SYSTEMATIC APPROACHES TO IMPROVE TEST CASE PRIORITIZATION USING REQUIREMENTS AND RISKS

A Dissertation
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Charitha Sasika Hettiarachchi

In Partial Fulfillment of the Requirements
for the Degree of
DOCTOR OF PHILOSOPHY

Major Department:
Computer Science

July 2016

Fargo, North Dakota

NORTH DAKOTA STATE UNIVERSITY

Graduate School

Title

SYSTEMATIC APPROACHES TO IMPROVE TEST CASE PRIORITIZATION USING REQUIREMENTS AND RISKS

By

Charitha Sasika Hettiarachchi

The supervisory committee certifies that this dissertation complies with North Dakota State University's regulations and meets the accepted standards for the degree of

DOCTOR OF PHILOSOPHY

SUPERVISORY COMMITTEE:

Dr. Hyunsook Do
Chair

Dr. Kendall Nygard

Dr. Saeed Salem

Dr. Simone Ludwig

Dr. Jacob Glower

Approved:

July 7 2016
Date

Dr. Brian Slator
Department Chair

ABSTRACT

The use of system requirements and their risks enables software testers to identify more important test cases that can reveal faults associated with risky components. Having identified important test cases, software testers can manage the testing schedule more effectively by running such test cases earlier so that they can detect then fix faults sooner, especially, in regression testing. Some work in this area has been done, but the previous approaches and studies have some limitations, such as an improper use of requirements risks in prioritization and an inadequate evaluation method.

To address the limitations, we implemented a new requirements risk-based prioritization technique and evaluated it considering whether the proposed approach detects faults earlier overall and also detects faults associated with risky components earlier. Then, we proposed an enhanced risk-based test case prioritization approach that estimates requirements risks systematically with a fuzzy expert system. Next, we performed an experiment on an enterprise cloud application to measure the fault detection rate of different test suites that are prioritized based on two requirements factors and requirements risks. Finally, we employed a systematic risk estimation mechanism using a fuzzy expert system to make our test prioritization process more efficient and more effective.

We also provide guidance and understanding to practitioners and researchers on the use of requirements and risk-based test prioritization on different types of software systems through the family of empirical studies performed in this dissertation. Further, we compared our novel requirements risks-based approaches with other existing industrial approaches. These comparisons will help practitioners and researchers to effectively employ prioritization techniques on their working environments.

ACKNOWLEDGEMENTS

First and foremost I want to thank my advisor Prof. Hyunsook Do. I would like to thank you for encouraging my research and for allowing me to grow as a research scientist. You have supported me academically and also provided research assistantship for over three years. I would like to thank Prof. Kendall Nygard for his advice since I started my Ph.D. degree at NDSU and his support in many ways. I would also like to thank my committee members, Prof. Saeed Salem, Prof. Simone Ludwig and Prof. Jacob Glower, for their time, insightful comments, and all the support they gave to me for past several years. I am also grateful to all professors who have educated me and the staff members of computer science department at NDSU. I would also like to acknowledge Prof. Janet Knodel for assisting me in several ways. Words cannot express how grateful I am to my beloved wife, Imali, and my beloved daughter, Ranolee, for all their support and all of the sacrifices that they have made over the years. Finally, I thank my mother-in-low, father-in-low, my mother, and my father for all the support they gave to me and for encouraging me to continue education and reach my goals.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Regression testing and maintenance are important activities to ensure high quality for modified software systems. Typically, these activities require a great deal of time, money, and effort, so they can be a significant burden for software companies that often have a time pressure with product release [10, 15]. One way to help this situation is to apply test case prioritization (TCP) that identifies more important test cases (e.g., test cases that are likely to detect more faults) and run them early within the limited time block. With this approach, companies can increase the chances to detect and fix faults early.

Due to their appealing benefits in practice, various test case prioritization techniques have been proposed and studied by researchers and practitioners [19, 60, 62, 66], and many empirical studies have shown the effectiveness of test case prioritization [16, 47]. While the majority of test case prioritization approaches utilize source code information, some researchers have investigated using other software artifacts produced during early development phases, such as system requirements and design documents [8, 36, 59]. For example, Srikanth et al. [59] present an approach that prioritizes test cases using system requirements as well as their importance and fault proneness, and Arafeen and Do [8] cluster test cases using requirements similarities and prioritize them by incorporating code information. Krishnamoorthi and Mary [36] present a technique that prioritizes test cases using requirements and several factors related to the requirements, such as implementation complexity and customer priority.

This trend is encouraging because such artifacts could provide a better understanding about the source of errors. Using system requirements and their risk information, software testers can manage the testing schedule better by identifying more important test cases that are likely to detect defects associated with the risks faced by the system (e.g., safety or security risks). To gain such benefits, a few researchers have started investigating the use of risks with the requirements, and these studies found that using risks along with requirements could improve the effectiveness of test case prioritization [14, 61, 67].

