University of Nebraska - Lincoln

DigitalCommons@University of Nebraska - Lincoln

CSE Conference and Workshop Papers

Computer Science and Engineering, Department of

4-2001

Understanding and Measuring the Sources of Variation in the Prioritization of Regression Test Suites

Sebastian Elbaum
University of Nebraska-Lincoln, selbaum@virginia.edu

David Gable
University of Nebraska-Lincoln, dgable@cse.unl.edu

Gregg Rothermel
Oregon State University, gerother@ncsu.edu

Follow this and additional works at: https://digitalcommons.unl.edu/cseconfwork

Part of the Computer Sciences Commons

Understanding and Measuring the Sources of Variation in the Prioritization of Regression Test Suites

Sebastian Elbaum
CSE Department
University of Nebraska
Lincoln, Nebraska
elbaum@cse.unl.edu

David Gable
CSE Department
University of Nebraska
Lincoln, Nebraska
dgable@cse.unl.edu

Gregg Rothermel
CS Department
Oregon State University
Corvallis, Oregon
grother@cs.orst.edu

Abstract

Test case prioritization techniques let testers order their test cases so that those with higher priority, according to some criterion, are executed earlier than those with lower priority. In previous work, we examined a variety of prioritization techniques to determine their ability to improve the rate of fault detection of test suites. Our studies showed that the rate of fault detection of test suites could be significantly improved by using more powerful prioritization techniques. In addition, they indicated that rate of fault detection was closely associated with the target program. We also observed a large quantity of unexplained variance, indicating that other factors must be affecting prioritization effectiveness. These observations motivate the following research questions: (1) Are there factors other than the target program and the prioritization technique that consistently affect the rate of fault detection of test suites? (2) What metrics are most representative of each factor? (3) Can the consideration of additional factors lead to more efficient prioritization techniques? To address these questions, we performed a series of experiments exploring three factors: program structure, test suite composition, and change characteristics. This paper reports the results and implications of those experiments.

1. Introduction

Test suite reuse is a common practice during regression testing [15]. Testers often reuse test suites by running all the test cases in those suites, which can require significant effort. Test case prioritization techniques [4, 17, 20] assist with test suite reuse by helping testers order their test cases such that those with higher priority, according to some criterion, are executed earlier than those with lower priority.

Test case prioritization can have several goals; one potential goal is that of increasing a test suite's *rate of fault detection* -- a measure of how quickly the test suite detects faults [17]. An improved rate of fault detection can provide earlier feedback on the system under regression test, and let developers begin debugging earlier than might otherwise be possible. It can also increase the likelihood that if testing is prematurely terminated, those test cases that offer the greatest fault detection ability in the available testing time will have been executed. In test cycles that are sufficiently long or expensive, such gains can be advantageous.

In previous studies [4, 6, 17, 18], we examined the abilities of several test case prioritization techniques to improve the rate of fault detection of test suites. The techniques that we investigated prioritized test suites based on various metrics, including code coverage, fault likelihood, and fault exposure potential. We showed that each technique could significantly improve the rate of fault detection of test suites during regression testing, and we determined which techniques were the most successful. Our experiments also showed that there was significant statistical evidence of an association between rate of fault detection and program under test: different programs offer different opportunities for test case prioritization.

These results were encouraging; however, the analysis of our data also showed that the target program and the prioritization technique explained only part of the observed variation in prioritization success (rate of fault detection achieved). A better understanding of the factors that influence the success of prioritization techniques could help both with the creation of new techniques, and the development of criteria for selecting appropriate techniques in particular circumstances. This motivates the search for other factors that affect the rate of fault detection of test suites. We hypothesize that, in addition to technique and program, there are at least two other factors affecting test case prioritization: change

characteristics and test suite composition. In our search of the literature, however, we could discover no previous work investigating the sources of variation affecting test case prioritization. In this work, we selected a set of metrics to account for these quantitative factors, and conducted an empirical study investigating the value of the chosen metrics in explaining the sources of additional variation.

2. Empirical Study

Our goal is to understand and quantify the sources of variation involved in the prioritization of test suites. To address this goal, we designed an empirical study that allowed us to manipulate and measure various potential sources of variation and prioritization techniques.

Presentation of the study is structured in four sections. This section presents our dependent and independent variables, including the list of metrics selected. Section 3 provides a summary of the collected data. Section 4 presents a principal components analysis performed on the collected metrics to understand the problem dimensionality, the information provided by each variable, and the relationships among variables. Section 5 presents the analysis of the value of the variables as predictors of the rate of fault detection through univariate and multiple regression analysis.

2.1. Dependent Variable

In previous work [17], we defined a measure with which to quantify and compare the rates of fault detection of test suites, called APFD (average percentage of faults detected); this measure serves as our sole dependent variable. APFD measures the average cumulative percentage of faults detected over the course of executing the test cases in the test suite in a given order. APFD values range from 0 to 100 percent: a higher APFD value means that faults were found by test cases occurring earlier in the order, and a lower APFD value means that faults were found by test cases occurring later in the order.

For illustration, consider a program with 10 faults (1-10), and a test suite of 5 test cases (A-E), with fault detecting abilities as shown in Figure 1.A.

Suppose we place the test cases in order A-B-C-D-E to form a prioritized test suite T1. Figure 4.B shows the percentage of detected faults versus the fraction of the test suite T1 used. After running test case A, two of the ten faults are detected; thus 20% of the faults have been detected after 0.2 of the test suite T1 has been used. After running test case B, two more faults are detected and thus 40% of the faults have been detected after 0.4 of the test suite has been used. In Figure 4.B, the area inside the inscribed rectangles (dashed boxes) represents the weighted percentage of the faults detected over the corresponding fraction of the test suite. The solid lines connecting the corners of the inscribed interpolate the gain in the percentage of detected faults. The area under the curve thus represents the weighted average of the percentage of faults detected over the life of the test suite. This area is the prioritized test suite's average percentage faults detected measure (APFD); the APFD is 50% in this example.

