A Cost Model to Compare Regression Test Strategies

Hareton K. N. Leung
Bell-Northern Research
P.O. Box 3511, Station C
Ottawa, Ontario, Canada
K1Y 4H7

Lee White
Dept. of Computer Eng. and Sci.
Case Western Reserve University
Cleveland, Ohio, U.S.A
44106

Abstract

In recent years, regression testing has been receiving increasing attention. Several *selective regression testing* strategies have been introduced. One question that demands an answer is the cost benefits of the selective regression testing strategies compared to the traditional *retest-all* strategy. This paper first presents a test cost model and then identifies the conditions under which the selective strategy is more economical than the retest-all strategy.

Key words: Regression testing, Test cost, Testing

1. Introduction

Most software systems tend to evolve as they are adapted to changing environment, changing needs, new concepts and new technologies. Software will grow in the number of functions, components and interfaces. Existing modules may be expanded for uses beyond their original design. Thus, modification to the software is inevitable. Estimates have been given that software maintenance consumes between 50 to 80% of the total software cost. A major component of maintenance is the retesting of the software. Efficient and effective regression testing can reduce the cost of maintenance.

Regression testing is a testing process which is applied after a program is modified. It involves testing the modified program with some test cases in order to re-establish our confidence that the program will perform according to the (possibly modified) specification.

A number of testing tools exist to assist in software development; however, few of them can be applied directly to regression testing. Among those which claim to be useful for regression testing, most provide no more than the capability to store the previous tests and rerun them after every modification [11, 2, 14, 15]. They do not provide any intelligent test selection capability, nor any estimation of the required testing effort. We will call this type of regression testing the *retest-all strategy*.

Recently, several strategies were introduced for regression testing at the unit level [4, 16, 1, 8]. These strategies all attempt to select a subset of the previous test for execution and do not repeat all the previous tests. Specific information is stored with the tests in order to identify the relevant tests with possible program changes. We will call this type of regression testing a *selective strategy*. Typically, a selective strategy will use fewer tests than the retest-all strategy.

An important issue that has been overlooked in current regression testing research is the cost of applying a selective strategy compared to that of the retest-all strategy. Assuming both strategies give the same test effectiveness, it is not clear whether a selective strategy will always produce some cost saving. In this paper, we attempt to identify the conditions under which a selective strategy is more economical than the retest-all strategy. We review the recently introduced regression testing strategies in the next section. Section 3 first presents a test cost model and then compares the cost of applying selective and retest-all strategies using that model. Section 4 discusses several practical issues and concludes with our plan for future research.

2. Overview of Selective Regression Test Strategies

Very few regression test tools have been described in the research literature. Most of the 'known' regression test tools are either under development or in a prototype form. None of them are of production quality. Also, most of them were introduced to aid in regression unit testing, and cannot be used for regression integration or system testing.

Yau and Kishimoto [16] recently introduced a regression testing tool which was based on the *input*

partition testing strategy [12]. This strategy divides the input domain of a module into different classes using both the program specification and code and requires one test be selected from each input class. The objective of regression testing is to execute each new or changed input partition at least once. Symbolic execution is used to identify those input domains which are not modified and to aid in test generation. This regression testing strategy selects a subset of the previous tests and some new tests to exercise the modified code. Similar to most regression tools, this tool concentrates on unit regression testing.

A regression testing tool is an essential component of any software maintenance environment. A *post-maintenance testing system* was described by Benedusi et al. [1], and also deals with unit regression testing. It implements a path testing strategy by selecting tests to exercise a specific set of paths. The regression testing strategy uses the following two steps: (1) identify those paths that have been added, those deleted and those modified; (2) update and re-run the test cases which exercise the modified and new paths. Algebraic expressions are used to represent the program before and after the program modification. By comparing these expressions using elementary algebraic operations, it is possible to classify paths affected by the modification into new, deleted, modified and unmodified paths. By storing test cases and their associated program paths in a table, it is straightforward to identify those tests needed to be re-run based on the changed code.