These approaches, however, consider one limited risk type (e.g., fault information collected from the previous version) [14, 61] or do not consider a direct relationship between requirements risks and test cases when they prioritize test cases [67]. Typically, software systems contain various types of risks, thus considering only one limited risk type fails to properly expose important potential risks in the software system. Also, without utilizing a direct relationship between requirements risks and test cases, prioritization

techniques could fail to produce an effective order of test cases that can expose faults early in risk areas. Further, like other test case prioritization research, these studies have evaluated the proposed approaches by measuring how fast the reordered test cases detect faults. However, the approaches utilize risk information to prioritize test cases, so they should be evaluated by measuring whether the detected faults are, indeed, from the locations where risks reside in the product.

1.1. Goals of this Dissertation

In this dissertation, we have proposed new requirements risk-based test prioritization approaches to address these limitations. The goals of this research are to propose new requirements risk-based test case prioritization approaches that can systematically estimate risks residing in software requirements and establish proper relationships between requirements risks and test cases to enhance the overall effectiveness of test case prioritization and detect more faults early in risky components of software systems under regression testing.

1.2. Approaches to Meet these Goals

We followed the following steps to accomplish the aforementioned goals.

First, we proposed a new test case prioritization technique that uses risk levels of potential defect types to identify risky requirements and prioritizes test cases based on the relationship between test cases and these requirements. From this empirical study, we determined that the requirements risk-based technique can positively impact the test prioritization in regressing testing. This research presented in Section 4.1 was published in the Proceedings of the Eighth International Conference on Software Security and Reliability (SERE) in 2014 [28].

Our second approach made the requirements risk estimation process more systematic and precise by reducing subjectivity using a fuzzy expert system. We found that the proposed systematic risk assessment approach that used a fuzzy expert system can address the subjectivity related limitations of our previous approach. We also found that this systematic approach can improve the effectiveness of test prioritization and the efficiency of testing process. This research presented in Section 4.2 was published in the Journal of Information and Software Technology (JIST) in 2016 [29].

Following that, we applied requirements risk-based test prioritization approaches in a very large industrial context using a cloud-based enterprise level application to investigate the effectiveness of requirements risk-based test prioritization on very large systems and we compared its performance with two other requirements-based test prioritization approaches. From the results of this research, we observed that both

requirements and requirements risk-based test prioritization approaches can perform better than one of the prevalent industrial test prioritization techniques (random approach). This research presented in Section 4.3 was published in the Journal of Information and Software Technology (JIST) in 2016 [57].

Finally, we proposed a test prioritization mechanism which was based on a fuzzy expert system (FES) that further reduced the involvement of human experts in risks estimation. From this research, we found that the use of requirements risks and fuzzy expert systems can improve the effectiveness of test prioritization. This research presented in Section 4.3 is under review in the Journal of Information and Software Technology (JIST).

To evaluate our new test prioritization approaches, we defined a new evaluation method that shows how fast the reordered test cases can detect faults in the risky areas. To investigate the effectiveness of our approaches, we designed and performed empirical studies using different types of applications such as open source and industrial programs with multiple versions and requirements documents.

1.3. Organization of this Dissertation

The rest of this paper is structured as follows. Chapter 2 of this dissertation describes related work. Chapter 3 describes the approach of this dissertation. Chapter 4, 5, 6, and 7 describe four empirical studies we performed with their results and implications. Chapter 8 discusses conclusions and future work.

2. BACKGROUND AND RELATED WORK

Test case prioritization provides a way to schedule test cases so that testers can run more important or critical test cases early. Various prioritization techniques have been proposed [66], and some of them have been used by several software organizations [39, 60]. The majority of prioritization techniques have used the information obtained from software source code [16, 49, 54]. For instance, one technique, *total statement coverage prioritization* reorders the test cases in the order of the number of statements they cover. One variation of this technique, *additional statement coverage prioritization* reorders the test cases in the order of number of new statements they cover. Other types of code information for aiding prioritization include code change history, code modification, and fault proneness of code [43, 54]. Beyond code-based information, other software artifact types, such as software requirements and design information, have also been utilized. For example, Srikanth et al. [59] proposed a test case prioritization approach using several requirements-related factors, such as requirements complexity and requirements volatility, for the early detection of severe faults. Krishnamoorthi and Mary [36] also proposed a model to prioritize test cases using the requirements specification to improve the rate of severe fault detection. Arafeen and Do [8] proposed an approach that clusters requirements based on similarities obtained through a text-mining technique and that prioritizes test cases using the requirements-tests relationship. These studies reported that using requirements information improved the effectiveness of prioritization.