Figure 4.C reflects what happens when the order of test cases is changed to E-D-C-A-B, yielding test suite T2, a "faster detecting" suite than T1 with APFD 66%. Figure 4.D shows the effects of using a prioritized test suite T3 whose test case ordering is C-E-D-A-B. By inspection, it is clear that this ordering results in the earliest detection of the most faults and illustrates an optimal ordering, with APFD 78%.

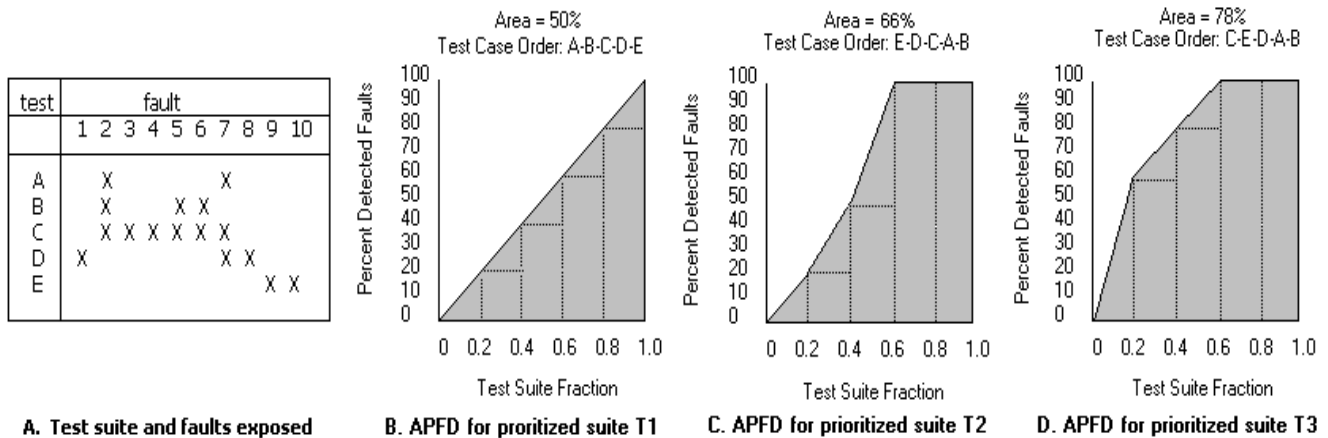Note that as we have defined it, APFD can be used

A. Test suite and faults exposed   B. APFD for proritized suite T1   C. APFD for prioritized suite T2   D. APFD for prioritized suite T3

Figure 1

only in cases in which it can be determined which faults are revealed by which test cases, as in controlled studies.

2.2. Independent Variables

The independent variables in our study are the subject programs, the prioritization techniques, the changes in the program introduced in a specific version, and the test suite characteristics.

The following sections expand the definition of each independent variable, relating the metrics used to characterize that independent variable within the context of this empirical study.

Subject programs. To obtain greater confidence that our results are not dependent on the target program, we

Table 1. Program metrics

Metric	Description
PRG.PROG_S	Number of new line characters in the source code
PRG.N_EXECST	Number of executable statements
PRG.AN_PATHS	Mean number of paths[1] in the control flow graph of a function over all functions
PRG.N_FUNCT	Number of functions in the program
PRG.A_FSIZE	Mean function size across all functions
PRG.S_FSIZE	Standard deviation of function size across all functions
PRG.A_FFAN	Mean function fan out over all functions

Table 2. Change metrics

Metric	Description
CHG.N_FUN_CH	Number of functions with at least one changed statement
CHG.P_FCH	Percentage of functions with at least one changed statement
CHG.P_SCH	Percentage of statements changed
CHG.N_INS	Number of statements inserted
CHG.N_DEL	Number of statements deleted
CHG.N_TOT_CH	Number of statements changed computed by using CHG.N_INS + D_DEL
CHG.P_GLOCOC	Percentage of global changes over the total number of changes
CHG.N_GLO_CH	Number of global changes measured by counting the changed statements that occur outside functions
CHG.AN_CHMOM	Mean number of changed functions over all functions

analyzed eight different programs written in C^2. Researchers at Siemens Corporate Research provided seven of these programs [9]; the eighth program is an application developed by the European Space Agency. For each subject program, we captured program characteristics using a set of metric tools used in previous measurement research efforts [2, 3]. Table 1 describes the metrics used.[3]

Versions and Changes. Each of the eight programs has a baseline version, and twenty-nine versions each containing multiple faults. By studying many versions containing varied types of changes within each program, we gain confidence that our analysis and results will not be dependent on the nature of the changes. The characteristics of the changes among versions were computed using syntactic differencing. Following the procedure detailed in [19], we used the UNIX program **diff** with the "unified" option flag to show which lines had been inserted into or deleted from the baseline version.

Table 2 presents a list of the metrics used to measure the distribution and characteristics of changes in the program versions. A "change" consists of one statement inserted into or deleted from the baseline version.[4]

Test Suites. For each baseline program, a large pool of test cases was available. For the seven Siemens programs, the Siemens researchers had created these pools in two stages. First, they created initial pools of black-box test cases using the category partition method and TSL tool [16]. They then augmented each pool with manually-created white-box test cases to ensure that each exercisable statement, edge, and definition-use pair in the base program or its control flow graph was exercised by at least 30 test cases. For the Space program, the initial pool consisted of 10,000 randomly generated test cases created by Vokolos and Frankl [19], augmented by additional white-box test cases sufficient to achieve coverage of each branch by at least 30 test cases [17].

