System Test Case Prioritization of New and Regression 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

Abstract

Test case prioritization techniques have been shown to be beneficial for improving regression-testing activities. With prioritization, the rate of fault detection is improved, thus allowing testers to detect faults earlier in the system-testing phase. Most of the prioritization techniques to date have been code coverage-based. These techniques may treat all faults equally. We build upon prior test case prioritization research with two main goals: (1) to improve user-perceived software quality in a cost effective way by considering potential defect severity and (2) to improve the rate of detection of severe faults during system-level testing of new code and regression testing of existing code. We present a value-driven approach to system-level test case prioritization called the Prioritization of Requirements for Test (PORT). PORT prioritizes system test cases based upon four factors: requirements volatility, customer priority, implementation complexity, and fault proneness of the requirements. We conducted a PORT case study on four projects developed by students in advanced graduate software testing class. Our results show that PORT prioritization at the system level improves the rate of detection of severe faults. Additionally, customer priority was shown to be one of the most important prioritization factors contributing to the improved rate of fault 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 [16]. Quality can be measured by the customer satisfaction with the resulting system based on the requirements that are incorporated successfully in the system [16]. Boehm proposes a value-based approach to software engineering that measures the value the system provides to the prospective customers [6]. 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. 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 [5, 6]. In this paper, we explore a value-driven approach to prioritizing software system test with the objective of improving user-perceived software quality. Software testing is a strenuous and expensive process [4, 8]. Research has shown that at least 50% of the total software cost is comprised of testing activities [12, 31]. Companies are often 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 and is forced to deliver the system with compromised software quality.

To optimize the time and cost spent on testing, prioritization of test cases in a test suite can be beneficial [2, 3, 9, 10, 24, 25]. Test case prioritization (TCP) involves the explicit planning of the execution of test cases in a specific order with the intention of increasing the effectiveness of software testing activities by improving the rate of fault detection earlier in the software process [24, 25]. TCP has been primarily applied to improve regression testing efforts [9, 10, 24, 25]. Regression testing is the process of retesting of a system or component to verify that changes made to the system code have not caused unintended effects and that the system is still compliant with the specified requirements [14]. Software engineers save test cases and re run these test cases as regression tests in later versions. Running the entire set of test cases on a new and/or revised version could be expensive. 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 [7].

We extend the code-coverage TCP techniques [24, 25] and apply TCP at a system-level for both new and regression tests. *The objective of this research is to* *develop and validate a system-level test case prioritization scheme to reveal severe faults earlier and to improve customer-perceived software quality.* We build upon the current TCP techniques [9, 10, 24, 25] and propose Prioritization of Requirements for Testing (PORT Version 1.1). PORT can be used to prioritize system-level black box test when traceability between requirements, test case, and test/field failures is maintained by the development team. The PORT algorithm is based upon four prioritization factors: (1) customer-assigned priority of requirements, (2) developer-perceived implementation complexity, (3) requirements volatility, and (4) fault proneness of requirements. PORT Version 1.0 [27-29] incorporated only three prioritization factors; PORT (Version 1.1) involves incorporating the fourth prioritization factor: fault proneness. We believe that directing test efforts using these four factors would enable more efficient identification of severe faults earlier in the software process. By focusing on customer-assigned priority, we aim to identify requirements, which would increase customer-perceived software quality. We hypothesize that applying test case prioritization by incorporating knowledge gained in the requirements phase would allow the team to decrease testing cost by reducing the likelihood of the requirements errors appearing in the field.

To determine the effectiveness of PORT Version 1.0, we conducted a two-phased feasibility study in four similar student projects developed in a senior graduate software testing class at North Carolina State University. Pairs of students developed a TCP application with an average size of approximately 2500 lines of Java code (LOC). Students turned in two final versions of their application (one good version and one with 20 injected defects) and tested other teams' applications. In the first phase of the study, we analyze the defects found by the students on other teams' projects. In the second phase, we analyze the ability of PORT to enable earlier discovery of the injected defects. We discuss these results in Section 5.