In addition to requirements and design information, other researchers have used software risk information to prioritize test cases in order to run test cases to exercise code areas with potential risks as early as possible [61, 67]. Many risk-based testing techniques have adopted Amland's [7] risk model that estimates risk exposure as a product of probability of faults in software components and the impact (e.g., cost or damage) of the corresponding fault if it occurs in the operational environment. In our approaches, we also used the risk exposure of both requirements and risk items to prioritize tests. Stallbaum et al. [61] proposed a technique, RiteDAP (risk-based test case derivation and prioritization), that can automatically generate test case scenarios from activity diagrams and can prioritize test cases using the risks associated with fault information. In this RiteDAP approach, to quantify the risk, probability of failure for each action is estimated by the usage frequency of each action, whereas the damage (impact) caused by that particular failure is estimated through its financial losses. Yoon et al. [67] used the relationship among requirements

risk exposure, risk items, and test cases to determine the order of test cases. Another paper [38] proposed a value-based software engineering framework to improve the software testing process. The proposed multi-objective feature prioritization strategy prioritizes the new features by considering the business importance, quality risks, testing costs, and the market pressure. Further, Felderer and Schieferdecker [23] 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. [20] 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. For example, the survey discusses a model-based security testing approach proposed by Zech [70] using risk analysis for cloud computing environments. In Zech's proposed approach, misuse cases are used on a model-driven approach for test code generation. These existing papers on risk-based testing demonstrate that the use of risks in the software systems can help find critical functional defects that may cause severe security or safety related issues.

Another research area that is relevant to our work is fuzzy expert systems. Fuzzy expert systems have been used in areas that require expert knowledge to make decisions while minimizing several issues, such as uncertainties and subjectivity, in the decision-making process. In general, fuzzy expert systems are applied to various domains, such as diagnosing diseases in the medical field [4, 32], risk assessment in aviation [26], risk assessment in construction projects [13], and selecting superior stocks on the stock exchange [21]. For instance, Adeli and Neshat [4] proposed a fuzzy expert system to diagnose heart disease. Recently, fuzzy expert systems have been used in software engineering areas such as software development effort prediction [5], software cost estimations [35], and risk analysis for e-commerce development [45]. For instance, Ahmed et al. [5] developed a fuzzy expert system to obtain accurate software cost and schedule estimation by managing the uncertainties and imprecision that exist in the early stages of software development. More recently, some researchers applied fuzzy expert systems to regression testing. Schwartz and Do [53] used a fuzzy expert system to determine the most cost-effective regression testing technique for different testing environments by addressing the limitations of existing, adaptive regression testing strategies. Xu et al. [65] applied a fuzzy expert system to deal with the inaccurate and subjective issues present during the test case selection process of regression testing. In this work, we used a fuzzy expert system with requirements and their risk information to improve risk estimation processes, thus improving the effectiveness of test case prioritization.

3. APPROACH

In this section we describe the overall approach of our research. Our goals of this dissertation are to propose new systematic requirements and risk-based test prioritization approaches by establishing proper relationships between requirements risks and test cases to increase the effectiveness of test prioritization in regression testing and also to detect more faults early in risky components at a faster rate. To accomplish these goals, we followed the steps given below in our approach:

- Investigate the effectiveness of requirements risk-based technique in test prioritization
- Perform an empirical study which emphasized reducing the threats to validity in the previous study
- Investigate the effectiveness of requirements risk-based test prioritization on a very large industrial system
- Investigate the impact of direct use of expert systems in requirements risk-based regression testing
- Devise a new evaluation technique to measure the effectiveness of test prioritization techniques in terms of detecting more faults in risky components

3.1. Investigate the effectiveness of requirements risk-based technique in test prioritization

In the first step of our approach, we investigated the direct relationship between requirements risks and test cases in order to produce prioritized test cases for regression testing not only to detect more faults early (improve regression testing effectiveness) but also to detect more faults in risky components of software systems.

To accomplish this step, at the beginning, we identified effective risk indicators and items that reflect crucial risks in software requirements. We conducted a controlled experiment on a mid-size software system that was compatible in estimating the risks of software requirements with respect to several risk indicators and risk items. We used the impact and the likelihood of requirements risks to assess requirements' risks and establish a direct relationship between software requirements risks and test cases in order to obtain an effective prioritized test suite.

We also devised a new evaluation technique to measure the effectiveness of the new prioritization technique in terms of detecting more faults in risky components. We applied this new evaluation technique to determine whether the proposed risk-based test prioritization technique is effective in detecting more faults in risky components of a software system.

In this research, we found that the use of requirements and their risks can help improve the effectiveness of test case prioritization. We also found that the use of requirements and their risks can help early detection of faults that reside in the risky components.

This research also laid the foundation to achieve our major goals of this dissertation. Section 4.1 describes this approach in detail.

3.2. Expand the previous study reducing the threats to validity

Once we determined that our requirements risk-based approach can be effective in test prioritization, our next step was to improve our previous approach by reducing subjectivity related limitations. Therefore, in this step, we investigated how artificial intelligence techniques (Fuzzy Expert System) which are capable of emulating the reasoning of human experts can contribute to reduction of subjectivity in risk estimation that occurs when using the requirements which are specified using natural language.

