from the Proceedings of the IEEE Computer Society 2nd International Workshop on Empirical Studies of Software Maintenance, Bari, Italy, October 3, 1997

A Comparative Study of Regression Test Selection Techniques

David Rosenblum
Department of Information & Computer Science
444 Computer Science
University of California, Irvine
Irvine, CA 92697-3425
dsr@ics.uci.edu
http://www.ics.uci.edu/~dsr

Gregg Rothermel
Department of Computer Science
Oregon State University
Dearborn Hall 307-A
Corvallis, OR 97331
grother@cs.orst.edu
http://www.cs.orst.edu/~grother

Abstract

Regression test selection techniques attempt to reduce the cost of regression testing by selecting a subset of an existing test suite for execution on a modified program. Over the past two decades, numerous regression test selection techniques have been described in the literature. Initial empirical studies of a few of these techniques have shown that they can be beneficial, but the studies were performed independently on dissimilar subjects. This paper describes the first comparative evaluation of two different regression test selection techniques on the same experimental subjects. In particular, two methods developed by the authors, `DejaVu` and `TestTube`, were evaluated and compared in terms of their *precision* in selecting test cases. The data reveal that in some instances `TestTube`'s relative lack of precision does not prevent it from competing with the more precise `DejaVu`, yet in other instances `DejaVu`'s superior precision gives it a clear advantage over `TestTube`. Such variation in performance seriously complicates the tester's choice of which technique to use, and it suggests avenues for further collaborative research and experimentation.

Keywords: software maintenance, regression testing, empirical studies.

1 Introduction

Regression testing is a necessary but expensive maintenance task, performed on modified programs to instill confidence that changes are correct and have not adversely affected unchanged portions of the program. *Regression test selection techniques* attempt to reduce the cost of regression testing by selecting a subset of an existing test suite for execution on a modified program. Over the past two decades, numerous regression test selection techniques have been described in the literature; Rothermel and Harrold recently published a comprehensive analysis of these techniques [5]. Initial empirical studies of a few of these techniques [1, 3, 4, 7, 6, 8] have shown that they can be beneficial, but that their costs and benefits vary widely over different programs, program versions, and test suites. These studies, however, were performed independently on dissimilar subjects; no studies have compared techniques empirically.

This paper describes the first comparative evaluation of two different regression test selection techniques on the same experimental subjects. The main goals of this study are to produce data revealing the relative merits of different regression test selection techniques, to further our understanding of the factors that influence the effectiveness of regression test selection techniques, and to guide future empirical work.

2 Background

In previous work, the authors have independently developed, implemented, experimented with, and reported on two distinct regression test selection techniques [1, 7]. Both techniques are *safe*, in the sense that they are guaranteed to select *all* test cases affected by a change (and possibly additional test cases as well). We summarize the techniques here; see the references for details.

DejaVu. Rothermel and Harrold developed a family of test selection algorithms, and implemented one algorithm as a tool called `DejaVu`.

DejaVu builds control flow graphs[1] for a program P and modified version P', collects test traces that associate tests in T with edges in the control flow graph for P, and performs synchronous depth-first traversals of the two graphs, comparing the program statements associated with nodes that are simultaneously reached in the two graphs. When the algorithm discovers a pair of nodes N and N' in the graphs for P and P', respectively, such that the statements associated with N and N' are not lexically identical, the algorithm selects all tests from T that, in P, reached N.

TestTube. Chen, Rosenblum and Vo developed the TestTube method of test selection specifically to address considerations of cost-effectiveness. TestTube is a general method of test selection based on coarse-grained analysis of the coverage relationship between test cases and the system under test, and it has been implemented for the C programming language. In the implementation of TestTube for C, each test case is associated with function definitions, global variable definitions, type definitions and preprocessor macro definitions that it covers or potentially covers. If a change is made to one of the entities associated with a test case, then the whole entity is considered to be changed, and the test case is selected (even though the test case may not cover the particular subpaths through the entity that are affected by the change). Thus, the TestTube method sacrifices some measure of precision in order to gain efficiency in test selection.