This rest of this paper is structured as follows. Section 2 gives a high-level overview of TCP. Section 3 presents the PORT technique. Sections 4 and 5 discuss the PORT validation algorithm and the case study conducted. Section 6 presents the summary and future work.

2. Related Work

This section describes code coverage-based TCP strategies [9, 10, 24, 25] and their benefits. Coverage-based TCP techniques involve ranking test cases based on the coverage they provide. For prioritized statement coverage, 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. Some of the other coverage techniques involve branch and function coverage. For branch and function coverage, tests are prioritized based on the program branches or program functions covered respectively.

The benefits of code coverage-based strategies were measured using a weighted Average of the Percentage of Faults Detected (APFD) [24]. 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 y-axis of a graph, and percentage of test suite run on x-axis of a graph. We analyze a similar validation metric to assess the efficacy of PORT, Weighted Percentage of Faults Detected (WPFD), as will be discussed in Section 5.

Several case studies demonstrated the benefits of code coverage-based TCP strategies [9, 10, 24, 25]. The researchers used various prioritization techniques to measure the APFD values and found statistically significant results that APFD values were not the same for all of the techniques. The code coverage-based TCP strategies were shown to improve the rate of fault detection, allowing the testing team to start debugging activities earlier in the software process and resulting in faster software release than otherwise possible [24].

If all faults are not equally severe, severity-neutral TCP strategies and associated APFD metric can provide misleading information [11]. As a result, Elbaum et al. [10, 24] incorporated fault severity in a TCP strategy [11]. Instead of representing *Test Suite Fraction* in the horizontal axis (as done for APFD), *Percentage of Total Test Case Cost Incurred* (as done for $APFD^c$) is represented. Additionally, instead of representing *Percent Faults Detected* on the vertical axis (as for APFD), *Percentage Total Fault Severity Detected* (as done for $APFD^c$) is represented. The $APFD^c$ measures the unit-of-fault-severity-detected-per-unit-test-cost [11]. The use of $APFD^c$ is primarily done for assessing the prioritization orders post hoc, i.e. when the severity and cost values are known. Elbaum et al. use six levels of fault severity where they use a linear approach as one of the ways to assign fault severity values. *Linear* approach involves assigning a severity value ranging from 1 to 6 to the faults. Another approach, *exponential*, involves assigning severity values from 2^0 through 2^5. We follow a similar approach of measuring the Total Severity of Faults Detected for an ordered test suite. We apply four levels of severity and use a 10-point scale to assign severity values to faults as discussed in Section 4.

3. Prioritizations of Requirements for Testing

Building on this work, we propose a multi-faceted, system-level prioritization technique called PORT. In this section, we explain the current set of PORT prioritization factors, the PORT prioritization factor collection process, and the algorithm underlying PORT.

3.1 Prioritization Factors

The algorithm underlying PORT Version 1.1 prioritizes based upon four factors: (1) customer-assigned priority on requirements, (2) requirement volatility, (3) developer-perceived implementation complexity, and (4) fault proneness of requirements. We discuss below these four factors, the reasoning of why they were chosen in our prioritization technique, and their importance to software testing.

•***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 sometimes or always used [19]. 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 [20, 21]. A focus on customer requirements for development has been shown to improve customer-perceived value and satisfaction [5, 6, 16]. By identifying and thoroughly testing the fraction of requirements that would be of highest importance to the customer, we aim to increase the business value generated to the customer. Additionally, if the testing efforts were stopped abruptly due to schedule pressures, the requirements of highest value to the customer would have been tested early and thoroughly.

•***Requirements volatility*** (RV) is based on the number of times a requirement has been changed during the development cycle. If a requirement has changed more than ten times, the volatility values for all requirements are quantified on a 10-point scale. Requirements volatility is an assessment of the requirements change once the implementation begins [17].