To do this, we performed an empirical study using a systematic risk assessment approach which is based on a fuzzy expert system. We found that the proposed systematic, risk-based approach was capable of outdoing the control techniques used in this research including state-of-art test prioritization technique. We also found that our proposed systematic approach has the ability to detect a significantly higher number of faults in risky components at low test execution rates. With this step, we were able to further verify the effectiveness of our requirements risk-based approach.

We also expanded the controlled experiment by incorporating industrial and open source software systems and added more control techniques for the experiment. More details of this empirical study is available in Section 4.2.

3.3. Investigate the effectiveness of requirements risk-based test prioritization on a very large industrial system

In this step, we investigated the effectiveness of the requirements risk-based test case prioritization approach on a large industrial system. We applied and compared the performance of requirements risk-based test case prioritization approach and two other requirements-based test prioritization techniques on a very large enterprise-level cloud-based software system-as-a-service (SaaS) application.

In the empirical study, we extracted risk information from the large system to perform risk-based prioritization. Section 4.3 describes this approach in detail.

3.4. Investigate the impact of direct use of expert systems in requirements risk-based regression testing

Based on our findings from previous steps, we further examined the effectiveness of semi-automated requirements risk assessment techniques and the direct use of expert systems in risk estimation without ignoring the expert knowledge of risk estimation during test prioritization.

At this final step, we focused on reducing the time and cost for test prioritization while further increasing test effectiveness in a systematic way. In this way, we were able to further verify that our requirements risk-based approach is effective and compelling in industrial settings.

From the results of this research, we found that the use of requirements risks and fuzzy expert systems can not only improve the effectiveness of test prioritization but also make the risk estimation process more efficient. We also observed that the use of fuzzy expert systems can help reduce the risk estimation process's subjectivity. More detailed information about this empirical study is available in Section 4.4.

4. EMPIRICAL STUDIES

We performed a family of empirical studies to investigate the approaches described in Section 3. The following subsections describe each approach in detail.

4.1. Effective Regression Testing Using Requirements and Risks

The use of system requirements and their risks enables software testers to identify more important test cases that can reveal faults associated with risky components. Having identified those test cases, software testers can manage the testing schedule more effectively by running such test cases earlier so that they can fix faults sooner. A few research studies found that using risks along with requirements could improve the effectiveness of test case prioritization [14, 61, 67]. These approaches, however, have several limitations such as not considering a direct relationship between requirements risks and test cases when they prioritize test cases or improper use of requirements risks in test prioritization. In addition, these previous approaches are lack of adequate evaluation method.

To address these limitations, we propose a new test case prioritization technique that uses risk levels of potential defect types to identify risky requirements and that prioritizes test cases based on the relationship between test cases and these requirements. In this research, we implemented a the new requirements risk-based prioritization technique and evaluated it considering whether the proposed approach can detect faults earlier overall.

To evaluate our approach, we define a new evaluation method that shows how fast the reordered test cases can detect faults in the risky areas. To investigate the effectiveness of our approach, we have designed and performed an empirical study using an open source program written in Java with multiple versions and requirements documents. Our results show that the new technique is effective in finding faults early and even better in finding faults in the risky components earlier than the existing techniques.

4.1.1. Approach

In this section, we describe the prioritization approach that uses system requirements risks, which has five main steps:

1. Estimate risks by correlating with requirements
2. Calculate the risk weights for requirements

3. Calculate the risk exposure values
4. Evaluate additional factors to prioritize requirements
5. Prioritize requirements and test cases

Figure 4.1 gives an overview of the proposed technique. The main steps of the approach are shown in light blue boxes while the inputs and outputs for each step are shown in the ovals. The first four steps are used for calculating requirements priorities, and the last step is used for prioritizing test cases based on the results produced by the first four steps. Note that "Req-Test Case Mapping" activity in the figure is not listed as a part of steps, but this activity is required for the last step ("Prioritize requirements and test cases") to provide the relationship between requirements and test cases. The following subsections describe the rest of the steps in detail.