We used these test pools as a source for test suites, creating 1000 branch-adequate suites for each program by randomly selecting test cases from these suites and adding them to the test suite if they added coverage, and continuing until complete branch coverage had been achieved. Duplicate test suites were discarded. We then

---

[1] Measured as a bounded inter-procedural loop free paths.

[2] For further details about the programs, versions, and tests suites see [17, 4].

[3] In some cases, to capture certain attributes for high-level entities we derived quantities from some lower level entities. For example, to capture program structure (a high-level entity) we computed the average function (low-level entity) size for that program, and assumed that the average function size has a random distribution across programs.

[4] The values reported by **diff** were transformed to just "insertions" and "deletions" to ensure their consistency.

selected, for each subject program, 50 of its associated test suites for use in this study.

We used the Aristotle program analysis system [8] to generate test coverage data for each of the subject programs and test cases. Additional library routines from the Aristotle system and newly developed scripts were used to match executed statements with the changed lines of code reported by the UNIX **diff** tool discussed above. Table 3 presents a list of the metrics used to quantify the test suite characteristics.

Prioritization Techniques. For this study we selected a varied subset of the prioritization techniques used in previous studies. We next discuss each technique briefly; further details can be found in [4].

Total function coverage (tc-f). By instrumenting a program we can determine, for any test case, the number of functions in that program that were exercised by that test case. We can prioritize these test cases according to the total number of functions they cover simply by sorting them in order of total function coverage achieved.

Total statement coverage (tc-s). Analogous to total function coverage prioritization but operating at the level of statements, this technique prioritizes test cases according to the total number of statements they execute.

Table 3. Test metrics

Metric	Description
TS.SUITE_S	Number of tests in the test suite
TS.P_TRCHF	Percentage of tests reaching a function that contains a change
TS.AN_CHFET	Mean number of changed functions executed by a test over a test suite
TS.SN_CHFET	Standard deviation of number of changed functions executed by a test over a test suite
TS.AN_CHSET	Mean number of changed statements executed by a test over a test suite
TS.SN_CHSET	Standard deviation of number of changed statements executed by a test over a test suite
TS.AN_FET	Mean number of functions executed by a test over a test suite
TS.SN_FET	Standard deviation of the number of functions executed by a test over a test suite
TS.AP_FET	Mean percentage of functions executed by a test over a test suite
TS.SP_FET	Standard deviation of the percentage of functions executed by a test over a test suite
TS.AP_SET	Mean percentage of statements executed by a test over a test suite
TS.SP_SET	Standard deviation of the percentage of statements executed by a test over a test suite
TS.AN_STET	Mean number of statements executed by a test over a test suite
TS.SN_STET	Standard deviation of number of statements executed by a test over a test suite

Additional function coverage (ac-f). Additional function coverage prioritization greedily selects a test case that yields the greatest function coverage, then adjusts the coverage data about subsequent test cases to indicate their coverage of functions not yet covered, and then repeats this process, until each function covered by at least one test case has been covered.

Additional statement coverage (ac-s). Analogous to additional function coverage prioritization but operating at the level of statements, this technique prioritizes test cases (greedily) according to the total number of additional statements they cover.

Total fault index (fi-t). Certain functions are more likely to contain faults than others. This fault proneness can be associated with measurable software attributes [1, 10, 12, 13]. Fault index prioritization attempts to take advantage of this association by prioritizing test cases based on the execution of fault prone functions. To represent fault proneness, we use a fault index based on principal component analysis [2, 14, 3]. Prioritization computes the sum of the fault indexes for each function each test case executes, and then sorts the test cases in decreasing order of these sums.

Additional fault-index (fi-a). Additional fault index prioritization is accomplished in a manner similar to additional function coverage prioritization. The set of functions that have been covered by previously executed test cases is maintained. If this set contains all functions (no test case adds anything to this coverage) the set is reinitialized to empty. To find the next best test case we compute, for each test case, the sum of the fault indexes for each function that test case executes, except for functions in the set of covered functions. The test case for which this sum is the greatest wins. This process is repeated until all test cases have been prioritized.

Optimal (optimal). As an experimental control, we consider an optimal ordering of the test cases in the test suite. We can obtain such an ordering in our experiments because we utilize programs with known faults and know which faults each test case exposes: this lets us determine the ordering of test cases that maximizes a test suite's rate of fault detection. In practice, of course, this is not a practical technique, but it provides an upper bound on the effectiveness of the other heuristics.

3. Data Collected

To capture suspected sources of variation we collected various data. In this section, we use descriptive statistics to present that data. Note that the original data did not constitute a normal distribution (it was skewed toward high APFD values) and it presented non-homogeneous

Table 4. Summary program metrics

Metrics	Programs								Mean	Std Dev
	replace	sched	sched2	space	tcas	tokens	tokens2	tot_inf		
PRG.N_EXECST	160	98	95	3152	48	171	122	64	488.75	1076.96
PRG.PROG_S	563	412	307	9126	173	563	510	406	1507.50	3081.19
PRG.N_FUNCT	21	18	16	136	9	18	19	7	30.50	42.91
PRG.A_FFAN	2.33	1.83	2.19	4.10	2.00	2.61	1.89	1.29	2.28	0.83
PRG.A_FSIZE	12.10	7.78	8.44	26.18	6.56	11.72	8.37	15.57	12.09	6.40
PRG.S_FSIZE	8.22	6.99	5.53	25.56	7.56	13.06	7.08	14.81	11.10	6.67
PRG.AN_PATHS	7.24	3.06	6.75	15.80	1.78	5.61	30.89	47.29	14.80	16.17