Reasoning: Roughly 50% of all faults identified in a project are errors introduced in the requirements phase [18]. 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 [30]. The biggest cause for project failures happens to be lack of user input, and changing or incomplete requirements [15, 18, 22]. Roughly 25% of the requirements for an average project change before project completion [22], and volatile requirements tend to make the testing activities difficult and cause the software to contain high defect density [17]. Changing requirements result in re-design, the addition or deletion of existing functions, and often an increase in the fault density in the program [17]. Severe defects that escape into the field can cost 100 times more to fix after delivery than correcting the problem in the requirements phase [26]. For non-severe defects, the ratio is 2:1 for fixing the defect in the field as opposed to pre-delivery [26].

•***Implementation complexity*** (IC) is a subjective measure of how difficult the development team perceives the implementation of requirement to be. 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: Requirements with high implementation complexity tend to have a higher number of faults [1]. 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 modules result in 80% of the faults [26, 32], and roughly 50% of the modules are defect free [26].

•***Fault proneness*** of requirements (FP) allows the development team to identify the requirements which have had customer-reported failures. 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 and in-house system test 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: Research has shown that fault-prone modules are more likely to cause field failures than the modules, which are not fault prone [20].

Through our research, we will examine and refine the set of PORT prioritization factors. Our goal is to identify the minimum set of factors which can be used to effectively prioritize system-level test cases.

3.2 PORT Prioritization Factor Collection Process

The process for collection and updating the PORT factors is shown in Figure 1. There are five stakeholders in the process and a system for providing automatic updates. (An open source tool is available for automatic updates). The roles of each of these stakeholders are defined below:

The *customer* states the system requirements, the priority of the each requirement, and any additions or changes to the requirements throughout development. The customer also reports field failures when the product has been delivered.

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* 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.

The *PORT tool* increments requirements volatility when a requirement is changed or added late and increments fault proneness when a test failure or field failure is reported. In the absence of a tool, these values can be tracked manually.

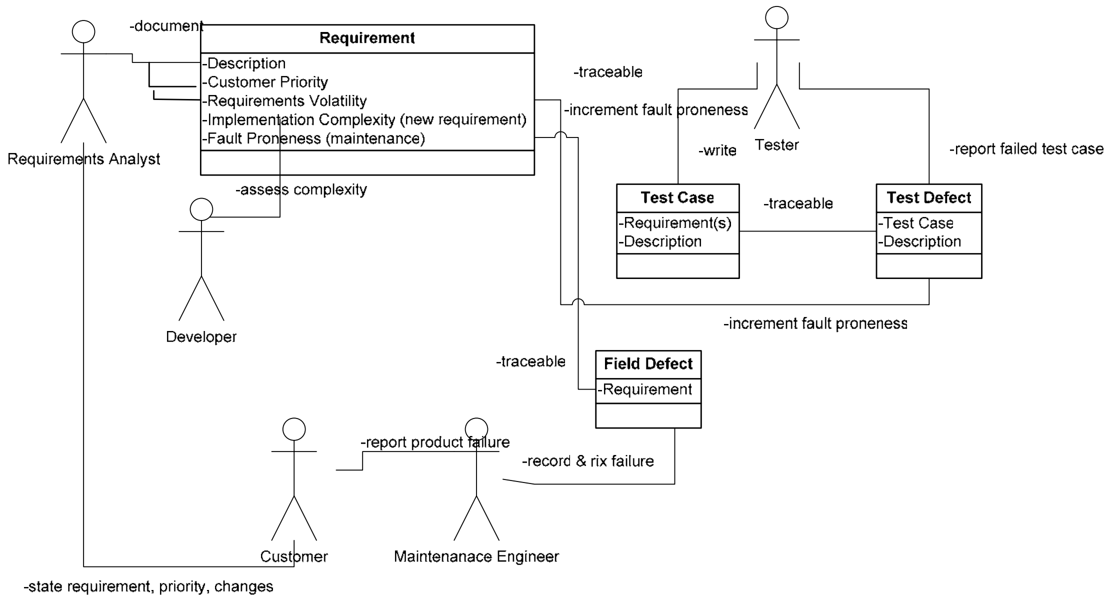


Figure 1: PORT Prioritization Factor Collection Process

3.3 PORT Algorithm

Factor values are assigned during the design analysis phase, and evolve continually during the software development process as the project evolves. Based on the project and customer needs, the development team assigns weight to the prioritization factor such that the assigned total weight (1.0) is divided amongst 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 example, if the requirements for a project have been very stable, then the development team might assign RV a relatively smaller portion of the total weight. A default value can be assigned, giving each factor equal weight.

