

# Empirical Studies of a Prediction Model for Regression Test Selection

Mary Jean Harrold<sup>1</sup> David Rosenblum<sup>2</sup> Gregg Rothermel<sup>3</sup> Elaine Weyuker<sup>4</sup>

## Abstract

*Regression testing* is an important activity that can account for a large proportion of the cost of software maintenance. One approach to reducing the cost of regression testing is to employ a *selective regression testing technique* that (1) chooses a subset of a test suite that was used to test the software before the modifications, and then (2) uses this subset to test the modified software. Selective regression testing techniques reduce the cost of regression testing if the cost of selecting the subset from the test suite together with the cost of running the selected subset of test cases is less than the cost of rerunning the entire test suite.

Rosenblum and Weyuker recently proposed *coverage-based predictors* for use in predicting the effectiveness of regression test selection strategies. Using the regression testing cost model of Leung and White, Rosenblum and Weyuker demonstrated the applicability of these predictors by performing a case study involving 31 versions of the KornShell.

To further investigate the applicability of the Rosenblum-Weyuker (RW) predictor, additional empirical studies have been performed. The RW predictor was applied to a number of subjects, using two different selective regression testing tools, *DejaVu* and *TestTube*. These studies support two conclusions. First, they show that there is some variability in the success with which the predictors work, and second, they suggest that these results can be improved by incorporating information about the distribution of modifications. It is shown how the RW prediction model can be improved to provide such an accounting.

**Keywords:** software maintenance, regression testing, selective retest, regression test selection

## 1 Introduction

*Regression testing* is an important activity that can account for a large proportion of the cost of software maintenance [5, 17]. Regression testing is performed on modified software to provide confidence that the software behaves correctly and that modifications have not adversely impacted the software's quality. One approach to reducing the cost of regression testing is to employ a selective regression testing technique. A *selective regression testing technique* chooses a subset of a test suite that was used to test the software before modifications were made, and then uses this subset to test the modified software.<sup>5</sup> Selective regression testing techniques reduce the cost of regression testing if the cost of selecting the subset from the test suite together with the cost of running the selected subset of test cases is less than the cost of rerunning the entire test suite.

Empirical results obtained by Rothermel and Harrold on the effectiveness of their selective regression testing algorithms, implemented as a tool called *DejaVu*, suggest that test selection can sometimes be effective in reducing the cost of regression testing by reducing the number of test cases that need to be rerun [24, 26]. However, these studies also show that there are situations in which their algorithm is not cost-effective. Furthermore, other studies performed independently by Rosenblum and Weyuker with a different selective

---

<sup>1</sup>Department of Computer and Information Science, Ohio State University.

<sup>2</sup>Department of Information and Computer Science, University of California, Irvine.

<sup>3</sup>Department of Computer Science, Oregon State University.

<sup>4</sup>AT&T Labs – Research.

<sup>5</sup>A variety of selective regression testing techniques have been proposed (e.g., [1, 3, 4, 6, 7, 8, 9, 10, 12, 13, 15, 16, 18, 21, 26, 27, 28, 29, 30]). For an overview and analytical comparison of these techniques, see [25].

regression testing algorithm, implemented as a tool called `TestTube` [8], also show that such methods are not always cost-effective [23]. When selective regression testing is not cost-effective, the resources spent performing the test case selection are wasted. Thus, Rosenblum and Weyuker argue in [23] that it would be desirable to have a predictor that is inexpensive to apply but could indicate whether or not using a selective regression testing method is likely to be worthwhile.

With this motivation, Rosenblum and Weyuker [23] propose *coverage-based predictors* for use in predicting the cost-effectiveness of selective regression testing strategies. Their predictors use the average percentage of test cases that execute covered *entities*—such as statements, branches, or functions—to predict the number of test cases that will be selected when a change is made to those entities. One of these predictors is used to predict whether a *safe* selective regression testing strategy (one that selects all test cases that cover affected entities) will be cost-effective. Using the regression testing cost model of Leung and White [19], Rosenblum and Weyuker demonstrate the usefulness of this predictor by describing the results of a case study they performed involving 31 versions of the KornShell [23]. In that study, the predictor reported that, on average, it was expected that 87.3% of the test cases would be selected. Using the `TestTube` approach, 88.1% were actually selected on average over the 31 versions. Since the difference between these values is very small, the predictor was clearly extremely accurate in this case. The authors explain, however, that because of the way their selective regression testing model employs averages, the accuracy of their predictor might vary significantly in practice from version to version. This is particularly an issue if there is a wide variation in the distribution of changes among entities [23]. However, because their predictor is intended to be used for predicting the long-term behavior of a method over multiple versions, they argue that the use of averages is acceptable.

To further investigate the applicability of the Rosenblum-Weyuker (RW) predictor for safe selective regression testing strategies, we present in this paper the results of additional studies. We applied the RW predictor to subjects developed by researchers at Siemens Corporate Research for use in studies to compare the effectiveness of certain software testing strategies [14]. For the current paper we used both `DeJaVu` and `TestTube` to perform selective regression testing. In the following sections we discuss the results of our studies.

## 2 Background: The Rosenblum-Weyuker Predictor

Rosenblum and Weyuker presented a formal model of regression testing to support the definition and computation of predictors of cost-effectiveness [23]. Their model builds on work by Leung and White on modeling the cost of employing a selective regression testing method [19]. In both models, the total cost of regression testing incorporates two factors: the cost of executing test cases, and the cost of performing analyses to support test selection. A number of simplifying assumptions are made in the representation of the cost in these models:

1. The costs are constant on a per-test-case basis.
2. The costs represent a composite of the various costs that are actually incurred; for example, the cost associated with an individual test case is a composite that includes the costs of executing the test case, storing execution data, and validating the results.

3. The cost of the analyses needed to select test cases from the test suite has a completely negative impact on cost-effectiveness, in the sense that analysis activities drain resources that could otherwise be used to support the execution of additional test cases.
4. Cost-effectiveness is an inherent attribute of test selection over the complete maintenance life-cycle, rather than an attribute of individual versions.

As in Rosenblum and Weyuker’s model [23], we let  $P$  denote the system under test and let  $T$  denote the regression test suite for  $P$ , with  $|T|$  denoting the number of individual test cases in  $T$ . Let  $M$  be the selective regression testing method used to choose a subset of  $T$  for testing a modified version of  $P$ , and let  $E$  be the set of entities of the system under test that are considered by  $M$ . It is assumed that  $T$  and  $E$  are non-empty and that every syntactic element of  $P$  belongs to at least one entity in  $E$ .

The Rosenblum-Weyuker (RW) model defined  $covers_M(t, e)$  as the coverage relation induced by method  $M$  for  $P$  and defined over  $T \times E$ , with  $covers_M(t, e)$  true if and only if the execution of  $P$  on test case  $t$  causes entity  $e$  to be *exercised* at least once. Rosenblum and Weyuker specify meanings for “exercised” for several kinds of entities of  $P$ . For example, if  $e$  is a function or module of  $P$ ,  $e$  is exercised whenever it is invoked, and if  $e$  is a simple statement, statement condition, definition-use association or other kind of execution subpath of  $P$ ,  $e$  is exercised whenever it is executed.

Letting  $E^C$  denote the set of covered entities, the RW model defined  $E^C$  as follows:

$$E^C = \{e \in E \mid \exists t \in T (covers_M(t, e))\}$$

with  $|E^C|$  denoting the number of covered entities. Furthermore,  $covers_M(t, e)$  can be represented by a 0-1 matrix  $C$ , whose rows represent elements of  $T$  and whose columns represent elements of  $E$ . Then, element  $C_{i,j}$  of  $C$  is defined to be:

$$C_{i,j} = \begin{cases} 1 & \text{if } covers_M(i, j) \\ 0 & \text{otherwise.} \end{cases}$$

Finally,  $CC$  was the *cumulative coverage* achieved by  $T$  (i.e., the total number of ones in the 0-1 matrix):

$$CC = \sum_{i=1}^{|T|} \sum_{j=1}^{|E|} C_{i,j}$$

As a first step in computing a predictor for safe strategies when a single entity had been changed, Rosenblum and Weyuker considered the expected number of test cases that would have to be rerun. Calling this average  $N_M$  they defined:

$$N_M = \frac{CC}{|E|}$$

Rosenblum and Weyuker emphasized that this predictor was only intended to be used when the selective regression testing strategy’s goal was to rerun *all affected* test cases.

A slightly refined variant of  $N_M$  was defined using  $E^C$  rather than  $E$  as the universe of entities.

$$N_M^c = \frac{CC}{|E^C|}$$

Then the fraction of the test suite that must be rerun was denoted  $\pi_M$ , the predictor for  $|T_M|/|T|$ :

$$\begin{aligned} \pi_M &= \frac{N_M^c}{|T|} \\ &= \frac{CC}{|E^C||T|} \end{aligned}$$

Rosenblum and Weyuker discussed results of a case study in which test selection and prediction results were compared for 31 versions of the KornShell using the `TestTube` selective regression testing method. As mentioned above, in this study, the test selection technique chose an average of 88.1% of the test cases in the test suite over the 31 versions, while the predicted value was 87.3%. They concluded that, because the difference between these values was very small, their results indicated the usefulness of their predictor as a way of predicting cost-effectiveness.

### 3 Two New Empirical Studies of the Rosenblum-Weyuker Predictor

The results of the Rosenblum-Weyuker case study were encouraging for two reasons:

1. The difference between the predicted and actual values was insignificant.
2. Because a large proportion of the test set would have to be rerun for regression testing, and it could be quite expensive to perform the analysis necessary to determine which test cases did not need to be rerun, it would often be cost-effective to use the predictor to discover this and then simply rerun the entire test suite rather than selecting a subset of the test suite.

