International Journal of Information Technology and Management

Vol. V, Issue No. I, August-2013, ISSN 2249-4510

A COMPLETE ANALYSIS OF TEST CHOICE AND EFFICIENT REGRESSION TESTING

International Journal of Information Technology and Management
Vol. V, Issue No. I, August-2013, ISSN 2249-4510

A Complete Analysis of Test Choice and Efficient Regression Testing

Meena Mehta

Assistant Professor (Adhoc) Maharaja Agarsen College, University of Delhi

Abstract – Regression testing is a costly but crucial problem in software development. Both the research community and the industry have paid much attention to this problem. However, are the issues they concerned the same? The paper try to do the survey of current research on regression testing and current practice in industry and also try to find out whether there are gaps between them. The observations show that although some issues are concerned both by the research community and the industry gay, there do exist gaps.

Regression testing is an important and expensive activity that is undertaken every time a program is modified to ensure that the modifications do not introduce new bugs into previously validated code. An important research problem, in this context, is the selection of a relevant subset of test cases from the initial test suite that would minimize both the regression testing time and effort without sacrificing the thoroughness of regression testing. Researchers have proposed a number of regression test selection techniques for different programming paradigms such as procedural, object-oriented, component-based, database, aspect, and web applications. In this paper, we review the important regression test selection techniques proposed for various categories of programs and identify the emerging trends.

---------------------------♦-----------------------------

INTRODUCTION

No matter how well conceived and tested before being released, softwarewill eventually have to be modified in order to fix bugs or respond to changes in user specifications. Regression testing must be conducted to confirm that recent program changes have not adversely affected existing features and new tests must be conducted to test new features. Testers might rerun all test cases generated at earlier stages to ensure that the program behaves as expected. However, as a program evolves the regression test set grows larger, *old* tests are rarely discarded, and the expense of regression testing grows. Repeating all previous test cases in regression testing after each minor software revision or patch is often impossible due to the pressure of time and budget constraints. On the other hand, for software revalidation, arbitrarily omitting test cases used in regression testing is risky. In this paper, we investigate methods to select small subsets of effective fault-revealing regression test cases to revalidate software.

Many techniques have been reported in the literature on how to select regression tests for program revalidation. The goal of some studies is to select every test case on which the new and the old programs produce different outputs, but ignore the coverage of these tests in the modified program. In general, however, this is a difficult, sometimes un-decidable, problem. Others place an emphasis on selecting existing test cases to cover *modified* program components and those may be affected by the modifications, i.e., they use coverage information to guide test selection. They are not concerned with finding test cases on which the original and the modified programs differ. Consequently, these techniques may fail to select existing tests that expose faults in the modified program. They may also include test cases that do not distinguish the new program from the old for re-execution.

In this paper, a combination of both techniques described above is used. We first select tests from the regression suite that execute any of the modifications in the old program and refer to this technique as a modification-based test selection technique. This includes tests that have to be used for revalidation, but it also contains some *redundant* tests on which the old and the new program produce the same outputs.

Then, depending on the available resources, a *tradeoff* between what we should ideally do in regression testing and what we can afford to do is applied to determine which tests, among those necessary, should be reexecuted first, and which ones have lower priority or are to be omitted from reexecution. Two techniques, test set minimization and prioritization, are used. Although both of them may exclude certain necessary regression tests, the value of using them is explained below.

BASIC METHOD AND TERMINOLOGY

The proposed hybrid approach is based on the selection and prioritization of the test cases for inter procedural programs. It is a version-specific technique that takes into account the variable usage in the old as well as the modified program, named as P_e and P_e'respectively. The technique requires that the test cases in the original test suite Te not only contain test case identification, expected input and expected output (as per past practice) but also the variable(s) that is (are) being checked by this test case and the module to which the variable belongs. It selects all those variables that are in the changed statements and then selects only those test cases that either correspond to these variables or to the variables computed from them recursively. Multiplelevel prioritization of the selected test cases is performed on the basis of variable usage. Variables are a vital source of changes in the program and this approach captures the effect of change in terms of variable computation. The approach takes into account the changes in the variables and its ripple effect. Appendix 1 defines some related terminology.

A computed variable table (CVT_e) is prepared (maintained through development testing) in which the list of variables computed from other variables is maintained. An array with the information of the number of times the variable is used in computation is also maintained during development testing in VDC_e (Variable Dependency Count). The algorithm is presented in Appendix 2 which demonstrates the technique. Initially, the resultant test suite is set to null. In step 2 of algorithm, a list of variables "V_e" is created from changed (inserted/modified/deleted) lines using array CLB which maintain changed line numbers.

If any variable is deleted permanently from the program by modification or deletion of any line, it results in modified versions of V_e, VDC_e, and CVT_e (by deleting the row corresponding to those variables). The selection step and priority1 assignment step (step 3) selects all those test cases that correspond to variables contained in modified Ve. These test cases are assigned Priority1 as 1 (step 3(i), (ii)). Step 3(iv) of the algorithm gets the variable computed from variables found above from modified CVT and sets Priority1 of corresponding test case as 2 onwards. If the same test case already exists then Priority1 is kept as the minimum of the two.