Table 5. Summary change metrics

Metrics	Mean	Std Dev.	Min	Max
CHG.P_FCH	16%	12%	0%	57%
CHG.P_SCH	2%	2%	0%	17%
CHG.N_INS	5.47	9.44	0	73
CHG.N_DEL	5.63	9.97	0	73
CHG.N_TOT_CH	11.10	19.30	1	146
CHG.P_GLOCOC	12%	24%	0%	100%
CHG.N_GLO_CH	1.18	2.89	0	12
CHG.AN_CHMOM	0.43	0.35	0.00	2.89

Table 6. Summary test metrics

Metrics	Mean	Std Dev.	Min	Max
TS.SUITE_S	21.14	38.04	50	166
TS.P_TRCHF	63%	35%	0%	100%
TS.SN_CHFET	0.67	0.39	0.00	2.46
TS.SN_CHSET	0.86	0.92	0.00	9.17
TS.AN_CHFET	1.05	0.80	0.00	5.58
TS.AN_CHSET	1.30	1.49	0.00	12.81
TS.AN_STET	146.68	226.74	32.20	971.49
TS.SN_STET	109.64	251.18	13.89	1060.21
TS.AN_FET	14.80	12.14	3.75	57.68
TS.SN_FET	6.27	4.35	1.60	21.42
TS.AP_SET	48%	8%	28%	66%
TS.SP_SET	22%	6%	9%	32%
TS.AP_FET	70%	10%	40%	86%
TS.SP_FET	31%	6%	15%	42%

variance. To address this problem, we performed a random sampling and assignment on APFD values to help distribute the idiosyncratic characteristics of the subjects and avoid biasing the outcome of the study.

Program Metrics. Table 4 summarizes the data values collected to characterize the subject programs. Space is clearly the largest program as measured by most of these metrics. For the other programs, all the metrics are substantially smaller, with the noticeable exception of average number of paths per function.

Change Characteristic Factor Metrics. Table 5 summarizes the metrics collected to characterize the changes made to a program in a particular version. For each program, change metrics were computed by comparing the baseline version of the program with each of the versions. The percentage of change was relatively small. On average, CHG.N_INS and CHG.N_DEL per version are about 5 statements, while the number of global changes was slightly over 1.

Test Suite Composition. Table 6 summarizes the test suite composition metrics. On average, about half of all statements are executed by each test case (TS.AP_SET). The number of changed statements executed by each test case (TS.AN_CHSET) is smaller, which is consistent with the small amount of change present in most versions. TS.P_TRCHF at 63% suggests that a large percentage of test cases are useful for validating the changes.

4. Understanding the dimensionality of the problem

The data collection process targeted 32 metrics from three different domains: program characteristics, test suite composition and change attributes. Although each of these metrics plays a role in explaining the variability in the collected data, a smaller set of metrics suffices to capture most of the observed variance. To find a smaller set of these metrics representing the true dimensionality of the data we employed principal component analysis [7,

11]. This classical multivariate analysis technique reduces the dimensionality of the data to its principal orthogonal components. (The mechanisms for generating principal components are outside the scope of this paper; it suffices to understand that principal component analysis groups variables such that all the variables within a group (component) are highly correlated but have small correlations with variables in other components. The result is a series of orthogonal components representing an underlying common attribute domain.). The first step in this process is to determine how many orthogonal sources of variation are really being measured y the set of metrics. Principal component analysis identified six principal components (PCs) from our set of metrics. This component structure contradicted some of

our expectations and indicated that some other sources of information may be important to predicting APFD.

To clarify the component structure we used a varimax rotation [11], which maximizes the metrics correlation coefficients, also known as variable loadings. Table 7 presents the component structure indicating the greater loadings by coloring the corresponding cells.

After performing the principal component analysis, we proceeded to interpret the components and the metrics loadings. PC1 explains almost half of the observed variation. There are two interesting findings within PC1.

First, most "program characteristics" (all except PRG.AN_PATHS) and several "test suite composition" metrics (TS.AN_FET, TS.SN_FET, TS.SUITE_S, TS.AN_STET, and TS.SN_STET) were both strongly

Table 7. Principal component analysis

Metrics	PC 1	PC 2	PC 3	PC 4	PC 5	PC 6
TS.SUITE_S	0.963	0.087	0.220	-0.048	-0.044	0.019
PRG.N_EXECST	0.952	0.107	0.226	-0.051	-0.070	0.019
PRG.N_FUNCT	0.951	0.104	0.226	-0.073	-0.102	-0.058
TS.AN_STET	0.951	0.109	0.224	-0.062	-0.085	0.041
PRG.PROG_S	0.950	0.112	0.226	-0.059	-0.072	0.038
TS.SN_STET	0.944	0.118	0.229	-0.055	-0.091	0.026
TS.AN_FET	0.942	0.062	0.214	-0.091	-0.123	-0.118
TS.SN_FET	0.896	0.118	0.230	-0.124	-0.188	-0.200
PRG.A_FFAN	0.895	-0.089	0.167	0.038	0.086	-0.343
PRG.A_FSIZE	0.831	0.107	0.161	-0.041	0.130	0.447
PRG.S_FSIZE	0.799	0.067	0.158	-0.079	0.227	0.438
TS.AP_FET	-0.732	-0.254	-0.222	0.055	0.026	0.127
TS.SP_SET	-0.750	0.224	-0.056	-0.099	-0.263	-0.273
TS.AP_SET	-0.778	0.269	-0.112	-0.039	-0.107	0.105
TS.SP_FET	-0.821	0.057	-0.079	-0.002	-0.172	-0.148
TS.P_TRCHF	-0.067	0.941	0.053	0.050	-0.115	0.009
TS.AN_CHFET	0.108	0.905	0.141	0.261	-0.045	-0.004
TS.AN_CHSET	0.366	0.700	0.235	0.177	0.052	-0.026
TS.SN_CHFET	0.071	0.662	0.187	0.406	-0.096	-0.082
CHG.N_INS	0.497	0.130	0.824	0.161	0.074	-0.015
CHG.N_TOT_CH	0.518	0.134	0.819	0.142	0.058	-0.003
TS.SN_CHSET	0.310	0.299	0.807	0.175	-0.048	-0.019
CHG.N_DEL	0.532	0.137	0.805	0.123	0.043	0.009
CHG.AN_CHMOM	-0.123	0.321	0.338	0.807	0.000	0.162
CHG.P_SCH	-0.207	0.135	0.148	0.806	0.261	-0.252
CHG.P_FCH	-0.353	0.341	-0.062	0.729	-0.065	0.342
CHG.N_FUN_CH	0.398	0.453	0.171	0.575	-0.183	-0.105
CHG.N_GLO_CH	0.033	-0.052	0.055	0.142	0.924	-0.096
CHG.P_GLOCOC	-0.052	-0.216	-0.003	-0.082	0.902	0.073
PRG.AN_PATHS	-0.003	-0.070	-0.029	0.028	-0.048	0.943
Eigenvalues	13.031	4.458	3.491	2.618	2.105	1.823
% Variance	47%	16%	13%	10%	8%	7%
Accumulated	47%	63%	76%	86%	94%	100%

