Requirements based test prioritization using risk factors: An industrial study

Hema Srikantha a, Charitha Hettiarachchib, Hyunsook Do c,∗

a IBM Product Strategy, Waltham, USA
b Computer Science, North Dakota State University, Fargo, ND, USA
c Computer Science and Engineering, University of North Texas, Denton, TX, USA

Article history:
Received 25 August 2014
Revised 17 September 2015
Accepted 19 September 2015
Available online 26 September 2015

Keywords:
Software testing
System testing
Test prioritization
Cloud application
SaaS

A B S T R A C T

Context: Software testing is an expensive and time-consuming process. Software engineering teams are often forced to terminate their testing efforts due to budgetary and time constraints, which inevitably lead to long term issues with quality and customer satisfaction. Test case prioritization (TCP) has shown to improve test effectiveness.

Objective: The results of our prior work on requirements-based test prioritization showed improved rate of fault detection on industrial projects; the customer priority (CP) and the fault proneness (FP) were the biggest contributing factors to test effectiveness. The objective of this paper is to further investigate these two factors and apply prioritization based on these factors in a different domain: an enterprise level cloud application. We aim to provide an effective prioritization scheme that practitioners can implement with minimum effort. The other objective is to compare the results and the benefits of these two factors with two risk-based prioritization approaches that extract risks from the system requirements categories.

Method: Our approach involved analyzing and assigning values to each requirement based on two important factors, CP and FP, so that the test cases for high-value requirements are prioritized earlier for execution. We also proposed two requirements-based TCP approaches that use risk information of the system.

Results: Our results indicate that the use of CP and FP can improve the effectiveness of TCP. The results also show that the risk-based prioritization can be effective in improving the TCP.

Conclusion: We performed an experiment on an enterprise cloud application to measure the fault detection rate of different test suites that are prioritized based on CP, FP, and risks. The results depict that all approaches outperform the random prioritization approach, which is prevalent in the industry. Furthermore, the proposed approaches can easily be used in the industry to address the schedule and budget constraints at the testing phase.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

Software testing can often be a tedious and expensive process [1,2], and can often add up to 50% of the total software cost [3,4]. With limited time and resources, companies are often unable to complete their testing efforts, resulting in software that might not meet customer needs. In order to facilitate effective testing, the concept of test case prioritization (TCP) is often applied to the suite of test cases such that the test cases are run in an order that improves the rate of fault detection [5–7]. By improving the rate of fault detection, the testing teams can mitigate many of the testing issues by reducing the time and cost associated with testing.

To date, the research conducted in TCP has primarily focused on improving regression testing efforts using white box, code level and coverage-based approaches [5,6,8]. Regression testing allows software engineers to test changes made to the system to ensure the code changes made do not introduce new faults in the software system [9]. Regression testing is a necessary and important maintenance activity for improving software quality, but it can require a substantial amount of time and effort as software systems and numbers of test cases grow. While TCP techniques help address this problem by identifying important test cases to run earlier, the majority of them use code coverage information [10], which can be expensive for practitioners to apply [11]. Further, a software system is built upon its requirements information could potentially
help identify more important or more error-prone test cases than just using source code information.

To address this problem, in our prior work, we introduced a prioritization of requirements for test (PORT) 1.0 scheme where we showed the efficacy of TCP at the system level by considering four factors for each requirement [13]. Test cases were prioritized based on the priority of the requirement that was derived by assessing the four factors, customer priority (CP), implementation complexity (IC), fault proneness (FP), and requirements volatility (RV) [13] for each requirement. Test cases that map to requirements with higher priority were ordered earlier for execution. We demonstrated the efficacy of PORT 1.0 technique on four large industrial projects to show the improved rate of fault detection, and thus test effectiveness [12,14]. From our prior study, with extensive sensitivity analysis, we learned that CP was the biggest contributor to improving the rate of fault detection [14]. These findings can be useful for the practitioners when they have limited time and resources to execute the entire tests during regression testing, but these studies were applied to the projects under the same application domain, so whether these results generalize to other application domains is an open question. Further, PORT 1.0 was applied mostly to software applications supporting hardware that usually have a longer release cycle (products having yearly releases). These software applications usually follow software process similar to waterfall model and tend to have a longer release cycle. While PORT 1.0 was validated in industrial projects on a hardware-centric domain, PORT 2.0 is validated on an enterprise cloud application for analytics that has customers around the globe for several years.

Our goal in this paper is to present PORT 2.0 where we apply only a set of factors toward prioritization. In our prior study, we found CP and FP as most significant factors; thus we share the results of PORT 2.0, which uses only these two factors for prioritization of tests. Additionally we validate the approach on software as a service application that follows a very iterative software process where release cycles are as frequent as monthly. In addition to utilizing these two factors, we also investigate whether the use of risk information extracted from the system can improve the effectiveness of test case prioritization. In this paper, we apply the concept of these prioritization techniques on an enterprise-level software system as a service (SaaS) application. The software application, which has several million lines of code, is an enterprise level marketing analysis system that has thousands of customers around the globe. The product team for this application is spread across five geographical locations with thousands of use cases being used by customers every day.

The contributions of this paper include development and validation of two requirements-based prioritization approaches and the validation on an enterprise-level cloud application. Our results indicate that the use of CP and FP can improve the effectiveness of test case prioritization. The results also show that the risk-based prioritization can be effective in improving the test case prioritization. Further, we found that there are some cost-benefit tradeoffs among these approaches, thus we believe that the findings from this study can help practitioners to select an appropriate technique under their specific testing environments and circumstances.

The remainder of the paper is structured as follows. Section 2 describes the research motivation and two different requirements-based test prioritization approaches. Sections 3 and 4 present our experiment including research questions, experiment setup, and analysis. Section 5 discusses the results of our study, and Section 6 describes background information and the related work. Finally, Section 7 presents conclusions and future work.

2. Research motivation and approach

In this section, we discuss our research motivation and then describe two proposed requirements-based approaches: PORT 2.0 that applies two factors CP and FP, toward prioritizing test cases, and a risk-based prioritization technique that prioritizes based on risks associated with requirements categories. To facilitate these approaches, we had developed a tool called requirements based testing (ReBaTe) and we provide an overview of ReBaTe tool architecture in Section 2.3.2.

2.1. Research motivation

Although SaaS has gained acceptance in the past years, it presents the community with a new set of software engineering challenges. Because SaaS is delivered to the customer on cloud, there is one version hosted online for all customers. Thus, to remain competitive, the vendors have to make frequent software updates as often as every two weeks, and maintain high software quality and reliability standards. While SaaS has benefits for customers, quality management has been a major challenge for providers. The system has to be available and functional 24 hours seven days a week. Because all customers are using one version of software, the impact of faults in the system is amplified and there is a need to minimize escape of faults to the field. Finally, frequent updates and software releases provide very little time to testing teams to effectively complete testing efforts, and so there is a need to have effective regression testing techniques. The test case prioritization techniques add much more value to the applications on cloud because of the timeliness involved in releasing updates and the need to ensure quality because of the broader impact of a single failure to the entire customer base. These issues make the regression testing efforts very difficult and there is a dire need for test case prioritization techniques.