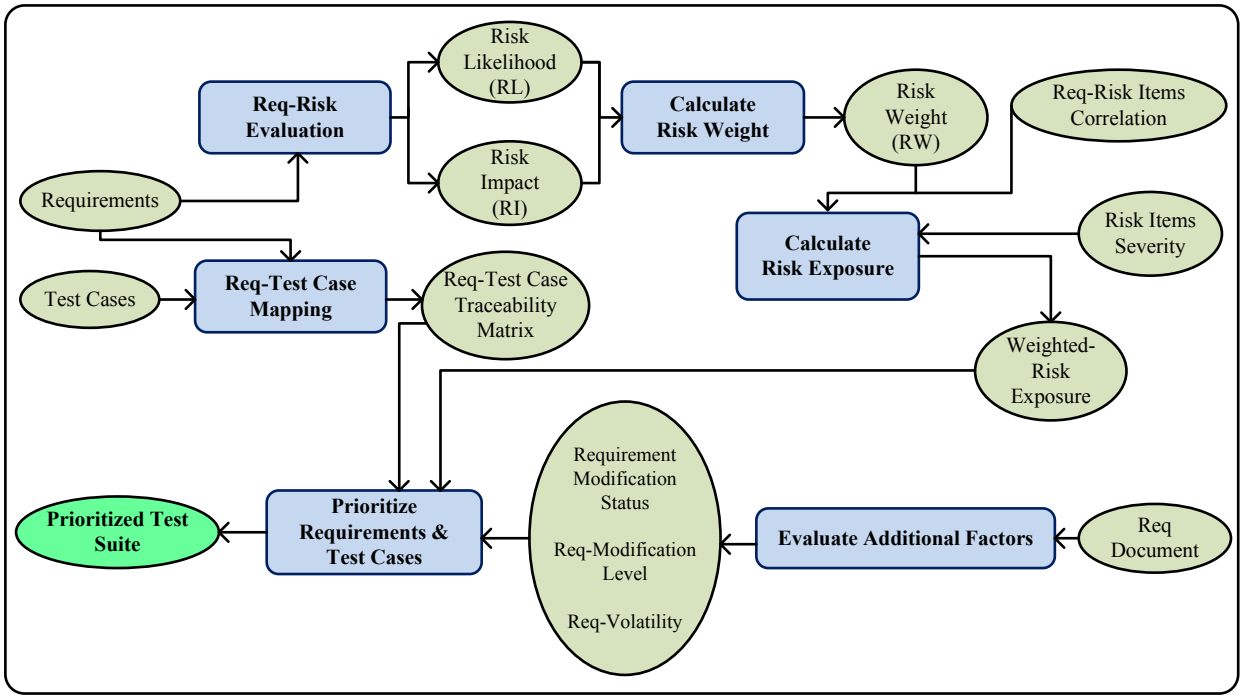

Figure 4.1. Overview of Requirements Risk-Based Approach

4.1.1.1. Estimate Risks by Correlating with Requirements

Risks related to each requirement of the system were estimated by two graduate students who have several years of software industry experience. The graduate students surveyed the risks related to software product requirements found in the literature [9, 63]. Two risk factors were considered in estimating requirement risks; Risk Likelihood (RL) and Risk Impact (RI), which are commonly used by other researchers [7].

Not all software risks have the same impact on the software product. Some risks can be acceptable, and some can be catastrophic. Further, their probabilities of occurrence can vary. Thus, for each requirement, we estimated the probability of a risk's occurrence associated with a requirement (RL) and the degree of the possible damage (RI) caused by a risk for a requirement.

In this experiment, to estimate the RL values, graduate students utilized risk indicators and weights as shown in Table 4.1. The four risk indicators are new requirements, potential security threats residing in the requirements, requirements complexity, and the size of the functions used to implement the requirements. The weights range from 1 to 5, with 5 being considered the highest importance. By examining these risk indicators in the system, we can understand the system's risk level. For example, often new requirements tend to introduce new faults to the system because new requirements have not been validated yet, so having new requirements could increase the risk level. Potential security threats residing in the requirements, such as threats related to confidentiality, integrity, availability, and privacy can also increase the risk level. When requirements complexity increases, developers might misunderstand the requirements and produce a wrong product; thus, requirements complexity can affect the risk level in the system. Further, similar to requirements complexity, large size functions that are associated with requirements could increase the risk level by introducing unexpected errors. In our experiment, Line of code (LOC) was used to measure the size of the functions.

Table 4.1. Weights of risk indicators

Indicators	Weights
New requirements	5
Potential security threats	5
Requirements complexity	3
Size	1

For each requirement, each indicator's contribution of risk level is quantified using a scale range from 1 to 10 where 1 indicate less probability whereas 10 indicates the highest probability. The Equation 4.1 is used to calculate the RL value.

$$RL_{(Req_j)} = \sum_{i=1}^{n} (W_i * R_{ji}) \tag{4.1}$$

where n is the number of indicators; W_i is the weight of the indicator, and R_{ji} is the risk level value of the requirement, Req_j, under indicator i.

The RL value for each requirement was calculated by two graduate students individually, and these two RL values were averaged to obtained the final $RL_{(Avg)}$ value. The following example shows the calculation of the RL value for the UC1S1 requirement ("The health care personnel creates a patient as a new user of the system.") and the second column of Table 4.2 shows the averaged RL values.

$RL_{(UC1S1)}$ = (5*0) + (5*5) + (3*5)+ (1*6)= 46

Table 4.2. Risk Weights of Requirements

Requirements	$RL_{(Avg)}$	$RI_{(Avg)}$	RW
UC1S1	47.5	4.75	2.26
UC2S1	46.5	8.00	3.72
UC3S4	74.5	8.75	6.52
UC6S1	57.5	6.00	3.45
UC6S2	66.5	6.75	4.49
UC8S1	52.5	4.25	2.23
UC9S1	75.5	6.25	4.72
:	:	:	:
UC34S6	67.5	5.25	3.54

The magnitude of RI for each requirement was estimated in terms of technical impact (i.e., how a potential failure related to a requirement affects the system's performance) and the impact on target users' activities associated with the requirement. Assuming both impact indicators are equally important RI of requirements are calculated using the Equation 4.2. For each requirement, impact level of each indicator is quantified using a scale range from 1 to 10 where 1 indicates the lowest impact and 10 indicates the highest impact level.

$$RI_{(Req_j)} = (T_j + A_j)/2 \tag{4.2}$$

where T_j is the technical impact level of Req_j and A_j is the impact level on target user activities in terms of Req_j.