Nevertheless, this study involved a *single selective regression testing method* applied to a *single subject program*, albeit a large and widely-used one for which there were a substantial number of actual production versions. In order to obtain a broader picture of the usefulness of the RW predictor, we conducted additional studies with other subject software and other selective regression testing methods. In particular, we performed two new studies with two methods, `DejaVu` and `TestTube`, applied to a suite of subject programs that have been used in other studies in the testing literature.

#### 3.1 Statement of Hypothesis

The hypothesis we tested in our new studies is the hypothesis of the Rosenblum-Weyuker study:

**Hypothesis:** Given a system under test  $P$ , a regression test suite  $T$  for  $P$ , and a selective regression testing method  $M$ , it is possible to use information about the coverage relation  $covers_M$  induced by  $M$  over  $T$  and the entities of  $P$  to predict whether or not  $M$  will be cost-effective for regression testing future versions of  $P$ .

The previous and current studies test this hypothesis under the following assumptions:

1. The prediction is based on a cost metric that is appropriate for  $P$  and  $T$ . Certain simplifying assumptions are made about costs, as described in Section 2.
2. The prediction is performed using data from a single version of  $P$  to predict cost-effectiveness for all future versions of  $P$ .
3. “Cost-effective” means that the cumulative cost over all future versions of  $P$  of applying  $M$  and executing the test cases in  $T$  selected by  $M$  is less than the cumulative cost over all future versions of  $P$  of running all test cases in  $T$  (the so-called *retest-all* method).

Program	Lines of Code	Number of Functions	Number of Modified Versions	Test Pool Size	Average Test Suite Size	Description
tcas	138	9	41	1608	6	altitude separation
schedule2	297	16	10	2710	8	priority scheduler
schedule1	299	18	9	2680	8	priority scheduler
totinfo	346	7	23	1054	7	information measure
printtokens2	483	19	10	4115	12	lexical analyzer
printtokens1	402	20	7	4130	16	lexical analyzer
replace	516	21	32	5542	19	pattern replacement

Table 1: Summary of subject programs.

### 3.2 Subject Programs

For our new studies, we used seven C programs as subjects that had been previously used in a study by researchers at Siemens Corporate Research [14]. Because the researchers at Siemens sought to study the fault-detecting effectiveness of different coverage criteria, they created *faulty* modified versions of the seven base programs by manually seeding the programs with faults, usually by modifying a single line of code in the base version, and never modifying more than five lines of code. Their goal was to introduce faults that were as realistic as possible, based on their experience. Ten people performed the fault seeding, working “mostly without knowledge of each other’s work” [14, p. 196].

For each base program, Hutchins et al. created a large *test pool* containing possible test cases for the program. To populate these test pools, they first created an initial set of black-box test cases “according to good testing practices, based on the tester’s understanding of the program’s functionality and knowledge of special values and boundary points that are easily observable in the code” [14, p. 194], using the *category partition method* and the Siemens Test Specification Language tool [2, 20]. They then augmented this set with manually-created white-box test cases to ensure that each executable statement, edge, and definition-use pair in the base program or its control flow graph was exercised by at least 30 test cases. To obtain meaningful results with the seeded versions of the programs, the researchers retained only faults that were “neither too easy nor too hard to detect” [14, p. 196], which they defined as being detectable by at least three and at most 350 test cases in the test pool associated with each program.

Table 1 presents information about these subjects. For each program, the table lists its name, the number of lines of code in the program, the number of functions in the program, the number of modified (i.e., fault seeded) versions of the program, the size of the test pool, the average number of test cases in each of the 1000 coverage-based test suites we generated for our studies, and a brief description of the program’s function. We describe the generation of the 1000 coverage-based test suites in greater detail below.

### 3.3 Design of the New Studies

In both studies, our analysis was based on measurements of the following variables:

**Independent Variable:** For each subject program  $P$ , test suite  $T$  for  $P$  and selective regression testing method  $M$ , the independent variable is the relation  $covers_M(t, e)$  defined in Section 2.

**Dependent Variables:** For each subject program  $P$ , test suite  $T$  for  $P$  and selective regression testing method  $M$ , there are two dependent variables: (1) the cost of applying  $M$  to  $P$  and  $T$ ,

and (2) the cost of executing  $P$  on the test cases selected by  $M$  from  $T$ .

We then used the model described in Section 2 plus our measurements of the dependent variables to perform analyses of cost-effectiveness. Each study involved different kinds of analysis, as described below.

For both studies, we used the Siemens test pools from which we selected smaller test suites. In particular, we randomly generated 1000 branch-coverage-based test suites for each base program from its associated test pool.<sup>6</sup> To create each test suite  $T_i$ ,  $1 \leq i \leq 1000$ , for program  $P$ , we applied the following algorithm:

1. Initialize  $T_i$  to  $\phi$ .
2. While uncovered coverable branches remain,
  - (a) Randomly select an element  $t$  from  $T$  using a uniform distribution,
  - (b) Execute  $P$  with  $t$ , recording coverage,
  - (c) Add  $t$  to  $T_i$  if it covered branches in  $P$  not previously covered by the other test cases in  $T_i$ ,
3. If  $T_i$  differs from all other previously generated test suites, keep  $T_i$ , and increment  $i$ ; otherwise, generate a new  $T_i$ .

Step 3 of the procedure ensures that there are no duplicate test suites for a program, although two different test suites may have some test cases in common.

For both of our studies, we computed the cost measures (dependent variables) for both `DejaVu` and `TestTube`, and we compared this information to test selection predictions computed using the RW predictor. To gather this information, we considered each base program  $P$  with each modified version  $P_i$  and each test suite  $T_j$ . For each  $P$  and each  $T_j$ , we computed the following:

$\pi_{\text{DejaVu}_j}$ , the percentage of test cases of  $T_j$  that the RW predictor predicts will be selected by `DejaVu` when an arbitrary change is made to  $P$ ;

$\pi_{\text{TestTube}_j}$ , the percentage of test cases of  $T_j$  that the RW predictor predicts will be selected by `TestTube` when an arbitrary change is made to  $P$ ;

$S_{\text{DejaVu}_{i,j}}$ , the percentage of test cases of  $T_j$  actually selected by `DejaVu` for the changes made to create  $P_i$  from  $P$ ; and

$S_{\text{TestTube}_{i,j}}$ , the percentage of test cases of  $T_j$  actually selected by `TestTube` for the changes made to create  $P_i$  from  $P$ .

Finally, we used these values to evaluate the accuracy of the RW predictor, as described in detail in the following sections.

### 3.3.1 Study 1

The goal of our first study was to determine the accuracy, on average, of the RW predictor for the subject programs, modified versions, and test suites for each of the selective regression testing approaches we considered. We therefore used the regression test selection information described above to compute the average

---

<sup>6</sup>Because our studies focused on the cost-effectiveness of selective regression testing methods rather than the fault-detecting effectiveness of coverage criteria, the realism of the modifications made to the Siemens programs is not a significant issue. What is more important is that they were made independently by people not involved in our study, thereby reducing the potential for bias in our results.

percentages of test cases selected by DeJaVu and TestTube over all versions  $P_i$  of  $P$ . For each  $P$  and each  $T_j$ , we computed the following:

$$\bar{S}_{\text{DeJaVu}_j} = \frac{\sum_{i=1}^{|\text{versions of } P|} S_{\text{DeJaVu}_{i,j}}}{|\text{versions of } P|} \quad (1)$$

$$\bar{S}_{\text{TestTube}_j} = \frac{\sum_{i=1}^{|\text{versions of } P|} S_{\text{TestTube}_{i,j}}}{|\text{versions of } P|} \quad (2)$$

The first step in our investigation was to see how much the percentage of test cases actually selected by each of the methods differed from the predicted percentage of test cases. For this analysis, we needed the following two additional pieces of data, which we computed for each  $P$  and each  $T_j$ :

$$\bar{D}_{\text{DeJaVu}_j} = \pi_{\text{DeJaVu}_j} - \bar{S}_{\text{DeJaVu}_j} \quad (3)$$

$$\bar{D}_{\text{TestTube}_j} = \pi_{\text{TestTube}_j} - \bar{S}_{\text{TestTube}_j} \quad (4)$$

$\bar{D}_{\text{DeJaVu}_j}$  and  $\bar{D}_{\text{TestTube}_j}$  represent the deviations of the percentages of the test cases predicted by the RW predictor for  $T_j$  from the average of the actual percentages of test cases selected by the respective method for all versions  $P_i$  of  $P$ . Because it is possible for  $\bar{D}_{\text{DeJaVu}_j}$  and  $\bar{D}_{\text{TestTube}_j}$  to lie anywhere in the range  $[-100, 100]$ , we wanted to determine the ranges into which the values for  $\bar{D}_{\text{DeJaVu}_j}$  and  $\bar{D}_{\text{TestTube}_j}$  actually fell. Thus, we rounded the values of  $\bar{D}_{\text{DeJaVu}_j}$  and  $\bar{D}_{\text{TestTube}_j}$  to the nearest integer  $I$ , and computed, for each  $I$  such that  $-100 \leq I \leq 100$ , the percentage of the rounded  $\bar{D}$  values with value  $I$ . For each  $P$ , using each of its  $\bar{D}_{\text{DeJaVu}_j}$  values, the result was a set  $\bar{H}_{\text{DeJaVu}}$ :

$$\begin{aligned} \bar{H}_{\text{DeJaVu}} = \{ & (r, prd) \mid r \text{ is the range value, } -100 \leq r \leq 100, \\ & prd \text{ is the percentage of rounded } \bar{D}_{\text{DeJaVu}_j} \text{ values at } r \} \end{aligned} \quad (5)$$

Similarly, for each  $P$ , using each of its  $\bar{D}_{\text{TestTube}_j}$  values, the result was a set  $\bar{H}_{\text{TestTube}}$ :

$$\begin{aligned} \bar{H}_{\text{TestTube}} = \{ & (r, prd) \mid r \text{ is the range value, } -100 \leq r \leq 100, \\ & prd \text{ is the percentage of rounded } \bar{D}_{\text{TestTube}_j} \text{ values at } r \} \end{aligned} \quad (6)$$

These sets essentially form a histogram of the deviation values. In the ideal case of perfect prediction, all deviations would be zero, and therefore each graph would consist of the single point  $(0, 100\%)$ .

