Towards the Prioritization of System Test Cases

Hema Srikanth[1], Laurie Williams[1], Jason Osborne[2]

[1] Department of Computer Science, North Carolina State University, Raleigh, NC 27695

[2] Department of Statistics, North Carolina State University, Raleigh, NC 27695

Email: {hlsrikan, lawilli3, jaosborn}@ ncsu.edu

Research Area: Software Testing and Reliability, Software Quality, Value-Based Testing

Abstract

During software development companies are frequently faced with lack of time and resources, which limits their ability to effectively complete testing efforts. Often, the engineering team is compelled to stop their testing efforts abruptly due to schedule pressures. We build upon prior test case prioritization research and present a system-level, value-driven approach to test case prioritization called the Prioritization of Requirements for Test (PORT). PORT involves analyzing and assigning value to each requirement based on four factors: requirements volatility, customer priority, implementation complexity, and fault proneness. System test cases are prioritized for execution based on the assigned priority value of the requirements they originate from such that the test cases for requirements with higher priority are executed earlier during system test. We applied PORT to four student team projects in an advanced graduate software testing class. Our results show that PORT prioritization at the system level improves the rate of detection of severe failures. Additionally, we found that customer priority was the most important contributor towards improved rate of failure detection.

1. Introduction

In today's changing business environment, time to market is a key factor to achieving project success. For a project to be most successful, quality must be maximized while minimizing cost and keeping delivery time short [14]. Quality can be measured by the customer satisfaction with the resulting system based on the requirements that are incorporated successfully in the system [14]. Boehm suggests that currently most of the software engineering research and practice is done in a value-neutral setting whereby all requirements, use cases, test cases, and defects are treated as equally important without considering the business value provided to the customer. Boehm proposes a value-based approach to software engineering that measures the value the system provides to the prospective customers [4]. Value-based engineering involves the prioritization of development activities, keeping in mind stakeholder value propositions. Value-based software engineering practices are believed to improve user-perceived software quality. These practices are believed to improve user-perceived software quality [3, 4]. This paper explores a value-driven approach to prioritizing software systems by directing test efforts on requirements that are of highest value to the customer, which is believed to improve user-perceived software quality.

Software testing is a strenuous and expensive process [2, 6]. Research has shown that at least 50% of the total software cost is comprised of testing activities [10, 33]. Companies are often faced with lack of time and resources, which limits their ability to effectively complete testing efforts. Prioritization of test cases in the order of execution in a test suite can be beneficial. Test case prioritization (TCP) involves the explicit prior planning of the execution order of test cases to increase the effectiveness of software testing activities by improving the rate of fault detection earlier in the software process [24, 25].

To date, TCP has been primarily applied to improve regression testing efforts [7, 24, 25] of white box, code-level test cases. Regression testing is the process of retesting of a system or component to verify that changes made to the code have not caused unintended effects and that the system is still compliant with the specified requirements [12]. Software engineers save test cases developed for prior versions and re-run these test cases as regression tests in later versions. Running the entire set of test cases on a revised version, however, could be cost and time-prohibitive. Currently, regression TCP techniques use structural coverage criteria to select the test cases [23]. Structural coverage techniques, such as statement or branch coverage, are applicable at the code level [5].

We build upon the current code-coverage TCP techniques [7, 8, 24, 25] and propose a black box approach called Prioritization of Requirements for Testing (PORT v 2.0) with the objective of *developing and validating a system-level test case prioritization scheme to reveal severe failures earlier.* PORT prioritizes system-level black box tests by considering four factors: (1) customer-assigned priority of requirements; (2) developer-perceived implementation complexity; (3) requirements volatility; and (4) fault proneness of requirements. PORT (v 1.0) [31] incorporated only three prioritization factors, PORT (v 2.0) incorporated the fourth factor: fault proneness.

PORT is an easy-to-use scheme wherein the factor values can be collected by the engineering team during the design phase with minimal effort. Directing test efforts using these four factors is believed to enable more efficient identification of severe failures earlier in the software process. By focusing on customer-assigned priority, we aim to identify requirements, which would increase *customer-value,* thus improving the overall business-value provided to the customers. Our set of research goals is listed below.

G_1: To develop and validate a method for identifying the most severe failures earlier in system test.

G_2: To determine the most effective PORT factors that contribute to an improved rate of detection of severe failures.

Via our prioritization scheme, we aim to increase test efficiency by improving the rate of failure detection, which is measured by the percentage of failures detected over the life of the test suite.

To determine the effectiveness of PORT (v 1.0), an academic case study was conducted on four similar student team projects developed in an advanced graduate software testing class at North Carolina State University. The rest of this paper is structured as follows. Section 2 discusses the background work. Section 3 presents the PORT scheme. Sections 4 and 5 present the PORT validation strategy and the case study. Section 6 presents the sensitivity analysis. Section 7 presents comparison of PORT scheme with Musa’s Software Reliability Engineered Testing (SRET) approach. Section 8 discusses summary and future work.

2. Related Work

This section provides an overview of coverage-based TCP techniques; software reliability engineered testing, and our earlier work that led to identification of the PORT prioritization factors.

2.1. Coverage-based Test Case Prioritization

Coverage-based TCP techniques [7, 8, 24, 25] involve ranking test cases based on the statement coverage they provide; test cases are ranked based on the number of statements executed/covered by the test case such that the test case covering the maximum number of statements would be executed first. For branch and function coverage techniques, tests are prioritized based on the program branches or program functions covered, respectively.

The benefits of code coverage-based TCP strategies were measured using a weighted Average Percentage of Faults Detected (APFD) [24] metric. The APFD value is a measure of how quickly the faults are identified for a given test suite set. The APFD values, range from 0 to 100 and are monitored during test suite execution. The APFD values represent the area under the curve by plotting *percentage of faults detected* on the y-axis of a graph, and *percentage of test suite run* on the x-axis. We analyze a similar validation metric to assess the efficacy of PORT, Weighted Percentage of Failures Detected (WPFD), as will be discussed in Section 4.

If all faults are not equally severe, severity-neutral TCP strategies and associated APFD metric can provide misleading information [9]. As a result, Elbaum et al. incorporate fault severity in a cost-cognizant TCP strategy [9]. Instead of representing *Test Suite Fraction* in the horizontal axis (as done for APFD), *Percentage of Total Test Case Cost Incurred* ($APFD^c$) is represented. Additionally, instead of representing *Percent Faults Detected* on the vertical axis (as for APFD), *Percentage Total Fault Severity Detected* ($APFD^c$) is represented. The $APFD^c$ measures the unit-of-fault-severity-detected-per-unit-test-cost [9]. The use of $APFD^c$ is can be used to assess the prioritization orders post hoc, i.e. when the severity and cost values are known, and cannot be used for predicting cost and severity values. In the validation of the $APFD^c$ metric, Elbaum et al. use six levels of severity to assign severity values to the faults in the program. We follow a similar approach of assigning severity values to the failures, which is discussed in Section 4.