In our prior validation of PORT 1.0 on industrial applications [12], we observed that CP was the highest contributing factor toward improved test effectiveness. Based on our prior research we found that these factors contributed toward quality and therefore were selected in PORT 1.0 [13,14]. Our motivation toward selecting these requirements engineering factors is discussed in detail in [13,14]. In this paper, we extend our prior study to PORT 2.0 that applies prioritization only on two factors – CP and FP, both are critical in understanding customer usage of the applications. It is imperative to present an approach that practitioners can find easy to use and apply with minimal effort, and also achieve testing effectiveness. In addition to these two factors, risk information associated with requirements could provide a means to identify important test cases that can reveal highly risky defects. By identifying and fixing such defects earlier, we can further speed up the regression testing process and release more reliable products. With these motivations, in this work, we investigate whether these factors can indeed improve the effectiveness of test case prioritization. We believe that our research outcomes can help practitioners apply this technique without requiring advanced mathematics or statistics, and with minimal effort of collecting and analyzing these factors in the test planning phase.

2.2. Prioritization strategies

In this work, we consider three PORT-based prioritization approaches and two risk-based approaches. The following subsections describe them in detail.

2.2.1. PORT-based prioritization

Our prior contribution to the test case prioritization problem was the development of PORT 1.0 [12,13,14]. Evaluation of PORT 1.0 enabled the software engineering team to assign values to four critical factors: customer priority (CP), fault proneness (FP), requirements volatility (RV), and implementation complexity (IC). The selection of the PORT factors was based on prior research and a more thorough discussion of the factors and their justifications can be found in our previous work [12,14]. In our prior work, we validated PORT 1.0 on
four industrial projects and results are discussed [12]. The results showed the effectiveness of PORT 1.0 over several randomly prioritized suites for all the industrial projects. We also showed that the results of sensitivity analysis on the prioritization factors; CP was the biggest contributor to the improved rate of fault detection [14]. In this paper, we extend the research by further investigating CP toward test effectiveness and how it compares with FP as both these factors demonstrate customer usage of the system. We briefly define these factors below; but additional descriptions and validation results are summarized in our previous work [12,14].

- Customer priority (CP): Provides a measure of how important a particular requirement is to the customer. According to Moisias, approximately 45% of the software functions are never used, 19% are rarely used, and only 36% of the software functions are always used [22]. A focus on customer requirements for development has been shown to improve customer-perceived value and satisfaction [23–25]. Customer priority is a value that is provided for each requirement by the customer facing team.

- Fault proneness (FP): Provides a measure of how faulty legacy requirements were in a prior release. Fault proneness uses both field failures and testing failures. Ostrand et al. have shown that test efficiency can be improved by focusing on the functionalities that are likely to contain higher number of faults [27].

In order to apply PORT scheme, the test managers need to collect these factor values during the test-planning phase. FP exists if the software application has had a prior release. CP values are collected via customer facing product management teams. Based on our observation, while applying PORT 1.0, most companies developing industrial applications that are sold for enterprise customers are required to collect customer-centric data to meet quality standards, and to have continuous quality feedback.

In this work, we evaluate the effectiveness of the proposed approach that uses two important factors, namely CP, FP, or a combination of CP and FP; these two factors were identified to be two significant contributing factors toward test effectiveness in our previous studies. The empirical evaluation has been conducted on an enterprise-level SaaS application, which investigates whether practitioners can improve test efforts by applying fewer factors to prioritize test cases.

2.2.2. Risk-based prioritization

Risk-based approaches used in software testing typically focus on risks associated with software requirements [20,21]. Amland [26,31] defined risk exposure as a product of probability of fault occurrence and the cost if the fault occurs in the production. In this paper, we consider the risks reside in requirements categories and prioritize requirements categories based on the risk levels of each category. Risk exposure indicates the risk level of requirements categories of our application. The following paragraphs explain how we estimate the risks of each category by means of risk likelihood and risk impact.

Amland’s risk model [26,31] is used to estimate the risk of requirements categories of the application and then prioritized the test cases based on the association between requirements categories and test cases. Risk exposure (RE) of each requirements category is calculated by multiplying the risk likelihood (RL) of a requirements category and the risk impact (RI) of the requirements category as shown in Eq. (1).

\[
RE_i = RL_i \times RI_i 
\]

where \(RE_i\) is the risk exposure of requirements category \(i\) and \(RL_i\) is the risk likelihood of the category \(i\), and \(RI_i\) is the risk impact of the category \(i\).

In order to estimate the risk likelihood of requirements categories, we used the number of test cases of requirements categories because the number of test cases of a requirement category reflects the functional complexity and the number of functionalities of that requirement category. Thus, the presence of higher number of functionalities or more complex functions in a requirement category increases the risk of failures in the requirement category. The risk likelihood of a requirements category is calculated by dividing the number of test cases in the requirements category by the total number of test cases of the system as shown in Eq. (2).

\[
RL(Cat)_i = \frac{T_i}{TT_i} \quad (2) 
\]

where \(RL(Cat)_i\) is the risk likelihood of requirements category \(i\) and \(T_i\) and \(TT_i\) are the number of test cases of the requirement category \(i\) and the total number of test cases of the system, respectively. For example, if the total number of test cases of the system is 100 and the requirement category \(A\) has 10 test cases, the risk likelihood of the requirements category \(A\) is 0.10 (10/100).

In order to estimate risk impact values for each requirement category, two approaches were considered. First, we considered the business-criticality of requirements categories because unsuccessful completion of critical requirements may lead to huge financial losses. The business-criticality values were derived from the customer priority values of the requirements categories; these customer priority values are proportional to the business-criticality. We called the risk impact of this approach as business-criticality based risk impact (RIBC). The customer priority of a requirements category indicates how important the requirement is to the end user, and the successful completion of high priority requirements can boost business opportunities. The business-criticality values range from 1 to 5. For example, if the customer priority of requirements category \(A\) is 5, then the RIBC of requirements category \(A\) also becomes 5. Based on the observation of the customer facing team, the requirements categories with higher customer priorities tend to be associated with important functionalities that are essential to meet the forecasted sales because lack of successful completion of these requirements might result into unsubscripton of a service by customers.

