An Overview of Regression Testing

Yuejian Li
Nancy J. Wahl
Middle Tennessee State University
wahl@mtsu.edu

Abstract

Regression testing is an important part of the software development life cycle. Many articles have been published lately detailing the different approaches. This article is an overview of regression testing in the following areas: types of regression testing; unit, integration and system level testing, regression testing of global variables, regression testing of object-oriented software, comparisons of selective regression techniques, and cost comparisons of the types of regression testing.

Introduction

As a software system ages, the cost of maintaining the software dominates the overall cost of developing the software. It is reported that the cost of maintenance is increasing [8, 9]. In the 1990s, the estimated expenditure exceeds 70% of total software costs [9]. A large percentage of that maintenance expense is devoted to testing. Regression testing is a testing process that is used to determine if a modified program still meets its specifications or if new errors have been introduced. Improvements in the regression testing process would help reduce the cost of software.

Economic considerations are not the only reason to improve regression testing. It is important to developers and users of software that their software works correctly and is reliable. Testing can be used to build confidence in the correctness of software and to increase software reliability. Regression testing is an important type of testing. Onoma et al. reported that the secret in delivering quality software is good regression testing [3]. Many researchers have tried to improve the regression testing process. The following sections contain an overview of regression testing techniques.

Types of regression testing

Regression testing is necessary when a program has been changed and is usually done during maintenance. If adaptive or perfective maintenance were performed then program specifications were modified. If corrective maintenance was performed then the specifications may not have been modified. Leung and White describe two types of regression testing based on whether specifications were changed [10]. Corrective regression testing applies when specifications are unmodified and test cases can be reused. Progressive regression testing applies when specifications are modified and new test cases must be designed.

When an existing suite of tests is available to be reused during regression testing, the question arises as to which tests should be run—all of them or some of them? Two regression testing strategies have developed as an answer to the question. The *retest-all* strategy reuses all tests, but this strategy may waste time and resources due to executing unnecessary tests. When the change to a system is minor, the *retest-all* strategy would be rather wasteful. The *selective* strategy uses a subset of the existing test cases to reduce the retesting cost. Chen et al. compared *selective* regression testing to *selective* compilation of software using the *make* utility [8]. *Make* recompiles only source files that have been changed. Similarly, in regression testing using a selective strategy a test unit must be rerun if and only if any of the program entities (e.g., functions, variables, etc.) it covers have been changed. The challenge is to identify the dependencies between a test case and the program entities it covers. Determining such dependency information requires sophisticated analyses of both the source code and the execution behavior of the test cases.

Rothermel and Harrold [14] and Onoma et al. [3] specified a typical selective regression testing process. In general, the process includes following steps:

- Identify affected software components after program P has been modified to P′.
- Select a subset of test cases T′ from an existing test suite T that covers the software components that are affected by the modification.
- Test modified program P′ with T′ to establish the correctness of P′ with respect to T′.
- Examine test results to identify failures.
- Identify and correct the fault(s) that caused a failure.
- Update the test suite and test history for P′.

From the process described above we can observe the following characteristics of selective regression testing: (1) identifying the program components that must be retested and finding those existing tests that must be rerun are essential, (2) when selected test cases satisfy retest criterion, new test cases are not needed, and (3) once regression testing is done, it is necessary to update and store the test information for reuse at a later time. An important retest requirement is to determine which software components are exercised by each test.

Unit, integration and system level regression testing

Three types of software testing are applied during the software development life cycle: unit testing, integration testing, and system testing. Testing at the unit, integration and system levels reveals different types of failures. Regression testing is one kind of testing that is applied at all three levels. White [4] suggested applying specification-based (system) testing before structure-based (unit) testing to get more test case reuse because system testing cases could be reused in unit testing but the converse is not true. McCarthy [2] suggested applying regression unit testing first to find faults early. In the following paragraphs various *selective* regression testing techniques at the unit, integration, and

system levels are discussed. (For an additional analysis of test selection techniques refer to the article by Rothermel and Harrold [14].)

Selective regression testing at the unit level

Unit testing is the process of testing each software module to ensure that its performance meets its specification [4]. Most existing regression testing techniques focus on unit testing.

Yau and Kishimoto [5] developed a method based on the input partition strategy. This approach divides the input domain of a module into a set of input partitions (classes) using the information from the program specification and code. The input partition P is derived by intersecting the two input partitions P_s and P_c which are generated respectively from the program specification and code [5]. The retest criterion of the testing method is to ensure that each new or changed input partition is executed by at least one test case. A cause-effect graph of the specifications is constructed and a test information table is created to help in selecting test cases. The specification partitions are the different possible combinations of input conditions from the graph [5]. The code partitions are the paths in the code that correspond to the specification partitions [5]. A test case is chosen if it exercises one of these paths. If the selected test cases do not satisfy the testing criterion, new test cases are generated. The authors demonstrated the feasibility of their method on FORTRAN programs that perform mathematical computations.