associated with PC1, even though we had expected these two classes of metrics to quantify different domains. This metric loading suggests that as program size increases, the expected number of test cases in the associated test suite will also increase, and also the average number of entities (statements or functions) executed by each test case will increase. The second interesting issue is that four of the "test suite composition" metrics (TS.SP_SET, TS.AP_FET, TS.AP_SET, and TS.SP_FET) have a high negative correlation to PC1. Since those metrics specify the percentage of code reached by a test case, it makes sense for them to decrease as a function of program size.

PC2 explains over 16% of the observed variation. The metrics identified as mapping predominantly to PC2 (TS.P_TRCHF, TS.AN_CHFET, TS.AN_CHSET, TS.SN_CHFET) are a subset of the "test suite composition" metrics. This subset consists of metrics that measure the number of changed statements executed by a test case. This differs from the PC1 testing metrics, which do not capture any information about changes, constituting a unique (and unanticipated) component that quantifies a combination of test execution and change characteristics.

PC3, PC4 and PC5 present high loading on the "change characteristics" metrics. PC3 explains almost 13% of the observed variance and most of its metrics (3 out of 4) are associated with "change characteristics". This subset of metrics measures the extent of changes within the program. The exception is TS.SN_CHSET, which was classified as a "test suite composition" metric. Although this metric does capture some information about changes, it is unclear why it is more correlated to PC3 than to PC2, which otherwise involves metrics that measure test coverage of changes. PC4 explains almost 10% of the observed variation and is highly correlated to another subset of the metrics identified as "change characteristics" (CHG.AN_CHMOM, CHG.P_FCH, and CHG.P_SCH). These metrics differ from those in PC3 because they measure the distribution of changes instead of the extent of the changes. PC5 includes the CHG.N_GLO_CH and CHG.P_GLOCOC metrics, which are also "change characteristics" metrics. These metrics capture unique characteristics such as global variables, global constants, and global structure type definitions.

PC 6 correlates to just one metric: PRG.AN_PATHS. Principal component analysis shows that the average number of paths is not correlated with any of the other metrics that we have considered, including the other "program characteristics" metrics. This is probably due to the fact that the number of paths per function is a control flow measure, which constitutes a source of variation not captured by the other program metrics.

In summary, the principal component analysis discovered 6 underlying uncorrelated sources of variation. These sources of variation did not match the ones that we had expected to find, in number or nature, because: (1) most of the metrics describing program characteristics and the metrics associated with the distribution of the testing activity behave in a similar fashion; (2) metrics describing how the testing activity is associated with the changes constitute a unique source of variation, and thus, testing metrics are present in two domains: one in association with programs and the other involving changes; and (3) there is more than one dimension describing change characteristics: the extent, distribution and global characteristics of changes constitute independent dimensions.

5. Regression Analysis

The principal component analysis provided insights into the underlying dimensions of the collected metrics. This section takes the analysis a step further by deriving relationships between the collected metrics and APFD. We use regression analysis to examine the relationships between one or more independent variables and the dependent variable. In other words, we use regression analysis to evaluate the collected metrics as predictors of our measurement criterion APFD. We performed this analysis in two phases. First, we individually regressed each variable against the measurement criterion APFD. Second, we created a multiple regression model with metrics representing various factors. Note that, in following this process, we are not attempting to evaluate the prioritization techniques [4, 17], nor are we trying to build a prediction model. Instead, this section explores the influence of certain factors (as represented by the chosen metrics) across the different prioritization techniques.

5.1. Univariate Regression

Table 8 synthesizes the univariate regression analysis. Each metric was analyzed independently under each technique to account for that metric's influence on APFD. For space reasons, we present only the squared multiple R (referred to as R-square or R2)[5]. The R-square value represents the amount of reduction in the variability of APFD obtained through the regressor variable. Next to each R-square value in the table and within parentheses is the number of observations made per technique for each metric. Given the large number of observations, the **p** values were all very small (smaller than 0.005) so we rejected the hypothesis that for each metric in the table, the coefficient was zero. Some of the cells in Table 8 have a gray shade to assist in data visualization. The three shades of gray, from darkest to lightest, correspond to R-

---

[5] Access to the rest of the model per variable can be obtained at [5].

square values in the ranges of [1, 0.8], (0.8, 0.7], and (0.7, 0.6], respectively.