After assigning Priority1, Priority2 are assigned, as stated in step 4 of the algorithm. The purpose of assigning Priority2 is to further prioritize the test cases that have the same value as Priority1. Priority2 is based on the dependency count as in the modified VDC_e. The variables which have highest dependency count are selected. The test cases corresponding to these selected variables are assigned Priority2 as 1. Then, the variables having next highest dependency count are selected. The test cases corresponding to them are assigned priority2 as 2 and so on. Step 4(i) to Step 4(iii) chooses all the test cases with same Priority1 and Step 4(v) further prioritize according to the dependency count. The resultant test suite T' has test cases having Priority1 and Priority2 assigned.

REGRESSION TESTING IN PRACTICE

Commercial Regression Testing Tools - In this section, a set of leading regression testing tools will be briefly reviewed. There are tons of testing tools nowadays and there are some other interesting tools, e.g., which are not discussed in this paper. The main reasons why the tools are selected in this paper are:

1. These tools are widely used in the industry and have high reputation by the users.

2. The free trial version is available online or some important documents which can give enough details are available online.

3. The tools can support the regression testing well.

Case Studies - In this section, several case studies will be done to show the effort to apply the regression testing technologies into the practice or the current regression practice in the industry. The tool they developed is called Echelon, a test prioritization system. The distinguish feature of the tool is that it analyze the binary code while most the regression prioritizing technologies analyze the source code. The Echelon is part of the Magellan tool set. The core of the Magellan is a SQL Server database which stores test coverage information for each test. All the program binaries and the source codes are stored separately. Magellan provides a set of tools which are commonly needed during the test process. It includes test coverage collection tool, GUI interface to map coverage data source code and test migration tool to migrate coverage data from older version to the new one. Echelon is actually the prioritization system of the tool set. The following graph is the architecture of the Echelon system.

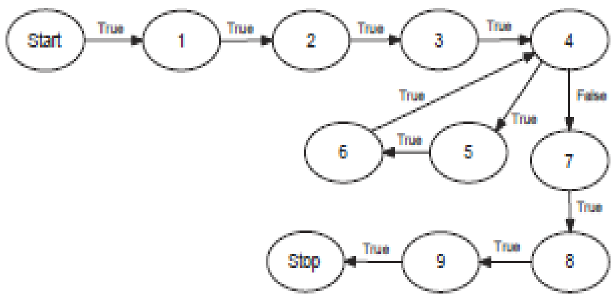
GRAPH MODELS FOR PROCEDURAL PROGRAMS

Graph models of programs have extensively been used in many applications such as program slicing, impact analysis, reverse engineering, computation of program metrics, regression test selection, etc. Analysis of graph models of programs is more efficient compared to textual analysis, and various types of relationships among program elements are also not explicit in the code. This has led to several representations such as Control Flow Graph (*CFG*), Program Dependence Graph (*PDG*) and System Dependence Graphs (*SDG*) being proposed for procedural programs. In the following, we briefly discuss the important graph models proposed for procedural programs.

Flow Graph - A flow graph for a program *P* is a directed graph (*N, E*) where the program statements correspond to the set of nodes *N* in the flow graph, and the set of edges *E* represent the relationships among the program statements. However, the nodes in a flow graph can also correspond to basic blocks in a program. Typically it is assumed that there are two distinguished nodes called *start* with in-degree zero and *stop* with out-degree zero. There exists a path from *start* to every other node in a flow graph, and similarly, there exists a path from every other node in the graph to *stop*.

Control Flow Graph - A control flow graph (*CFG*) is a flow graph that represents the sequence in which the different statements in a program get executed. That is, it represents the flow of execution of control in the program. In fact, a *CFG* captures all the possible flows of execution of a program.

The *CFG* of the program *P* is the flow graph *G* = (*N, E*) where an edge (m, n) $\in$ *E* indicates possible flow of control from node *m* to node *n*. Figure 4 represents the *CFG* of the program shown in Figure. Note that the existence of an edge (x, y) in a *CFG* does not necessarily mean that control *must* transfer from *x* to *y* during a program run.



Data Dependence Graph - Dependence graphs are used to represent potential dependencies between the elements of a program. In the following, we discuss data and control dependencies between program elements and their graph representations.

Data Dependence: Let *G* be the *CFG* of a program *P*. A node $n \in G$ is said to be data dependent on a node $m \in G$, if there exists a variable *var* of the program *P* such that the following hold:

1. The node *m* defines *var*,
2. The node *n* uses *var*,
3. There exists a directed path from *m* to *n* along which there is no intervening definition of *var*.

THREATS TO VALIDITY

On carefully analyzing the behavior of the two techniques, we observed that the proposed technique gives better results for the programs containing intensive variable computations. Further, the technique does not build the new test cases required for the code added due to modification. Moreover, the types of decision statements may affect the percentage of coverage achieved. Coverage depends on the type of decision statements: Some decisions are taken after the execution such as in “do…while,” and “for,” and some before execution such as “while.” There are other options available in the programming language such as “switch statement,” “multiple condition decision statement,” “if…else,” and so on, which give different coverage for the same test case.