### 3.3.2 Study 2

In Study 1, we treated the RW predictor as a *general* predictor in an attempt to determine how accurate it is for predicting test selection percentages for *all future* versions of a program. In the earlier KornShell study [23], it was determined that the relation  $\text{covers}_M(t, e)$  changes very little during maintenance. In particular, Rosenblum and Weyuker found that the coverage relation was extraordinarily stable over the 31 versions of KornShell that they included in their study, with an average of only one-third of one percent of the elements in the relation changing from version to version, and only two versions for which the amount of change exceeded one percent. For this reason, Rosenblum and Weyuker argued that coverage information from a single version might be usable to guide test selection over several subsequent new versions, thereby saving the cost of redoing the coverage analysis on each new version.

However, in circumstances where the coverage relation is not stable, it may be desirable to make predictions about whether or not test selection is likely to be cost-effective for a *particular* version, using version-specific information. The goal of our second study was therefore to examine the accuracy of the RW predictor as a *version-specific* predictor for our subject programs, modified versions, and test suites. The intuition is that in some cases it might be important to utilize information that is known about the specific changes made to produce a particular version.

We considered each base program  $P$ , with each modified version  $P_i$  and test suite  $T_j$ , as we had done in Study 1, except that we did not compute averages over the percentages of test cases selected over all versions of a program. Instead, the data sets for this study contain one deviation for each test suite and each version of a program.

As in Study 1, the first step in our investigation was to see how much the percentage of test cases actually selected by each of the methods differed from the predicted percentage of test cases. For this analysis, we needed the following additional pieces of data, which we computed for each  $P$ , each  $P_i$ , and each  $T_j$ :

$$D_{\text{DejaVu}_{i,j}} = \pi_{\text{DejaVu}_j} - S_{\text{DejaVu}_{i,j}} \quad (7)$$

$$D_{\text{TestTube}_{i,j}} = \pi_{\text{TestTube}_j} - S_{\text{TestTube}_{i,j}} \quad (8)$$

$D_{\text{DejaVu}_{i,j}}$  and  $D_{\text{TestTube}_{i,j}}$  represent the deviations of the percentages of the test cases predicted by the RW predictor for  $T_j$  from the actual percentages of test cases selected by the respective method for the versions  $P_i$  of  $P$ . As in Study 1, to determine the ranges into which the values for  $D_{\text{DejaVu}_{i,j}}$  and  $D_{\text{TestTube}_{i,j}}$  actually fell, we rounded the values of  $D_{\text{DejaVu}_{i,j}}$  and  $D_{\text{TestTube}_{i,j}}$  to the nearest integer  $I$  and computed, for each  $I$  such that  $-100 \leq I \leq 100$ , the percentage of the rounded  $D$  values with value  $I$ . For each  $P$ , using each of its  $D_{\text{DejaVu}_{i,j}}$  values, the result was a set  $H_{\text{DejaVu}}$ :

$$H_{\text{DejaVu}} = \{(r, prd) \mid r \text{ is the range value, } -100 \leq r \leq 100, \\ prd \text{ is the percentage of rounded } D_{\text{DejaVu}_{i,j}} \text{ values at } r\} \quad (9)$$

Similarly, for each  $P$ , using each of its  $D_{\text{TestTube}_{i,j}}$  values, the result was a set  $H_{\text{TestTube}}$ :

$$H_{\text{TestTube}} = \{(r, prd) \mid r \text{ is the range value, } -100 \leq r \leq 100, \\ prd \text{ is the percentage of rounded } D_{\text{TestTube}_{i,j}} \text{ values at } r\} \quad (10)$$

### 3.4 Threats to Validity

There are three types of potential threats to the validity of our studies: (1) threats to *construct validity*, which concern our measurements of the constructs of interest (i.e., the phenomena underlying the independent and dependent variables); (2) threats to *internal validity*, which concern our supposition of a causal relation between the phenomena underlying the independent and dependent variables; and (3) threats to *external validity*, which concern our ability to generalize our results.

#### 3.4.1 Construct Validity

Construct validity deals directly with the issue of whether or not we are measuring what we purport to be measuring. The RW predictor relies directly on coverage information. It is true that our measurements

of the coverage relation are highly accurate, but the coverage relation is certainly not the only possible phenomenon that affects the cost-effectiveness of selective regression testing. Therefore, because this measure only partially captures that potential, we need to find other phenomena that we can measure for purposes of prediction.

Furthermore, we have relied exclusively on the number of test cases selected as the measure of cost reduction. Care must be taken in the counting of test cases deemed to be “selected”, since there are other reasons a test case may not be selected for execution (such as the testing personnel simply lacking the time to run the test case). In addition, whereas this particular measure of cost reduction has been appropriate for the subjects we have studied, there may be other testing situations for which the expense of a test lab and testing personnel might be significant cost factors. In particular, the possibility of using spare cycles might affect the decision of whether or not it is worthwhile to use a selective regression testing method at all in order to eliminate test cases, and therefore whether or not a predictor is meaningful.

### 3.4.2 Internal Validity

The basic premises underlying Rosenblum and Weyuker’s original predictor were that (1) the cost-effectiveness of a selective regression testing method, and hence our ability to predict cost-effectiveness, are directly dependent on the percentage of the test suite that the selective regression testing method chooses to run, and that (2) this percentage in turn is directly dependent on the coverage relation. In this experiment we take the first premise as an assumption, and we investigate whether the relation between percentage of tests selected and coverage exists and is appropriate as a basis for prediction. The new data presented in this paper reveal that coverage explains only part of the cost-effectiveness of a method and the behavior of the RW predictor. Future studies should therefore attempt to identify the other factors that affect test selection and cost effectiveness.

### 3.4.3 External Validity

The threats to external validity of our studies are centered around the issue of how representative the subjects of our studies are. All of the subject programs in our new studies are small, and the sizes of the selected test suites are small. This means that even a selected test suite whose size differs from the average or the predicted value by one or two elements would produce a relatively large percentage difference. (The results of Study 1 are therefore particularly interesting because they show small average deviations for most of the subject programs.)

For the studies involving the Siemens programs, the test suites were chosen from the test pools using branch coverage, which is much finer granularity than `TestTube` uses, suggesting that there is a potential “mismatch” in granularity that may somehow skew the results. More generally, it is reasonable to ask whether our results are dependent upon the method by which the test pools and test suites were generated, and the way in which the programs and modifications were designed. We view the branch coverage suites as being reasonable test suites that could be generated in practice, if coverage-based testing of the programs were being performed. Of course there are many other ways that testers could and do select test cases, but because the test suites we have studied are a type of suite that could be found in practice, results about predictive power with respect to such test suites are valuable.

The fact that the faults were synthetic (in the sense that they were seeded into the Siemens programs) may also affect our ability to investigate the extent to which change information can help us predict future changes. In a later section we will introduce a new predictor that we call the weighted predictor. This predictor depends on version-specific change information. Because it seemed likely that conclusions drawn using synthetic changes would not necessarily hold for naturally occurring faults, we did not attempt to use the Siemens programs and their faulty versions to empirically investigate the use of the weighted predictor. Nevertheless, the predictor itself is not dependent on whether the changes are seeded or naturally occurring, and thus our results provide useful data points.

## 4 Data and Results

### 4.1 Study 1

Figure 1 presents data for  $\overline{D}_{\text{DejaVu}_j}$  (Equation (3)) and  $\overline{D}_{\text{TestTube}_j}$  (Equation (4)), with one graph for each subject program  $P$ ; the Appendix gives details of the computation of this data using one of the subject programs, `printtokens2`, as an example.

Each graph contains a solid curve and a dashed curve. The solid curve consists of the connected set of points  $\overline{H}_{\text{DejaVu}_j}$  (Equation (5)), whereas the dashed curve consists of the connected set of points  $\overline{H}_{\text{TestTube}_j}$  (Equation (6)). Points to the left of the “0” deviation label on the horizontal axes represent cases in which the percentage of test cases predicted was less than the percentage of test cases selected by the tool, whereas points to the right of the “0” represent cases in which the percentage of test cases predicted was greater than the percentage of test cases selected by the tool. To facilitate display of the values, a logarithmic transformation has been applied to the y axes. No smoothing algorithms were applied to the curves.

For all  $P$  and  $T_j$ , the  $\overline{D}_{\text{DejaVu}_j}$  were in the range  $[-20,33]$  and the  $\overline{D}_{\text{TestTube}_j}$  were in the range  $[-24,28]$ . However, as we shall see in Figure 2, these ranges are a bit misleading because there are rarely any significant number of values outside the range  $(-10,0]$  or  $[0,10)$ , particularly for `TestTube`.