As the second approach, we employed a fault proneness (FP) based approach where risk impact is associated with the field failures fraction that is calculated by dividing the number of field failures of a particular requirements category by the total number of field failures of the system. In this approach, the risk impact of a requirements category is calculated using Eq. (3).

\[
RIFP(Cat)_i = \frac{FF_i}{TFF_i} \quad (3) 
\]

where \(RIFP(Cat)_i\) is the fault proneness based risk impact of requirements category \(i\) and \(FF_i\) and \(TFF_i\) are the number of field failures of the requirement category \(i\) and the total number of field failures of the system, respectively. For instance, assuming requirements category \(A\) has 30 field failures and the system has a total of 150 field failures, the fault proneness based risk impact of requirements category \(A\) is calculated using Eq. (3) as follows.

\[
RIFP(Cat)_A = \frac{30}{150} = 0.2 
\]

After calculating RL and RI values, the risk exposure values for both business-criticality and the fault proneness based approaches are calculated using Eq. (1). When calculating the risk exposure of a requirements category, both approaches use the same risk likelihood value that is calculated by using Eq. (2). From our example, risk exposure values for requirements category \(A\) are calculated as follows. Both approaches use the same risk likelihood value (0.10) with different risk impact values.

\[
REBC(Cat)_A = 0.10 \times 5 = 0.50 
\]

\[
REFP(Cat)_A = 0.10 \times 0.2 = 0.02 
\]

Finally, these risk exposure values of requirements categories are used to prioritize requirements categories. One risk-based prioritization technique is based on the REBC values while the other risk-based
prioritization technique is based on the REFP values. Thus, two distinct sets of prioritized requirements categories are obtained by prioritizing the requirements categories using REBC and REFP values separately. After prioritizing the requirements categories, the test cases correspond to the requirements categories are prioritized to obtain the prioritized test suites. For example, if requirements category C obtains the highest REBC value under the business-criticality based prioritization technique, then the test cases associated with the requirements category C obtain the highest priority in the prioritized test suite. Hence, all test case groups are prioritized according to the priorities of the requirements categories.

2.3. ReBaTe implementation

2.3.1. Overview of ReBaTe

The purpose of the tool, Requirement Based Testing (ReBaTe), is to implement the PORT based prioritization scheme for practitioners to apply on a software project. The tool provides end-to-end traceability among requirements, test cases, and defects. ReBaTe provides a centralized repository of software artifacts and supplies a user-interface for managers, developers, and quality engineers to access this repository throughout the software development cycle. The simplicity of the PORT framework entails that ReBaTe can be developed on any platform using either a software package (such as Java, Perl, C, etc.) or a spreadsheet application (such as Microsoft Excel) or a database application (such as Microsoft Access). In order to maintain the platform independence we utilize the Microsoft Access framework to develop the tables, forms and queries that will be used to generate the prioritized test cases. Moreover, ReBaTe implementation can easily be extended to support risk-based prioritization. The centralized repository of ReBaTe provides necessary inputs for the risk-based prioritization. The ReBaTe tool provides a basic framework that is applicable to most companies, but allows the user to personalize based on a specific project needs.

2.3.2. Architectural overview

ReBaTe adheres to a model-view-controller (MVC) design pattern, which allows an application to be broken into three parts—the model, the view and the controller as illustrated in Fig. 1. The controller handles user inputs from the chosen input device and maps these events into commands that are sent to the model and view as necessary. The model manages the internal state and responds to queries about its state and reacts to instructions by changing its state. Finally, the view attaches itself to the model and is responsible for presenting the model to the user.

![Fig. 1. The model-view-controller architecture.](image)

2.3.3. Development technologies

The core of ReBaTe implementation involves four PORT factors described in [12] and a calculation of the PFV, WP, and TSFD scores, which are utilized to prioritize test cases. Our priori work on PORT 1.0 discusses [12] in great detail on how the PFV, WP, and TSFD are computed. The computation of these values is similar for both PORT 1.0 and 2.0. We provide a brief overview of the three below:

### Prioritization factor value (PFV)

The PFV is used to determine the order of the test cases. In order to compute the PFV score we utilize the four factor weights for RV, CP, IC, and FP along with their associated factor weights [13,14]. The factor weights can be computed using the severity values of each failure. We used an exponential of 2 to assign severity values; but based on different teams and products, a different approach to assigning severity can be applied. Additional details on these computations are also provided in [14]. In this paper, for validation purposes, we used APFD (discussed in Section 3.2.2) to measure the effectiveness.

The severity values are assigned scores as follows:

- Severity 1 (SV1): 16
- Severity 2 (SV2): 8
- Severity 3 (SV3): 4
- Severity 4 (SV4): 2

Thus, for a given project the development team will have a list of failures and their associated severity. To compute the TSFD for a project with t failures we utilize Eq. (6):

$$TSFD = \sum_{i=1}^{t} SV_i$$  

where $SV_i$ represents the prioritized value for requirement $i$, and FactorValue$_{ij}$ represents the value for one of the four factors for the requirement $i$.

### Weighted priority (WP)

By using the PFV score we can compute the WP based on the following assumptions. Let us assume that the project has $n$ requirements, and a test case $j$ can cover $i$ requirements. Let $X_i$ represent the set of requirements that is covered by $j$. Then WP$_j$ can be computed by using Eq. (5):

$$WP_j = \sum_{k=x_j}^{n} PFV_k \times \sum_{y=1}^{p} PFV_y$$  

where WP$_j$ indicates the priority of running the test case, test cases are run with highest priority first. If a one-to-one relationship exists between test cases and requirements, the values for WP and PV will be the same.

### Total severity of failures detected (TSFD)

The FP scores are computed using the severity values of each failure. We used an exponential of 2 to assign severity values; but based on different teams and products, a different approach to assigning severity can be applied. Additional details on these computations are also provided in [14]. In this paper, for validation purposes, we used APFD (discussed in Section 3.2.2) to measure the effectiveness.

Thus, for a given project the development team will have a list of failures and their associated severity. To compute the TSFD for a project with $t$ failures we utilize Eq. (6):
In this study we validate our prioritization approaches using APFD that is discussed in Section 3.2.2. In the ReBaTe tool there were eight core database tables used in the ReBaTe tool. Each table in the ReBaTe tool can be modified to include additional information as needed. The tables represent the basic information needed in order to successfully run the tool. The essential relations of the database that are managed through the ReBaTe application are products, requirements, test cases, and defects. All the following share one-to-many relationship: the products and requirements relations, the requirements and test cases relations, the test cases and defects relations, the employees and test cases, and the employees and defects relations.

We talk briefly about the overall architecture for ReBaTe below.

A. Database design

The database design for ReBaTe can be achieved with any database application. Any backend application can be applied.

B. Primary relations

The essential database relations managed for ReBaTe application include products, requirements, test cases, and defects. All three relations (products and requirements, requirements and test cases, and test cases and defects) share a one-to-many relationship. Additionally, for a multi-user environment an Employees application can also be created. Employees and test cases share a one-to-many relationship as does employees and defects.

Fig. 2 illustrates the primary relations found in the tool. In this example, we utilize a simple Microsoft Access database to build the framework. The figure is a truncated view and does not include all available fields.

C. UI support

To provide consistent usability experience, individual forms are provided to store values for products and employees. These values stored in these forms are used in the products, requirements, test cases, and defects table based on the relationships shown in Fig. 2. Fig. 3 illustrates the products entry form below. We provide selection-lists and combo-boxes in required places for data collection minimize the number of errors in data entry.