2.2. Software Reliability Engineered Testing

Musa prioritizes system tests using Software Reliability Engineered Testing (SRET) [19], a business value-based approach. Test planning using SRET involves estimating the relative customer use of the functions in a software product via an operational profile. An operational profile is a set of operations and their probabilities of occurrence [19] where an operation is a task that the system performs. Operational profile estimation is done collectively by testers, system engineers, architects and customers. Once the operational profile has been estimated, testers determine the total number of test cases that can be written and executed according to budgetary and resource constraints. Then, the number of test cases written for each operation is a relative proportion of the total number of test cases based upon the use percentage for that operation in the operational profile. Once the tests have been written, they are executed in a random order. SRET was applied extensively at AT&T. One particular project at AT&T, International Definity, showed that the use of SRET in one product release resulted in increased reliability and customer satisfaction when compared with the previous release of the same product. Additionally, product sales increased by a factor of 10, system costs decreased by a factor of two, total project development time decreased by 30%, and maintenance costs were reduced by a factor of 10 [20].

We use a similar approach in test planning in that we consider customer-assigned priority in our prioritization scheme where we determine which requirements are more important to the customer and apply the obtained information in our TCP scheme. Despite the positive business results from applying SRET, some consider the creation and maintenance of the operational profile to be resource intensive and propose non-operational [34] techniques. PORT is a non-operational profile system test case prioritization scheme.

2.3. Case Study to Identify Prioritization Factors

A postmortem analysis was conducted on an industrial project that was comprised of 152 thousand lines of code (KLOC) [27-30]. The goal was to determine the developmental factors that resulted in increased number of severe failures. The findings of this study were factored in the determination of our PORT prioritization factors.

For this industrial project, a total of 1,030 failures were identified in system test: 16 of them were classified as Severity 1, 259 as Severity 2, 608 as Severity 3, and 157 as Severity 4 failures. For this case study, we identified (1) the percentage of the total failures that were identified during beta test, and (2) the most prevalent factors that caused some modules to have higher failure density than others. Thirteen percent of Severity 1 and 23% of the Severity 2 failures were found during beta testing. The company's goal for future releases is to reduce the detection of severe failures during beta testing to less than 5%.

The failures were analyzed and mapped back to eight modules that comprised the system. Our results show, that 78% of the Severity 1 and Severity 2 failures were found in 25% of the modules; these modules were reported by the engineering team as the most volatile and complex modules, motivating requirements volatility and implementation complexity to be considered in our system-level prioritization scheme.

3. Prioritization of Requirements for Test (PORT)

This section presents the PORT factors and the reasoning behind the factor selection. Also the factor collection process and the algorithms used to prioritize test cases using PORT scheme is discussed.

3.1. PORT Factors

3.1.1. Requirements volatility (RV) is based on the number of times a requirement has been changed in the development cycle. Requirements volatility is an assessment of the requirements change once the implementation begins [15].

Reasoning: Roughly 50% of all faults identified in a project are errors introduced in the requirements phase [16]. On average severe defects that escape into the field can cost 100 times more to fix after delivery than correcting the same problem in the requirements phase [26]. Studies conducted by the Standish Group report that 30% of all projects are cancelled before completion, and 70% of the remaining projects fail to deliver the required system functionality. The most significant factor to cause these project failures were attributed to changing requirements [32]. Other studies show the cause for project failures to be lack of user input, and changing or incomplete requirements [13, 16, 21]. Roughly 25% of the requirements for an average project change before project completion [21], and volatile requirements tend to make the testing activities difficult and cause the software to contain high defect density [15]. Changing requirements result in re-design, and often an increase in the fault density in the program [15] which is also the true in the case study conducted and discussed in Section 2.3 [27-30].

3.1.2 Customer-assigned priority (CP) is a measure of the importance of a requirement to the customer. The customer assigns a value for each requirement ranging from 1 to 10 where 10 is the requirement with the highest customer priority.

Reasoning: Approximately 45% of the software functions are never used, 19% are rarely used, and only 36% of the software functions are always used [17]. A fault that lies along the path of normal execution results in frequent failures, and the majority of the effort should be spent in finding these faults [18, 19]. A focus on customer requirements for development has been shown to improve customer-perceived value and satisfaction [3, 4, 14]. By identifying and more thoroughly testing the fraction of requirements that would be of highest importance to the customer earlier in testing, we aim to increase the business value generated to the customer. If the testing efforts were shortened due to schedule pressures, the requirements of highest value to the customer would have been tested early and thoroughly.

3.1.3. Developer-perceived implementation complexity (IC) is a subjective measure of how difficult the implementation of a requirement is perceived to be by the development team. Each requirement is analyzed to assess the anticipated implementation complexity and is assigned a value ranging from 1 to 10; the larger value indicates higher complexity. IC is a prioritization factor for requirements being implemented for the first time or for all requirements in the first release.

Reasoning: Several studies indicate that requirements with high complexity (examined post hoc) tend to have a higher number of faults. Amland [1] conducted a case study to find that the functions with high number of faults were the functions with higher McCabe complexity [1]. Twenty percent of the system modules result in 80% of the faults [1, 26, 30, 35], and roughly 50% of the modules are defect free [26]. To date, no studies have related pre-development developer-perceived complexity and fault proneness; we will examine this relationship in our work.

3.1.4. Fault proneness of requirements (FP) allows the development team to identify the requirements which have had customer-reported failures in the previous release. As the system evolves into several versions, the developers can use the data collected from prior versions to identify requirements that are likely to be error prone. FP is based on the number of field failures found in the code that implements a requirement. FP is not considered for new requirements, only for those requirements that have already been in a released product.

Reasoning: Ostrand et al. has shown that test efficiency can be improved by focusing on the functionalities that are likely to contain higher number of faults [22]. The functionalities that contain higher number of faults in one release are more likely to be troublesome in the future releases as well.

3.2. PORT Prioritization Factor Collection Process

The process for collection and updating the PORT factors is shown in Figure 1.