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

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

PFV_i represents 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 value-matrix representation of PFV for requirements is shown in Equation 2 below where PFV (*P*) is the product of value (*V)* and weight (*w*).

$$P = Vw$$

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

The computation of PFVi for a requirement is used in computing the Weighted Priority (WP) of its associated test cases. WP of the test case is the product of two elements: (1) the average PFV of the requirement(s) the test case maps to and (2) the requirements-coverage a test case provides. Requirements coverage is the fraction of the total project requirements exercised by a test case. Let there be n total requirements for a product/release, and test case j maps to i requirements. WPj is an indication of the priority of running a particular test case. WPj is represented by the Equation 3.

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

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. Validation Algorithm

Refinement and validation of PORT will proceed via the analysis of the severity of faults detected for a product. For analysis purposes, each failure is assigned a severity value (SV) on a 10-point scale as shown below:

- Highly severe (Severity 1): SV value of 10;
- Medium severe (Severity 2), SV of 6;
- Less severe (Severity 3), SV of 4; and
- Least severe (Severity 4), SV of 2.

Total Severity of Faults Detected (TSFD) is the summation of severity values of all faults identified for a release. Equation 4 shows TSFD for a product/release, where t represents total number of faults identified for the product/release.

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

The Average Severity of Faults Detected (ASFD) is computed for each requirement to analyze if the requirement with a higher computed PFV actually had higher average severe faults when the product was system tested or used by the customer. The ASFD for requirement i (ASFDi) is the ratio of the summation of severity values of faults identified for that requirement divided by the TSFD. The computation of ASFD is shown in Equation 5, where m is the number of faults mapped to requirement i.

$$ASFD_i = \left[\frac{\sum_{j=1}^{m} SV_j}{TSFD} \right] \qquad (5)$$

ASFD is used to analyze the effectiveness of PORT technique by mapping the total percentage of severe faults identified for a requirement to the PFV for that requirement. The case study showing the use of ASFD is discussed in Section 5.

5. PORT Feasibility Study

To measure the effectiveness of PORT (Version 1.0), four similar projects with average size of approximately 2500 lines of Java code (LOC) were analyzed. The projects were developed by 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). The factor weights were assigned based on discussion between the customer and students.

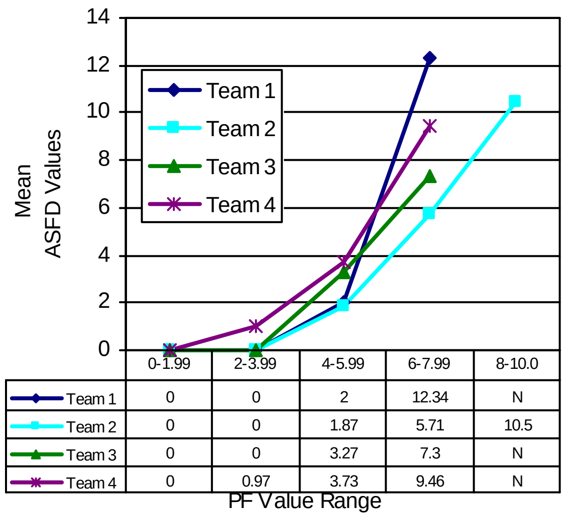
The student projects were analyzed in two phases. In the first phase we evaluate the faults identified by external testers and map the ASFD for requirement to their respective PFV. The results are presented in Section 5.1. In the second phase, the research team tested the four systems by applying PORT technique and via a random prioritization strategy to measure the effectiveness of PORT technique. The results of this phase are discussed in Section 5.2.