The graphs show that, for our subjects, the RW predictor was quite successful for both the `DejaVu` and `TestTube` selection methods. The predictor was least successful for the `printtokens2` program for which it predicted an average of 23% more test cases than `DejaVu` actually selected. This was the only deviation that exceeded 10% using the `DejaVu` approach. For `schedule1`, the prediction was roughly 9% high, on average, compared to the `DejaVu`-selected test suite. `DejaVu` selected an average of roughly 10% more test cases than predicted for `schedule2`, 7% more for `totinfo`, 7% more for `tcas`, 3% more for `printtokens1`, and 4% fewer for `replace` than the RW predictor predicted.

For `TestTube`, the predictor also almost always predicted within 10% of the actual average number of test cases that were actually selected. The only exception was for the `totinfo` program, for which the average deviation was under 12%. For the other programs, the average deviations were 5% for the `printtokens1` program, 5% for the `printtokens2` program, 4% for the `replace` program, 7% for the `tcas` program, 10% for `schedule1` and 1% for `schedule2`. We consider these results encouraging, although not as successful as the results described by Rosenblum and Weyuker for the `KornShell` case study. Recall that in that study there were a total of 31 versions of `KornShell`, a large program with a very large user base, with all changes made to fix real faults or modify functionality. None of the changes were made for the purpose of the study.

Another way to view the data for this study is to consider deviations of the predicted percentage from the

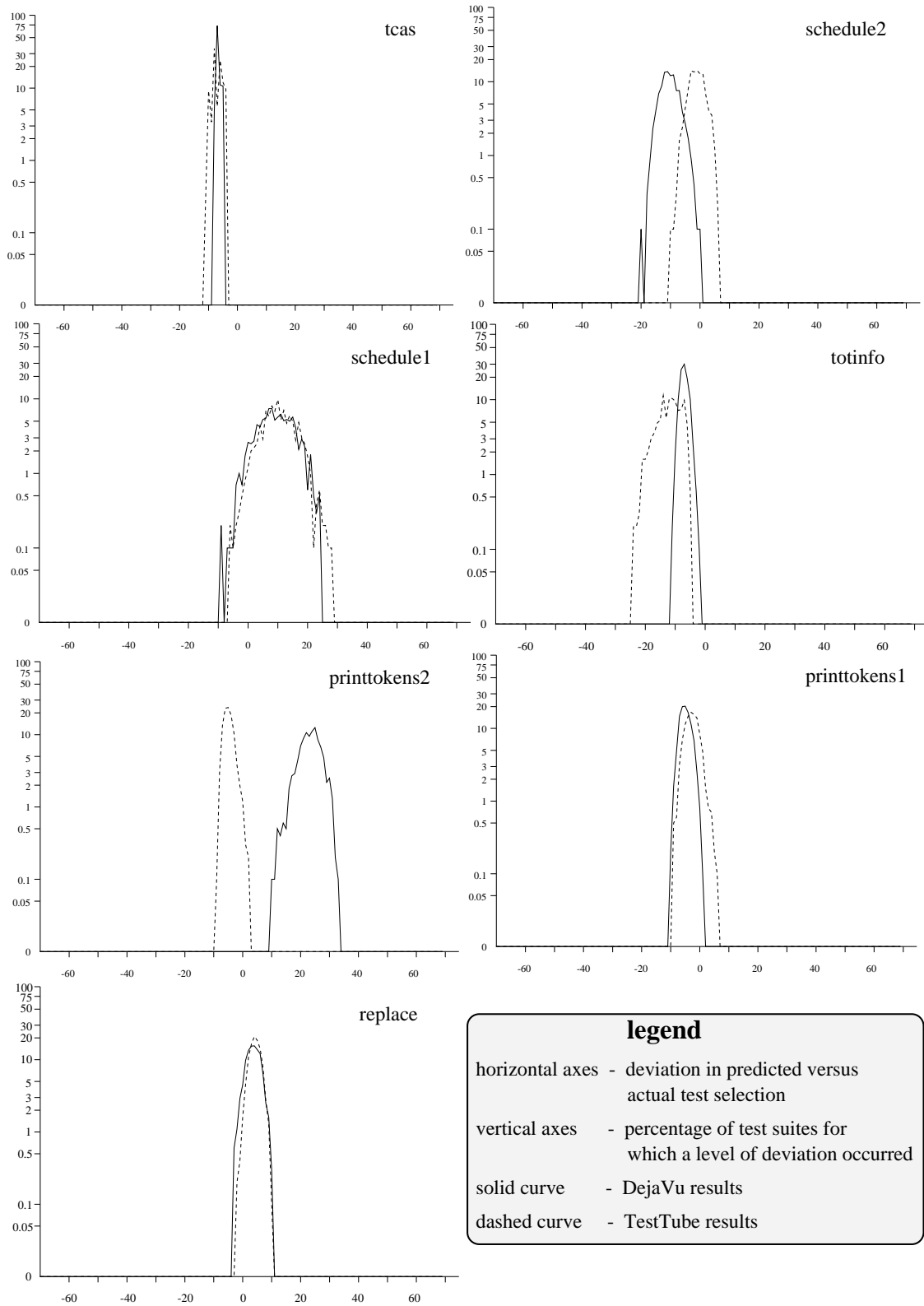


Figure 1: Deviation between predicted and actual test selection percentages for application of DejaVu and TestTube to the subject programs. The figure contains one graph for each subject program. In each graph, the solid curve represents deviations for DejaVu, and the dashed curve represents deviations for TestTube.

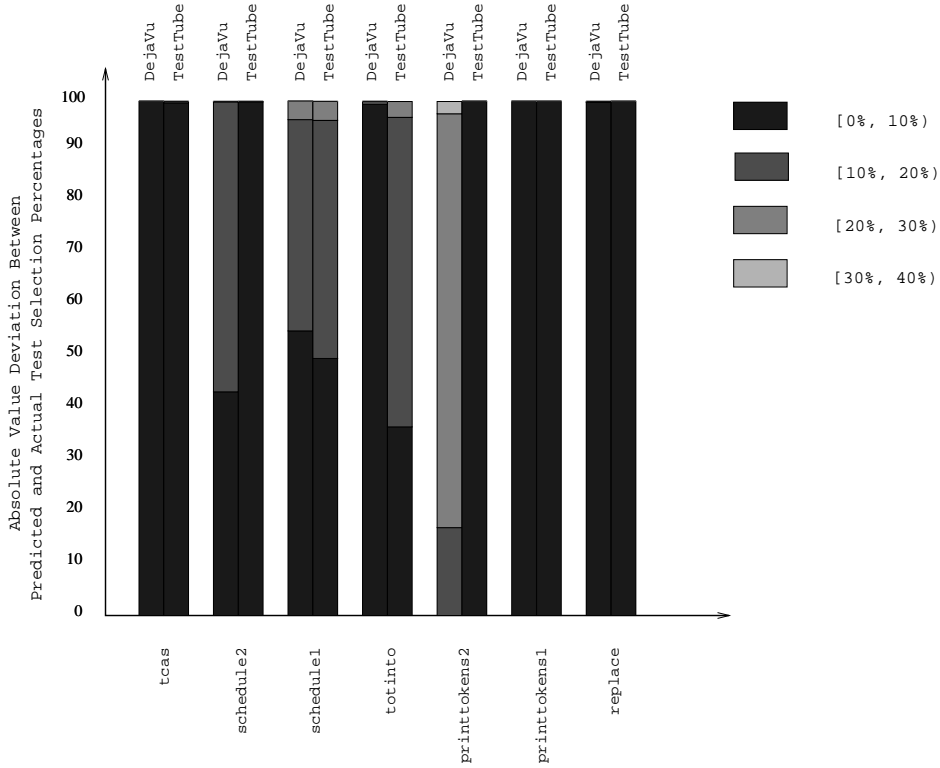


Figure 2: Absolute value deviation between predicted and actual test selection percentages for application of DeJaVu and TestTube to the subject programs. The figure contains two bars for each subject program: the left bar of each pair represents the absolute value deviation of DeJaVu results from the RW predictor, and the right bar represents the absolute value deviation of TestTube results from the RW predictor. For each program  $P$ , these numbers are averages over all versions  $P_i$  of  $P$ . Each bar represents 100% of the test suites  $T_j$ , with shading used to indicate the percentage of test suites whose deviations fell within the corresponding range.

actual percentage without considering whether the predicted percentage was greater or less than the actual percentage selected. These deviations constitute the *absolute value deviation*. To compute the absolute value deviation, we performed some additional computations:

For each  $P$  and each  $T_j$ , we first computed  $Abs\overline{D}_{DeJaVu_j} = |\overline{D}_{DeJaVu_j}|$  and  $Abs\overline{D}_{TestTube_j} = |\overline{D}_{TestTube_j}|$ . We then tabulated the percentage of the  $Abs\overline{D}_{DeJaVu_j}$  and the  $Abs\overline{D}_{TestTube_j}$  that fell in each of the ranges  $[0\%,10\%)$ ,  $[10\%,20\%)$ , ...,  $[90\%,100\%]$ .

Figure 2 depicts these results as segmented bar graphs. The figure contains two bars for each subject program: the left bar of each pair represents the absolute value deviation of DeJaVu results from the RW predictor, and the right bar represents the absolute value deviation of TestTube results from the RW predictor. For each program  $P$ , these numbers are averages over all versions  $P_i$  of  $P$ . Each bar represents 100% of the test suites  $T_j$ , with shading used to indicate the percentage of test suites whose deviations fell within the corresponding range. For instance, in the case of `printtokens2`, 100% of the test suites showed less than 10% deviation for TestTube, whereas for DeJaVu, 14% of the test suites showed deviations between 10% and 20%, 82% showed deviations between 20% and 30%, and 4% showed deviations between 30% and 40%.