McCarthy [2] provides a way to automate unit regression testing using *make*. When using this method, a small test program is created for a module then a makefile is written with targets indicating the test case's dependency on the module it tests. The test program accepts input from a data file via "standard in" and sends output to "standard out" that is captured in a file. After unit tests have passed the first time, *make* is run with the accepted target. That makes reference copies (often called "canon files") of the test results [2]. When changes are made in the future, the unit test files corresponding to the changed modules are rerun and corresponding outputs are generated. The authors then run another target called the regression target to compare the new test outputs to the outputs in the canon files and list the differences in a regression report file [2]. The report file shows immediately what changed. If the changes are correct, they are accepted and new canon files are made; otherwise, the problem is corrected and the regression tests will be run again. The test programs, sample data, canon files should be placed in a source code control system so that earlier versions can be recreated [2].

Gupta, Harrold and Soffa [11] proposed an approach to *data flow-based regression testing* using program slicing techniques. Like traditional data flow testing strategies, this approach computes definition-use (def-use) associations in a program and uses certain data adequacy criteria to select particular def-use associations to test. To implement this approach the *ForwardWalk* and *BackwardWalk* algorithms are used to identify the affected def-use associations that must be retested [11]. The backward slicing starts from the program edit point P for variable V (i.e., the changed statement containing the variable), and works backward to detect statements that may affect the value of V in P [11]. The forward slicing starts from the same edit point P and works forward to find the uses and subsequent def-uses, which are affected by the change [11]. Because the slicing based algorithms explicitly identify both directly and indirectly affected def-use associations, it is easy to generate all tests for the affected def-use associations during regression testing [11]. Thus, when applying this technique the authors neither have to maintain a test history nor recompute complete data flow information to enable *selective* retesting [11].

Korel and Al-Yami [18] developed an approach for automated generation of tests at the unit level of regression testing and applies when the specifications are not changed for a module (corrective regression testing). Their approach finds tests that generate different results when run on the original and modified modules. Each test case reveals 1 or more failure [18]. The purpose of their approach is to find input that causes the modified module to behave differently from the original module.

Ball [19] reworked Rothermel's and Harrold's algorithm [9] that is described in the section on regression testing at the integration level. Rothermel's and Harrold's algorithm uses control flow graphs to select tests for regression testing. Ball also used control flow graphs for test selection. He relates control flow graphs and their intersection to deterministic finite automaton and the intersection of two regular languages [19]. Control flow graphs are created for the original module and the modified version of the module. The intersection of the graphs determines which paths need to be retested and which do not [19]. The test cases, which execute the former paths, are selected.

Regression testing at the integration level

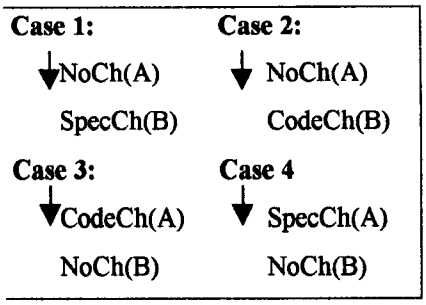
Integration testing is the testing applied when individual modules are combined to form larger and larger working units until the entire program is created. Integration testing detects failures that were not discovered during unit testing. Integration testing is important because approximately 40% of software errors can be traced to module integration problems discovered during integration testing [10].

Leung and White [1, 10, 12] introduced the firewall concept to regression testing at the integration level. A firewall is used to separate the set of modules affected by program changes from the rest of the code. The modules enclosed in the firewall could be those that interact with the modified modules or those that are direct ancestors or direct descendants of the modified modules. After analyzing the possible relationships between a modified module and a related unmodified module and their specifications, Leung and White identified four basis boundary cases and used them to build a firewall [1, 10, 12]. The four basis cases are shown in Figure 1 [1, 10, 12]. In all cases module A invokes B.

Because program changes can involve program specification changes and code changes, the basis boundary cases for the construction of the firewall must include both kinds of changes. Any unchanged module that calls a changed module or is called by a changed module is an affected module. The firewall concept is simple and easy to use, especially when the change to a program is small [1, 10, 12]. By retesting only the modules and interfaces inside the firewall, the cost of regression integration testing can

be reduced.