The following example shows the calculation of RI for the UC1S1 requirement. Similar to RL calculation, the averaged RI values are showed in the third column of Table 4.2.

$RI_{(UC1S1)}$ = (4 + 5)/2 = 4.5

The $RL_{(Avg)}$ and $RI_{(Avg)}$ values calculated at this step are used in the next step to calculate the risk weights of requirements to quantify the risk level of each requirement.

4.1.1.2. Calculate the Risk Weights for Requirements

Using two risk weight factors that we obtained from the previous step, risk likelihood (RL) and risk impact (RI), we calculate risk weight (RW) by following Amland's risk model [7].

The RW values are obtained using the following equation:

$$RW_{(Req_j)} = RL_{(Avg)j} * RI_{(Avg)j} * 0.01 \tag{4.3}$$

where $RL_{(Avg)j}$ is the averaged risk likelihood of Req_j and $RI_{(Avg)j}$ is the averaged risk impact of Req_j.

Table 4.2 shows a portion of RW values for *iTrust* requirements (version 1). The $\text{RL}_{(Avg)}$ values range from 0 to 140. A value of 0 indicates no likelihood of risk, whereas 140 indicates the highest likelihood of risk. The $\text{RI}_{(Avg)}$ values range from 0 to 10. A value of 0 indicates no impact of risk, whereas 10 indicates the highest impact of risk. In this example, for the UC1S1 requirement, we obtained 47.5 and 4.75 as RL and RI, respectively. The RW value for UC1S1 was obtained by applying Equation 4.3 as follows:

$\text{RW}_{(UC1S1)}$ = 47.5*4.75*0.01 = 2.26

The RW values range from 0 to 14. A value of 0 means no risk, and 14 means the highest risk. The RW value for UC1S1 is 2.26, which indicates a low risk. In Table 4.2, UC3S4 has the highest RW (6.52), and UC8S1 has the lowest RW (2.23). In fact, the UC3S4 requirement allows a licensed health care professional (LHCP) to access laboratory results of patients, to see upcoming appointments, and to reject or accept comprehensive reports. There is a relatively high risk of misusing a patient's confidential information, such as laboratory results, by LHCP or a hacker. Due to security and complexity related risks reside in this requirement, it obtained relatively a high RL value. If the confidentiality of patients' sensitive data are breached or patients' privacy is violated, severe consequences such as monetary and reputation losses may occur. The performance of the system may go down and the users of the system may not be able to use some functionalities. Thus, the requirement obtained a relatively high RI value.

4.1.1.3. Calculate the Risk Exposure Values

So far, we explained how to calculate RW values for the requirements. Although the RW values of the requirements illustrate how risky each requirement is in terms of the risks identified for the requirements themselves, RW does not provide information about whether the requirements are associated with potential defect types of the system. (Note: One particular requirement can be associated with multiple defect types,

which indicates that such a requirement has a high potential to expose several defects.) Using this information, we can identify the requirements that are related to common failures of a software product, thus we can assign high priorities, which will be used in prioritizing test cases to such requirements.

To obtain such information, we adopted the process defined by Yoon et al. [67], and improved it by considering weights for each risk item based on our experiences. The modified process is as follows:
(1) Identify software risk items (RiIM) as shown in Table 4.3. Risk items are potential defect types of a particular system. We used software risk items derived from studies based on different applications and standards [9, 30, 63]. The input problem is one example. During system operation, input data can cause several problems. For instance, a system may crash or perform erroneously if the input data are not validated before they are executed or if the data are beyond the valid boundary.
(2) Calculate risk exposure values for the risk items. Here, risk exposure (RE) values quantify the risk level for the risk items.

To find the probability of fault occurrence, we considered the association of risk items and requirements. A requirement may associate with several risk items. A risk item can have many requirements associated with it. When a particular risk item associated with more requirements, its probability of fault occurrence will increase. In order to determine the impact of risk items, the risk weight (RW) of requirements is employed. The impact of risk items increase when its associated requirements have high risk weights. Table 4.4 shows a matrix to calculate the risk exposure (RE) values for risk items and the weighted risk exposure (W-RE) values for requirements.