Analytical comparisons. In [5], Rothermel and Harrold discuss theoretical issues relevant to regression test selection, present a framework for analytically comparing regression test selection techniques, and use the framework to compare existing techniques. According to that analysis, both DejaVu and TestTube belong to a class of regression test selection algorithms described as *safe*: the defining characteristic of such algorithms being that, under certain well-defined conditions, they select every test from the original test suite that can expose faults in the modified program. Rothermel and Harrold show that among safe approaches, DejaVu is more *precise* than TestTube, in that it selects the fewest unnecessary tests. However, DejaVu obtains its extra precision at additional cost: TestTube is more efficient than DejaVu.

[1]A control flow graph is a directed graph in which nodes represent program statements, and edges represent the flow of control between statements.

3 Empirical Studies

Analytical results notwithstanding, we require empirical data to determine the relative costs and benefits, in practice, of regression test selection techniques. Ultimately, we would like to measure the effort required to select tests, and the reduction in testing effort achieved by eliminating unnecessary tests. To obtain such data, we must assemble a collection of experimental subjects, on which all algorithms being compared function. We must then exercise reasonable implementations of the algorithms on those subjects under controlled conditions, and gather data about the analysis and test execution time. If we do not possess implementations, we may be able to obtain some data by simulating techniques.

Objective. Our initial studies, reported in this paper, focus on *precision*. The objective of these studies is to measure and compare the number of tests selected by DejaVu and TestTube; this number is an indicator of the relative savings we can expect from the techniques.

Subjects. Table 1 describes the experimental subjects used in our studies. Each subject consists of a base program, a number of modified versions, a large test pool, and a number of test suites.

The base programs, modified programs, and test pools described on the first seven lines of Table 1 were assembled originally by Hutchins et al. [2], for use in experiments with dataflow- and controlflow-based test adequacy criteria. These subjects were subsequently modified and equipped with test suites by Rothermel and Harrold, for use in studies of regression testing[7]. For each of the subjects, 1000 branch-coverage-adequate test suites[2] were generated. Table 1 lists the average size of the branch-coverage-adequate test suites provided with each of the first seven subjects.

Hutchins et al. sought to study error detection; thus, they created *faulty* modified versions of base programs by manually seeding faults into the base programs. For the purpose of our studies, we view

[2]A test suite T is branch-coverage-adequate for program P if every dynamically feasible outcome of every predicate in P is exercised by at least one test in T.

Program Name	*Number of Functions*	*Lines of Code*	*Number of Versions*	*Test Pool Size*	*Test Suite Avg Size*	*Description of Program*
replace	21	516	32	5542	19	pattern replacement
printtok2	20	483	7	4115	12	lexical analyzer
printtok1	21	402	10	4130	16	lexical analyzer
schedule2	16	297	10	2710	8	priority scheduler
schedule1	18	299	9	2650	8	priority scheduler
totinfo	16	346	23	1054	7	information measure
tcas	8	138	41	1608	6	altitude separation
player	766	49316	5	1033	1033	transaction manager

Table 1: Experimental subjects.

the faulty modified versions of base programs as ill-fated attempts to create modified versions of the base programs.

The program described on the last line of Table 1, `player`, is a component of the software distribution for the Internet-based game, `Empire`. Several modified versions of `Empire` have been created, to fix bugs or add functionality. Most of these versions alter the `player` program. For the base version listed in Table 1, five "real" versions, created by different programmers for different reasons, were collected. Table 2 summarizes significant statistics about these five versions.

The test pool for `player` contains "black box" tests, constructed using the `Empire` information files, which describe the 154 commands recognized by `player`, and discuss parameters and special side effects of each command. The information files were treated as informal specifications; for each command, tests were created to exercise each parameter and special feature, and test erroneous parameter values and conditions. Because the complexity of commands and parameters varies widely over the `player` commands, this process yielded between 1 and 30 tests of each command, and ultimately produced a test suite of 1035 tests.

Version	*Functions Modified*	*Lines of Code Changed*
1	3	114
2	2	55
3	11	726
4	11	62
5	42	221

Table 2: Modified versions of `player`.

Method. We can obtain data on the precision of `DejaVu` by applying the `DejaVu` implementation to our experimental subjects; however, our `TestTube` implementation does not yet function on these subjects. Work is in progress to enable the direct application of `TestTube`. In the meantime we have discovered a way to simulate `TestTube`'s test selection algorithm, to measure the precision of its test selection. To simulate `TestTube`, we modified the `DejaVu` code, such that whenever `DejaVu` locates changed code during its graph walk, the modified version selects all tests that enter the procedure that contains the code. Although the modified `DejaVu` requires greater execution time than that expected for `TestTube`, it selects exactly the tests that `TestTube` would select. Thus, although the simulation does not provide data on the relative efficiency of the analysis tools, it does provide data on their relative precision.