Rothermel and Harrold [9] developed DejaVu1 and DejaVu2 for use in regression testing at the unit and integration levels respectively. Both utilize control flow graphs (CFGs) to select tests from the original test set. DejaVu1 and DejaVu2 neither require prior knowledge of where code has been modified nor depend on a particular test adequacy criterion [9]. This decreases the overhead in regression testing. DejaVu2 runs *SelectInterTests* procedure to automatically build a CFG for both the original program P and its modified version P'. It also maintains a test history table to record which test cases were related to each traversed edge in P. Starting from entry procedures P_e and P'_e, the system compares the two CFGs node by node and edge by edge. If there is any difference, the corresponding test cases in the history table are selected. Finally, all selected test cases are returned for reuse. The authors proved that the algorithms employed by DejaVu2 are safe for controlled regression testing (i.e., P' is tested under the same conditions as P) [9].



NoCh(A) represents "No Change in Module A."

CodeCh(A) represents "Only Code Change in A.

SpecCh(A) represents "Change in the Specification of A"

Figure 1. Basis Boundary Cases for Firewall

Regression testing at the system level

System testing is testing of the entire software system against the system specifications. System testing must verify that all system elements have been properly integrated and perform allocated functions. System testing can be performed without the knowledge of the software implementation at all.

TestTube is a system developed at AT&T Bell Laboratories to perform system-level regression testing [8]. TestTube is an example of a selective retesting technique. It can be used with unit-level regression testing as well. In TestTube, test selection is based on analysis of the coverage relationship between test cases and the system under test [8]. TestTube partitions a software system into basic code entities, then monitors the execution of a test case, analyzes its relationship with the system under test, and determines which subset of the code entities the test covers. For a C program, each test case is associated with function definitions, global variables, type definitions and preprocessor macro definitions that it covers or possibly covers. If a change is made to one of the entities associated with a test case, then the test case is selected. TestTube is an example of how automatic regression testing can be applied. The C program is instrumented by the annotation preprocessor for C (app) [8]. When a test case is run a function trace is produced that shows which functions are covered by the test. The C information abstraction (cia) is used to create a database of C entities that shows dependencies among the entities [8]. If a program and its revision each have a database, then a program is run to determine differences in the databases and thus what has been changed. The authors plan to extend TestTube to nondeterministic systems such as real-time telecommunications software [8].

Regression testing of global variables

White and Leung [12] found that global variables could be treated as input/output parameters. A global variable is an output parameter in a module where the variable is defined and an input parameter for a module that uses the variable. They extended their firewall method of regression testing to retesting global variables [12]. The global variables can be retested at the unit-level by running the tests that exercise the changed code and the instruction referencing the global variable [12]. A global variable can be retested at the integration level. If any of its defining modules have been changed than all of its using modules must be retested [12]. The authors show that regression testing of global variables is very time-consuming and, therefore, costly [12]. The admonition against using global variables is well founded.

Comparison of regression testing techniques

When there are several regression testing techniques, the problem is choosing an appropriate technique for practical application. Several articles have been written that contain comparisons of different regression testing techniques [6, 13, 14, 15]. The following sections contain discussions of these articles.

Comparison of selective regression testing techniques

Rothermel and Harrold [14] provided a framework for evaluation of different selective regression testing techniques. They identified four criteria to form the framework. Among the criteria, *inclusiveness* measures the extent to which a technique chooses tests that will expose faults caused by program changes, *precision* measures the ability of a technique to avoid choosing tests that will not reveal the faults caused by the changes, *efficiency* measures the computational cost of a technique, and *generality* measures the ability of a technique to handle different language constructs and testing applications [14]. The code-based selective regression testing techniques can be grouped into three classes (i.e., coverage, minimization, and safe) depending on the testing goal [14]. The framework can be used to determine which technique should be used for a particular application. For example, if we require very reliable code, we may stick with a safe selection strategy (with high inclusiveness of test cases). If we need to reduce regression testing time, we may choose a minimization technique (with more precision in the selection of tests) [14].

DejaVu and TestTube were evaluated in terms of their precision in selecting test cases for the same experimental subjects [15]. Rosenblum and Rothermel found that DejaVu was better in some cases while TestTube was better in others [15]. They agreed that more collaboration of this type was needed. According to the authors this was the first such study of two selective regression techniques applied to the same program [15].

Comparison of selective and retest-all strategies in terms of cost-effectiveness

Several selective regression testing techniques have been pro-

posed. Selective regression testing techniques have been extensively studied because researchers believe that selective techniques can reduce the cost of regression testing. *Selective* regression techniques allow the testers to reuse a subset of the original test cases. However, if we consider the time and resources required for test selection, we find that the *selective* techniques do not always reduce the total cost of regression testing. If the cost in analysis for test case selection is too high, or the number of selected test cases is not significantly smaller than the number of original test cases, the selection is not cost effective.