5.1 Phase I

At the completion of the project, each student group tested two other student projects by using their own black box system tests. Each group acted as an external, objective test team, thus improving the objectivity of the results. When the testers identified a failure, they assigned a severity level to the failure. On average, 15 faults were identified for each project. These defects were analyzed to determine which requirement/s the defects mapped to. The PFV and ASFD for each of the 19 requirements and the TSFD was computed for each project.

The requirements were grouped into one of five categories based on the range of PFV for that requirement where a lower PFV range indicates a lower priority for this requirement to be tested. For each PFV range, the mean ASFD values for requirements in that range were computed for each project. Figure 2 shows the results for the four projects. The curves indicate that for all four team projects, as the PFV range increases, the ASFD values increase as well.

For Team 2, the average severity was highest for the PFV range of 8-10. For the other three teams (1, 3, and 4), the average severity was highest for the PFV range of 6-7.99. For these three teams, there were not any requirements that mapped to the PFV range of 8-10. This change in PFV for the requirements among teams is a result of the different value-set assigned for IC and RV factors. So the highest possible range for these three teams (1, 3 and 4) was 6-7.99 because of the lower values of IC and RV assigned by the students. The results, as depicted in Figure 2, show that requirements with higher PFV range have higher ASFD values, and vice versa [28, 29].



	0-1.99	2-3.99	4-5.99	6-7.99	8-10.0
Team 1	0	0	2	12.34	N
Team 2	0	0	1.87	5.71	10.5
Team 3	0	0	3.27	7.3	N
Team 4	0	0.97	3.73	9.46	N

Figure 2: Comparison of ASFD and PFV Range

5.2 Phase II

In the second phase of the study, we compared the PORT technique with a random prioritization strategy towards testing student projects. As an extra credit assignment, the students were asked to create a faulty version of their project by injecting 20 faults. The students were instructed that for each project, at least 50% of the faults to be of Severity 1 and 2, and the other 50% comprised of Severity 3 faults.

To test these four student projects, approximately 50 system test cases were written for the student projects by the lead researcher. 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 considering if the test cases were run in the order prescribed by PORT and if they were run in random order. The goal was to identify the effectiveness of PORT in improving the rate of detection of severe faults 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 faults.

•Contribution of prioritization factors towards the effectiveness of PORT.

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

5.2.1. Rate of Detection of Severe Faults

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

Setup: Using the values and weights for the prioritization factors provided by students and the research team, the PFV for the 19 requirements was computed. Based on the PFV of the requirements and the requirements coverage provided by the system test cases, WP value was computed for each test case. The PORT technique calls for execution of the test cases in the descending order of the WP value. The random strategy involved a random permutation of test cases for execution.

The four faulty applications were tested by running the system test cases to find the induced faults in the application. The faults 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 fault identified was noted. After all test cases were executed and all the induced faults were identified and their SV noted, analysis was done to measure the rate of fault detection. A graph was plotted with fraction of test suite executed on x-axis, and percentage of TSFD on y-axis. The rate of detection of faults is computed using a metric called *Weighted Percentage of Faults Detected* (WPFD), which is the area under the curve when plotting a graph with percentage of TSFD on the y-axis and fraction of test suite executed on the x-axis. 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 technique. We applied statistical analysis 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-fourth and all the test cases. For the purposes of depicting the results graphically, we show a sample comparison of a WPFD for a random run and WPFD for PORT for a team in Figure 3. For this comparison the WPFD for PORT is better than WPFD for the chosen random run.