Table 4.3. Software product-risk items

Risk Item	Abbreviation
Input Problem	IP
Output Problem	OP
Calculation	Calc
Interactions	Inac
Error Handling	ErHa
Startup/ShutDown	St/Sh

The matrix lists requirements, risk weights for requirements, and a set of risk items. Each risk item ($RiIM_x$) has a severity value (SV_x) that indicates how risky the item is. The severity values are defined in Table 4.5. If a risk item is associated with a certain requirement, the C_{mx} value is 1; otherwise, the value is 0. Again, multiple associations between risk times and requirements can exists. The last row in the

Table 4.4. Risk Exposure and Weighted Risk Exposure Matrix

Requirements	Risk Weights	Risk Items				W-RE
		RiIM1 (SV1)	... ...	RiIMx (SVx)	RiIMn (SVn)	
Req_1	RW(Req_1)	C_{11}		C_{1x}	C_{1n}	W-RE(Req_1)
...	...	...	...	...	...	...
Req_y	RW(Req_y)	C_{y1}	...	C_{yx}	C_{yn}	W-RE(Req_y)
Req_m	RW(Req_m)	C_{m1}	...	C_{mx}	C_{mn}	W-RE(Req_m)
		RE(RiIM1)	...	RE(RiIMx)	RE(RiIMn)	

matrix shows RE values for risk items, and RE values are calculated using Equation 4.4; the last column shows W-RE values that are calculated by incorporating RE values with severity values of risk items for each requirement. The final outcome of this step, the W-RE values, is used to prioritize requirements.

Equations 4.4 and 4.5 are used to calculate the risk exposure values for risk items and the weighted risk exposure values for requirements.

$$RE_{(RiIM_x)} = \sum_{i=1}^{m} (RW_{(Req_y)} * C_{yx}) \tag{4.4}$$

where m is the number of requirements, $RW_{(Req_y)}$ is the risk weight of Req_y, and C_{yx} is 1 when requirement Req_y is associated with the risk item $RiIM_x$ or 0 otherwise.

$$W - RE_{(Req_y)} = \sum_{i=1}^{n} (RE_{(RiIMx)} * C_{yx} * SV_x) \tag{4.5}$$

where n is the number of risk items for the system; $RE_{(RiIMx)}$ is the risk exposure value of the risk item, $RiIM_x$; C_{yx} indicates the correlation between the requirement, Req_y, and the risk item, $RiIM_x$; and SV_x is the severity value of the risk item, $RiIM_x$.

Table 4.5. Severity of risk items

Severity Value	Description
1	Slightly critical risk item
2	Moderately critical risk item
3	Very critical risk item
4	Most critical risk item

Table 4.6 shows a sample data set collected from our experiment. In this example, we can see that the output problem risk item is associated with all requirements except for UC6S2. Thus, the risk exposure value of the output problem risk item can be calculated by using Equation 4.4 as follows:

$RE_{(OP)}$ = (2.26*1) + (2.26*1) + (3.72*1) + (3.72*1) +...+ (4.49*0) +...+ (3.54*1) = 628.74

Table 4.6. Example of Risk Exposure and Weighted Risk Exposure of iTrust

Requirement	TC ID	RW	OP	ErHa	Inac	IP	Calc	Sh/St	W-RE
Risk Items Severity Values			4	1	3	4	3	2	
UC1S1	TC1	2.26	1	1	0	1	0	0	4876.48
UC1S1	TC2	2.26	1	1	0	1	0	0	4876.48
UC2S1	TC5	3.72	1	1	1	1	0	0	6591.85
UC2S1	TC6	3.72	1	1	1	1	0	0	6591.85
UC3S4	TC7	6.52	1	0	1	0	0	0	4230.32
UC3S4	TC8	6.52	1	0	1	0	0	0	4230.32
UC6S1	TC9	3.45	1	0	0	0	0	0	2514.95
UC6S1	TC10	3.45	1	0	0	0	0	0	2514.95
UC6S2	TC11	4.49	0	0	1	0	0	0	1715.37
UC6S2	TC12	4.49	0	0	1	0	0	0	1715.37
UC8S1	TC13	2.23	1	1	0	1	1	0	5375.18
UC9S1	TC14	4.72	1	1	1	0	0	0	4804.13
UC9S1	TC15	4.72	1	1	1	0	0	0	4804.13
:	:	:	:	:	:	:	:	:	:
UC34S6	TC122	3.54	1	1	1	1	0	0	6591.85
Risk Exposure (RE)			628.74	573.82	571.80	446.93	166.24	79.87	

Because the output problem has a high RE value compared to other risk items, it implies that the output problem is a highly risky area for this product.

After calculating RE values, we calculate the W-RE values for each requirement by following Equation 4.5. For instance, we obtain the W-RE value for UC1S1 as follows.

W-RE$_{(UC1S1)}$ = (628.74*1*4) + (573.82*1*1) + (571.80*0*3) + (446.93*1*4) + (166.24*0*3) + (79.87*0*2) = 4876.48

After calculating the W-RE values for requirements, we prioritize requirements by their W-RE in descending order. Table 4.7 shows a portion of our data. From the table, we can see that each requirement has one or more corresponding test cases (the last subsection explains how to map these two) and requirements are appeared by their W-RE in descending order. When multiple requirements have the same value (e.g., UC33S1 and UC26S2 have the same value, 7250.29, in the table), we need to decide which one should be picked first among others. To aid that process, we consider additional factors for prioritization, and the next section explains them.