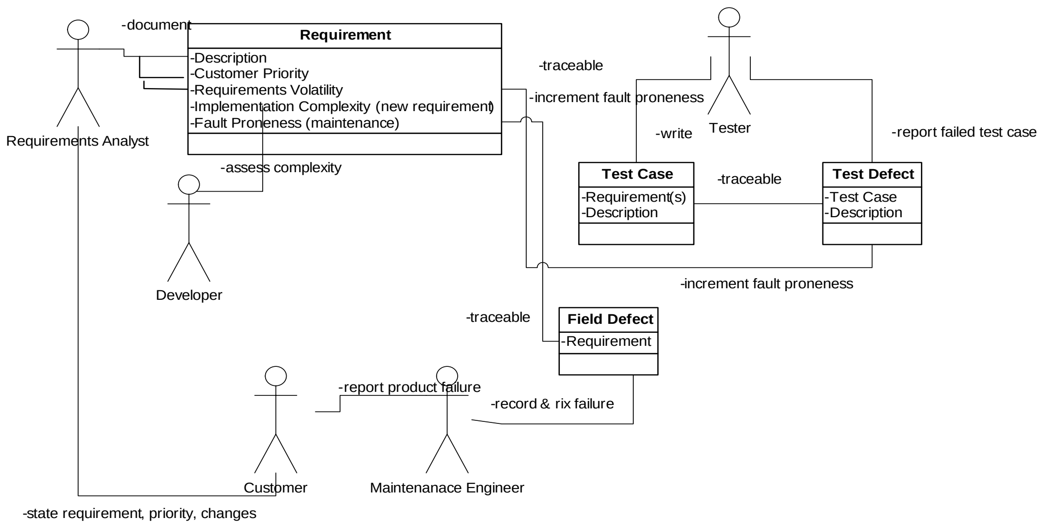

Figure 1: PORT Prioritization Factor Collection Process

An open source tool, ReBaTe[1] (Requirements Based Testing), supports the PORT scheme to the software development. ReBaTe allows users to automatically update PORT factors like requirements volatility and fault proneness as requirements change and failures are reported. The *tool* (ReBaTe) increments requirements volatility when a requirement is

[1] http://sourceforge.net/projects/rebate/

changed and increments fault proneness when failure is reported. In the absence of a tool, these values can be tracked manually. ReBaTe also displays the PORT-computed ordering of system test cases.

There are five stakeholders in the process; the stakeholder roles are defined below:

- The *customer* states the system requirements, the priority for the each requirement, and any additions or changes to the requirements throughout development. The customer also reports field failures after product delivery.
- The *requirements analyst* records the requirements and associated priorities and any changes to requirements.
- The *maintenance engineer* fixes defects when field failures are reported and maps the failure back to the requirement(s) that were impacted by the failure.
- The *developer/architect* provides a subjective assessment of how complex a requirement is to implement.
- The *tester* writes test cases for each requirement, mapping the requirement to its test case(s), and runs the test cases. Test case failures are reported, mapping the test case failure to the test case that revealed the failure.

Figure 2 presents a flow chart to represent the PORT structure.

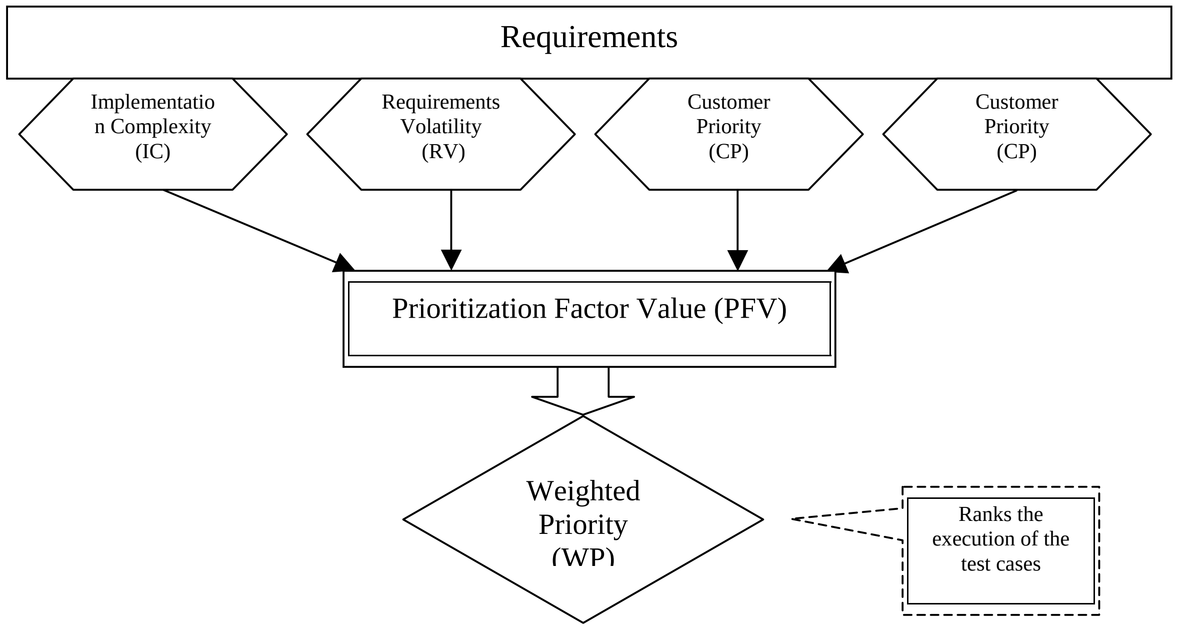
Requirements
Implementation Complexity (IC)
Requirements Volatility (RV)
Customer Priority (CP)
Customer Priority (CP)
Prioritization Factor Value (PFV)
Weighted Priority (WP)
Ranks the execution of the test cases

Figure 2: High level PORT Overview

The four factor values are used to compute the Prioritization Factor Value (PFV), which is then used to produce a ranked list of test cases. Factor values are assigned during the design analysis phase, and evolve continually during the software process. Based on project and customer needs, the development team assigns weight to the prioritization factor such that the assigned total weight (1.0) is divided among the four factors. Factor weight, which is unique for each project, allows the PORT user to customize the priority of each factor for a particular project. For e.g. if the requirements for a project have been stable, then the engineering team might assign RV a relatively smaller portion of the total weight. A default value can be assigned, giving each factor equal weight. Sensitivity analysis was performed to determine an effective way to allocate factor weights, as discussed in Section 6.