Users are presented with easy to use drop-down combo-boxes pre-populated with relevant employee names and descriptions to facilitate this mapping between test cases and testers. Any administrator can take the source code and tailor to the variables that their organization applies and does not need to modify source code much.

D. Operational profile

ReBaTe allows users to capture operational profile data, to support software reliability assessment. The ReBaTe UI records each user operation, such as the addition of a defect or the update of a requirement, during the course of regular operation. This information is stored and aggregated to provide usage trends over time. Operational profiles are used to direct development and testing efforts as part of a value-driven approach to software development [6,7]. The operational usage data can be exported by creating simple queries.

E. Access control

Based on the user’s assigned role, specific actions in the UI will be enabled, while others are disabled. Each user can have one or more of the four roles: manager, tester, developer, and administrator. The roles allow user to access specific system operations. For example, a manager may add or edit requirements, edit test cases, defects, etc.; however testers may add or edit test cases that could be limited to projects. Because ReBaTe’s access control model is defined in the underlying database, the
roles and the respective privileges can be modified by simply editing the tuples to meet the project’s needs.

F. Usage

The UI uses a query driven explorer to allow users to browse for the database contents. Database tables can be updated with built in forms. Since the system is platform independent, the users have the ability to add additional tables, queries, forms, etc. as the need arises. The ReBaTe tool provides a basic framework that is applicable to most companies, but allows users to customize easily as needed.

2.3.4. ReBaTe problems and limitations

The ReBaTe tool that is developed in Microsoft Access has a selected set of problems and limitations:

(1) File size limit: Microsoft Access has a 2GB file size limit, as a result large organizations with a large number of projects, requirements, test cases, defects, etc. may need to utilize a different framework.

(2) Software requirements: The tool was developed with the understanding that most organizations will have Microsoft Office 2003 or higher.

3. Empirical study

In this section, we discuss our controlled experiment setup for two requirements-based test prioritization approaches: PORT-based and risk-based prioritization approaches, considering the following research questions:

[RQ 1] Can we improve the rate of fault detection by ordering test cases based on CP?
[RQ 2] Can we improve the rate of fault detection by ordering test cases based on FP?
[RQ 3] Is there a correlation between the two factors CP and FP? Can the project teams employ only one of the factors and obtain test effectiveness?
[RQ 4] Is a risk-based approach better to improving rate of fault detection and how does it compare with PORT 2.0?

3.1. System under test

We applied our scheme to an enterprise level IBM analytics application. The application is cloud-based and deployed as a service for thousands of enterprise customers around the world. The application needs to be reliable and functional 24/7 and has to be validated in all environments including mobile. The application provides intelligent analytics to marketers and retailers and allows them to improve their marketing needs. This solution, with over a million lines of code, is currently being used by thousands of organizations across the world. Since the application is hosted as a service the teams have to make system updates frequently (at least once a month) to remain competitive. The software team is distributed around the world that further adds to the challenges.

From a software engineering research perspective, such a vast suite of products and customers means a large repository of real world software usage and failure data. The dataset used in this study consists of 15 broadly defined requirements categories with over 1700 test cases. The suite of test cases uncovers over 100 failures, and over hundred problems reported by customers have been analyzed to compute FP.

3.2. Variables

In this section, we describe independent and dependent variables.

3.2.1. Independent variable

Given our research questions, this study manipulated one independent variable: test case prioritization technique. We consider one control and five heuristic techniques (three PORT-based and two risk-based) as follows:

- Control
  - Random (Tran): The random order serves as our experimental control technique (we randomly order the test cases). Based on the experience of the first author as a consultant and working in the industry for over 10 years, it has been observed that random approach is prevalent in the industry.

- Heuristics
  - PORT-based
    - FP-based prioritization (Tp-fp): This technique uses fault proneness (FP) for prioritization.
    - CP-based prioritization (Tp-cp): This technique uses customer priority (CP) for prioritization.
    - FP + CP-based prioritization (Tp-fp-cp): This technique uses both fault proneness (FP) and customer priority (CP) for prioritization. Details on these factors are provided in Section 2.2.1

  - Risk-based
    - Business-criticality based, risk-based prioritization (Tr-bc): This technique uses risk information for prioritization while using customer priority data to estimate the risk impact.
    - Fault proneness based, risk-based prioritization (Tr-fp): This technique uses risk information for prioritization while using field failures data to estimate the risk impact.

3.2.2. Dependent variable

Our dependent variable is average percentage of fault detection (APFD). APFD [5,6,15] shows how rapidly a prioritized test suite detects faults during the execution of the test suite. The APFD values range from 0 to 100 and are monitored during test suite execution and 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 [5,6,15]. A higher APFD score indicates higher rate of fault detection.

To illustrate this measure, consider an example program with 10 faults and a test suite of five test cases, A through E, with fault detecting abilities, as shown in Fig. 4. Suppose we order the test cases as C-E-B-A-D. After executing test case C, we have uncovered 70% of the faults while utilizing only 20% of the test cases. Next, when test case E is run we have uncovered 100% of the faults while executing only 40% of the test cases. In contrast, if we order the test cases as C-E-B-A-D, after running test case E we uncover 30% of the faults. Next, after running test case C we uncover 100% of the faults. The area inside the rectangles represents the weighted percentage of faults detected over the fraction of the test suite executed. The solid lines represent the gain in detecting the percentage of faults. The area under the curve represents the weighted average percentage of faults detected (APFD). A higher APFD score indicates higher rate of fault detection. Thus, in this case ordering test cases as C-E-B-A-D yields a better APFD score of 84% than ordering test cases as E-C-B-A-D that yields an APFD score of 76%.

3.3. Experimental setup and procedure

In this section, we discuss the experimental setup for PORT 2.0 scheme. We use an enterprise level application to validate effectiveness of the prioritization scheme. Our dataset consists of requirements categories such that each requirements category maps to the associated set of test cases, the test cases that uncovered defects for each requirements category, and the number of reported field failures

and the customer priority for each requirements category. Given the granularity of each individual requirement we combine similar requirements into a single category. For example, we consider a broad requirement as 'import file' and many sub requirements like 'import doc file', or 'import pdf file' are classified into one requirements category 'import file'. Due to legal and proprietary reasons we utilize a code name A to O for each requirements category. In this study, we evaluated 15 broad requirements categories and analyzed over 100 defects that arose from system black box test cases. However, due to legal reasons, we have not provided the actual number of defects, but provided a defect ratio. Table 1 illustrates the dataset utilized in this study. Each column in the table is defined as follows:

1. Requirements categories: Broad categories that are used to combine similar requirements together.
2. Test cases: The total number of test cases associated with the particular requirements category.
3. Defect ratio: The proportion of test cases in that requirements category that uncovered defects. For example, nearly 20% (or 60) of the test cases in requirements category A uncovered defects.
4. Fault proneness (FP): The propensity for failure based on the number of field failures for the requirements category. A lower number is indicative of fewer failures in the field. The FP score is computed based on the number of field failures. First, for

![Table 1](image)

![Graph](image)

Fig. 4. APFD example (Rothermel et al.).