Most of the metrics with higher R-square values had a high correlation with PC1, in particular metrics such as PRG.A_FSIZE that reflect the structure of the program in terms of its distribution (means, standard deviation, and so forth). The same holds for most of the metrics that describe the relationship between the program and test cases such as TS.AP_FET. Some of the metrics highly correlated with PC2 also had relatively high R-square values, but only within some of the techniques (and especially with optimal). For example, TS.SN_CHFET was approximately 0.6 for all the techniques. Interestingly, the metrics that correlated highly with PC3 presented lower R-square values, which means that the sizes of the changes did not correlate directly with APFD. In PC4, which reflected the distribution of changes, the

number of functions that changed shows an R-square value greater than 0.6 across all the techniques. The metrics mapped to PC5 did not show high R-square values, which indicates that the extent to which changes are global did not have a large impact on APFD. The same can be said about PC6.

It is also worth observing that the optimal technique has the highest R-square value for the majority of the shaded cells. This might be due to the fact that orderings computed by the optimal technique are computed knowing the location of the faults, and this minimizes a source of noise present in all the other techniques. Although the prediction of APFD under the optimal technique itself does not make much practical sense (it is only a control technique), we could use our ability to predict optimal based on our metrics to set an upper threshold for prioritization possibilities when the location

Table 8. Univariate regression

PCs	Metric	Technique						
		Ac-f	Ac-s	Fi-a	Fi-t	Optimal	Tc-f	Tc-s
		R2 (1779)	R2 (1639)	R2 (1683)	R2 (1452)	R2 (532)	R2 (1512)	R2 (1470)
1	TS.SUITE_S	0.361	0.34	0.366	0.364	0.319	0.369	0.36
	PRG.N_EXECST	0.261	0.247	0.273	0.272	0.253	0.276	0.271
	PRG.N_FUNCT	0.485	0.461	0.482	0.475	0.404	0.474	0.47
	TS.AN_STET	0.418	0.391	0.417	0.413	0.355	0.415	0.41
	PRG.PROG_S	0.298	0.281	0.307	0.305	0.278	0.309	0.303
	TS.SN_STET	0.274	0.258	0.284	0.28	0.26	0.284	0.281
	TS.AN_FET	0.632	0.621	0.63	0.624	0.556	0.618	0.614
	TS.SN_FET	0.657	0.663	0.654	0.644	0.614	0.638	0.636
	PRG.A_FFAN	0.778	0.796	0.787	0.795	0.889	0.783	0.781
	PRG.A_FSIZE	0.777	0.756	0.78	0.78	0.816	0.772	0.768
	PRG.S_FSIZE	0.736	0.738	0.747	0.744	0.78	0.734	0.733
	TS.AP_FET	0.705	0.749	0.706	0.707	0.953	0.693	0.694
	TS.SP_SET	0.627	0.677	0.617	0.609	0.894	0.595	0.597
	TS.AP_SET	0.681	0.719	0.676	0.667	0.941	0.656	0.656
	TS.SP_FET	0.674	.714	0.669	0.661	0.921	0.649	0.651
2	TS.P_TRCHF	0.5	0.596	0.496	0.493	0.749	0.485	0.483
	TS.AN_CHFET	0.458	0.51	0.441	0.451	0.573	0.448	0.44
	TS.AN_CHSET	0.395	0.391	0.368	0.388	0.369	0.386	0.389
	TS.SN_CHFET	0.597	0.584	0.596	0.603	0.736	0.597	0.591
3	CHG.N_INS	0.356	0.322	0.334	0.339	0.271	0.327	0.322
	CHG.N_TOT_CH	0.356	0.317	0.336	0.338	0.265	0.329	0.23
	TS.SN_CHSET	0.493	0.437	0.458	0.486	0.415	0.453	0.456
	CHG.N_DEL	0.348	0.308	0.333	0.332	0.257	0.326	0.313
4	CHG.AN_CHMOM	0.565	0.487	0.541	0.54	0.543	0.557	0.542
	CHG.P_SCH	0.426	0.338	0.394	0.397	0.358	0.409	0.405
	CHG.P_FCH	0.553	0.521	0.532	0.512	0.595	0.53	0.525
	CHG.N_FUN_CH	0.657	0.643	0.639	0.631	0.648	0.651	0.638
5	CHG.N_GLO_CH	0.143	0.136	0.153	0.147	0.184	0.152	0.163
	CHG.P_GLOCOC	0.146	0.157	0.145	0.147	0.219	0.149	0.154
6	PRG.AN_PATHS	0.457	0.418	0.461	0.441	0.533	0.439	0.447

of faults is not known.

In summary and based on this univariate regression study, it seems that metrics reflecting the normalized program characteristics[6], and the characteristics of the test suite in relation to the program (expected coverage measures) and in association with the changes, are the main contributors to the variation in APFD. In other words, most of the metrics with high R-square values have high loadings on PC1 and a few from PC2 and PC4 also have relatively high R-square values. Those metrics with the highest R-square values are important because they can explain the most variance in APFD and offer better prediction of APFD. The other 55% of the metrics were not able to explain even half of the variation in APFD, as measured by their R-square value. It is also worth noting that in the majority of cases, the metrics prediction power was consistent across techniques; metrics with high R-square under one technique were likely to be good predictors of APFD independent of the chosen prioritization technique.

5.2. Multiple Regression

This section can be considered an extension of the previous univariate regression section, in that here we create a regression model that considers a set of metrics acting as predictors of the APFD criterion. We conjecture that by using a set of metrics that captures different sources of variation, we will be able to generate a more appropriate model to explain APFD behavior. To create the multiple regression model, we first selected a subset of the metrics to act as predictor variables because a regression model with many metrics (we had more than 30) would make the results difficult to interpret, and likely unstable.