Harrold and Soffa are developing an incremental data flow tester which can be used for regression testing purposes [4, 5]. This tool combines data flow testing with incremental data flow analysis to aid in unit and integration regression testing. During the initial testing, the system stores the previous data flow analysis results and test cases. After a module is modified, the system analyzes the effects of the changes on the *test history* and determines new and existing definition-use pairs for retesting. Existing test cases are reused whenever possible. The system identifies those definition-use pairs that have not been exercised. This tool does not help with test generation and uses only a white-box testing method (structural testing based on data-flow coverage.) Harrold and Soffa [6] have recently extended their technique to include interprocedural testing. Data flow testing is performed across procedure boundaries to integrate those procedures.

Another regression tool has been introduced by Hartmann and Robson [7]. This tool extends Fischer's method [3] for test selection from the previous test set. It assumes that a test set which covers all program segments is available. The tool then takes this test set as input and produces a minimum test set which covers the same program segments.

Fischer [1] uses zero-one integer programming to find a minimum set of tests which can cover all the program segments. Four tables are used to record the control flow relation between program segments, reachability between segments, test cases which cover the different segments, and variable use and define information within each segment, respectively. A series of inequality constraint expressions is created, one for each program segment, relating those tests which traverse the segment. The objective function relates the cost of running all the tests to individual tests. Given the program segments that have been modified, standard linear-programming techniques are used to solve for the objective function which can be used to derive the minimum number of test cases that must be rerun to validate the modification. Recall that linear program problems usually terminate with a non-integer solution. To arrive at a zero-one solution, additional constraints and iteration may be needed. However, cutting plane methods sometimes work well and sometimes not at all; branch and bound techniques can spend too much time solving linear programs.

The prototype tool under development by Hartmann and Robson [7] extends Fischer's method to include programs written in the C programming language, programs with several modules, and segments which have multiple use of variables. Since this tool also uses zero-one linear programming, it suffers from the same set of computational problems associated with other zero-one linear programming problems. This tool has the potential to aid in system regression testing by treating each module as a segment for test coverage purposes. Observe that this tool implicitly assumes a white-box testing technique (segment cover) is used and cannot be generalized to include black-box testing.

We can summarize the characteristics of the above regression tools as follows:

1 Most of them apply one testing method (either white-box or black-box testing).
2 Most restrict themselves to unit testing and do not address integration or system testing.
3 None of them is in production use, indicating the difficulty in producing such a tool. This can be partly explained by the fact that any regression tool should be coupled with a testing tool because both regression testing and original testing should use the same techniques.

Recently, the authors [8, 9] have conducted a fundamental study of regression testing. They have identified two types of regression testing: *progressive regression testing* which involves a modified specification, and *corrective regression testing* where the specification does not change. In general, corrective regression testing should be an easier process than progressive regression testing because more test cases

can be reused. Regression strategies which incorporated both functional and structural testing were developed. The authors have also analyzed the issues involved in regression testing at the integration level. Common errors and faults that may occur in using another module were identified and regression testing strategies for various basic types of modifications were also described. The authors [10] have since extended their work to programs with global variables.

3. Comparing the Relative Cost of Regression Testing Strategies

A practical question that has not been answered in previous regression testing research is the cost of applying a particular regression testing strategy. It seems that a selective strategy will require more time and resources for test selection in order to realize a reduction in the number of test cases executed. A benefit is accrued only if the effort spent in test selection is less than the cost for executing the extra test cases used by the retest-all strategy. In this section, we provide an analysis of the trade-off between using the selective strategy and the retest-all strategy.

The cost of applying any testing strategy will be modelled in Section 3.1. Section 3.2 compares the relative cost of a selective strategy and the retest-all strategy. The conditions under which the selective strategy is more economical than the retest-all strategy will be derived.