3.3.PORT Algorithm

For every requirement, Equation 1 is used to calculate a Prioritization Factor Value (PFV) for that requirement.

$$PFV_i = \sum_{j=1}^{4} (FactorValue_{ij} * FactorWeight_j) \qquad (1)$$

In Equation 1, PFV_i represents the prioritization factor value for requirement *i*, which is the summation of the product of factor value and the assigned factor weight for each of the factors. $FactorValue_{ij}$ represents the value for factor *j* for requirement *i*, and $FactorWeight_j$ represents the factor weight for *jth* factor for a particular product. PFV is a measure of the importance of testing a requirement. A matrix representation of the dependence of the PFV vector P on the value vector V and the weight vector w is shown in Equation 2 below.

$$P = Vw$$

$$\begin{pmatrix} PFV_1 \\ . \\ . \\ . \\ PFV_n \end{pmatrix}_{(n*1)} = \begin{pmatrix} R_1^{CP} & R_1^{IC} & R_1^{RV} & R_1^{FP} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ R_n^{CP} & R_n^{IC} & R_n^{RV} & R_n^{FP} \end{pmatrix}_{(n*4)} \begin{pmatrix} w_{CP} \\ w_{RC} \\ w_{FP} \\ w_{Rv} \end{pmatrix}_{(4*1)} \qquad (2)$$

The computation of PFV for a requirement is used in computing the Weighted Priority (WP) of its associated test cases. WP of the test case is the PFV contribution of the requirement(s) the test case maps to as shown in Equation 3. Let there be *n* total requirements for a product/release, and test case *j* maps to *i* requirements.

$$WP_j = \left[\frac{\sum_{x=1}^{i} PFV_x}{\sum_{y=1}^{n} PFV_y} \right] \tag{3}$$

WP_j is an indication of the priority of running a particular test case. The test cases are ordered for execution based on the descending order of WP values such that the test case with the highest WP value is run first and so on.

4. PORT Validation Metrics

Refinement and validation of PORT will proceed via the analysis of the severity of failures detected for a product. For analysis purposes, each failure is assigned a weighted severity value (SV) as follows:

- Highly severe (Severity 1): Severity 1 is assigned to a failure when a customer can no longer use the product and/or testing must cease until the defect causing the failure is fixed. For Severity 1 failures, we assign a SV of 2^4.
- Medium severe (Severity 2): Severity 2 is assigned to a failure when there is a work around for the failure and the product can be used with the work around. For Severity 2 failures, we assign a SV of 2^3.
- Less severe (Severity 3): Severity 3 is assigned to a failure for which a fix can be done in later versions. For Severity 3 failures, we assign a SV of 2^2.
- Least severe (Severity 4): Severity 4 is assigned to a failure for which a fix may be done at later versions or not done at all. A SV of 2^1 is assigned to failures with Severity 4.

The PORT validation approach involves computing Total Severity of Failures Detected (TSFD) for the project. TSFD is the summation of severity values (SV) of all failures identified for the product/release. Equation 4 shows TSFD for a product/release, where *t* represents total number of failures identified for the product/release.

$$TSFD = \sum_{l=1}^{t} SV_l \qquad (4)$$

The case study showing the use of TSFD is discussed in Section 5.

5. PORT Case Study

To measure the effectiveness of PORT (v 1.0), four similar projects with average size of approximately 2500 lines of code (LOC) were analyzed. The projects were developed by the students in an advanced graduate-level software testing class at North Carolina State University. The class was divided into four pairs of students, and each team was given the same 19 requirements to develop a TCP tool. For each project, the students assigned values for RV and IC. The research team (who acted as the customer) assigned values for customer priority (CP). FP was not applicable as the product was going through the first release and field data was not available. The factor weights were assigned based on discussion between the customer and students. The case study limitations and results are discussed in Sections 5.1 and 5.2 respectively.

5.1. Case Study Limitations

We consider the following limitations of our case study:

- *Construct validity* or establishing measures that reflect the theory under test. Our measures are defined in Section 3 to reflect our theories about utilizing four system-level factors to identifying the most severe failures earlier in system test.
- *Internal validity* or the occurrence that all the potential factors that might influence the data are controlled except the one under study. Internal validity is a strength of academic studies, such as ours. Our case study involves analyzing multiple versions of the same program all measured on the same criteria.
- *External validity* or how well the results of the study can be generalized to the world outside the research situation. Conversely, external validity is a weakness of academic studies. The programs involved in this case study are not as complex and volatile as industrial programs.

- *Experimental reliability*, which assesses whether another investigator would get similar results by following the experimental procedures with the same case study. The procedures outlined in this paper would enable another researcher to get similar results. As will be discussed in Section 5.2, we utilized a random selector, which would inhibit direct duplication of our results. However, we ran a sufficient number of trials to feel confident that a replication would produce similar results.

5.2. Case Study Results

In the case study, the PORT scheme was compared with a random prioritization strategy towards testing student projects. As an extra credit assignment, the students created a faulty version of their project by injecting 20 failures. The students were instructed that for each project, at least 50% of the injected failures were to be of Severity 1 and 2, and the other 50% comprised of Severity 3 failures. To test these four student projects, approximately 50 system test cases were written without any knowledge of the injected failures. The test cases were written to test at least one success condition and one failure condition for each requirement. At least one fourth of the test cases mapped to multiple requirements. The test cases were run and the test failures were recorded. The test results were then analyzed to investigate the differences due to ordering of test cases. The goal was to identify the effectiveness of PORT in improving the rate of detection of severe failures when compared with random prioritization approach. For each of the four projects, the following two factors were determined for both prioritization approaches:

- Rate of detection of severe failures.
- Contribution of prioritization factors towards the effectiveness of PORT

The experimental setup and the results for each of these two factors are discussed below.

5.3.1. Rate of Detection of Severe Failures

Goal: This section involves identifying the effectiveness of PORT scheme in improving the rate of detection of severe failures by testing four faulty applications.

Setup: Using the values and weights for the prioritization factors provided by the students and the research team, the PFV for the 19 requirements was computed. The WP for the system test cases was computed. The PORT scheme involves execution of the test cases in the descending order of the WP value. The random strategy involved a random ordering of test cases for execution.

The four faulty applications were tested by running the test cases to find the injected failures in the application. The failures found were mapped to their respective requirements. The TSFD was computed for all four projects. After executing each case, the test case status was noted: Pass/Fail. If a test case failed, the SV of the failure identified was noted. As discussed in Section 4, a Severity 1 failure was assigned a SV of 2^4, followed by Severity 2 failure with SV of 2^3, Severity 3 failure with SV of 2^2, and Severity 4 failure with SV of 2^1.