Metrics were selected based on just one criterion: having the highest prediction capability (as indicated by the R-square values in the univariate analysis) within the groups of metrics that mapped to the same PC[7]. Initially, we selected 3 metrics: PRG.A_FFAN, TS.P_TRCHF and CHG.N_FUN_CH, each with the highest R-square value within its domain. By selecting metrics from different components we capture several factors affecting our criterion, which is likely to lead to more stable regression models. Then, we added TS.AP_FET to account for the possible effect of the test suite coverage metrics that had high negative loadings with PC1.

Table 9 presents 8 multiple regression models, one for each of the techniques and one overall model. The column

labeled "N" indicates the number of observations available to construct each model. In addition to the multiple R (an indicator of how well the model fits the data), we have included the adjusted R-square values to account for the addition of metrics to the model, so that more variables will not necessarily guarantee a higher prediction coefficient. The adjusted R-square is a stricter estimator of the model prediction power than the optimistic R because it considers the number of regressors. The *p* value for each model is presented in the next column. The last four columns introduce the model coefficients for each of the chosen metrics.

For each of the models *p* was less than 0.001, indicating that there was a linear relationship between the response variable, APFD, and the subset of chosen independent variables. For all but one model, the adjusted R-square indicates that approximately 80% of the variation in APFD can be explained by that model. The noticeable exception is the model for the optimal technique, where over 99% of the variation can be explained. Note also that the overall model, which does not discriminate among techniques, still has a high adjusted R-square.

In addition to evaluating the goodness of fit of the model through the correlation coefficients, we evaluated the adequacy of the overall model by analyzing the residuals and performing a detection of influential observations. We identified 40 observations that had a disproportionate influence on the model by using Cook's test [11]. Those 40 observations corresponded to the same version of the program space -- the one with the largest number of faults. These observations, although extreme, are valid and cannot be dismissed from the observation set. This fact indicates that we might need additional metrics in the model to capture those outliers appropriately. We also realized that since our model did not account for APFD bounds, some estimations were over the proper range. Slight adjustments to the model (piecewise regression) to account for those discontinuities would address this disparity.

In a nutshell, the implications of this section's analysis are twofold. First, we have determined that more powerful models to explain the variation in APFD can be developed through the combination of different metrics. Second, we expect this understanding to lead us to the creation of more powerful prioritization techniques that may be able to improve the rate of fault detection by including and joining the sources of variation captured by these new metrics (which are not a part of our current prioritization techniques).

---

[6] The results were fairly consistent across all programs. Individual tables per program are included in [5].

[7] Stepwise regression could have been used to keep the metrics that contributed the most to explaining APFD variation. However, preliminary results using stepwise regression did not reduce the number of metrics enough to make the model comprehensible.

Table 9. Multiple regression

Model Technique	N	Multiple R	Adjusted R-Square	p	Coefficients			
					CHG.N_FUN_CH	PRG.A_FFAN	TS.AP_FET	TS.P_TRCHF
Ac-f	1779	0.892	0.795	0.001	4.8	14.8	18.1	-15.6
Ac-s	1639	0.901	0.811	0.001	2.1	14.2	25.5	-3.2 [8]
Fi-a	1683	0.896	0.803	0.001	4.0	16.0	23.1	-18.8
Fi-t	1452	0.905	0.818	0.001	4.3	16.9	27.9	-24.8
Optimal	532	0.997	0.994	0.001	-0.8	14.9	84.4	8.8
Tc-f	1512	0.898	0.805	0.001	5.4	15.3	24.5	-21.8
Tc-s	1470	0.898	0.806	0.001	5.1	15.5	28.2	-24.8
Overall	10067	0.896	0.803		4.0	15.8	26.9	-17.9

6. Threats to Validity

In this section, we summarize in three groups the potential threats to the validity of our study.

Internal Validity (causal relationship between independent and dependent variables). Our regression models provide statistical evidence of the relationships between various metrics and our APFD measure. However, this type of model cannot guarantee causality. We limited this threat by providing an environment in which most factors (e.g., test suites, changes and techniques) were controllable. We also ensured that the assumptions necessary to perform regression analysis held, by performing random sampling and transforming the data (to ensure normality, linearity and constant variance), choosing a small number of modeling variables for the multiple regression, and employing principal component analysis to address the multicolinearity problem. A second (and hopefully minor) internal threat involves the tools and processes used in the data collection. We have continually validated the tools for metric generation. However, our tools for syntactic differencing are based on other tools for which the level of accuracy can be questioned (e.g., diff reports). We have validated many small inputs to control this threat.

External validity (results generalization). The representativeness of our subject programs is the major external threat to validity for our study. Although we were able to manipulate our test suites, versions and techniques, our results are somewhat limited in terms of the number and nature of subject programs that were considered. Furthermore, the fact that a large percentage of the changes we considered constituted faults may have caused prediction models based on change metrics to be optimistically biased. This threat also prevents us from

blindly discarding some of the metrics (e.g., global change metrics) that had low prediction capabilities due to the subject programs' specific characteristics (e.g., small number of global variables or global changes). A third threat involves our test cases and test suites. Although these suites are constructed from a mix of tests, they may not represent distributions of test cases that would occur in practice. In general, however, such threats to external validity as these can be addressed only by additional studies on additional subjects.

Construct validity (measure appropriateness). The dependent variable APFD is not the only possible measure of rate of fault detection and it does not capture every aspect of prioritization effectiveness. For example, APFD does not account for fault severity, test cases with different costs, and the value of re-detection of faults by additional test cases. Also, the metrics that we choose to quantify the sources of variation affecting APFD were the best metrics that we could generate with the tools available in our environment. Although more appropriate metrics might exist, we provided sufficient confidence in our metric selection through the verification of the sources of variation we captured (principal component analysis) and through the amount of APFD variation the selected metrics could explain (regression analysis).