3.1. A Test Cost Model

Many factors affect the cost of testing a software system. We distinguish two sets of costs: *direct* and *indirect* costs. A *direct* cost includes the test analyst's time for performing all activities related to testing and the resources such as the computer and test lab required for executing the tests. An indirect cost includes:

- the management overhead for managing the overall testing process,
- the database cost for storing the test-related information such as test cases, the result of the static analysis, and execution histories, and
- test tool development.

In the planning phase of a project, indirect costs are typically not factored into the total cost. In our initial modeling effort, we will concentrate on the direct costs. We do not expect the indirect costs will affect our results in a major way. Section 3.2 will discuss the implications of incorporating these costs to our model.

The direct activities involved in testing a software system include not only test selection, test execution and result analysis, but also analysis time to determine the software system behavior to be tested. The cost of applying a set of tests to a software system consists of the following components:

(1) *System Analysis Cost, Ca*

Before a test set can be selected, the test analyst must become familiar with the system specification, design and possibly the program. Time must be spent studying the various requirements and design documents. The knowledge gained in this phase will also allow the test analyst to judge whether the program behavior is correct. In general, the larger the system under test, the higher is this cost.

(2) *Test Selection Cost, Cs*

After gaining some knowledge about the system, the test analyst can then select the test cases for testing the actual behavior of the system. Some cost is incurred in working out the test input, and identifying the correct output or system behavior. This cost component will depend largely on the chosen test strategy.

(3) *Test Execution Cost, Ce*

This cost component includes the cost of setting up the environment for testing (such as loading and compiling required modules and entering the proper data tables) and the cost of computing resources for the actual execution of the system under test. The cost can be quite high for some applications. For example, in the telecommunication industry, the cost of setting up a testing lab to simulate an actual communication network can be as high as several million dollars.

(4) *Result Analysis Cost, Cr*

The last step of testing involves checking the behavior of the system under test against the specified behavior. The cost factors for Cr are: the tester's time in collecting the test outputs, the tester's time to compare the test output to the system specification, and the computing resource required for recording the system behavior under test.

These cost components are dependent on several factors and some of them are interrelated. The testing strategy used has a direct effect on the cost. For example, since a black-box strategy only requires an analysis of the specification, the system analysis cost for such a strategy will be less than that of a testing strategy which uses both white-box and black-box testing. Note that the

number of test cases also depends on the testing strategy. A test strategy which requires that every definition-use pair be executed generally needs more tests than one which only requires all instructions be executed.

All test cost components are related to the number of test cases. Although it seems that Ca is independent of the number of tests because the cost of understanding both the problem specification and the system is roughly the same for designing either 10 test cases or 100 test cases, one can argue that a complex system will require more effort for understanding and also many more tests than a simple system. Thus, the number of tests and the system analysis cost are related to the same factor - the complexity of the system.

Also, the other three cost components, Cs, Ce, and Cr, should be dependent on the number of test cases; the higher the number of test cases, the higher are the respective costs. In this paper, we will assume this relationship to be linear; in general, some experiments should be conducted to establish this relationship.

The system analysis cost Ca and the result analysis cost Cr are related in a subtle way. Suppose the test analyst has spent a small amount of time in the system analysis phase. When the result is obtained, more time is required to determine whether the output is correct, because the analyst first has to understand the expected program behavior by studying the specification and other program documents. Thus in reducing Ca, it is likely that Cr will increase. By thoroughly performing the system analysis phase, the test analyst will better understand the expected behavior of the system, and it will be straightforward to check whether the program is correct during the result analysis phase.

From the previous discussion, we will model the result analysis cost Cr as consisting of two sub-components: the cost Cu for understanding the program and specification in order to judge whether the program behavior or output is correct, and the cost Cc for comparing each test output to the expected output. Thus, Cr=Cu+Cc. The *checking cost Cc* should be proportional to the number of test cases. As argued earlier, Cu is directly related to Ca because a thorough system analysis upfront will reduce the effort for understanding the system during the result analysis phase.