Leung and White [6] analyzed the various factors that can affect the cost of regression testing, and proposed a simple model to compare the cost between the *selective* regression testing strategy and the *retest-all* strategy. They claimed that a benefit is accrued only if the effort spent in test selection is less than the costs for executing the extra test cases and for checking the results of the extra test cases used by the *retest-all* strategy [6]. This model established a basis for cost comparison between the two regression testing strategies. However, because this model makes some simplifying assumptions that may be inappropriate for some systems, it has limitations.

Rosenblum and Weyuker [13] extended Leung and White's model [6] by developing cost-effective predictors to determine the cost-effectiveness of a *selective* regression testing strategy. Their design of the predictors is based on some fundamental assumptions about the nature of test coverage and the nature of changes made to a system. Recall that the selection in selective regression testing is driven by two kinds of analysis: coverage analysis and change analysis. Rosenblum and Weyuker's experiences lead them to find that, in practice the relationship between the test suite and the entities the test suite covers (i.e., the coverage of tests) in a software system changes very little during maintenance, except when new features are added to the system [13]. This means that the coverage of a test case in different versions of a system changes very little. They also observed that the ability of a method to find a subset of test cases from a test suite is governed by the nature of the coverage relationship [13]. For example, if there is a great deal of overlay in the sets of entities of the system that each test case covers, they do not expect a safe strategy to be able to exclude very many test cases [13].

Based on those considerations, Rosenblum and Weyuker presented their algorithm for predicting the cost-effectiveness of a selective regression testing technique for a given software system [13]. The algorithm is the following: given a system under test, a candidate selective regression testing method, and a stable coverage relation, it suffices to evaluate the cost-effectiveness of the testing method on one version of the system in order to predict its cost-effectiveness for all future versions of the system [13]. For a safe strategy, their predictor first predicts the cost-effectiveness of the strategy when a change is made to a single entity. If it shows the strategy is cost-effective, then further computation will be performed when changes are made to multiple entities. During the two steps, if there are any data indicating that the testing strategy is not cost-effective, the candidate strategy will be discarded [13]. Rosenblum and Weyuker's work is significant since it provides a method for determining whether or not a selective regression testing technique is cost effective.

Regression testing in object-oriented software

Object-oriented concepts such as inheritance and polymorphism present unique problems in maintenance and thus regression testing of object-oriented programs. Several regression testing techniques have been extended to retesting of object-oriented software.

Rothermel and Harrold [16] extended the concept of selective regression testing to object-oriented environments. Their algorithm constructs program or class *dependence graphs* and uses these graphs to determine which test cases to select. A program dependence graph represents the data and control dependencies [16]. A program is assumed to have 1 entry and 1 exit. This is not true for classes since a class can have multiple entry points (i.e., one for each of its public methods). A class dependence graph must be constructed [16]. To test a graph, driver programs are developed to invoke the methods in a class in different sequences. The class dependence graph links all these driver programs together by selecting one driver as the root of the graph and adding edges from it to the public methods in the class [16]. Now the methods can be invoked in different sequences. Rothermel and Harrold emphasized that when a class is changed, the class itself, its subclass and the application program that uses this class has to be retested with respect to inheritance, polymorphism and dynamic binding, and data encapsulation [16].

Abdullah and White [17] extended the firewall concept to retesting object-oriented software. A firewall is an imaginary boundary that encloses the entities that must be retested in a modified program. Unlike the firewall in a procedure-based program, the firewall in an object-oriented program is constructed in terms of classes and objects. When a change is made to a class or an object, the other classes or objects that interact with this changed class or object must be enclosed in the firewall. Because the relations between classes and objects are different, (the relation between objects is messages) there should be different ways to construct the firewalls for classes and objects. Object-oriented programs must be retested by testing classes (unit level), inheritance relationships, and objects (integration level) [17].

Summary

Regression testing is an important maintenance activity and has received extensive attention recently. The goal of regression testing is to verify changes and to show that the changes do not adversely affect other parts of the software.

Regression testing techniques can be classified in different ways. Is the technique strictly code-based or does it consider the specifications? Is automation used to make the process easier? If a test suite is available which strategy will be used—run all tests (retest-all strategy) or select certain tests (selective strategy)? What program entities does the test case cover? If the selective strategy is used, what criteria are used to select the test cases? Testing occurs at different levels; therefore, regression testing needs to occur at the same levels. How is this accomplished? How should changes to global variables be tested?