After all test cases were executed and all the induced failures were identified and their SV noted, analysis was done to measure the rate of failure detection. The rate of failure detection is computed using a metric called *Weighted Percentage of Failures Detected* (WPFD), which is the area represented under the curve when plotting a graph with *percentage of TSFD* against *fraction of test suite executed*. We compare the WPFD for PORT against the WPFD for a random prioritized set. Twenty unique random prioritization sets for each of the four projects were generated to allow for statistical comparison. The mean WPFD values for the 20 random different orderings are compared against the WPFD achieved for the PORT scheme. Statistical analysis was done to determine the effectiveness of PORT in comparison to 20 different sets of randomly prioritized test cases.

Results: For each team, percentage of TSFD was determined at different stages of test suite execution: after executing one-fourth, one-half, three-fourths and all the test cases. The results of the WPFD for both a random TCP and PORT strategy for all four projects are graphed in Figure 3. For the purposes of depicting the results graphically, the mean WPFD for 20 random permutations was compared with the WPFD for PORT.

As the Figure 3 shows, the WPFD achieved via PORT for all four projects is higher than mean WPFD for 20 random permutations.

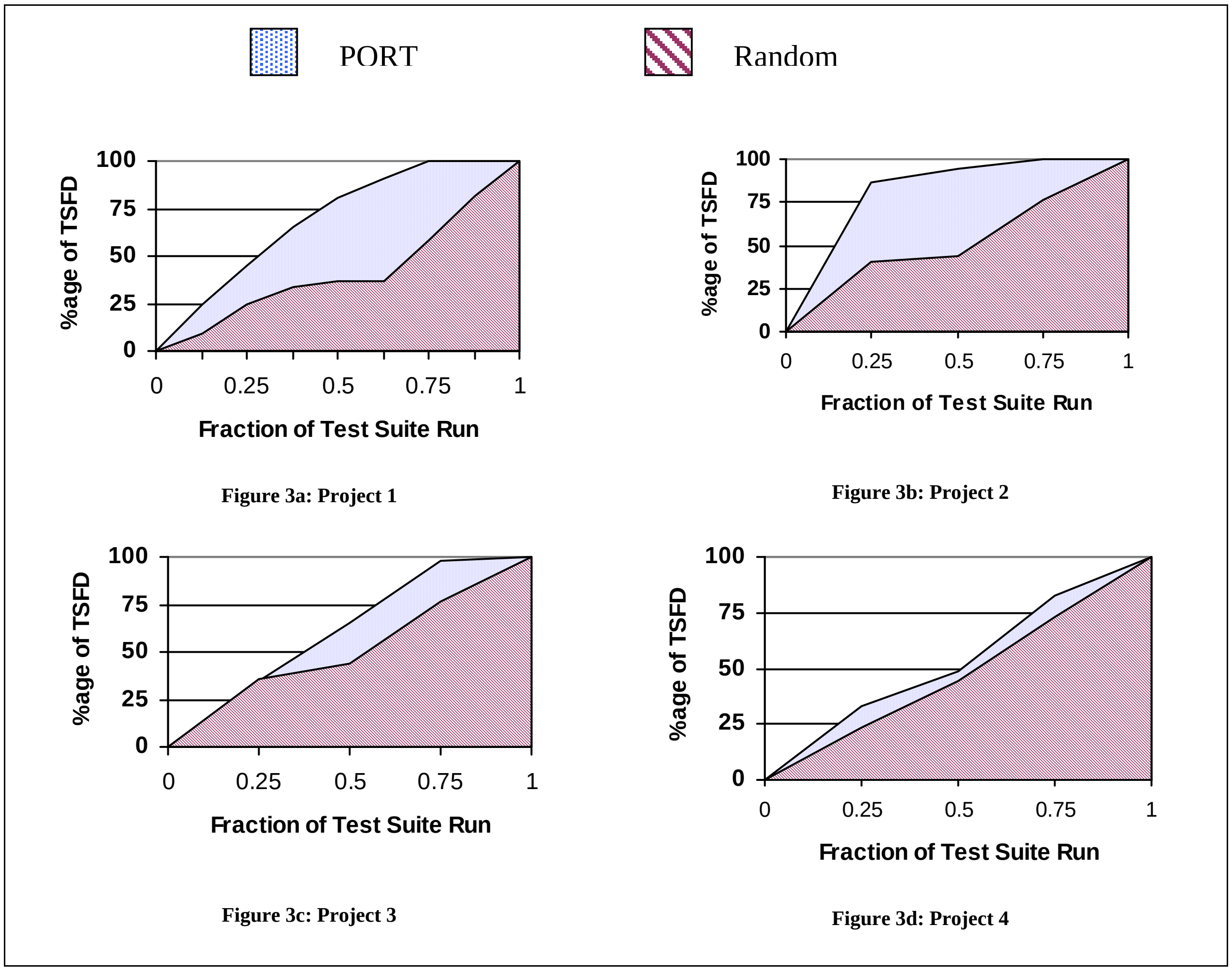

Figure 3a: Project 1

Figure 3b: Project 2

Figure 3c: Project 3

Figure 3d: Project 4

Figure 3: Comparison of PORT and Random

Table 1 provides WPFD for 20 different random TCP sets and one set of PORT scheme for all four projects. The mean WPFD values for Random TCP (for n=20) are also listed. The null and alternative hypotheses to determine the effectiveness of PORT over Random TCP set (for n = 20) are as follows:

H_0: WPFD for PORT = Mean WPFD for Random TCP

H_a: WPFD for PORT > Mean WPFD for Random TCP

We find statistically significant results in favor of H_a, PORT is better than random prioritization ($p < 0.01$). The results indicate that PORT strategy leads to improved rate of severe failure detection for all four projects.