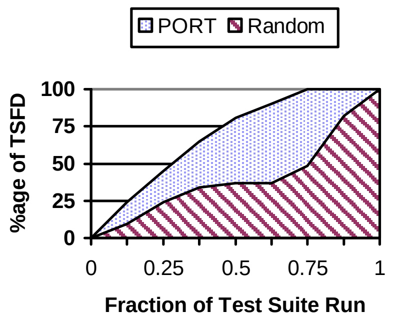


Figure 3: WPFD for PORT and Random

The WPFD results achieved for 20 random sets were used to compare the effectiveness of PORT by comparing the mean WPFD of 20 permutations with WPFD for PORT. The PORT technique involves generating a repeatable prioritized ordering; thus, we have only one set of PORT WPFD results. The mean WPFD values for Random TCP (for n=20) are listed in

Table 2. To investigate the effectiveness of PORT, the following null and alternative hypotheses were considered:

H_0: WPFD for PORT = Mean WPFD for Random TCP

H_a: WPFD for PORT > Mean WPFD for Random TCP

Table 2: WPFD for PORT and Random

	Team 1		Team 2		Team 3		Team 4	
PFV Range	A	R	A	R	A	R	A	R
8-10	N	N	**51.7**	**26.3**	N	N	9.1	5.3
7-7.99	**43**	**15.8**	**48.3**	**26.3**	**63**	**36.8**	**46.3**	**36.8**
6-6.99	30.6	21	0	10.5	22.5	21	**24**	**26.3**
5-5.99	26.4	36.9	0	15.8	14.6	26.3	8.3	10.5
4-4.99	0	5.3	0	5.3	0	5.3	4.1	5.3
0-3.99	0	21.1	0	15.8	0	10.5	8.3	15.8

R – Percentage of Total Requirements in PFV Range

A- Average Severity of Faults Detected in Percentage

N – N/A: N represents that zero requirements mapped to that particular PFV range, and therefore the ASFD for that range is Not Available.

Random TCP Set #	Team 1: WPFD	Team 2: WPFD	Team 3: WPFD	Team 4: WPFD
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	**67.12**	**83.49**	**63.41**	**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**

We find statistically significant results in favor of H_a, PORT is better than random prioritization ($p < 0.001$). The results indicate that PORT strategy leads to improved rate of severe fault detection for all four projects. The difference between PORT and mean for these twenty permutations is significant ($p < 0.001$).

Alternatively, the sign test may be used to investigate the null hypothesis that the WPFD for is no better than that for a randomly chosen prioritization. The sign test is a nonparametric procedure that makes no assumptions about the distribution of WPFD. For example, in the 20 randomly chosen prioritizations for Team 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. The last row of Table 2 gives the results for a sign-test for each of the four teams

5.2.2. Analysis of PORT Factors

Goal: This section involves: (1) investigating whether requirements with higher ASFD originate from requirements with higher range of PFV, and (2) identifying the mean contribution of the prioritization factors towards the PFV of the requirements.

Setup: The ASFD for each requirement provides a measure of the severity of faults detected from that requirement. For all four projects while applying PORT, the faults identified were mapped back to their respective requirements, and the ASFD for each requirement was computed. For all four projects, we computed the contribution of each of the three prioritization factors towards the PFV for each of the nineteen requirements and the mean contribution of the over all project requirements.

Table 3: **ASFD and % of Total Requirements**

Results: This section discusses the results of the effectiveness of the prioritization factors in the PORT technique. The results are divided in two parts: (1) Table 3 shows the mapping of ASFD values and PFV for requirements for all four projects, and (2) Figure 4 shows the contribution of the prioritization factors towards PFV of the requirements for all projects. The requirements are grouped in the PFV range as shown in Table 3.

For each of the PFV range of requirements, the mean ASFD values for requirements in that range is computed. The results presented in Table 3 show that the higher percentage of ASFD originates from requirements with PFV range of 6 or higher. For example for Team 2, 100% of the ASFD originates from requirements with PFV range greater than 7, although roughly only 50% of the total requirements fall under the range of 7 or higher.

The mean contribution results for all three prioritization factors are represented in Figure 4. The CP was ranked as the biggest contributor for all four projects, followed by IC and RV. On average customer priority 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 implementation complexity.