<table>
<thead>
<tr>
<th>Requirements categories</th>
<th>Test cases</th>
<th>Defect ratio</th>
<th>FP</th>
<th>CP</th>
<th>FP + CP</th>
<th>RL</th>
<th>R₁KC</th>
<th>R₄FP</th>
<th>R₁KC</th>
<th>R₄FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>302</td>
<td>0.19</td>
<td>5</td>
<td>5</td>
<td>5.00</td>
<td>0.169</td>
<td>5</td>
<td>0.427</td>
<td>0.84263</td>
<td>0.07196</td>
</tr>
<tr>
<td>B</td>
<td>79</td>
<td>0.27</td>
<td>2</td>
<td>3</td>
<td>2.70</td>
<td>0.044</td>
<td>3</td>
<td>0.067</td>
<td>0.13225</td>
<td>0.00297</td>
</tr>
<tr>
<td>C</td>
<td>56</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1.40</td>
<td>0.031</td>
<td>2</td>
<td>0.005</td>
<td>0.06250</td>
<td>0.00016</td>
</tr>
<tr>
<td>D</td>
<td>128</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>4.10</td>
<td>0.071</td>
<td>5</td>
<td>0.090</td>
<td>0.35714</td>
<td>0.00642</td>
</tr>
<tr>
<td>E</td>
<td>30</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>3.10</td>
<td>0.017</td>
<td>4</td>
<td>0.034</td>
<td>0.06696</td>
<td>0.00056</td>
</tr>
<tr>
<td>F</td>
<td>191</td>
<td>0.16</td>
<td>3</td>
<td>5</td>
<td>4.40</td>
<td>0.070</td>
<td>5</td>
<td>0.180</td>
<td>0.53292</td>
<td>0.01616</td>
</tr>
<tr>
<td>G</td>
<td>22</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>2.40</td>
<td>0.012</td>
<td>3</td>
<td>0.045</td>
<td>0.03683</td>
<td>0.00055</td>
</tr>
<tr>
<td>H</td>
<td>13</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>2.40</td>
<td>0.007</td>
<td>3</td>
<td>0.022</td>
<td>0.02176</td>
<td>0.00016</td>
</tr>
<tr>
<td>I</td>
<td>246</td>
<td>0.01</td>
<td>2</td>
<td>1</td>
<td>1.70</td>
<td>0.137</td>
<td>2</td>
<td>0.034</td>
<td>0.27455</td>
<td>0.00463</td>
</tr>
<tr>
<td>J</td>
<td>45</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>3.40</td>
<td>0.025</td>
<td>4</td>
<td>0.067</td>
<td>0.10045</td>
<td>0.00169</td>
</tr>
<tr>
<td>K</td>
<td>185</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>2.40</td>
<td>0.030</td>
<td>3</td>
<td>0.011</td>
<td>0.30771</td>
<td>0.00116</td>
</tr>
<tr>
<td>L</td>
<td>49</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.027</td>
<td>1</td>
<td>0.005</td>
<td>0.02734</td>
<td>0.00014</td>
</tr>
<tr>
<td>M</td>
<td>50</td>
<td>0.32</td>
<td>0</td>
<td>2</td>
<td>1.40</td>
<td>0.028</td>
<td>2</td>
<td>0.005</td>
<td>0.05580</td>
<td>0.00014</td>
</tr>
<tr>
<td>N</td>
<td>33</td>
<td>0.12</td>
<td>1</td>
<td>4</td>
<td>3.10</td>
<td>0.018</td>
<td>4</td>
<td>0.011</td>
<td>0.07366</td>
<td>0.00021</td>
</tr>
<tr>
<td>O</td>
<td>363</td>
<td>0.02</td>
<td>1</td>
<td>4</td>
<td>3.10</td>
<td>0.203</td>
<td>4</td>
<td>0.011</td>
<td>0.81027</td>
<td>0.00228</td>
</tr>
<tr>
<td>Average</td>
<td>1.4</td>
<td>3.27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weight</td>
<td>0.3</td>
<td>0.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
each requirements category we divide the reported field failures by the total number of field failures. As an example, suppose requirements category A had 5 reported failures and the project had a total of 50 reported failures. The ratio for category A is then 0.1. Next, all the field failure values are normalized to a scale of 0–5 to create the FP score, a value 0 is assigned to categories with no field failures.

(5) Customer priority (CP): The priority of each requirements category as perceived by the customer. A higher number indicates the requirements category was most important to the customer. Table 1 below shows the CP for each of the requirements categories. The value was provided by the customer facing team for each of the category based on their assessment of ‘impact of the set of requirements to customer usage and satisfaction’. It is a qualitative assessment based on the perception of the customer facing team.

(6) Combined FP + CP: An additive effect of FP and CP score. For each requirements category, the FP score is multiplied by the CP weight. The FP weight is computed as follows. First, we find the average of all FP values. In this case, the average of all FP values is 1.4. Next, we find the average of all CP values. In this case the average of all CP values is 3.27. Finally, we compute the FP weight by dividing the FP average score by the sum of the FP and CP average scores. Thus, we would divide the FP average (1.4) by the sum of the FP and CP average (1.4 + 3.27 = 4.67) to obtain an FP weight of 0.3. The same methodology is applied for the CP weight calculation. We generate the final FP + CP score as follows. We take the raw FP score and multiply by the CP weight and add it to the raw CP score multiplied by the CP weight. Thus, for category A we multiply its FP score (5) by the CP weight (0.3) and add it to its CP score (3) multiplied by the CP weight (0.7) to get a final FP + CP score of 5.00.

(7) Risk likelihood (RL): The probability of fault occurrence in a requirements category. RL is estimated using the number of test cases in requirements categories as explained in Section 2.2.2.

(8) Risk impact (RIBC): The impact or loss if faults occur in a requirements category. This approach uses the customer priority data to estimate the risk impact as described in Section 2.2.2.

(9) Risk impact (REBC): The risk impact estimated by means of field failures data.

(10) Risk exposure (RE): The quantitative measurement of risks resides in a requirements category. The multiplication of RL and RI of a particular requirements category produces the RE of the category as shown in Eq. (2) in Section 2.2.2. The tenth column contains the business-criticality based, risk exposure values (REBC) of each requirements category.

(11) Risk exposure (REFP): The fault proneness based, risk exposure value of a requirements category.

Table 1 presents the different requirements categories, the total number of test cases of each category, the defect ratio, FP, CP, FP + CP, RL, RLbc, RLfp, REbc, and REfp. For example, for requirements category A, we have 302 test cases, the defect ratio is 0.19, the FP score is 5, the CP score is also 5, the combined FP + CP score is 5.00, the RL is 0.169, the RLbc is 5, the RLfp is 0.427, the REbc is 0.84263 and the REfp is 0.07196.