Table 1: WPFD for 20 Random Sets of TCP and PORT TCP

Random TCP Set #	Project 1: WPFD	Project 2: WPFD	Project 3: WPFD	Project 4: WPFD
1	40.29	30.55	47.52	49.24
2	52.21	11.30	59.07	43.70
3	35.88	28.94	45.52	48.60
4	38.97	80.80	53.20	46.44
5	36.97	66.44	57.72	45.34
6	36.77	57.63	56.91	48.93
7	36.77	55.53	32.72	59.07
8	41.77	57.48	52.73	47.26
9	43.29	52.52	60.53	53.86
10	43.81	48.13	51.73	50.03
11	18.43	34.67	55.17	53.47
12	30.38	39.33	55.32	56.54
13	33.97	45.32	53.20	48.42
14	36.83	50.04	49.39	48.39
15	45.79	58.59	48.77	45.62
16	49.93	52.08	50.95	45.71
17	53.85	54.58	51.01	52.92
18	40.06	53.95	41.03	45.13
19	85.57	64.27	53.70	55.93
20	68.25	77.85	46.55	48.78
MEAN WPFD-Random TCP	**43.49**	**51.00**	**51.14**	**49.67**
# of Times PORT is better than Random order	18	20	20	17
PORT TCP: WPFD	**66.71**	**83.5**	**64.12**	**54.43**
Test Statistic	**4.99**	**8.48**	**8.84**	**7.49**
p value	**< 0.001**	**< 0.001**	**< 0.001**	**< 0.001**
Sign Statistic	**18**	**20**	**20**	**17**
sign-test p-value	**=0.0002**	**<0.0001**	**<0.0001**	**=0.0013**

Alternatively, the sign test may be used to investigate the null hypothesis that the WPFD for PORT is no better than that for a randomly chosen prioritization. The sign test is a simple nonparametric procedure that makes no assumptions about the distribution of WPFD. For example, in the 20 randomly chosen prioritizations for Project 1, PORT was observed to have a better WPFD 18 times. Using the binomial distribution, the probability of observing 18 or more successes under the null hypothesis of equivalence is $p = 0.0002$, a highly significant result indicating that the WPFD with PORT is higher than the median of all permutations. The last row of Table 1 gives the results for a sign-test for all four projects.

5.3.2 Analysis of PORT Factors

Goal: This section involves identifying the mean contribution of the prioritization factors towards the PFV of the requirements.

Setup: For all four projects, the contribution of each of the three prioritization factors towards the PFV for each of the nineteen requirements was computed. Also, for all projects, the mean contribution of the three prioritization factors towards all project requirements was computed.

Results: This section discusses the results of the effectiveness of the prioritization factors in the PORT scheme. Figure 4 shows the mean contribution of the three prioritization factors towards the PFV of the requirements for all four projects.

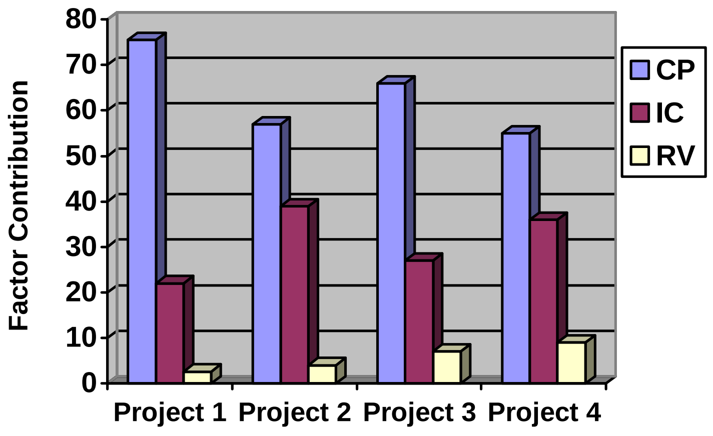

Figure 4: Contribution of PFs for Four Projects

The CP was ranked as the biggest contributor for all four projects, followed by IC and RV. On average, the CP contribution was at least 55% of the total PFV for all four projects. At least 22% of the total PFV contribution for all projects came from IC. The smallest contribution came from RV, which was less than 10% for all four projects. The RV had a lesser value, as the requirements for the project were very stable. Also, the project scope was limited as it was a part of a graduate course curriculum, and the students had less than 10 weeks to finish the project.

6. Sensitivity Analysis of PORT Factor Weights

In this section, we present sensitivity analysis conducted on strategies for assigning factor weights. The purpose of this analysis is to determine whether the factor weights can sway the prioritization provided by the PORT algorithm such that the software engineer is misguided. Additionally, we aim to reduce subjectivity by providing a more repeatable and objective process for assigning weights. For the sensitivity analysis, four possible allocations for factor weights are considered. The PFV for the requirements are computed for these four factor weight allocations for the four projects discussed in Section 5. The four different allocations for factor weights are as follows:

- *Team assigned weight*: the factor weights are assigned by the research team and students in the case study discussed in Section 5.
- *Equal weight: the* factor weights are equal. This approach is similar to having no factor weights. We chose this as a reference point because it removes all subjectivity in weighting the factors.
- *Mean-based weight*: the factor weights are allocated to correspond with the mean value of each factor for the 19 requirements. Intuitively, factors that tend to have higher weights might be more important to the analysis.
- *Variance-based weight*: the factor weights are assigned based on the variance of factor values for each project. At one extreme, if the variance between factor values is zero, this factor can be dropped from the analysis because there is no differentiation in this factor among the requirements being analyzed. Intuitively, the greater the variance, the more a particular factor can be used to differentiate the requirements.
    - Relative to variance, for one trial we chose the ”contradictory” approach and assign higher weights to factors with lower variance to assess whether inappropriate weights lead to misleading information.
    - We also allocated factor weights corresponding to the variance of the factor values for each project.

The results of sensitivity analysis for the four case study projects are presented in Table 2. For all four projects, the WPFD achieved based on different possible allocations of factor weights are displayed.

Table 2: WPFD for Different Factor Weights Distribution

	WPFD Values for Various Factor Weights				
	Equal Weights	PORT Weights	Mean-based Weights	Variance-based Weights	Contradictory Variance-based Weights
Project 1	66.30	66.71	**67.12**	67.08	65.75
Project 2	78.57	83.49	**85.13**	70.68	69.18
Project 3	63.54	64.12	**64.27**	63.24	62.14
Project 4	49.08	54.43	**56.38**	50.34	49.02

The results show that a *mean-based* assignment of weights achieves the highest WPFD for all four projects. Mean-based assignment involves allocating weights by computing the mean factor value for all project requirements. The weights are allocated to the factors proportionally based on the mean factor value. The *contradictory variance-based* approach to assigning weights resulted in the lowest WPFD values for all four projects. The *team-assigned* approach to allocating weights involves assigning weights based on discussion amongst the customer and engineering team. The *team-assigned* weight has shown to be better than the *equal weight* and *the contradictory variance-based* approach. The *intuitive variance-based weight* is comparable to *team-assigned* weight in that it is better than *equal weight* and *contradictory variance-based* approach. The *team-assigned* weight is better than *intuitive variance based weight* for three out of four projects. However *team-assigned* weight approach is not better than the mean-based approach. We will use mean-based approach to assigning weights in our future projects as that approach has yielded the highest rate of failure detection in comparison to the other four approaches.

