Test Case Prioritization Using Relevant Slices", ISSTA, Portland, July, 2006.

[42] Li, Z., Harman, M., Hierons, and Robert M., "Search algorithms for regression test case prioritization", IEEE Transactions on Software Engineering, vol.33, No.4, pp. 225–237, 2007.

[43] Krishnamoorthi, R., and Mary, S.A., "Factor oriented requirement coverage based system test case prioritization of new and regression test cases", Information and Software Technology, Vol.51, No.4, pp. 799–808, April 2009.

[44] Kavitha, R., and Sureshkumar, N., "Test Case Prioritization for Regression Testing based on Severity of Fault", International Journal on Computer Science and Engineering, Vol.2, No.5, pp. 1462-1466, 2010.

[45] Kavitha, R., and Sureshkumar, N., "Factors Oriented Test Case Prioritization Technique in Regression Testing". European Journal of Scientific Research Vol. 55, No.2, pp.261-274, 2011.

[46] Raiyani, A. G., Pandya, S. S., "Proritization technique for minimizing number of test cases" International Journal of Software Engineering Research & Practices, Vol. 1, N0. 1, Jan, 2011.

[47] Linnenkugel, U., and Mullerburg, M., "Test data selection criteria for (software) integration testing". ISCI '90 Proceedings of the first international conference on systems integration on Systems integration '90, pp. 709–717, Apr. 1990.

[48] Ostrand, T. J., and Weyuker, E. J., "Using dataflow analysis for regression testing". In: Sixth Annual Pacific Northwest Software Quality Conference, pp. 233–247, Portland, September, 1988.

[49] Taha, A. B., Thebaut, S. M., and Liu, S. S., "An approach to software fault localization and revalidation based on incremental data flow analysis". In Proceedings of the 13th Annual International Computer Software and Applications Conference, pp. 527–534, Orlando, FL, USA, Sept 20-22, 1989.

[50] Laski, J. and Szermer, W., "Identification of program modifications and its applications in software maintenance," in Proceedings of the IEEE Conference on Software Maintenance, (Orlando, FL, USA), pp. 282–290, November 1992.

[51] Bates, S., and Horwitz, S., "Incremental program testing using program dependence graphs." In Proceedings of Symposium on the Prin. Of Program Language, pp. 384–396, Jan, 1993.

[52] Gupta, R., Harrold, M., and Soffa, M., "Program slicing-based regression testing techniques". J. Software Testing Verification Reliability, Vol. 6, No. 2, pp. 83–111, June 1996.

[53] Rothermel, G., and Harrold, M. J., "Selecting tests and identifying test coverage requirements for modified software". ISSTA '94 Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis, PP 169–184, Seattle, WA, Aug, 1994.

[54] Yoo, S., and Harman, M., Test data augmentation: Generating new test data from existing test data. Technical Report TR-08-04, Dept. of Computer Science, King's College London, July 2008.

[55] Apiwattanapong, T., Santelices, R., Chittimalli, P. K., Orso, A., and Harrold, M. J., "Matrix: Maintenance-oriented testing requirements identifier and examiner." TAIC-PART '06 proceeding of the Testing: Academic and Industrial Conference-Practice and Research Techniques, pp. 137–146, Windsor, Aug 29-31, 2006.

[56] Person, S., Dwyer, M. B., Elbaum, S., and P˘as˘areanu, C. S., "Differential symbolic execution". Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, No.1, pp. 226–237, Atlanta, Georgia, USA, Nov 9-15, 2008.

[57] Santelices, R., Chittimalli, P. K., Apiwattanapong, T., Orso, A., and Harrold, M. J., "Test-suite augmentation for evolving software". In Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering, PP. 218-227, L'Aquila Sept 15-19, 2008.

[58] Xu, Z., and Rothermel, G., "Directed test suite augmentation". In Asia-Pacific. Software Engineering Conference, pp. 406-413, Penang, Dec 1-3, 2009.

[59] Sen, K., Marinov, D., and Agha, G., "CUTE: A concolic unit testing engine for C". ESEC/FSE-13 Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, Vol. 30, No. 5, pp. 263–272, September. 2005.

[60] Myers, E. W., "An O(ND) difference algorithm an its variations," Algorithmica, Vol.1, No.2, pp. 251–266, 1986.

[61] Horwitz, S., "Identifying the semantic and textual differences between two versions of a program," in Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, (White Plains, NY, USA), Vol. 25, No. 6, pp. 234–246, June 1990.

[62] Jackson, D. and Ladd, D. A., "Semantic diff: A tool for summarizing the effects of modifications". In Proceedings of the International Conference on Software Maintenance, pp. 243.252, Victoria, B.C., September 1994

[63] Kung, D., Gao, J., Hsia, P., Chen, C., and Toyoshima, Y., "Change impact identification in object-oriented software maintenance," In Proceedings of IEEE International Conference on Software Maintenance, IEEE Computer Society Press, pp. 202-211, Victoria, BC, Canada, September 19-23, 1994.

[64] Kung, D., Gao, J., Hsia, P., and Wen, F., Toyoshima, Y., and Chen, C., "On regression testing of object-oriented

programs," Journal of Systems and Software, Vol.32, No.1, pp. 21-40, Jan. 1996.

[65] Wang, Z., Pierce, K., and McFarling, S., "BMAT – a binary matching tool for stale profile propagation," Journal of Instruction-Level Parallelism, Vol. 2, May 2000.

[66] Maletic, J. I. and Collard, M. L., "Supporting source code difference analysis," in Proceedings of the 20th IEEE International Conference on Software Maintenance, (Chicago, IL, USA), pp. 210–219, September 2004.

[67] Apiwattanapong, T., Orso, A., and Harrold, M. J., "A Differencing Algorithm for Object-Oriented Programs," 19th International Conference on Automated Software Engineering (ASE'04), pp. 2-13, Linz, Austria, Sept 20-24, 2004.

[68] Iqbal, A., Identifying modifications and generating dependency graphs for impact analysis in a legacy environment. Master's thesis, McMaster University, 2011.

[69] Carzaniga, A., Rosenblum, D. S., and Wolf, A. L., "Design and evaluation of a wide-area event notification service," ACM Transactions on Computing Systems, Vol.19, pp. 332–383, August 2001.

[70] Krishnamoorthi1, R., and Mary, S.A., "Regression Test Suite Prioritization using Genetic Algorithms", International Journal of Hybrid Information Technology, Vol.2, No.3, PP. 35-51 July, 2009.

[71] Xu, Z., Cohen, M. B., and Rothermel, G.,"Using a Genetic Algorithm for Test Suite Augmentation" Genetic and evolutionarty computation conference, (GECCO), Portland, Oregon, USA, July 7-11, 2010.

[72] Sujata., Kumar, M., Kumar, V., "Requirements based Test Case Prioritization using Genetic Algorithm", International Journal of Computer Science and Technology, Vol.1,No.2,pp. 189- 191, December 2010.

[73] Kaur, A., and Goyal, S., "a Genetic Algorithm for Regression test case Prioritization Using Code Coverage" International Journal on Computer Science and Engineering (IJCSE), Vol. 3, No. 5, May 2011.