Our process for performing studies is as follows. For each base program P, modified version P', and test suite T, we ran all tests in T on an instrumented version of P, and saved the test trace information. We ran `DejaVu` on P, P', T, and the test trace information, and counted the number of tests selected. We then ran our `TestTube` simulation on P, P', and T, and the test trace information, and counted the number of tests selected.

Study 1: TestTube vs DejaVu on Coverage-Based Test Suites

Study 1 investigated the precision of `TestTube` versus the precision of `DejaVu` on programs 1 through 7, using coverage-based test suites. Figure 1 depicts the results of the study. The figure contains a separate graph for each base program. In each

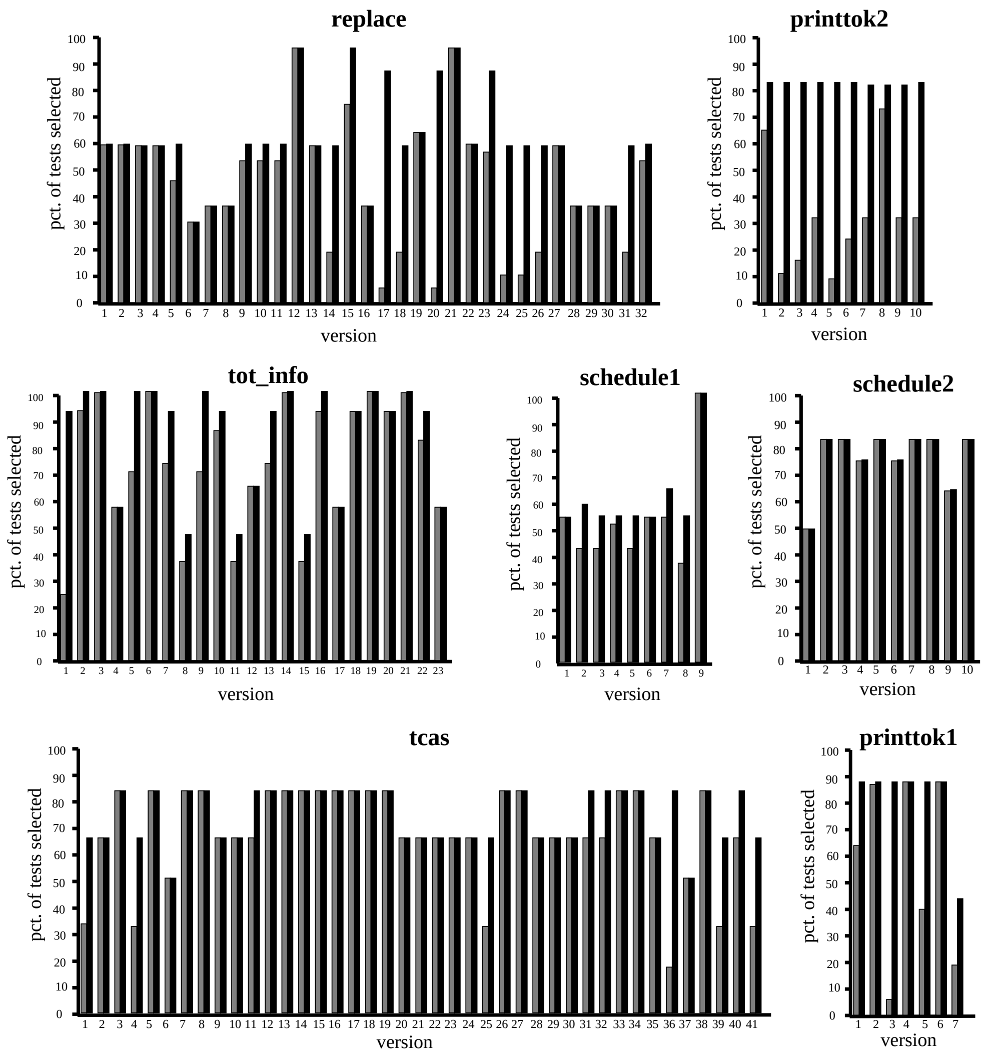

Figure 1: Test selection statistics for subjects 1-7, with coverage-based test suites. DejaVu (grey bars) versus TestTube (black bars).