We further investigated the contribution of the three PFs for each method of allocation of factor weights. The factor contribution for all four projects for different allocations of factor weights is shown in Table 3 below. The contribution is measured based on the average contribution of the factors towards the PFV of all project requirements.

Table 3: Factor contribution for different factor weights

		Project 1	Project 2	Project 3	Project 4
Equal Weights	RV	12.79%	15.25%	30.43%	21.58%
	IC	22.80%	39.09%	24.82%	33.69%
	CP	**64.41%**	**45.66%**	**44.75%**	**44.73%**
Mean-based Weights	RV	2.72%	3.87%	9.66%	6.71%
	IC	11.11%	39.95%	22.11%	34.20%
	CP	**86.17%**	**56.18%**	**68.22**	**59.09%**
Contradictory Variance-based Weights	RV	3.84%	3.94%	10.59%	6.80%
	IC	37.59%	**51.51%**	44.07%	41.79%
	CP	**58.57%**	44.55%	**45.34%**	**51.41%**
Variance-based Weights	RV	2.90%	3.86%	9.68%	6.75%
	IC	15.71%	37.42%	22.65%	38.42%
	CP	**81.39%**	**58.72%**	**67.67%**	**54.83%**
Team-assigned Weights	RV	3.14%	3.87%	9.9%	6.72%
	IC	21.49%	39.38%	28.47%	35.46%
	CP	**75.37%**	**56.75%**	**61.63%**	**57.81%**

The results show that CP is the biggest contributor for all four projects for all factor weight allocations except for one project (Project 2) of the *contradictory variance-based* allocation technique in which the implementation complexity is the largest contributor. For *mean-based* allocation of factor weights, customer priority has the highest contribution (at least over 55%) in the overall PFV. The second highest contributor for *mean-based* weight allocation is implementation complexity, followed by requirements volatility, which contributed the least for this project. These results are in agreement with what we achieved via PORT application (as discussed in Section 5.2). The *equal* weight approach also showed customer priority to be the biggest contributor, followed by implementation complexity. Based on these results, mean-based approach of factor weights results in the highest rate of failure defection, and we will use the mean-based approach to factor weights in our future case studies.

7. Comparison of SRET and PORT

In this section we compare PORT scheme to Musa's SRET approach. We applied Musa's SRET approach of test selection to the same four student projects. SRET involves choosing test cases corresponding to the operational profile and then executing the chosen test cases in random order. The operational profile for the requirements was identified by the students and the research team collectively during a class period. We modified the existing test suite such that the number of test cases selected for execution for each requirement is proportional to the operational profile for the requirement. For example, if the requirement R_1 had an operational profile of 5%, the number of test cases that mapped to requirement R_1 in a test suite of 100 test cases is five.

Once the new set of test cases was identified, we generated 20 random permutations of this new set for statistical comparison. We computed the WPFD for each permutation. Furthermore, we computed the WPFD by applying the PORT scheme towards this revised (new) test set. We compared the WPFD for the 20 random permutations with the WPFD for PORT scheme and the results are shown in Table 4.

A sign-test was applied to determine statistical significance. We find statistically significant results in favor of PORT scheme in comparison to SRET ($p < 0.001$). A sign test indicated that the median WPFD of all permutations of test cases chosen according to SRET is less than the observed WPFD for the PORT scheme.

Table 4: WPFD for 20 SRET Random Sets and PORT TCP

SRET Random Set #	Team 1: WPFD	Team 2: WPFD	Team 3: WPFD	Team 4: WPFD
1	48.41	54.05	51.88	53.93
2	45.72	52.77	50.29	43.14
3	41.43	65.11	50.05	53.08
4	54.34	65.70	54.29	60.28
5	57.13	68.74	60.94	57.42
6	49.36	44.23	39.29	47.01
7	48.43	52.54	59.29	45.09
8	57.94	55.46	46.52	50.40
9	52.78	51.90	53.88	54.24
10	42.76	51.54	52.58	55.59
11	52.80	57.92	62.29	46.47
12	49.92	50.43	59.70	45.47
13	51.83	33.24	45.52	43.64
14	44.30	47.28	43.76	48.64
15	55.09	57.98	46.11	41.94
16	46.94	62.71	53.88	52.01
17	50.01	45.16	49.88	38.05
18	49.32	51.02	47.58	50.53
19	58.69	45.40	55.05	54.52
20	47.41	44.22	51.82	53.89
MEAN WPFD-Random TCP	**50.23**	**52.87**	**51.73**	**49.77**
# of Times PORT is better than Random order	20	20	19	17
PORT TCP: WPFD	**61.68**	**73.09**	**62.21**	**54.15**
Sign Statistic	**20**	**20**	**19**	**17**
sign-test p-value	**<0.0001**	**<0.0001**	**<0.0002**	**=0.0013**

8. Summary

In this paper we propose the PORT scheme for prioritizing system level test cases to improve the rate of detection of severe failures. We use three prioritization factors in our PORT (v 1.0) scheme: requirements volatility, developer-perceived implementation complexity and customer priority. PORT (v 1.0) was applied to four similar student team projects that were developed in an advanced graduate software-testing course. In our next version of PORT (v 2.0), we have incorporated fault proneness as the fourth factor where fault proneness would apply to requirements that are currently in a release product.

Our validation results for PORT (v 1.0) involved evaluating the effectiveness of PORT scheme towards meeting our research goals: (1) improve the rate of detection of severe failures, and (2) assess the contribution of prioritization factors. The results indicate that PORT scheme leads to significant improvement in rate of detection of severe failures in comparison to random ordering of test cases for the four projects. The results also show that CP is the biggest contributor towards the effectiveness of PORT followed by IC. These results suggest that the PORT scheme could improve the effectiveness of testing activities by focusing on: (1) functionalities that are of highest value to the customer, and (2) improving the rate of detection of severe failures. Rectifying severe failures earlier is believed to improve customer-perceived software quality, thereby increasing the overall business value provided to the customers. We conducted sensitivity analysis on factor weights to analyze whether the factor weights can sway the prioritization provided by the PORT algorithm such that the software engineer is misguided. We considered four possible allocations for factor weights and the results indicated that the mean-based approach to assigning weights yielded the highest WPFD. Additionally, Musa's SRET approach of test selection was applied to the same four student projects. The PORT scheme was compared with 20 random permutations of test case ordering on the test suite selected based on SRET technique. The WPFD for the 20 random permutations of test set selected via SRET was compared with the WPFD for PORT scheme and the results were statistically significant in favor of PORT scheme in comparison to SRET ($p < 0.01$).