PORT and risk-based approaches require two different experimental setups. The following two Sections 3.3.1 and 3.3.2 explain the experimental setup for each of the approaches in detail.

3.3.1. PORT experimental setup

In this section, we discuss the steps taken to setup our experiment for PORT:

• Create 30 random orderings of the test suites (Tran), and compute the average percentage of faults detected (APFD) [Section 3.2.2] score for each random ordering. In our study, we found that a given fault mapped to one test case. For each requirements category, we order the faults to the respective test cases in that category. Next, the entire set of 1792 test cases is randomly ordered 30 times to create the 30 unique randomly ordered test sets. Finally, the APFD score is computed by utilizing the APFD formula for each of the 30 random orderings.

• Prioritize test cases based on their CP scores (Tp-cp). The CP values are provided for each requirements category by the customer facing teams, based on the perceived impact of the requirement to customer usage and satisfaction. The test cases are then ordered such that test cases with higher CP values are executed first. In cases where more than one requirements category had the same CP value, we applied random ordering. We computed APFD value for the test suite ordering that was prioritized using CP.

• Prioritize test cases based on their FP scores (Tp-fp). FP is computed based on the number of reported field failures. The number of field failures for each requirements category is divided by the total number of field failures. For example, suppose we have 50 total field failures and 5 of those failures were associated with requirements category A. The field failure ratio for category A would be 0.1. Next, the ratios are normalized on a scale of 0–5. Requirements with no field failures are assigned a category of 0. Due to legal reasons the exact number of field failures cannot be reported. The test cases will be ordered and run such that test cases associated with higher FP values are run first. The APFD value was computed for the test suite ordering that applied FP to prioritize test cases.

• Prioritize test cases based on the combined effect of FP and CP (Tp-fp-cp). We first find the average of all CP scores, in this instance 3.27, and all FP scores, in this instance 1.4 as shown in Table 1. Next, we divide the average CP score by the sum of the average CP and FP score to obtain a weight of 0.7. Similarly, for FP we obtain a weighted score of 0.3. Finally, we compute a combined FP and CP score by multiplying each factor score by its weight and adding them.

3.3.2. Risk-based prioritization experimental setup

In the risk-based prioritization approach, risks exist in the requirements categories of the system are estimated using two factors (risk likelihood and risk impact) as explained in Section 2.2.2. The seventh column of Table 1 shows the risk likelihood values of all requirements categories. In our experiment, resultant risk likelihood values range from 0.007 to 0.203. Risk impact is estimated using two different approaches: business-criticality value-based and fault proneness-based. The eighth column of Table 1 shows the business-criticality based risk impact (REBC) values of all requirements categories. The ninth column of the table shows REfp values of all requirement categories. The resultant risk impact values range from 0.005 to 0.427.

After completing the estimations of RL and RI, risk exposure (RE) values for each requirements category are calculated by multiplying RL of a category by its RI using Eq. (1) explained in Section 2.2.2. The tenth column of Table 1 shows the business-criticality based risk exposure values (REBC) and the last column shows the fault proneness based, risk exposure values (REfp) of all requirements categories. In this experiment, the risk exposure values of business-criticality based approach (REBC) range from 0.02176 to 0.84263 whereas fault proneness based, risk exposure (REfp) values range from 0.00014 to 0.07196.

Requirements categories are prioritized based on the RE value obtained by each category. Two separate prioritized test suites, Tr-bc and Tr-fc that are based on REbc and REfp, respectively, are prepared for execution. The test cases associated with requirements categories with higher RE values are executed first. If more than one category had the same RE value, then the test cases are prioritized randomly. The APFD values are computed for both risk-based prioritized test suites.
Table 2
Test case prioritization results.

<table>
<thead>
<tr>
<th>Average of 30 random samples (Tran)</th>
<th>PORT (Tp-cp)</th>
<th>(Tp-fp)</th>
<th>(Tp-fp-cp)</th>
<th>Risk-based (Tr-bc)</th>
<th>(Tr-fp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>APFD values</td>
<td>51</td>
<td>69</td>
<td>80</td>
<td>72</td>
<td>71</td>
</tr>
</tbody>
</table>

3.4. Threats to validity

3.4.1. Construct validity

The customer priority was quantified by the customer facing teams that comprise of sales team members, product managers, and product specialists. Based on the experience of the first author and experience in product teams for over ten years, the customer facing teams have years of experience working in the field and are also technical experts; therefore their analysis is mostly quantifiable. Although we used quantified customer priority values, because these values were assigned by the customer facing teams, the results can be subjective. Further, the risk-based prioritization approach used the number of test cases to derive risk likelihood whereas the customer priority values were used to derive business-criticality based on risk impact (RIBC) values. The accuracy of deriving the risk likelihood and the risk impact values from the aforementioned sources could slightly vary from system to system. This threat can be reduced by utilizing different system information. For instance, the cost of functional failures associated with requirements categories could be a good alternative to derive the risk impact.

3.4.2. Internal validity

We grouped requirements into 15 broad categories. More specific categorization can affect the outcome of prioritization techniques used in this study. Both the prioritization schemes require a few factor values to be collected during test planning phase. It is our observation that most enterprise applications that have been thru one release tend to have the very minimal customer data to assess end user quality.

3.4.3. External validity

We used one large size industrial application (SaaS) for the experiment. The data used in this study were obtained from a real world software application, which is widely used by the customers around the world. Therefore, similar results could be expected from other software systems with similar domain [41,42]. However, our findings might not be applied to other application domains, and control for this threat can be achieved only through additional studies with other types of industrial applications and open source applications (e.g., applications that can be obtained from open source software hosts, such as SourceForge).

4. Data and analysis

In this section, we summarize the results of our study. We use APFD, as discussed in Section 3.2.2 to measure the effectiveness of the prioritization techniques. The results of our test case prioritization techniques are shown in Table 2. The table shows that the average APFD value for 30 unique random orderings, and APFD values for one test suite ordering using CP, FP, FP + CP, and risks information. A higher APFD value indicates that the prioritization technique resulted in higher rate of fault detection. The results show that the APFD value using PORT had the highest rate of fault detection when utilizing FP to prioritize the test suite. For example, a prioritized test suite using FP alone (Tp-fp) resulted in an APFD value of 80, likewise applying CP resulted in 69 whereas combination of FP + CP resulted in 72. Furthermore, the results show that the fault proneness based, risk-based approach (Tr-fp) obtained an APFD value of 77, which is close to the highest APFD value (80) of the experiment, and the business-criticality based, risk-based approach (Tr-bc) resulted in an APFD value of 71, which is better than the random approach (Tran) and CP-based approach (Tp-cp). Compared to the control technique, i.e. random ordering (Tran), all heuristics performed better.

Now, we briefly discuss our results for each of our research questions and we discuss further implications of the results in Section 5.

RQ1: Can we improve the rate of fault detection by ordering test cases based on CP?