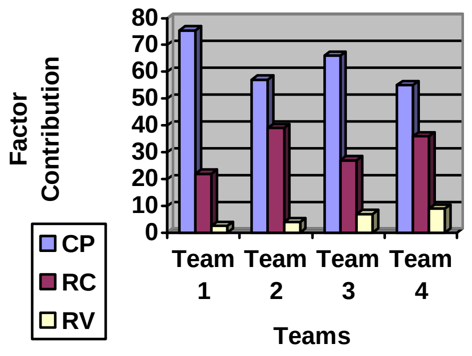


Figure 4: Contribution of Prioritization Factors

The least contribution came from requirements volatility, 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 implementation. Due to the stable nature of the student projects, these results do not induce us to remove the RV factor from future case studies of PORT; RV is likely to be a significant factor in industrial projects.

6. Acknowledgements

We would like to thank the North Carolina State University (NCSU) Software Engineering Realsearch Group for their helpful suggestions on this paper. This material is based upon work partially supported by the National Science Foundation under CAREER Grant No. 0346903. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

7. Summary

In this paper we propose the PORT technique for prioritizing system level test cases to improve the rate of fault detection of severe faults. We use three prioritization factors in our PORT (Version 1.0): requirements volatility, implementation complexity and customer priority. PORT (Version 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 (Version 1.1), we will incorporate FP as the fourth factor where FP would apply to requirements that are currently in a release product. For these systems with second or higher release, the requirements, which were part of a prior release, would be analyzed for FP while the newly-introduced requirements for that current release would be evaluated for IC.

Our initial results for PORT (Version 1.0) originate from a two-phase feasibility study. The first phase involved analyzing faults identified from four systems as a result of peer testing. The second phase involved executing approximately 50 system test cases on four faulty applications to identify rate of fault detection of induced faults.

The Phase 1 results indicted that on average higher severity faults were mapped to requirements with higher range of PFV. The results achieved in this phase motivated us to apply PORT for test suite planning and execution of four team projects.

In the Phase 2 of our feasibility study, we evaluate the effectiveness of PORT technique towards meeting our research goals: (1) improve the rate of detection of severe faults, and (2) assess the contribution of prioritization factors. The Phase 2 results indicate that PORT technique leads to improved rate of detection of severe faults in comparison to random ordering of test cases at a statistically significant level for the four projects. Phase 2 results also show that CP is the biggest contributor towards the effectiveness of PORT followed by IC. These results are supportive that the PORT technique could improve the effectiveness of testing activities by focusing on: (1) functionality that is of highest value to the customer, and (2) improving the rate of detection of severe faults. Rectifying severe faults is believed to improve customer-perceived software quality.

Prioritization at the system level can also be beneficial because the PORT technique 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 technique 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 [13]. Future studies will be conducted on industrial projects.

8. References

[1] S. Amland, "Risk Based Testing and Metrics," 5th International Conference EuroSTAR ’99, Barcelona, Spain, 1999, pp. 1-20.

[2] F. Basanieri, A. Betolino, and E. Marchetti, "CoWTeSt: A Cost Weighed Test Strategy," Escom-Scope 2001, London, England, April 2001, pp. 387-396.

[3] F. Basanieri, A. Betolino, and E. Marchetti, "The Cow_Suite Approach to Planning and Deriving Test Suites in UML Projects," Fifth International Conference on the Unified Modeling Language - the Language and its applications UML 2002, Dresden, Germany, September 2002, pp. 383-397.

[4] B. Beizer, *Software Testing Techniques*. New York, NY: Van Nostrand Reinhold, 1990.

[5] B. Boehm, "Value-Based Software Engineering," *ACM Software Engineering Notes*, vol. 28, no. 2, pp. 1-12, March 2003.

[6] B. Boehm and L. Huang, "Value-Based Software Engineering: A Case Study," *IEEE Computer*, vol. 36, no. 3, pp. 33-41, March 2003.