Prioritization at the system level can also be beneficial because the PORT scheme requires the team to conduct system analysis and to write concrete test cases. The act of writing concrete test cases immediately after requirements specification can lead to the identification of ambiguous and unclear requirements, allowing requirements errors to be identified and rectified earlier. The PORT scheme allows the engineering team to monitor the requirements covered in system test; the ability to monitor requirements covered in system test is believed to be one of the challenges faced by the industry [11]. PORT is being applied to industrial projects at companies including ABB, I-Cubed and Tekelec.

References

1. S. Amland, "Risk Based Testing and Metrics," presented at 5th International Conference EuroSTAR ’99, Barcelona, Spain, 1999.
2. B. Beizer, *Software Testing Techniques*. New York, NY: Van Nostrand Reinhold, 1990.
3. B. Boehm, "Value-Based Software Engineering," *ACM Software Engineering Notes*, vol. 28, pp. 1-12, March 2003.
4. B. Boehm and L. Huang, "Value-Based Software Engineering: A Case Study," *IEEE Computer*, vol. 36, pp. 33-41, March 2003.
5. Y. Chen, R. Probert, and D. Sims, "Specification based Regression Test Selection with Risk Analysis," presented at Conference of the Center for Advanced Studies on Collaborative Research, Ontario, Canada, 2002.
6. R. Craig and S. Jaskiel, *Systematic Software Testing*. Norwood, MA: Artech House Publishers, 2002.
7. S. Elbaum, A. Malishevsky, and G. Rothermel, "Prioritizing Test Cases for Regression Testing," *Proceedings of the ACM International Symposium on Software Testing and Analysis*, vol. 25, pp. 102-112, August 2000.
8. S. Elbaum, A. Malishevsky, and G. Rothermel, "Test Case Prioritization: A Family of Empirical Studies," *IEEE Transactions on Software Engineering*, vol. 28, pp. 159-182, February, 2002.
9. S. Elbaum, A. Malishevsky, and G. Rothermel, "Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization," presented at 23rd International Conference on Software Engineering, Ontario, Canada, May 2001.
10. M. Harrold, "Testing: A Roadmap," presented at International Conference on Software Engineering, Limerick, Ireland, 2000.
11. P. Hsia, A. M. Davis, and D. C. Kung, "Status report: Requirements Engineering," *IEEE Software*, vol. 10, pp. 75-79, November 1993.
12. IEEE, "IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology," 1990.
13. C. Jones, "Software Challenges: Strategies for Managing Requirements Creep," *IEEE Computer*, vol. 29, pp. 92 - 94, June 1996.
14. J. Karlsson and K. Ryan, "A Cost-Value Approach for Prioritizing Requirements," *IEEE Software*, vol. 14, pp. 67-74, Sep-Oct 1997.
15. Y. K. Malaiya and J. Denton, "Requirements volatility and defect density," presented at 10th Intl' Symposium on Software Reliability Engineering, Boca Ratan, Florida, November 1999.
16. G. Mogyorodi, "Requirements-Based Testing: An Overview," presented at 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems, Santa Barbara, California, August 2001.
17. F. Moisiadis, "Prioritising Use Cases and Scenarios," presented at 37th International Conference on Technology of OO Languages and Systems, Sydney, NSW, 2000.
18. J. C. Munson and S. Elbaum, "Software reliability as a function of user execution patterns and practice," presented at 32nd Annual Hawaii International Conference of System Sciences, Maui, HI, 1999.

19. J. Musa, *Software Reliability Engineering*. New York, NY: McGraw-Hill, 1999.
20. J. Musa, "Software-Reliability-Engineered Testing," *IEEE Computer*, vol. 29, pp. 61-68, November 1996.
21. J. O'Neal and D. Carver, "Analyzing the Impact of Changing Requirements," presented at IEEE International Conference on Software Maintenance, Los Alamitos, California, 2001.
22. T. Ostrand, E. Weyuker, and R. Bell, "Where the Bugs Are," presented at Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis, Boston, MA, July 2004.
23. G. Rothermel and M. Harrold, "Selecting Tests and Identifying Coverage Requirements for Modified Software," presented at ACM International Symposium on Software Testing and Analysis, Seattle, WA, August 1994.
24. G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Test Case Prioritization," *IEEE Transactions on Software Engineering*, vol. 27, pp. 929-948, October, 2001.
25. G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Test Case Prioritization: An Empirical Study," presented at International Conference on Software Maintenance, Oxford, UK, September 1999.
26. F. Shull, V. Basili, B. Boehm, W. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, and M. Zelkowitz, "What We Learned about Fighting Defects," presented at IEEE Symposium on Software Metrics, Ottawa, Canada, June 2002.
27. H. Srikanth, "Requirements Based Test Case Prioritization," presented at Student Research Forum in 12th ACM SIGSOFT Int’l Symposium on the Foundations of Software Engineering, Newport Beach, California, 2004.
28. H. Srikanth and L. Williams, "Requirements Based Test Case Prioritization," presented at Poster Session and Ph. D Forum in Grace Hopper Celebration of Women in Computing, Chicago, Illinois, 2004.
29. H. Srikanth, "Requirements-Based Test Case Prioritization," presented at Doctoral Symposium in International Conference of Software Engineering, St. Louis, 2005.
30. H. Srikanth and L. Williams, "On Economic Benefits of System Level Test Case Prioritization," presented at International Conference on Software Engineering, St. Loius, MO, 2005.
31. H. Srikanth, L. Williams, and J. Osborne, "System Test Case Prioritization of New and Regression Tests," presented at International Symposium of Empirical Software Engineering, Noosa Heads, Australia, 2005.
32. Standish.Group, "CHAOS." http://www.standishgroup.com/chaos.htm.
33. L. Tahat, B. Vaysburg, B. Korel, and A. Bader, "Requirement-Based Automated Black-Box Test Generation," presented at 25th Annual International Computer Software and Applications Conference, Chicago, Illinois, 2001.
34. M. Vouk and A. T. Rivers, "Construction of Reliable Software in Resource-Constrained Environments," in *Case Studies in Reliability and Maintenance*, W. R. Blischke and D. N. P. Murthy, Eds. Hoboken, NJ: Wiley-Interscience, John Wiley and Sons, 2003, pp. 205-231.
35. E. Wong, J. Horgan, M. Syring, W. Zage, and D. Zage, "Applying design metrics to predict fault-proneness: a case study on a large-scale software system," *Software Practice and Experience*, vol. 30, pp. 1587-1608, 2000.