The results of this study show that for many of the subject programs, modified versions, and test suites, the absolute value deviation for both DeJaVu and TestTube was less than 10%. In these cases, the RW

model explains a significant portion of the data. However, in a few cases, the absolute value deviation was significant. For example, as mentioned above, for `printtokens2`, the absolute value deviation from the predictor for `DejaVu` was between 20% and 30% for more than 80% of the versions.

One additional feature of the data displayed in Figure 1 bears discussion. For all programs other than `printtokens2`, the curves that represent deviations for `DejaVu` and `TestTube` are (relatively) close to one another. For `printtokens2`, in contrast, the two curves are disjoint and (relatively) widely separated. Examination of the code coverage data and locations of modifications for the programs reveals reasons for this difference.

Sixteen of the nineteen `printtokens2` functions are executed by a large percentage (on average over 95%) of the test cases in the program’s test pool; the remaining three functions are executed by much lower percentages (between 20% and 50%) of the test cases in that test pool. All modifications of `printtokens2` occur in the sixteen functions that are executed by nearly all test cases. Thus, the actual test selections by `TestTube`, on average, include most test cases. The presence of the latter three functions, and the small number of test cases that reach them, however, causes a reduction in the average number of test cases per function, and causes the function-level predictor to under-predict by between 0% and 10% the number of test cases selected by `TestTube`.

Even though nearly all test cases enter nearly all functions in `printtokens2`, several of these functions contain branches that significantly partition the paths taken by test cases that enter the functions. Thus, many of the statements in `printtokens2` are actually executed by fewer than 50% of the test cases that enter their enclosing functions. When modifications occur in these less-frequently executed statements, `DejaVu` selects much smaller test suites than `TestTube`. (For further empirical comparison of `TestTube` and `DejaVu`, see [22].) This is the case for approximately half of the modified versions of `printtokens2` utilized in this study. However, the presence of a large number of statements that are executed by a larger proportion of the test cases causes the average number of test cases per statement to exceed the number of test cases through modified statements. The end result is that the statement-level predictor over-predicts the number of test cases selected by `DejaVu` by between 5% and 27%. Of course, the precise locations of modifications in the subjects directly affect the results. Therefore, the fact that all changes were synthetic is of concern when trying to generalize these results.

## 4.2 Study 2

Like Figure 1, Figure 3 contains one graph for each subject program. The graphs also use the same notation as was used in Figure 1, using a solid curve to represent the percentage of occurrences of  $D_{DejaVu,i,j}$  over deviations for all test suites  $T_j$  and using a dashed curve to represent the percentage of occurrences of  $D_{TestTube,i,j}$  over deviations for all test suites  $T_j$ . Again, a logarithmic transformation has been applied to the y axes. Figure 4 depicts these results as a segmented bar graph, in the manner of Figure 2.

The results of this study show that, for the subject programs, modified versions, and test cases, the deviations and absolute value deviations for individual versions for both `DejaVu` and `TestTube` are much greater than in Study 1. This is not surprising because in this study the results are not averaged over all versions as they were in Study 1. For example, consider `tcas`, `printtokens1`, and `replace`. In Study 1, the average absolute value deviation from the predicted percentage for each of these programs is less than 10% using either `DejaVu` or `TestTube`. However, when individual versions are considered, the percentage of test

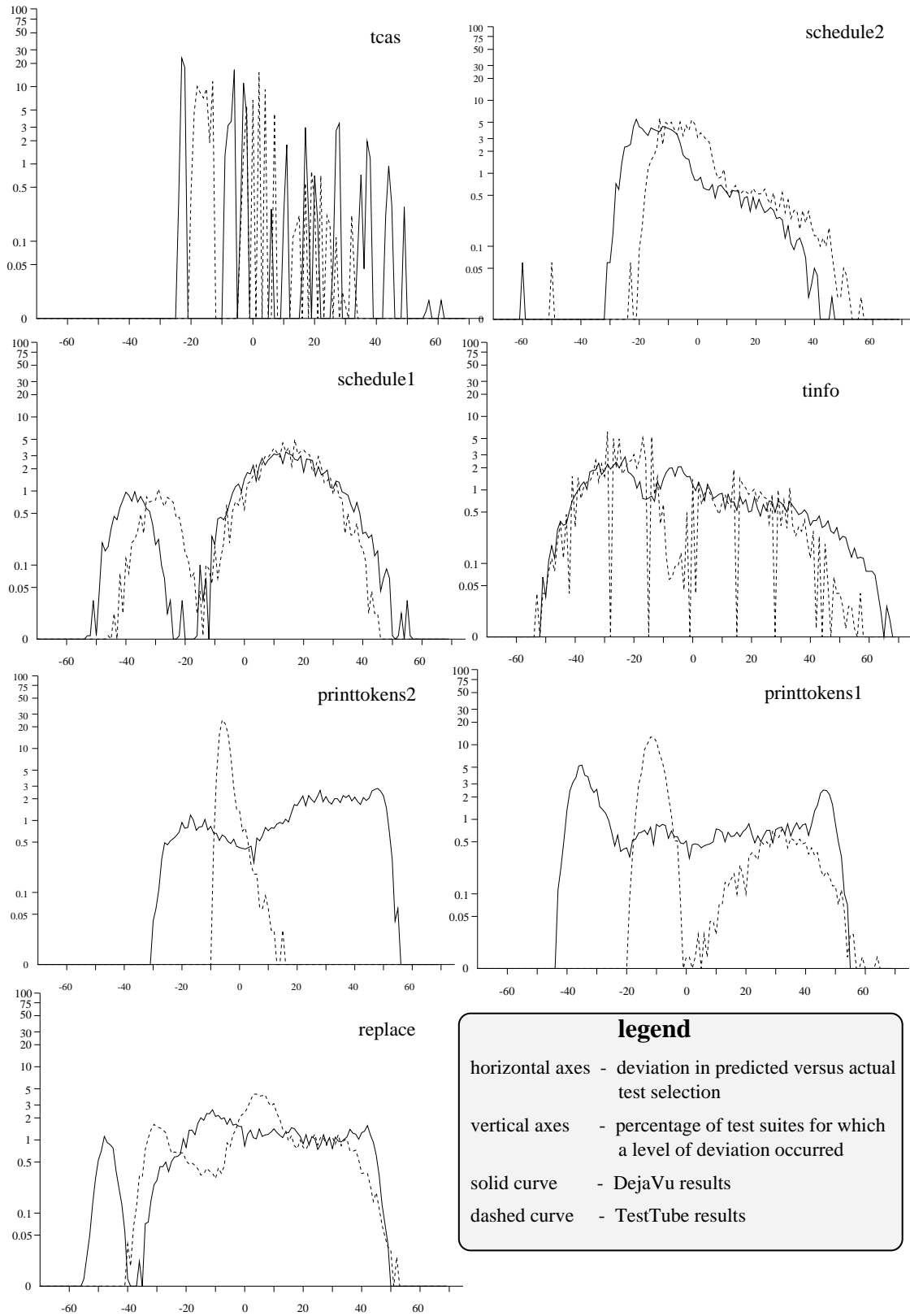


Figure 3: Version-specific absolute value deviation between predicted and actual test selection percentages for application of DejaVu and TestTube to the subject programs. The figure contains one graph for each subject program. In each graph, the solid curve represents deviations for DejaVu, and the dashed curve represents deviations for TestTube.

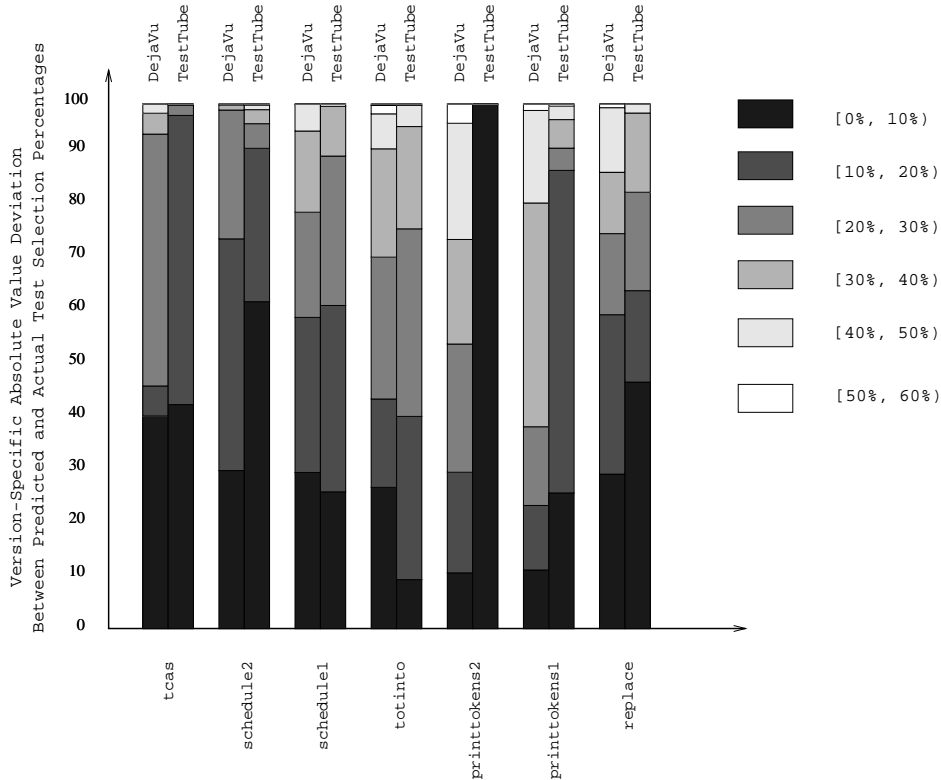


Figure 4: Version-specific absolute value deviation between predicted and actual test selection percentages for application of `DejaVu` and `TestTube` to the subject programs. The figure contains two bars for each subject program: the left bar of each pair represents the absolute value deviation of `DejaVu` results from the RW predictor, and the right bar represents the absolute value deviation of `TestTube` results from the RW predictor. For each program  $P$ , these numbers are averages over all versions  $P_i$  of  $P$ . Each bar represents 100% of the test suites  $T_j$ , with shading used to indicate the percentage of test suites whose deviations fell within the corresponding range.

cases selected by `DejaVu` for these programs varies significantly, up to 64%, from the percentages predicted. Deviations and absolute value deviations for the other subjects show similar differences. In Figure 3, the range of deviations can be seen. In most cases there are at least a few versions that have a small number of instances for which the deviations are significant. The bar graphs in Figure 4 show more clearly how frequently these large absolute value deviations occur.