Table 2 shows that the use of customer priority (CP) can improve the effectiveness of test case prioritization compared to the control technique (Tran). The APFD value for CP-based approach (Tp-cp) was 18 points higher than the random approach (Tran).

RQ2: Can we improve the rate of fault detection by ordering test cases based on FP?

Table 2 shows that the rate of fault detection by prioritizing test cases using FP alone (Tp-fp) resulted in the highest APFD value (80).

RQ3: Is there a correlation between the two factors CP and FP? Can the project teams employ only one of the factors and obtain test effectiveness?

Prioritizing test cases using both CP and FP (Tp-fp-cp) resulted in an APFD score of 72, which is slightly better than the CP-based approach (Tp-cp). We also investigated whether these two factors has a correlation. As shown in Fig. 5, we found that a strong positive correlation (0.7) exists between CP and FP. Thus, the testing team can obtain effectiveness over random approach by prioritizing using either CP or FP based on the time and effort the practitioners have to invest toward applying the techniques.

RQ4: Is a risk-based approach better to improving rate of fault detection and how does it compare with PORT?

Table 2 shows that the risk-based approach that used business-criticality data can perform better than the random approach and CP-based approach. The APFD value for business-criticality based, risk-based approach (Tr-bc) was 20 and 2 points higher than the random approach (Tran) and CP-based approaches, respectively. The fault proneness based, risk-based approach (Tr-fp) obtained an APFD value of 77, which is close to the highest fault detection rate of the experiment (80).

5. Discussion

In this section, we discuss the empirical results of two requirements-based test case prioritization approaches: PORT 2.0 and risk-based prioritization. We applied these techniques to an enterprise level cloud application that has customers around the world. Our results indicate that the CP-based approach (Tp-cp) was better than the random approach (Tran) and its improvements over the random approach (Tran) was 35.3%. The combination of CP and FP (Tp-fp-cp) produced the third best result and its improvement over the random (Tran), the CP-based (Tp-cp) and the business-criticality, risk-based (Tr-bc) approaches were 41.2%, 4.3%, and 1.4%, respectively. The FP-based prioritization technique (Tp-fp) produced the best result
over all other heuristics and its improvement over Tran, Tp-cp, Tp-fp-cp, Tr-bc, and Tr-fp are 56.9%, 15.9%, 11.1%, 12.7%, and 1.3%, respectively. All heuristics outperformed the Tran prioritization, which is the prevalent approach applied by practitioners in the industry.

The results also show that for applications that have already been released (post version 1.0) where practitioners have field data, it is recommended that they use FP to prioritize test cases for execution. However, for products that are being released for the first time (i.e., v 1.0), where the software teams lack field data, it is still beneficial to use CP as a factor to prioritize test cases because CP shows a higher rate of fault detection in comparison to random prioritization. Both these factors enable the teams to understand customer usage scenarios and allow the teams to prioritize test cases based on customer usage. Also both factors are easy to collect and implement which, we believe, is one of the additional benefits of PORT 2.0 scheme in that it allows for easy implementation of the scheme. PORT 2.0 can be applied to any application domain using FP data (quantitative field failure data) or CP that will require domain knowledge and can be subjective in nature. PORT application would require few days’ effort by a QA manager. To gain acceptance of the technique, it is critical to identify a technique that can be easily applied by practitioners. We believe, based on our experience in the industry, that PORT 2.0 is an easy to apply scheme that requires minimum effort by practitioners and yet allows the teams to achieve test efficiency.

Similar to PORT approaches, both risk-based approaches outperformed the random approach (Tran). Their improvement rates over the Tran were 39.2% (Tr-bc) and 51.0% (Tr-fp). When we compared risk-based approaches to PORT approaches, we can see that the fault proneness risk-based approach (Tr-fp) produced better results than the approach used CP alone (Tp-cp) and the Tp-fp-cp approach (the improvement rates were 11.6%, and 6.9%, respectively), but we also observed that the fault proneness risk-based approach (Tr-fp) did not produce better results than the FP approach (Tp-fp). Further, the business-criticality based risk-based approach (Tr-bc) outperformed the CP-based (Tp-cp) approach (the improvement rate was 2.9%).

These results are somewhat surprising because we expected that the use of risk information would improve the effectiveness of test case prioritization compared to all other types of heuristics as we observed from our previous study [19]. We speculate that our assumption about fault costs affected this outcome. While we utilized risk factors to identify more important test cases for test prioritization in this study, we evaluated the prioritized tests under the assumption that fault costs are all the same because the information regarding fault severity was not available in this study. However, if such data is available, we might have obtained different findings; in our previous study [19], we learned that the risk-based approach indeed helped find more severe faults earlier. Such claim, however, should be further investigated through future studies.

By examining our results, we can say that when the practitioners have scheduling issues, they can use Tp-cp, Tp-fp or Tp-fp-cp approaches rather than using the risk-based approaches that need more time and effort. Also, because risk-based approaches are quantitative in nature and do not require much domain expertise and experiences, when such qualitative information is not available, risk-based approaches would be a viable choice. Further, if risk-based approaches can detect more severe faults earlier, mission or safety critical systems could gain some benefits from these approaches.

With the constant need to improve time to market, practitioners are in need to find easy to use regression testing technique that does not require teams to have advanced statistical or mathematical skills, or require a lot of implementation effort. TCP techniques that involve minimal effort for setup are likely to gain acceptance and adoption with the software test teams. We believe that the proposed techniques are easy to apply for practitioners to gain effectiveness. In software industry, it may take several days to complete regression testing activities. Our results showed that the proposed approaches can find faults early and reduce the regression testing process’s time of a large application with over a million lines of code. Therefore, this time saving will lead to save significant amount of money in particular, for companies who develop large software systems. In this research, we only consider regular system faults. However, if we consider the severity of faults, our approaches could be much more cost-effective because finding and fixing sever faults is more expensive than ordinary faults. Therefore, by adopting our approaches in large
and critical systems such as medical diagnosis systems, more faults can be detected early and significantly improve the quality of such applications and also the saving of the dollar amount would be much higher.

6. Background and related work

6.1. Coverage-based TCP techniques

Test case prioritization involves ordering of test cases to achieve a predefined goal like rate of fault detection. The current techniques are primarily coverage based. Coverage-based TCP techniques involve ranking of test cases by using the number of statements execute by the test case such that the test case covering the maximum number of statements would be executed first [5,15]. For branch and function coverage techniques, tests are prioritized using program branches or program functions covered, respectively.

To date, many researchers have proposed and studied various coverage-based TCP techniques, and numerous empirical studies showed that prioritization techniques that use coverage information can improve the effectiveness of test case prioritization [5,15,32,33]. A recent survey conducted by Yoo and Harman [34] provides a comprehensive overview of test case prioritization techniques.