In summary, our model is based on the following key assumptions:

(1) the cost of applying a test strategy *S* which uses a set T of tests is

$$C(S) = Ca(T)+Cs(T)+Ce(T)+Cu(T)+Cc(T)$$

(2) Cs, Ce and Cc are linearly dependent on the number of test cases.

3.2. Cost of Regression Strategies

In this section, we first analyze the cost of applying the retest-all strategy and that of a selective strategy, and then compare the relative cost of these two strategies. The following assumptions will be made in our comparison:

- both selective and retest-all strategies are equally effective; this implies that both strategies use the same testing technique, and the same strategy to select new tests,
- the retest-all strategy does not do any analysis before applying all the previous tests, and
- the selective strategy spends some effort in selecting a subset of the previous tests for execution.

During regression testing, some existing program components (e.g., statements, modules) may be modified and deleted, and new program components may be added to the program. The retest-all strategy uses two test sets: the old tests (To) which contains all the previous tests, and a set of new tests (Tn) for testing the modified specification and/or modified code. We will call To the *total previous set*. Thus the testing cost is

$$C(\text{retest-all}) = [Ca(To)+Cs(To)+Ce(To)+Cu(To)+Cc(To)] + [Ca(Tn)+Cs(Tn)+Ce(Tn)+Cu(Tn)+Cc(Tn)]$$

Since according to the retest-all strategy, the test analyst does not analyze the previous tests before applying them and no effort is needed to select these tests, Ca(To)=Cs(To)=0. However, as argued in the previous section, one consequence of skipping the system analysis and test selection phases is that the test analyst will pay a heavy price later in the result analysis phase; that is, Cu(To) will be high. Thus the cost of applying retest-all can be reduced to

$$C(\text{retest-all}) = [Ce(To)+Cu(To)+Cc(To)] + [Ca(Tn)+Cs(Tn)+Ce(Tn)+Cu(Tn)+Cc(Tn)]$$

Next, consider the cost of applying a selective regression strategy where a subset Ts of To, together with the new tests Tn', are used to test the modified software system. Ts is selected based on the selective regression strategy. If a data flow based regression testing strategy [4] is used, Ts may contain those previous tests which traverse the changed and new define-use pairs. We will call Ts the *selected previous test*. Since we assume both the retest-all and the selective strategies use the same strategy to identify new tests, Tn'=Tn. Thus, the cost of applying the selective strategy is:

C(selective) = [Ca(Ts)+Cs(Ts)+Ce(Ts)+Cu(Ts)+Cc(Ts)]
+[Ca(Tn)+Cs(Tn)+Ce(Tn)+Cu(Tn)+Cc(Tn)]

Observe that the sum of Ca(Ts) and Cu(Ts) should be approximately the same as Cu(To) because in both cases, the test analyst has to understand the effect of the modification on the validity of the previous test cases. Also notice that the second set of terms in C(retest-all) and C(selective) are identical.

Based on these observations, the cost of applying the selective strategy is less than that of the retest-all strategy if

$$C(\text{selective}) - C(\text{retest-all}) < 0$$

or

$$[Cs(Ts)+Ce(Ts)+Cc(Ts)]-[Ce(To)+Cc(To)] < 0 \quad (1)$$

because $Ca(Ts)+Cu(Ts) \approx Cu(To)$. (1) can be rewritten as

$$Cs(Ts) < [Ce(To)-Ce(Ts)]+[Cc(To)-Cc(Ts)] \quad (2)$$

In other words, the selective strategy is more economical than the retest-all strategy if the cost for selecting a subset of the previous tests is less than the cost for executing and checking the extra previous tests needed for the retest-all strategy.

To gain more insight into inequality (2), we will invoke the second assumption: Cs(T), Ce(T) and Cc(T) are directly proportional to the number of tests in test set T. Thus, we write

$$Cs(T) = s\,|T|$$
$$Ce(T) = e\,|T|$$
$$Cc(T) = c\,|T|$$

where s, e and c are constants representing the cost per test case for the selection cost, execution cost and checking cost, respectively, and |T| represents the cardinality of the test set T. The exact values of these constants will depend on the complexity of the system, the particular implementation, and the testing strategy. Also, these constants will vary with the experience and ability of the test analyst.