In Figure 3, the data for `printtokens2` is again particularly interesting. In this case, the curve for `TestTube` is peaked and relatively narrow, whereas the curve for `DejaVu` is nearly flat and relatively wide. As discussed in the preceding section, for both techniques, these differences reflect differences in the degree of variance in the coverage relations at the statement and function level, as well as differences in the location of modifications. In this case, however, considering prediction on a version-specific basis causes the deviation in prediction at the statement level, where the variance in coverage is large, to be flat. Lack of variance in coverage at the function level prevents the `TestTube` curve from being flat.

## 5 Improved Predictors

As discussed in Section 3.4, the assumptions underlying our measurement of costs may pose threats to the validity of our conclusions about predictions of cost-effectiveness using the RW predictor. Furthermore, in some of the subject programs of our studies, there was significant absolute deviation of the results of the selective regression testing tools (`DejaVu` and `TestTube`) with respect to test selection values predicted by the RW predictor. Therefore, we believe that there may be factors affecting cost-effectiveness that are not being captured by the RW predictor. These factors, if added to the model, could improve the accuracy of both general and version-specific predictors. The RW predictor accounts for test coverage but does not account for the locations of modifications. Therefore, one obvious refinement would be to incorporate information about modifications into the predictor. We saw in Study 2 that the specific changes made to create a particular version may have significant effects on the accuracy of prediction in practice. Thus, we believe that an extended *weighted predictor* might be more accurate for both general and version-specific prediction. Such a predictor would incorporate information about the locations of the changes and weight the predictor accordingly.

To this end, in this section we extend the RW predictor by adding weights that represent the relative frequency of changes to the covered entities. For each element  $e_j \in E^C$ ,  $w_j$  is the relative frequency with which  $e_j$  is modified, and it is defined such that  $\sum_{j=1}^{|E^C|} w_j = 1$ . The original, unweighted RW model, discussed in Section 2, computes the expected number of test cases that would be rerun if a single change is made to a program. To do this, the model uses the average number of test cases that cover each covered entity  $e_j \in E^C$ . This average is referred to as  $N_M^C$ . The weighted analogue of  $N_M^C$  is a weighted average,  $WN_M^C$ , which we define as follows:

$$WN_M^C = \sum_{j=1}^{|E^C|} w_j \sum_{i=1}^{|T|} C_{i,j}$$

where  $C_{i,j}$  is defined as before:

$$C_{i,j} = \begin{cases} 1 & \text{if } \text{covers}_M(i,j) \\ 0 & \text{otherwise.} \end{cases}$$

Note that the inner sum represents the total number of test cases covered by  $e_j$ ; multiplying that sum by  $w_j$  provides  $e_j$ 's weighted contribution to the total number of test cases selected overall.

For this weighted average, the fraction of the test suite  $T$  that must be rerun, denoted by  $\Pi_M$ , is given as follows:

$$\Pi_M = \frac{WN_M^C}{|T|}$$

Note that the original, unweighted RW predictor,  $\pi_M$ , is a version of this weighted predictor in which  $w_j$  is  $\frac{1}{|E^C|}$  for all  $e_j$  (which represents an assumption that each entity is equally likely to be changed):

$$\Pi_M = \frac{WN_M^C}{|T|}$$

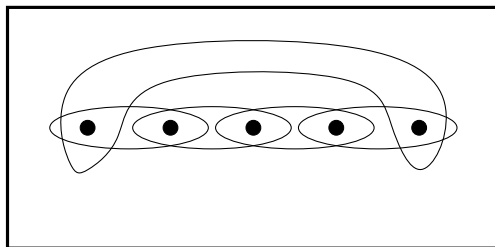


Figure 5: Coverage Pattern A.

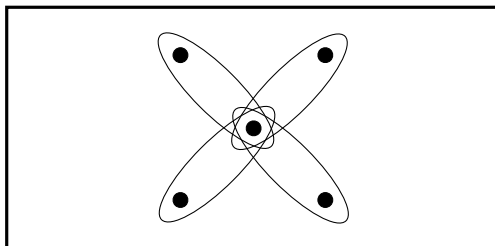


Figure 6: Coverage Pattern B.

$$\begin{aligned}
 &= \frac{\sum_{j=1}^{|E^c|} \frac{1}{|E^c|} \sum_{i=1}^{|T|} C_{i,j}}{|T|} \\
 &= \frac{\sum_{j=1}^{|E^c|} \sum_{i=1}^{|T|} C_{i,j}}{|E^c||T|} \\
 &= \frac{CC}{|E^c||T|} \\
 &= \pi_M
 \end{aligned}$$

To see the impact of the difference between  $\pi_M$  and  $\Pi_M$ , consider coverage patterns *A* and *B*, shown respectively in Figures 5 and 6, where each dot represents an entity and each closed curve a test case. Assume that the patterns are generalized over a large number of entities,  $n$ . As discussed by Rosenblum and Weyuker [23], the value of  $\pi_M$  predicted for each pattern is  $2/n$ . In Pattern *A*, the test cases are distributed evenly over the entities, and thus,  $\Pi_M$  and  $\pi_M$  are the same, and yield the exact number of test cases that would be selected by either `DejaVu` or `TestTube`, regardless of the relative frequency of changes (and hence, regardless of the values assigned to the  $w_j$ ).

In Pattern *B*, the test cases are not distributed evenly over the entities, and in contrast with Pattern *A*, the RW predictor never predicts the exact fraction selected for any changed entity, and it is significantly inaccurate for a change to the “core” element of that pattern. Suppose, however, that instead of assuming that the frequency of change is equal for all entities, we had information about the relative frequency of modifications to individual entities. In this case, using the weighted predictor, we could compute a more

accurate estimate of the fraction of the test suite that would be selected. For example, if we knew that changes are always made to two of the non-core entities (with one changed exactly as often as the other) and that no other entities are ever changed, then the weights would be  $1/2$  for the two changed entities and 0 for all other entities. And thus, we would predict that (for the case of a single entity change)  $1/n$  of the test suite would be selected, rather than  $2/n$  as predicted by the unweighted predictor.

## 5.1 Improved General Prediction

Provided we can obtain values for weights that accurately model the distribution of future modifications to a program, we can use the weighted predictor,  $\Pi_M$ , to improve general prediction. One approach is to utilize change history information about the program, often available from configuration management systems. Assuming that the change histories do accurately model the pattern of future modifications (a result suggested by the work of Harrison and Cook [11]), we can use this information to compute weights for  $\Pi_M$ . If the change histories are recorded at the module level,  $\Pi_M$  can be used as well to predict the percentage of test cases selected on average by a tool, such as `TestTube`, that considers module-level changes to the system. If the change histories are recorded at the statement level,  $\Pi_M$  can be used to predict the percentage of test cases selected on average by a tool, such as `DejaVu`, that considers statement-level changes to the system. In either case, the weighted predictor can be used to incorporate data that may account for change-location information, without performing full change analysis. Thus, it can be used to assess whether it will be worthwhile to perform *all* of the analysis needed by a selective regression testing tool.

In practice, weights may be collected and assumed to be fixed over a number of subsequent versions of a program, or they may be adjusted as change history information becomes available. In this context, an important consideration involves the extent to which weights collected at a particular time in the history of a program can continue to predict values for future versions of that program, and the extent to which the accuracy of predictions based on those weights may decrease over time. Future empirical study of this issue is necessary.

## 5.2 Improved Version-Specific Prediction

We can also use the weighted predictor,  $\Pi_M$ , as a version-specific predictor. For this version-specific predictor, one approach computes the  $w_i$  using the configuration management system. We assign a weight of  $1/k$  to each entity that has been changed (where  $k$  is the total number of entities changed), and we assign a weight of 0 to all other entities in the system. Using these weights,  $\Pi_M$  computes the exact percentage of test cases that will be selected by a test selection tool that selects at the granularity of the entities. For example, if the entities are modules, then  $\Pi_M$  will predict the exact percentage of test cases that will be selected by a test selection tool, such as `TestTube`, that considers changes at the module level. If the entities are statements, then  $\Pi_M$  will predict the exact percentage of test cases that will be selected by a test selection tool, such as `DejaVu`, that considers changes at the statement level. If the cost of determining the *number* of test cases that will be selected is cheaper than the cost of actually *selecting* the test cases, this approach can be cost-effective.