7. Conclusions

In this study, we have explored factors that might affect a test suite's rate of fault detection as measured by APFD. Our study employed a large set of metrics to capture the sources of variation corresponding to those factors affecting APFD. We first learned that many of the selected metrics encapsulated similar dimensions and were in some way redundant in their information content. In addition, we discovered that the identified dimensions (factors) were not exactly those we had anticipated. We also determined which metrics accounted for the greatest variation for our criterion: in other words, which metrics

---

[8] The t statistic indicated that the individual variable was not useful (significant) for the model.

were the best predictors of APFD. Finally, we created a multiple regression model to illustrate how several metrics can be used to obtain greater prediction power.

The results of this study have three major implications. First, the metrics capturing the new factors might lead to the development of more powerful prioritization techniques. Novel techniques could incorporate the information provided by those new metrics that best predict APFD, enhancing the probability of generating a more effective test suite order. For example, since the fan-out metric showed a high correlation with the effectiveness of our current prioritization techniques, we could devise a new prioritization technique that takes advantage of fan-out data. Such a prioritization technique could sort test cases so that those that execute functions with high fan-out are given priority. Furthermore, this prioritization technique could also incorporate the information regarding the location of the changes. Second, the identification and understanding of the sources of variation that impact APFD will help us to develop guidelines to assist the practitioner in determining which techniques are likely to be more appropriate for a given environment (program, test suite, and changes). Third, the high prediction capabilities of the regression model for the optimal technique open new opportunities for the evaluation of test suite orderings, because it can accurately estimate an upper threshold for prioritization potential without knowing the location of the faults.

There are still many challenges ahead of us. First, we must find a way to modify existing techniques to incorporate the new information. Second, we must design experiments and perform additional analyses to develop rules that will guide the technique selection process. Last, we need to replicate this work on a larger sample of subjects to reinforce the current empirical evidence. Through these efforts we hope to provide practitioners with useful, cost-effective methodologies for prioritizing test cases.

Acknowledgements

References

[1] L.Briand, J. Wust, S. Ikonomovski and H.Lounis. Investigating quality factors in object oriented designs: an industrial case study. In *Proc. Int'l. Conf. Softw. Eng.*, pages 345-354, May 1999.

[2] S. Elbaum and J. Munson. Code churn: A measure for estimating the impact of code change. In *Proc. Int'l. Conf. Softw. Maint.*, pages 24-31, Nov. 1998.

[3] S.G. Elbaum and J.C. Munson. Software evolution and the code fault introduction process. *Emp. Softw. Eng.,* 4(3): 241-262, Sept. 1999.

[4] S. Elbaum, A. Malishevsky, and G. Rothermel. Prioritizing test cases for regression testing. *Proc. Int'l Symp. Softw. Testing and Analysis*, pages 102-112, Aug. 2000.

[5] S. Elbaum, D. Gable and G. Rothermel. On the sources of variation in the prioritization of regression test suites. Tech. Rep. TRW-SW-2000-1. University of Nebraska – Lincoln, CSE.

[6] S. Elbaum, A. Malishevsky, and G. Rothermel, Incorporating varying test costs and fault severities into test case prioritization, *Proc. Int'l Conf. Softw. Eng.*, May, 2000 (to appear).

[7] Everitt, B. and Dunn, G. Applied Multivariate Data Analysis. Edward Arnold. 1991.

[8] M. Harrold and G. Rothermel. Aristotle: A system for research on and development of program analysis based tools. Tech. Rep. OSU-CISRC-3/97-TR17, Ohio State University, Mar. 1997.

[9] M.Hutchins, H.Foster, T.Goradia and T.Ostrand. Experiments on the effectiveness of dataflow and controlflow based test adequacy criteria. In *Proc. Int'l Conf. Softw. Eng.*, pages 191-200, May 1994.

[10] Software Engineering Standards, volume 3 of *Std.1061: Standard for Software Quality Methodology.* Institute of Electrical and Electronics Engineers, 1992.

[11] Johnson, R. and Wichern, D. Applied Multivariate Statistical Analysis. Prentice Hall. 1992.

[12] T.M. Khoshgoftaar and J.C. Munson. Predicting software development errors using complexity metrics. *J. Selected Areas in Comm.*, 8(2):253-261, Feb. 1990.

[13] J. Munson. Software measurement: Problems and practice. *Annals of Softw. Eng.*, 1(1):255-258, 1995.

[14] A. Nikora and J. Munson. Software evolution and the fault process. In *Proc. Twenty Third Annual Softw. Eng. Workshop, NASA/Goddard Space Flight Center,* 1998.

[15] K. Onoma, W.-T. Tsai, M. Poonawala, and H. Suganuma. Regression testing in an industrial environment. *Comm. ACM*, 41(5):81-86, May 1998.

[16] T. Ostrand and M. Balcer. The category-partition method for specifying and generating functional tests. *Comm. ACM*, 31(6): 676-686, June 1988.

[17] G. Rothermel, R.Untch, C.Chu, and M.J. Harrold. Test case prioritization: an empirical study. In *Proc. Int'l Conf. Softw Maint.,* pages 179-188, Aug. 1999.

[18] G. Rothermel, R. Untch, C. Chu, and M. J. Harrold. Test case prioritization. ACM Trans. Softw. Eng. and Meth. (to appear).

[19] F. Vokolos and P. Frankl. Empirical evaluation of the textual differencing regression testing technique. In *Proc. Int'l Conf. Soft. Maint.*, pages 44-53, Nov. 1998.

[20] W. E. Wong, J. R. Horgan, S. London, and A. Agrawal. A study of effective regression testing in practice. In *Proc. Eighth Int'l Symp. Softw. Rel. Eng.*, pages 230-238, Nov. 1997.