graph, each version of the program occupies a position along the x-axis and is represented by a pair of vertical bars. The two bars depict the percentage of coverage-based tests selected by `DejaVu` (grey bars) and by `TestTube` (black bars) in that version, *on average over the 1000 test suites.*

As foretold by analytical results, `DejaVu` is never less precise than `TestTube`: it never selects more tests. In fact, although the figure does not show this, the set of tests selected by `DejaVu` is always a subset (not necessarily proper) of the set of tests selected by `TestTube`. However, the relative precisions of the two techniques vary widely over various modified versions. For example, on the seven versions of `printtok1`, the two techniques select nearly equivalent (within 1%) sets in three cases. `DejaVu` is between 30% and 50% more precise than `TestTube` in two cases, and more than 50% precise than `TestTube` in two cases. In contrast, on `schedule2`, `DejaVu` is never more than 1% more precise than `TestTube`.

We can explain this variation in part by noting that some of the versions involved small changes to conditional expressions inside the function `main`, which caused `TestTube` to select all tests that cover `main` (i.e., the full test suite). Because of `DejaVu`'s greater precision, it was able to select only those test cases directly affected by changes in `main`.

Study 2: TestTube vs DejaVu on a Larger Scale

Study 2 investigated the precision of TestTube versus the precision of DejaVu on the `player` program. Figure 2 depicts the results of the study: for each version, two bars depict the percentage of the 1033 `player` tests selected by `DejaVu` (grey bars) and by `TestTube` (black bars). Again, as foretold by analytical results, `DejaVu` always selects fewer tests than `TestTube`. On versions 4 and 5, in fact, `TestTube` selects all tests, whereas `DejaVu` reduces test set size by over 90%. On versions 1, 2, and 3, however, `DejaVu` offers only minor savings.

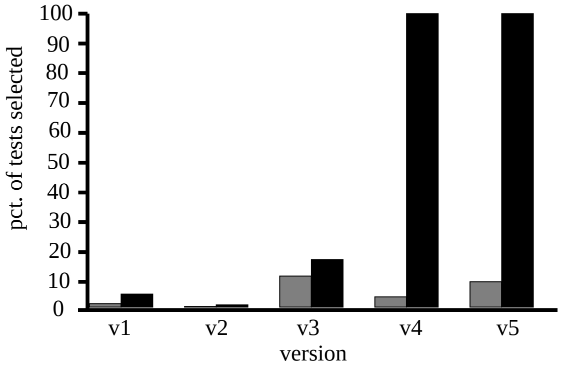

Figure 2: Test selection statistics for `player`; DejaVu (grey bars) versus TestTube (black bars).

4 Discussion

We have described results of a comparative evaluation of two regression test selection techniques: `DejaVu` and `TestTube`. In this initial study, we set up infrastructure to enable us to carry out a comparative evaluation, and we focused on comparing the techniques in terms of their precision in selecting test cases. A proper comparison must of course consider additional factors, including a comparison between the cost of applying the techniques and the testing costs saved by eliminating test cases. Based on our initial results, we observe that the choice of which regression test selection technique to use on a particular system under test is complicated by the following somewhat contradictory facts:

1. "Coarse-grained" techniques such as `TestTube`, despite their relative lack of precision, are frequently competitive with "fine-grained" techniques such as `DejaVu` in terms of their ability to produce significant reductions in the test cases that are selected.

2. Fine-grained techniques sometimes substantially outperform coarse-grained techniques.

In addition, we observe the following related fact, which further complicates the choice of technique to use:

3. The relative difference in the performance of two techniques on any one version of a program is highly sensitive to the specific changes that are made to create the version.

These facts raise the following questions:

1. What are the proper criteria to be used for comparing the two different techniques? For instance, is `TestTube` to be preferred for testing the `player` system because "usually" it is nearly as precise as `DejaVu`, or is `DejaVu` to be preferred because it periodically offers much more substantial reductions than `TestTube`? The answer to this question depends in part on what the average-case behavior of the techniques will be over all future versions. Yet this is something that can only be predicted probabilistically rather than revealed analytically.