Substituting the above expressions, we can rewrite inequality (2) as

$$s\,|Ts| < e\,(|To| - |Ts|) + c\,(|To| - |Ts|)$$
$$= (e+c)\,(|To| - |Ts|)$$

or

$$s < (e+c)\left(\frac{|To|}{|Ts|} - 1\right) \quad (3)$$

The inequality (3) depends on the values of s, e, c and the ratio |Ts|/|To|. Suppose the selective strategy only repeats half of the previous test (|Ts|=0.5|To|), then it is more cost effective than the retest-all strategy if

$$s < (e+c)$$

Figure 1 summarizes the conditions under which the selective strategy is more economical than the retest-all strategy. The vertical axis represents s values and the horizontal axis denotes the ratio |Ts|/|To|. The graph $(e+c)\left(\frac{|To|}{|Ts|} - 1\right)$ is plotted against |Ts|/|To|. If the selection cost of a selective strategy falls below the graph, using such a selective strategy will provide some saving in test cost over the retest-all strategy. For example, if s=(e+c) and the selected Ts=0.4 To, then a selective regression strategy will cost less than the retest-all strategy. Inequality (3) can be used to identify which regression strategy should be applied for a given project.

In the above analysis, we have implicitly assumed that the execution cost factor e is the same for both selective and retest-all strategies. In reality, these factors will likely have slightly different values because we should include the effects of static analysis, data structures and test execution histories in terms of computer execution as required by selective strategies over retest-all strategies. We next show how this difference will affect inequality (3).

Let e_r and e_s denote the execution cost factors for retest-all and selective strategies respectively. Clearly $e_s > e_r$. Inequality (2) can now be rewritten as

$$s\,|Ts| < (e_r|To| - e_s|Ts|) + c\,(|To| - |Ts|) \quad (2')$$

Now, let $e_s = e_r + \delta$, (2') can be reduced to

$$s\,|Ts| < e_r(|To| - |Ts|) - \delta\,|Ts| + c\,(|To| - |Ts|)$$

or

$$s < (e_r + c)\left(\frac{|To|}{|Ts|} - 1\right) - \delta \quad (3')$$

Notice that the only difference between (3') and (3) is the corrective term δ. If the value of δ is very small compared to e_r, then there will not be any major changes in our results. However, if δ is large, then the curve in Figure 1 will have to be moved downward by δ. This would make the selective strategy less cost effective than the retest-all strategy. Only experimentation will show the true magnitude of δ.