[7] Y. Chen, R. Probert, and D. Sims, "Specification based Regression Test Selection with Risk Analysis," Conference of the Center for Advanced Studies on Collaborative Research, Ontario, Canada, 2002, pp. 1-14.

[8] R. Craig and S. Jaskiel, *Systematic Software Testing*. Norwood, MA: Artech House Publishers, 2002.

[9] 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, no. 5, pp. 102-112, August 2000.

[10] S. Elbaum, A. Malishevsky, and G. Rothermel, "Test Case Prioritization: A Family of Empirical Studies," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159-182, February, 2002.

[11] S. Elbaum, A. Malishevsky, and G. Rothermel, "Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization," 23rd International Conference on Software Engineering, Ontario, Canada, May 2001, pp. 329-338.

[12] M. Harrold, "Testing: A Roadmap," International Conference on Software Engineering, Limerick, Ireland, 2000, pp. 61-72.

[13] P. Hsia, A. M. Davis, and D. C. Kung, "Status report: requirements engineering," *IEEE Software*, vol. 10, no. 6, pp. 75-79, November 1993.

[14] IEEE, "IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology," 1990.

[15] C. Jones, "Software Challenges: Strategies for Managing Requirements Creep," *IEEE Computer*, vol. 29, no. 6, pp. 92 - 94, June 1996.

[16] J. Karlsson and K. Ryan, "A Cost-Value Approach for Prioritizing Requirements," *IEEE Software*, vol. 14, no. 5, pp. 67-74, Sep-Oct 1997.

[17] Y. K. Malaiya and J. Denton, "Requirements volatility and defect density," 10th Intl' Symposium on Software Reliability Engineering, Boca Ratan, Florida, November 1999, pp. 285-298.

[18] G. Mogyorodi, "Requirements-Based Testing: An Overview," 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems, Santa Barbara, California, August 2001, pp. 286-295.

[19] F. Moisiadis, "Prioritising Use Cases and Scenarios," 37th International Conference on Technology of OO Languages and Systems, Sydney, NSW, 2000, pp. 108-119.

[20] J. C. Munson and S. Elbaum, "Software reliability as a function of user execution patterns and practice," 32nd Annual Hawaii International Conference of System Sciences, Maui, HI, 1999, pp. 255-285.

[21] J. Musa, *Software Reliability Engineering*. New York, NY: McGraw-Hill, 1999.

[22] J. O'Neal and D. Carver, "Analyzing the Impact of Changing Requirements," IEEE International Conference on Software Maintenance, Los Alamitos, California, 2001, pp. 190-195.

[23] G. Rothermel and M. Harrold, "Selecting Tests and Identifying Coverage Requirements for Modified Software," ACM International Symposium on Software Testing and Analysis, Seattle, WA, August 1994, pp. 169-184.

[24] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Test Case Prioritization," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929-948, October, 2001.

[25] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Test Case Prioritization: An Empirical Study," International Conference on Software Maintenance, Oxford, UK, September 1999, pp. 179 - 188.

[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," IEEE Symposium on Software Metrics, Ottawa, Canada, June 2002, pp. 249-258.

[27] H. Srikanth, "Requirements Based Test Case Prioritization," Student Research Forum in 12th ACM SIGSOFT Int'l Symposium on the Foundations of

Software Engineering, Newport Beach, California, 2004, pp.

[28] H. Srikanth, "Requirements-Based Test Case Prioritization," Doctoral Symposium in International Conference of Software Engineering, St. Louis, MO, 2005, pp.

[29] H. Srikanth and L. Williams, "On Economic Benefits of System Level Test Case Prioritization," International Conference on Software Engineering, St. Loius, MO, 2005, pp.

[30] Standish.Group, "CHAOS." http://www.standishgroup.com/chaos.htm.

[31] L. Tahat, B. Vaysburg, B. Korel, and A. Bader, "Requirement-Based Automated Black-Box Test Generation," 25th Annual International Computer Software and Applications Conference, Chicago, Illinois, 2001, pp. 489-495.

[32] 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, no., pp. 1587-1608, 2000.