2. Is it possible to develop a hybrid approach (in both space and time) that combines the best features of coarse-grained and fine-grained techniques? For instance, fine-grained techniques could be used for the analysis of `main` functions and other "core" entities, while coarse-grained techniques could be used for the analysis of infrequently used entities. Perhaps profiling data and/or an operational profile could be used to guide the choice of analysis techniques on different entities.

All of this suggests that much additional research is needed to uncover the various factors that influence the relative variation in performance of different techniques over multiple versions of a software system. The research by Rosenblum and Weyuker on predicting the cost-effectiveness of regression test selection techniques represents initial effort in studying these issues [4].

Comparative evaluations of regression testing techniques require substantial infrastructure in the form of programs, modified versions, fault data, and test suites. Assembling such an infrastructure is difficult; however, once assembled, the infrastructure can serve as a basis for benchmark experiments. We suggest that all of our experimental subjects, with their versions and test suites, constitute the beginnings of a benchmark suite for use in comparative studies of regression testing. Additional work is necessary, however, to correctly estimate the extent to which results gathered on these subjects generalize to larger populations. Additional subjects that account for the factors (some as yet undiscovered) that affect test selection techniques will be needed.

We further suggest that studies of the sort we have conducted can serve as a model for benchmark experiments with regression test selection techniques. In addition to measuring precision, however, such experiments should also measure algorithm run-time, costs of executing selected and non-selected tests, and inclusiveness of techniques (how well the techniques do at selecting the tests, in T, that reveal the faults present in a version.)

In the future, we plan to continue our comparative evaluation of `DejaVu` and `TestTube`, with the following action items:

- We will apply the implementation of `TestTube` for C to the `player` system, thereby obtaining accurate cost data that could not be obtained by the simulation of `TestTube`.

- We will gather data on *all* relevant costs and benefits involved in applying `DejaVu` and `TestTube` to `player`, in order to compare the techniques in terms of their overall cost-effectiveness.

- Since data are available from a full cost study of the application of `TestTube` to the `KornShell` [3, 4], we will apply `DejaVu` to the `KornShell` and gather all relevant cost data, in order to provide an additional point of comparison between `DejaVu` and `TestTube`.

- We will continue the analysis of our data in an attempt to further explain why `DejaVu` sometimes substantially outperforms `TestTube` in test selection and why at other times it selects the same or nearly the same test cases as `TestTube`.

- We will apply and enhance the prediction model described by Rosenblum and Weyuker [4] in an attempt to develop ways of predicting performance variations among different regression test selection techniques.

We expect these additional studies to dramatically increase our understanding of the nature of selective regression testing.

Acknowledgments

We thank the anonymous referees for several helpful comments that improved the presentation. This work was supported in part by a grant from Microsoft, Inc., and by a National Science Foundation Faculty Early Career Development Award, CCR-9703108, to Oregon State University.

References

[1] Y.F. Chen, D.S. Rosenblum, and K.P. Vo. TestTube: A system for selective regression testing. In *Proceedings of the 16th International Conference on Software Engineering*, pages 211–222, May 1994.

[2] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proceedings of the 16th International Conference on Software Engineering*, pages 191–200, May 1994.

[3] D. Rosenblum and E. J. Weyuker. Lessons learned from a regression testing case study. *Empirical Software Engineering Journal*, 2(2), 1997.

[4] D. Rosenblum and E. J. Weyuker. Using coverage information to predict the cost-effectiveness of regression testing strategies. *IEEE Transactions on Software Engineering*, 23(3):146–156, March 1997.

[5] G. Rothermel and M.J. Harrold. Analyzing regression test selection techniques. *IEEE Transactions on Software Engineering*, 22(8):529–551, August 1996.

[6] G. Rothermel and M.J. Harrold. Experience with regression test selection. *Empirical Software Engineering Journal*, 2(2), 1997.

[7] G. Rothermel and M.J. Harrold. A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology*, 6(2):173–210, April 97.

[8] L.J. White, V. Narayanswamy, T. Friedman, M. Kirschenbaum, P. Piwowarski, and M. Oha. Test Manager: a regression testing tool. In *Proceedings of the Conference on Software Maintenance - 1993*, pages 338–347, September 1993.