It is worth noting that Rosenblum and Weyuker found, in their experiments with KornShell, that it was typically not necessary to recompute the coverage relation frequently, because it remained very stable over the 31 versions they studied. If this is typical of the system under test, then this should make version-specific

predictors extremely efficient to use and therefore provide valuable information about whether or not the use of a selective regression testing strategy is likely to be cost-effective *for the current version* of the system under test.

An alternative approach assumes that method  $M$  can be supplemented with an additional change analysis capability that is more efficient but less precise than  $M$ 's change analysis. This supplementary change analysis is used during the critical phase of regression testing – after all modifications have been made to create  $P'$ , the new version of  $P$ .<sup>7</sup> The results of the supplementary change analysis can be used to assign weights to the entities in the system, which are then used for prediction as described above.

Using the weighted predictor,  $\Pi_M$ , as a version-specific predictor will be especially appropriate for test suites whose test cases are not evenly distributed across the entities, such as the case illustrated by Pattern  $B$ , where test selection results for specific versions may differ widely from average test selection results over a sequence of versions.

## 6 Conclusions

In this paper we presented results from new empirical studies that were designed to evaluate the effectiveness and accuracy of the Rosenblum-Weyuker (RW) model for predicting cost-effectiveness of a selective regression testing method. The RW model was originally framed solely in terms of code coverage information, and evaluated empirically using the `TestTube` method and a sequence of 31 versions of `KornShell`. In the new studies, two selective regression testing methods were used (`TestTube` and `DejaVu`), and seven different programs were used as subjects. For the experimental subjects we used in the new studies, the original RW model frequently predicted the average, overall effectiveness of the two test selection techniques with an accuracy that we believe is acceptable given that the cost assumptions underlying the RW model are quite realistic for our subjects. However, the predictions of the model occasionally deviated significantly from observed test selection results. Moreover, when this model was applied to the problem of predicting test selection results for particular modified versions of the subject programs, its predictive power decreased substantially, particularly for `DejaVu`. These results suggest that the distribution of modifications made to a program can play a significant role in determining the accuracy of a predictive model of test selection. We therefore conclude that to achieve improved accuracy both in general, and when applied in a version-specific manner, prediction models must account for *both* code coverage *and* modification distribution.

In response to this result, we showed how to extend the Rosenblum-Weyuker predictor to incorporate information on the distribution of modifications. However, to judge the efficacy of this extended predictive model in practice, we require additional experimentation. For this purpose, the subjects used in the studies reported in this paper will not suffice. Rather, we require versions of a program that form a succession of changes over their base versions, as the versions of `KornShell` did. We are currently building a repository of such programs and versions that, when complete, will provide subjects suitable for further empirical investigation of predictive models for regression testing in general, and of our weighted predictor in particular.

Future studies must be directed not only toward further validation of the RW predictor and the improved predictors described in this paper, but toward the development of a more realistic cost model for regression

---

<sup>7</sup>Rothermel and Harrold divide regression testing into two phases for the purpose of cost analysis. During the *preliminary phase*, changes are made to the software, and the new version of the software is built. During the *critical phase*, the new version of the software is tested prior to its release to customers [25].

	Test Suites								
	$T_1$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$	$T_7$	$T_{8-999}$	$T_{1000}$
$\pi_{\text{DejaVu}_j}$	54.8	61.6	53.0	54.0	58.0	55.1	56.0	...	54.8
$S_{\text{DejaVu}_{i,j}}$									
Version 1	58.3	81.8	62.5	60.0	84.6	62.5	72.7	...	50.0
Version 2	8.3	9.1	12.5	10.0	7.7	12.5	9.1	...	7.1
Version 3	16.7	18.2	25.0	10.0	23.1	12.5	18.2	...	21.4
Version 4	41.7	18.2	25.0	30.0	38.5	37.5	36.4	...	35.7
Version 5	8.3	9.1	12.5	10.0	7.7	12.5	9.1	...	7.1
Version 6	16.7	45.5	25.0	20.0	30.8	25.0	18.2	...	14.3
Version 7	41.7	18.2	25.0	30.0	38.5	37.5	36.4	...	35.7
Version 8	66.7	81.8	62.5	70.0	76.9	75.0	81.8	...	71.4
Version 9	41.7	18.2	25.0	30.0	38.5	37.5	36.4	...	35.7
Version 10	33.3	54.5	50.0	40.0	23.1	50.0	36.4	...	28.6
$S_{\text{DejaVu}_j}$	33.3	35.5	32.5	31.0	36.9	36.2	35.5	...	30.7
$D_{\text{DejaVu}_j}$	21.5	26.1	20.5	23.0	21.1	18.9	20.5	...	24.1

Table 2: Partial data used in the computation of graphs in Figure 1.

testing. This will require extensive field studies of existing large systems in order to create a better picture of the different factors driving cost-effectiveness, such as test suite size, test case execution times, testing personnel costs, and the availability of spare machine cycles for regression testing.

## 7 Acknowledgments

This work was supported in part by a grant from Microsoft, Inc., by NSF under NYI award CCR-9696157 to Ohio State University, CAREER award CCR-9703108 to Oregon State University, and Experimental Software Systems award CCR-9707792 to Ohio State University and Oregon State University, and by an Ohio State University Research Foundation Seed Grant. Thanks to Huiyu Wang who performed some of the experiments, to Panickos Palletas and Qiang Wang who advised us on the statistical analysis, to Jim Jones who performed some of the statistical analysis, and to Monica Hutchins and Tom Ostrand of Siemens Corporate Research for supplying the subject programs and other data necessary for the experimentation.

## 8 Appendix: Details of the Computation of Data for Figures 1 and 3

To compute the data used for the graphs in Figure 1, we used a procedure described in Section 3.3.1. As further explanation, we give details of the computation of that data for one subject program, `printtokens2`.

For our experiments, we used 1000 coverage-based test suites,  $T_1, \dots, T_{1000}$ . Table 2 shows data for a subset of these test suites:  $T_1 - T_7$ , and  $T_{1000}$ . For each test suite  $T_j$ , we used the RW predictor to predict the number of test cases that would be selected by `DejaVu` when an arbitrary change is made to `printtokens2`. We then used this number to determine  $\pi_{\text{DejaVu}_j}$ , the percentage of test cases that the RW predictor predicts will be selected by `DejaVu` when an arbitrary change is made to `printtokens2`. The first row of Table 2 gives these percentages for  $T_1 - T_7$  and  $T_{1000}$ .

We had ten versions of `printtokens2` (see Table 1). We next ran DeJaVu on these ten versions, with each of the 1000 test suites, and, for each version  $i$  and test suite  $j$ , recorded the number of test cases selected. We then used this number to compute, for each  $i$  and  $j$ ,  $S_{\text{DeJaVu}_{i,j}}$ , the percentage of test cases selected. The ten rows for Versions 1 - 10 in Table 2 give these percentages. For example, from the table, we can see that, for  $T_1$ ,  $S_{\text{DeJaVu}_{i,1}}$  ranges from 8.3% to 66.7%. Using Equation (1), we then computed the  $\bar{S}_{\text{DeJaVu}_j}$  for each test suite  $T_j$ . Table 2 gives these percentages for each  $T_j$ .

We then used Equation (3) to compute, for each  $T_j$ , the difference between the percentage predicted by the RW predictor and the average percentage selected by DeJaVu (i.e.,  $\bar{D}_{\text{DeJaVu}_j}$ ). Table 2 shows that, for  $T_1 - T_7$  and  $T_{1000}$ , the  $\bar{D}_{\text{DeJaVu}_j}$  range from 18.9% to 26.1%.

Finally, we created the set,  $\bar{H}_{\text{DeJaVu}}$  (Equation (5)). The ordered pairs in this set are obtained by first rounding the percentages of the  $\bar{D}_{\text{DeJaVu}_j}$ , then determining the number of those rounded percentages that have range value  $-100 \leq r \leq 100$ , and then determining the percentage of those percentages that occur for each value of  $r$ . Thus, for `printtokens2`,  $\bar{D}_{\text{DeJaVu}_6}$  rounds to 19,  $\bar{D}_{\text{DeJaVu}_3}$ ,  $\bar{D}_{\text{DeJaVu}_5}$ , and  $\bar{D}_{\text{DeJaVu}_7}$  round to 21,  $\bar{D}_{\text{DeJaVu}_1}$  rounds to 22,  $\bar{D}_{\text{DeJaVu}_4}$  rounds to 23,  $\bar{D}_{\text{DeJaVu}_{1000}}$  rounds to 24, and  $\bar{D}_{\text{DeJaVu}_2}$  rounds to 26. Thus, there will be ordered pairs in  $\bar{H}_{\text{DeJaVu}}$  with first coordinates 19, 21, 22, 21, 24, and 26, and the number of rounded percentages for  $T_1 \dots T_{1000}$  are used to compute the percentage of times (among the 1000 test suites) each percentage occurs, which is then used in the computation of the second coordinates of the ordered pairs. We used these ordered pairs to plot the solid curve for `printtokens2` in Figure 1.

We used a similar approach to obtain the data for Figure 3 except that we did not compute the averages of the deviations. To compute the data used for the graphs in Figure 3, we used a procedure described in Section 3.3.2. As further explanation, we give details of the computation of that data for one subject program, `printtokens2`.

Table 3 shows data for a subset of the 1000 coverage-based test suites:  $T_1 - T_7$  and  $T_{1000}$ . For each test suite  $T_i$ , we used the RW predictor to predict the number of test cases that would be selected by DeJaVu when an arbitrary changes is made to `printtokens2`. We then used this number to determine  $\pi_{\text{DeJaVu}_j}$ , the percentage of test cases that the RW predictor predicts will be selected by DeJaVu when an arbitrary change is made to `printtokens2`. The first row of Table 3 gives these percentages for  $T_1 - T_7$  and  $T_{1000}$ .