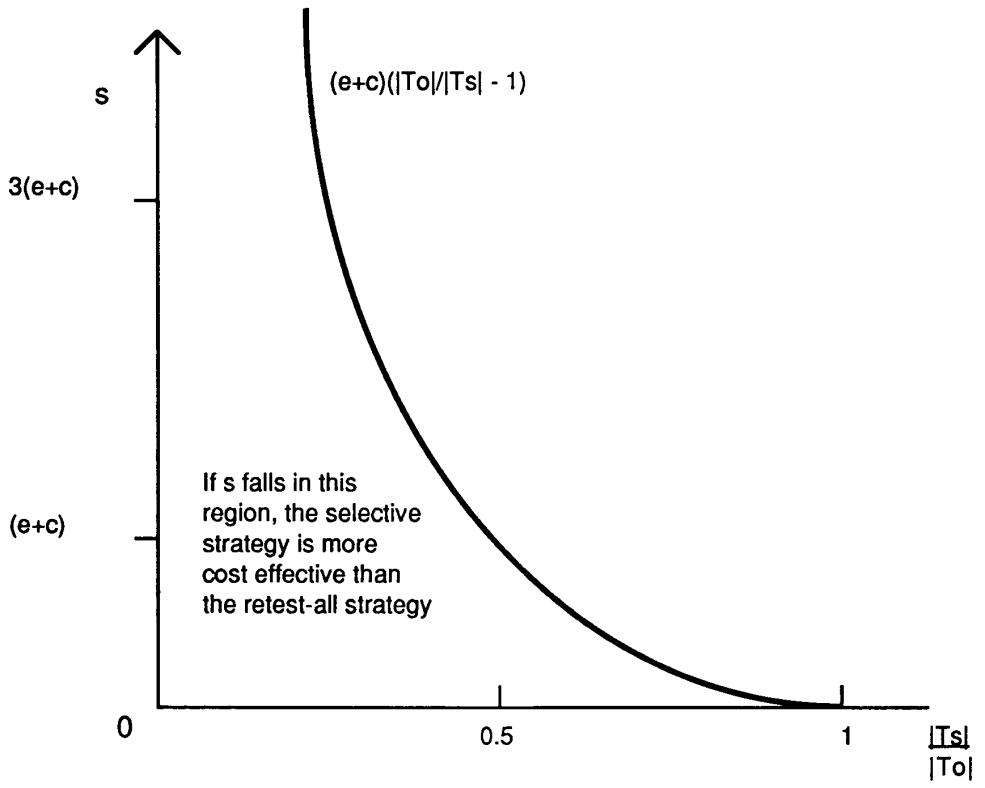

Figure 1. Cost Relationship Between the Retest-all and Selective Strategies

The work presented here represents the first step of our cost modeling effort. We plan to further refine the cost model by incorporating other indirect factors such as the management overhead, the database cost for storing test-related information, and the test tool development cost. We do not expect that incorporating these indirect costs will greatly affect our results. The key determinant of management overhead is the length of the testing interval. The longer the test interval, the higher is the management overhead. Depending on the complexity and the application type, more tests to be executed usually means a longer test interval. This seems to suggest that the selective strategy will be more attractive than the retest-all strategy if this indirect cost is added to our cost model.

Adding the second indirect cost (database cost) to our model seems to increase the attractiveness of retest-all strategy since the selective retest strategy needs more storage and a more complex database. The selective retest strategy typically stores information such as static analysis, test targets, and test history with each test case. This information may double the size of the test case database.

The test tool development cost will not have a major impact on our results. Since we have assumed that both retest-all and selective retest use the same test technique and strategy, then the same test tool development will be required for both. Thus, this indirect cost represents a constant factor for both regression testing strategies.

Despite the above observations, a detailed analysis of these indirect cost factors may shed new insights into the cost model.

4. Concluding Remarks and Future Research

Despite the increasing research in regression testing, an important question that has been overlooked is the cost benefit of applying selective regression testing

strategies over the retest-all strategy. In this paper, we have proposed a test cost model which can be used to compare the retest-all strategy to a selective strategy. We have found that the relative cost depends on four factors: the selection, the execution, the checking cost, and the ratio of the size of the selected previous test to the size of the total previous test.

To establish the relative cost advantage of different test strategies, actual cost data should be available. Starting a data collection program is the first step for identifying the test cost. There are two simple ways to collect the cost data Cs, Ce and Cc: a weekly tracking of the cost data and a post-mortem survey. The first method tends to generate more accurate data since the test analysts will be supplying the information on a weekly basis after each test activity. However, this method will likely encounter resistance because it involves a great overhead in that the test analysts have to expend some extra effort every week.

The post-mortem survey at the end of testing is less accurate because the test analysts will have to estimate the effort for each activity. This method will receive better acceptance from the test community because it is less disruptive and likely to generate usable data.

Once sufficient data has been collected and validated on a significant number of projects, then this data can be used to predict the cost of future projects. It is likely the values of s, e, and c will depend on the project type, selective strategy and environment. In using historical data, it is essential that data from similar applications under similar conditions and development environments be used.