REGRESSION TEST SUITE

A regression test suite of 1000 distinct tests was created based on the operational profile of how the space program was used. An operational profile, as formalized by Musa and used in our experiment, is a set of the occurrence probabilities of various software functions. To obtain an operational profile for space we identified the possible functions of the program and generated a graph capturing the connectivity of these functions. Each node in the graph represented a function. Two nodes, A and B, were connected if control could flow from function A to function B. There was a unique start and end node representing functions at which execution began and terminated, respectively.

A path through the graph from the start node to the end node represents one possible program execution. To estimate the occurrence probability of the software functions, each arc was assigned a transition probability, i.e., the probability of control flowing between the nodes connected by the arc. For example, if node A is connected to nodes B, C and D, and the probabilities associated with arcs A-B, A-C, and A-D are, respectively, 0.3, 0.6 and 0.1, then after the execution of functionA the programwill perform functions B, C or D, respectively, with probability 0.3, 0.6, and 0.1. There was a total of 236 function nodes. Transition probabilities were determined by interviewing the program users.

RTS TECHNIQUES FOR OBJECT-ORIENTED PROGRAMS

The object-oriented paradigm is founded on several important concepts such as encapsulation, inheritance, polymorphism, dynamic binding, etc. These concepts lead to complex relationships among various program elements, and make dependency analysis more difficult. Moreover, in object-oriented development, reuse of existing libraries, class definitions, program executables (blackbox components), etc. are emphasized to facilitate faster development of applications. These libraries and components frequently undergo independent modifications to fix bugs and enhance functionalities.

This creates a new dimension in regression testing of object-oriented programs that use these third party components or libraries, since the source code for such libraries are often not available. These features, therefore, raise challenging questions on how to effectively select regression test cases that are safe for such programs.

CONCLUSION

Regression testing is a costly but crucial problem in software development. There are a lot of researches addressing this area while in industry regression testing is also a crucial process. The paper is an initial work to survey both sides. And observations show that while there are some issues both sides are concerned, there are still some gaps between them. The gaps may be good direction for the research, or more work should be done to try to apply the technology from lab to the industry. In view of the fact that static analysis of large software systems is computationally expensive, model-based RTS techniques appear to be a promising approach that not only scales well, but is more efficient. Furthermore, of late MDD has been receiving a lot of attention. In MDD, there exists a close relationship between the design model(s) and code in the sense that any change to the model gets reflected in the code and vice versa. Therefore, instead of performing RTS on code, test selection could be automatically performed based on design models. Model-based RTS can also help to take into consideration several aspects of program behavior (like state transitions, message paths, task criticality, etc.) that are not easily identified from static code analysis.

In this paper, we have proposed and validated a technique, which is an extension of an existing technique proposed by us in an analogous study. The technique proposed in this work is compared with a technique given in literature by Rothermal et. al. The main results of this work are:

- Numbers of test cases selected are less for the proposed technique than the compared one.
- The technique selected less number of test cases as compared to other technique.
- The rate of fault detection using the technique is higher for the resultant test suite

REFERENCES

- Srivastava and J. Thiagarajan. Effectively Prioritizing Tests in Development Environment. In Proceedings of the International Symposium on Software Testing and Analysis, pages 97–106, July 2002.
- G. Rothermel and M. J. Harrold, A Safe, Efficient Regression Test Set Selection Technique, ACM Transactions on Software Engineering and Methodology, V.6, no. 2, April 1997, pages 173-210.
- G. Rothermel, Efficient effective regression testing using safe test selection techniques, PhD thesis, Clemson University, 1996.
- G.Rothermel and M. J. Harrold, Analyzing Regression Test Selection Techniques, IEEE Transactions on Software Engineering, V.22, no. 8, August 1996, pages 529-551.
- J.Bible, G. Rothermel, D. Rosenblum, Coarse- and fine-grained safe regression test selection. ACM Transactions on Software Engineering and Methodology 10 (2), (2001) 149-183.
- K. Abdullah and L.White. A firewall approach for the regression testing of object-oriented software. In Proceedings of 10th Annual Software Quality Week, page 27, May 1997.
- M. R. Gary and D. S. Johnson, “Computers and Intractability,” Freeman, New York, 1979.
- P.A Brown and D. Hoffman. The application of module regression testing at TRIUMF. Nuclear Instruments and Methods in Pysics Research, Section A, . A293(1-2):377- 381, August 1990.
- R. Binder. Testing Object-Oriented Systems:Models, Patterns, and Tools. Addison-Wesley, 1999.
- R. Gupta, M. J. Harrold, and M. L. Soffa, “An approach to regression testing using slicing,” in Proceedings of the Conference on SoftwareMaintenance, pp 299-308, Orlando, FL, November 1992.
- T. Ball. On the limit of control flow analysis for regression test selection. In ISSTA ’98: Proceedings of the 1998 ACM SIGSOFT international symposium on Software testing and analysis, pages 134–142, 1998.