Next, we ran DeJaVu on the ten versions of `printtokens2`, and, for each version, recorded the number of test cases selected. We then used this number to compute  $S_{\text{DeJaVu}_{i,j}}$ ,  $j = 1 \dots 10$ , the percentage of test cases selected. The ten rows for Versions 1 - 10 in Table 3 give these percentages.

We then used Equation (7) to compute, for each version  $i$  and each  $T_j$ ,  $D_{\text{DeJaVu}_{i,j}}$ , the difference between the percentage predicted by the RW predictor and the percentage selected by DeJaVu. Table 3 shows that, for  $T_1 - T_7$  and  $T_{1000}$ , these percentages range from -26.6% to 52.5%.

Finally, we created the set,  $H_{\text{DeJaVu}}$  (Equation (9)). These ordered pairs are obtained by first rounding the percentages of the  $D_{\text{DeJaVu}_{i,j}}$ , determining the number of those rounded percentages that have range value  $-100 \leq r \leq 100$ , and then determining the percentage of those percentages that occur for each value of  $r$ . For example, for `printtokens2`,  $D_{\text{DeJaVu}_{1,2}}$ ,  $D_{\text{DeJaVu}_{8,2}}$ , and  $D_{\text{DeJaVu}_{8,6}}$  round to -20 and  $D_{\text{DeJaVu}_{4,5}}$ ,  $D_{\text{DeJaVu}_{7,5}}$ ,  $D_{\text{DeJaVu}_{9,5}}$ ,  $D_{\text{DeJaVu}_{4,7}}$ ,  $D_{\text{DeJaVu}_{7,7}}$ ,  $D_{\text{DeJaVu}_{9,7}}$ , and  $D_{\text{DeJaVu}_{10,7}}$  round to 20. Thus,  $H_{\text{DeJaVu}}$  contains ordered pairs with -20 and 20 as the first coordinates, and the number of rounded percentages for  $T_1 \dots T_{1000}$  are used to compute the percentage of times (among the 1000 \* 10 test-suite/version pairs) each percentage occurs, and used in the computation of the second coordinates of the ordered pairs.

	Test Suites								
	$T_1$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$	$T_7$	$T_{8-999}$	$T_{1000}$
$\pi_{\text{DejaVu}_j}$	54.8	61.6	53.0	54.0	58.0	55.1	56.0	...	54.8
$S_{\text{DejaVu}_{i,j}}$									
Version 1	58.3	81.8	62.5	60.0	84.6	62.5	72.7	...	50.0
Version 2	8.3	9.1	12.5	10.0	7.7	12.5	9.1	...	7.1
Version 3	16.7	18.2	25.0	10.0	23.1	12.5	18.2	...	21.4
Version 4	41.7	18.2	25.0	30.0	38.5	37.5	36.4	...	35.7
Version 5	8.3	9.1	12.5	10.0	7.7	12.5	9.1	...	7.1
Version 6	16.7	45.5	25.0	20.0	30.8	25.0	18.2	...	14.3
Version 7	41.7	18.2	25.0	30.0	38.5	37.5	36.4	...	35.7
Version 8	66.7	81.8	62.5	70.0	76.9	75.0	81.8	...	71.4
Version 9	41.7	18.2	25.0	30.0	38.5	37.5	36.4	...	35.7
Version 10	33.3	54.5	50.0	40.0	23.1	50.0	36.4	...	28.6
$D_{\text{DejaVu}_{i,j}}$									
Version 1	-3.5	-20.2	-9.5	-6.0	-26.6	-7.4	-16.9	...	4.8
Version 2	46.5	52.5	40.5	44.0	50.3	42.6	46.9	...	47.7
Version 3	38.1	43.4	28.0	44.0	34.9	42.6	37.8	...	33.4
Version 4	13.1	43.4	28.0	24.0	19.5	17.6	19.6	...	19.1
Version 5	46.5	52.5	40.5	44.0	50.3	42.6	46.9	...	47.7
Version 6	38.1	16.1	28.0	34.0	27.2	30.1	37.8	...	40.5
Version 7	13.1	43.4	28.0	24.0	19.5	17.6	19.6	...	19.1
Version 8	-11.9	-20.2	-9.5	-16.0	-18.9	-19.9	-25.8	...	-16.6
Version 9	13.1	43.4	28.0	24.0	19.5	17.6	19.6	...	19.1
Version 10	21.5	7.1	3.0	14.0	34.9	5.1	19.6	...	26.2

Table 3: Partial data used in the computation of graphs in Figure 3.

We used this procedure to obtain the data for the rest of the graphs in Figure 3 for DeJaVu, and used a similar procedure to obtain the data for the graphs for TestTube.

## References

- [1] H. Agrawal, J. Horgan, E. Krauser, and S. London. Incremental regression testing. In *Proceedings of the Conference on Software Maintenance - 1993*, pages 348–357, September 1993.
- [2] M. Balcer, W. Hasling, and T. Ostrand. Automatic generation of test scripts from formal test specifications. In *Proceedings of the Third Symposium on Software Testing, Analysis, and Verification*, pages 210–218, December 1989.
- [3] T. Ball. On the limit of control flow analysis for regression test selection. In *Proceedings of the 1998 International Symposium on Software Testing and Analysis*, March 1998.
- [4] S. Bates and S. Horwitz. Incremental program testing using program dependence graphs. In *Proceedings of the 20th ACM Symposium on Principles of Programming Languages*, January 1993.
- [5] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, New York, NY, 1990.
- [6] P. Benedusi, A. Cimitile, and U. De Carlini. Post-maintenance testing based on path change analysis. In *Proceedings of the Conference on Software Maintenance - 1988*, pages 352–361, October 1988.
- [7] D. Binkley. Semantics guided regression test cost reduction. *IEEE Transactions on Software Engineering*, 23(9), August 1997.

- [8] 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.
- [9] K.F. Fischer, F. Raji, and A. Chruscicki. A methodology for retesting modified software. In *Proceedings of the National Telecommunications Conference B-6-3*, pages 1–6, November 1981.
- [10] R. Gupta, M.J. Harrold, and M.L. Soffa. An approach to regression testing using slicing. In *Proceedings of the Conference on Software Maintenance - 1992*, pages 299–308, November 1992.
- [11] W. Harrison and C. Cook. Insights on improving the maintenance process through software measurement. In *Proceedings of the Conference on Software Maintenance - 1990*, pages 37–45, November 1990.
- [12] M.J. Harrold and M.L. Soffa. An incremental approach to unit testing during maintenance. In *Proceedings of the Conference on Software Maintenance - 1988*, pages 362–367, October 1988.
- [13] J. Hartmann and D.J. Robson. Techniques for selective revalidation. *IEEE Software*, 16(1):31–38, January 1990.
- [14] 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.
- [15] J. Laski and W. Szermer. Identification of program modifications and its applications in software maintenance. In *Proceedings of the Conference on Software Maintenance - 1992*, pages 282–290, November 1992.
- [16] J.A.N. Lee and X. He. A methodology for test selection. *The Journal of Systems and Software*, 13(1):177–185, September 1990.
- [17] H.K.N. Leung and L.J. White. Insights into regression testing. In *Proceedings of the Conference on Software Maintenance - 1989*, pages 60–69, October 1989.
- [18] H.K.N. Leung and L.J. White. A study of integration testing and software regression at the integration level. In *Proceedings of the Conference on Software Maintenance - 1990*, pages 290–300, November 1990.
- [19] H.K.N. Leung and L.J. White. A cost model to compare regression test strategies. In *Proceedings of the Conference on Software Maintenance - 1991*, pages 201–208, October 1991.
- [20] T.J. Ostrand and M.J. Balcer. The category-partition method for specifying and generating functional tests. *Communications of the ACM*, 31(6), June 1988.
- [21] T.J. Ostrand and E.J. Weyuker. Using dataflow analysis for regression testing. In *Sixth Annual Pacific Northwest Software Quality Conference*, pages 233–247, September 1988.
- [22] D. Rosenblum and G. Rothermel. An empirical comparison of regression test selection techniques. In *Proceedings of the International Workshop for Empirical Studies of Software Maintenance*, pages 89–94, October 1997.
- [23] D. S. 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.
- [24] G. Rothermel and M. J. Harrold. Empirical studies of a safe regression test selection technique. *IEEE Transactions on Software Engineering*, 24(6):401–419, June 1998.
- [25] G. Rothermel and M.J. Harrold. Analyzing regression test selection techniques. *IEEE Transactions on Software Engineering*, 22(8):529–551, August 1996.
- [26] 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 1997.

- [27] B. Sherlund and B. Korel. Logical modification oriented software testing. In *Proceedings: Twelfth International Conference on Testing Computer Software*, June 1995.
- [28] A.B. Taha, S.M. Thebaut, and S.S. Liu. An approach to software fault localization and revalidation based on incremental data flow analysis. In *Proceedings of the 13th Annual International Computer Software and Applications Conference*, pages 527–534, September 1989.
- [29] F. Vokolos and P. Frankl. Pythia: A regression test selection tool based on textual differencing. In *ENCRESS '97, Third International Conference on Reliability, Quality, and Safety of Software Intensive Systems*, May 1997.
- [30] S.S. Yau and Z. Kishimoto. A method for revalidating modified programs in the maintenance phase. In *COMPSAC '87: The Eleventh Annual International Computer Software and Applications Conference*, pages 272–277, October 1987.