Lastly, the different regression techniques should be evaluated to try to decide which technique is more cost-effective and in what circumstances a test should be applied. *Selective* regression testing techniques identify affected program components after the software is changed and select a subset of existing test cases that relate to the changed components to retest the software. Applying a *selective* strategy usually can reduce the cost of retesting compared to the *retest-all* strategy because a smaller test suite is executed. However, a *selective* regression testing technique might not be cost-effective if the effort made in test selection exceeds the cost of executing the extra test cases used by the *retest-all* strategy. There are tradeoffs between regression testing techniques.

Regression testing is often thought of as a maintenance activity, but ideally it should occur whenever there is change to code or specifications. Regression testing should not be restricted to the maintenance phase of the software development life cycle.

Bibliography

[1] White, L. and H. K. N. Leung. Regression Testability. *IEEE Micro*, April 1992, pp. 81-84.

[2] McCarthy, A. Unit and Regression Testing. *Dr. Dobb's Journal*, February 1997, pp. 18-20, 82, & 84.

[3] Onoma, A.K., Tsai, W., Poonawala, M.H., and H. Suganuma. Regression Testing in an Industrial Environment. *Communications of the ACM*, Vol. 41, No. 5., May 1998 , pp. 81-86.

[4] White, L.J. Software Testing and Verification. *Encyclopedia of Computer Science and Technology*. Allen Kent, James G. Williams, Carolyn M. Hall, and Rosalind Kent, editors, Marcel Dekker, Inc., 1995, pp. 267-323.

[5] Yau, S.S. and Z. Kishimoto. A Method for Revalidating Modified Programs in the Maintenance Phase. *Proceedings COMPSAC 87*, pp. 272-277.

[6] Leung, H.K.N. and L. White. A Cost Model to Compare Regression Test Strategies. *Proceedings of the Conference on Software Maintenance-91*, 1991, pp. 201-208.

[7] Jiang, J., Zhou, X. and D. J. Robson. Program Slicing for C — The Problems in Implementation. *Proceedings of the Conference on Software Maintenance*, 1991, pp. 182-190.

[8] Chen,Y. F., Rosenblum,D. S.0 and K. P Vo. TestTube: A System for Selective Regression Testing. *Proc. 16th International Conference Software Engineering*, Sorrento, Italy, May 1994, pp. 211-220.

[9] Rothermel, G. and M. J. Harrold. A Safe, Efficient Regression Test Selection Technique. *ACM Transactions on Software Engineering and Methodology*, Vol. 6, No. 2, April 1997, pp. 173-210.

[10] Leung, H.K.N. and L. White. A Study of Integration Testing and Software Regression at the Integration Level. *Proc. Conf. Software Maintenance*, San Diego, Nov. 1990, pp. 290-301.

[11] Gupta, R., Harrold, M.J., and M. L. Soffa. Program Slicing-Based Regression Testing Techniques. *Journal of Software Testing, Verification and Reliability*, vol. 6, no.2, June 1996, pp. 83-112.

[12] White, L.J. and H. K. N. Leung. A Firewall Concept for Both Control-Flow and Data-Flow in Regression Integration Testing. *Proc. Conf. Software Maintenance-92*, 1992, pp. 262-271.

[13] Rosenblum, D.S. and E. J. Weyuker. Using Coverage Information to Predict the Cost-Effectiveness of Regression Testing Strategies. *IEEE Trans. Software Eng.*, vol. 23, no. 3, March 1997, pp. 146-156.

[14] Rothermel, G. and M. J. Harrold. Analyzing Regression Test Selection Techniques. *IEEE Trans. Software Eng.*, v 22, n 8, August 1996, pp. 529-551.

[15] Rosenblum, D. and G. Rothermel. A Comparative Study of Regression Test Selection Techniques. *Proc. IEEE Computer Society 2nd Int'l Workshop on Empirical Studies of Software Maintenance*, Bari, Italy, Oct. 1997, pp. 89-94.

[16] Rothermel, G. and M. J. Harrold. Selecting Regression Tests for Object-Oriented Software. *Int'l Conf. on Software Maintenance*, 1994, pp. 14-25.

[17] Abdullah, K. and L. White. A Firewall Approach for the Regression Testing of Object-Oriented Software. *Software Quality Week Conference*, San Francisco, CA., November 1997.

[18] Korel, B. and A. M. Al-Yami. Automated Regression Test Generation. *ISSTA 98*, 1998, Clearwater Beach, FL. pp. 143-152.

[19] Ball, T. On the Limit of Control Flow Analysis for Regression Test Selection. *ISSTA 98*, 1998, Clearwater Beach, FL. pp. 134-142.