For a given change situation, inequality (3) can be used to select which regression testing method to be used. The values for e, c and s can be selected according to the project type from historical data. Since |To| is known from previous testing, the only parameter to be determined is |Ts|. |Ts| can be estimated by an analysis of the extent of changes to the system specification, requirement and source code. If the test case database stores the relation between the test cases and their test targets (e.g., specification or code components to be tested by each test case), then it is possible to estimate |Ts| through simple database look-up operations. Otherwise, a crude estimate can be derived by first determine the proportion of changes to the total specification and code, and then use the same proportion for |Ts|/|To|.

In this paper, we have assumed that there is a linear relationship between the costs Cs, Ce, Cc and the number of test cases. This relationship can only be established experimentally, in order to reaffirm the result we have obtained in this paper between the retest-all and selective regression test strategies.

Two areas which warrant study are the relationship between the reliability of the software and test strategies, and the relationship between the cost of testing and the achieved quality. Although high reliability is desirable, it is equally important to attain the reliability at a reasonable cost. Unfortunately, the state-of-the-art in testing does not relate the increase in software quality to applying a particular testing strategy or the cost of such a strategy.

We plan to apply our model to evaluate the cost benefit of the selective regression testing strategies to produce a cost hierarchy of different strategies.

References

[1] P. Benedusi, A. Cimitile, and U. De Carlini, "Post-maintenance testing based on path change analysis," *Proc. Conf. Software Maintenance*, pp. 352-361, Phoenix, 1988.

[2] T. Dogsa and I. Rozman, "CAMOTE-computer aided module testing and design environment," *Proc. Conf. Software Maintenance*, pp. 404-408, Phoenix, 1988.

[3] K. F. Fischer, F. Raji, and A. Chruscicki, "A methodology for retesting modified software," *Proc. National Telecommunications Conf.*, pp. B-6-3(1-6), New Orleans, LA, Nov. 1981.

[4] M. J. Harrold and M. L. Soffa, "An incremental approach to unit testing during maintenance," *Proc. Conf. Software Maintenance*, pp. 362-367, Phoenix, 1988.

[5] M. J. Harrold and M. L. Soffa, "An incremental data flow testing tool," *6th Int. Conf. Testing Computer Software*, Washington, D.C., 1989.

[6] M. J. Harrold and M. L. Soffa, "Interprocedural data flow testing," *Proc. third Symposium on Software Testing, Analysis and Verification*, pp. 158-167, Key West, 1989.

[7] J. Hartmann and D. J. Robson, "Techniques for selective revalidation," *IEEE Software*, pp. 31-38, January, 1990.

[8] H. K. N. Leung and L. White, "Insights into regression testing," *Proc. Conf. Software Maintenance*, pp.60-69, Miami, FL, Oct. 1989.

[9] H. K. N. Leung and L. White, "A study of integration testing and software regression at the integration level," *Proc. Conf. Software Maintenance*, pp.290-301, San Diego, Nov. 1990.

[10] H. K. N. Leung and L. White, "Insights into testing and regression testing global variables," *Journal of Software Maintenance*, vol. 2, pp.209-222, Dec. 1990.

[11] B. Raither and L. Osterweil, "TRICS: a testing tool for C," *Proc. First European Software Engineering Conf.*, pp. 254-262, Strasbourg, France, Sept. 1987.

[12] D. J. Richardson and L. A. Clarke, "Partition analysis: a method of combining testing and verification," *IEEE Trans. Software Eng.*, vol. SE-11 (12), pp.1477-1490, 1985.

[13] *Software Engineering Automated Tools Index*, Software Research Associates, San Francisco, 1982.

[14] H. G. Stuebing, "A modern facility for software production and maintenance," *Proc. COMPSAC 80*, pp. 407-418, Chicago, IL, 1980.

[15] H. G. Stuebing, "A software engineering environment (SEE) for weapon system software," *IEEE Trans. Software Eng.*, vol. SE-10 (4), pp. 384-397, July 1984.

[16] S. S. Yau and Z. Kishimoto, "A method for revalidating modified programs in the maintenance phase," *Proc. COMPSAC 87*, pp. 272-277, Tokyo, Japan, 1987.