6.2. Operational profile based TCP

Musa argued that introducing the operational usage in prioritizing test cases would enable more effective testing [16,17]. In this approach the operational profile is first estimated by collecting data from testers, system engineers, architects, and customers. Once the operational profile has been generated the test cases are written and allocated in a manner such that the most used operations have the most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases. However, despite the benefits of operational profile driven testing as shown in AT&T, International definity most number of test cases.

6.3. Prior work on requirements-based prioritization [PORT 1.0]

Delivering high quality software on time and making the customers happy depend on several key factors such as early detection of faults in a software system. In this regard, the inevitable association between requirement engineering and testing significantly affects the success and the high quality of the software of interest. In the literature, numerous techniques have been proposed to improve the testing process in collaboration with requirement engineering. Sarmiento et al. [35] developed a tool that generates test cases using software requirements specification that is written in natural language and reduces the time and effort of testing process. Granda et al. [36] proposed an approach that automatically generates abstract test cases from requirements models with an intention to reduce the risk of defects cost effectively in software systems. These requirements-based testing approaches demonstrate that the use of software requirements can improve software quality and testing process by detecting defects early and by reducing testing effort.

Similar to these approaches, our prior work [12–14] also utilized requirements information so that we can improve the effectiveness of test case prioritization techniques. We introduced four factors for test case prioritization and found that CP was the biggest contributor of improved rate of fault detection. In this paper, we further investigate the effectiveness of CP and compare that with FP. Both CP and FP assess the importance of a requirement based on field usage. CP is a value that is provided for each requirement by the customer facing team, and can be subjective in nature. FP, however, is based on the number of failures reported by customers for a requirement and represents a quantitative assessment of field quality and usage. In our earlier research, we observed that CP was the biggest contributor to improving the rate of fault detection [14]. In this paper, our extension of PORT includes analyzing CP and FP toward improving the rate of fault detection and identifying if there is a correlation between these two factors such that practitioners can apply one or the other and obtain test effectiveness. The two factors can be collected during the design phase (customer priority) or after the product has been released (fault proneness).

6.4. Prior work on risk-based prioritization

As explained earlier, the majority of test case prioritization techniques have utilized source code information, but recently other types of software artifacts have been considered, such as software requirements and design documents. For example, Krishnamoorthy and Mary [28] proposed a cost effective technique to prioritize system level test cases based on software requirements related factors to improve the rate of severe fault detection at the early stage of testing life cycle. Srikant et al. [13] presented a system-level prioritization technique by using system requirements and fault proneness. Along with the use of software requirements, other researchers have used risk information to prioritize test cases so that they can run tests that exercise risky areas as early as possible. Stallbaum et al. [29] generated test cases using activity diagrams and prioritized tests based on risks associated with faults. Felderer and Schieferdecker [37] presented a framework that organizes and categorizes the risk-based testing to aid the adoption of appropriate risk-based approaches according to the circumstances. Erdogan et al. [38] conducted a systematic literature review on the combined use of risk analysis and testing. This survey identified, classified, and discussed the existing approaches in terms of several factors such as main goals and the maturity levels of the approaches. Li et al. [39] proposed a value-based software engineering framework to improve the software testing process. Their proposed strategy prioritizes the new features by considering several factors such as business importance and quality risks. The quality risks of their study are estimated through several risk factors such as personnel proficiency, size, and design quality of the new features, and the results showed that their approach can help handle the testing process effectively and efficiently while increasing the customer satisfaction. Yoon et al. [30] extracted risk information from requirements, identified relationships between risks and tests, and then prioritized tests by the risk severity. Stallbaum and Metzger [40] introduced a new automated risk assessment approach using requirements metrics and illustrated the benefit of the approach in risk-based and requirements-based testing.

These existing risk-based testing approaches demonstrate that the use of risks in the software systems can help find critical functional defects that may cause severe security or safety critical defects. While these existing approaches provide a good understanding of the importance of the use of risk information during testing, the risk information utilized by the majority of existing approaches is qualitative, which can lead to subjective evaluations of the approaches. In our approach, unlike the risk analysis of existing risk-based test case prioritization approaches, we use quantitative data (except for business-criticality) extracted from the system artifacts that are associated with requirements to estimate risks reside in requirements categories. Customer priority (CP) data corresponds to the business-criticality and the quantitative assessment of CP is based on the perception of the customer facing team. At the testing phase, all data need to perform the risk analysis are available for the testing team to execute the risk-based test case prioritization without performing any additional and heavily subjective risk estimation methods. Therefore, the testing team can obtain a prioritized test suite almost effortlessly within a very short time period and also with a limited budget.
7. Conclusions

Our earlier work on PORT 1.0 focused on requirements-based prioritization, and we demonstrated improved test effectiveness on four industrial systems. In this paper, we extended our PORT approach from PORT 1.0 to PORT 2.0, and replicated our prior study to present PORT 2.0 that only uses two factors, CP and FP to prioritize test cases. The proposed approach produced promising results and several findings. Our results indicate that both factors in isolation and in conjunction provide a more effective test case prioritization scheme than a random approach. Our results also show that FP produces the highest rate of fault detection. When we used both CP and FP for test prioritization, the result was better than the prioritization with CP alone. We also observed that there is a strong correlation between CP and FP, which means that practitioners can employ either CP or FP to improve testing effectiveness. In addition to investigating the use of these two factors (CP and FP), we presented a risk-based prioritization that is a system-level approach. Our results show that the proposed risk-based approaches produced better results over random approach, which is prevalent in the industry and also performed better than the CP approach of PORT 2.0.

In this study, we validated PORT 2.0 with an enterprise cloud application that has been in use by end users across the globe for many years. While PORT 1.0 was validated in a more hardware centric domain, PORT 2.0 has been applied toward analytics solutions used by marketing industry. Our approaches have been evaluated using one industrial project, thus our findings might not be applied to other applications. While we empirically evaluated a variation of the risk-based approach using open source, mid-size health record system in our previous work [19], to address this limitation further, we plan to evaluate our approaches considering additional application domains as we discussed in Section 3.4.

We believe that practitioners can effectively improve test case prioritization by applying the risk-based approaches and the investment in applying the scheme is considerably low. It has been our observation that for any scheme to gain acceptance, it needs to be applied by practitioners with minimal effort and without advanced mathematics or statistics, while achieving effective results. It is our experience during validation of PORT 1.0 and now with PORT 2.0 that practitioners needed to spend very limited effort to apply this scheme in test planning phase. Further, practitioners did not require additional skills to apply this scheme.

We extended the current coverage-based approaches to demonstrate effectiveness with a requirements-based scheme. This paper takes the prior work further to demonstrate that risk-based approaches can be effective with non-coverage based prioritization schemes. This study was validated on one enterprise level industrial SaaS application, with several million lines of code, with real customers around the globe using this product every day. We plan to evaluate our approaches considering additional application domains as we discussed in Section 3.4.

Acknowledgments

We would like to thank IBM for their contributions to the research. This work was supported, in part, by NSF CAREER Award CCF-1564238 to University of North Texas. The authors wish to thank Sean Banerjee for his contributions to this paper.

References