4.1.1.4. Evaluate Additional Factors to Prioritize Requirements

We utilize the W-RE values as a primary factor to prioritize requirements, but if multiple requirements have the same value, we need to have a way to break the tie. One simple way is to randomly choose a candidate and repeat the process until all remaining requirements with the same value are chosen. In

Table 4.7. Example of Prioritized Test Suite - iTrust

Requirement	TC-ID	W-RE	RM	RML	VL
UC33S1	TC117	7250.29	1	10	6
UC26S2	TC94	7250.29	0	0	6
UC21	TC30	7090.55	1	10	7
UC21	TC80	7090.55	1	10	7
:	:	:	:	:	:
UC12	TC26	4735.35	0	0	3
UC30S3	TC100	4236.64	1	10	6
UC30S1	TC99	4236.64	1	4	6
UC3S3	TC43	2947.63	0	0	3
UC24	TC91	2860.24	0	0	7
UC25	TC92	2860.24	0	0	6
:	:	:	:	:	:
UC2S2	TC77	1787.72	0	0	2

this study, however, we consider three factors that can be potentially effective in finding error-prone requirements [8, 25, 36, 58]: requirement modification status, level of modification, and level of requirement volatility.

Requirement modification and its modification level were determined by comparing two consecutive versions. For new requirements, we assigned the highest level of modification because new requirements can cause serious source code modifications and can introduce new faults to the system [8]. Volatile requirements are susceptible to have faults in later versions of the system. However, the impact of volatile requirements in terms of introducing system faults is relatively low compared to requirement modifications. We used modification status as the second factor for prioritization, followed by the level of modification and the level of requirement volatility. If the requirements are still tied after applying all the factors, they are ordered randomly.

Table 4.7 shows the values for these three factors applied to each requirement. RM, requirement modification status, has two values to indicate whether the requirement is modified (1) or not (0). RML indicates the level of modification. VL indicates the level of requirements volatility. These values ranges from 0 to 10. The value 0 indicates the lowest level and 10 indicates the highest level. Again, for the first two requirements, UC33S1 and UC26S2, in the table, because they have the same W-RE value, their priority is decided by comparing the next factor, RM. UC33S1 has 1 and UC26S2 has 0 for RM, so UC33S1 is picked first for prioritization.

4.1.1.5. Prioritize Requirements and Test Cases

After collecting all the values we described, we prioritize requirements using the W-RE values and the additional factors. Next, to obtain prioritized test cases, we need a traceability matrix that shows mapping relationships between requirements and test cases. In our case, we used the traceability matrix that came with *iTrust*. (The developers created requirements and tests mapping information).

Often, multiple test cases are created for a single requirement, meaning that the prioritization values for all tests with the same requirement are the same. Thus, after mapping test cases to their corresponding requirements, we randomly order tests with the same priority. For *iTrust*, only one or two test cases are associated with the same requirement.

4.1.2. Empirical Study

In this study, we investigate the following research questions:

RQ1: Can requirements risk-based test case prioritization improve the rate of fault detection for test suites?

RQ2: Can requirements risk-based test case prioritization find more faults in the risky components early?

4.1.2.1. Object of Analysis

Table 4.8. Experiment Object and Associated Data

Versions	Size (KLOCs)	Requirements	Test Cases	Mutation Faults	Mutation Groups
v1	24.42	91	122	54	13
v2	25.93	105	142	71	12
v3	26.70	108	157	75	12

An open source application *(iTrust)* was utilized for this experiment. The *iTrust* program is a patient-centric electronic health record system which was developed by the RealSearch Research Group at North Carolina State University. Four versions of the *iTrust* system (version 0, 1, 2, and 3) were used in our experiment. The test cases used in this study were functional test cases associated with requirements and written by *iTrust* system developers. Each version's metrics are listed in Table 4.8. The metrics for *v0*, which is the base version of *iTrust*, are not listed in the table because regression testing starts with the second release of the system. We, however, use information from *v0* to obtain mutants for *v1*. Our experiment requires faults in the program, so we utilize mutation faults from our previous study [8].

4.1.3. Variables and Measures

4.1.3.1. Independent Variable

Our study manipulated one independent variable, test case prioritization technique. We considered four control techniques and one heuristic prioritization technique as follows:

- Control Techniques
  - Original (*Torig*): The object program provides the testing scripts. *Torig* executes test cases in the order in which they are available in the original testing script.
  - Code metric (*Tcm*): This technique uses a code metric that we defined in our previous study [8]. The code metric is calculated using three types of information obtained from source code, Line of Code (LOC), Nested Block Depth (NBD), and McCabe Cyclomatic Complexity (MCC), which are considered good predictors for finding error-prone modules [52, 69].[1]
  - Requirements-based clustering: We consider two requirements-based clustering techniques proposed by Arafeen and Do [8] as follows:
    - Tcl-orig-prior (*Tcop*): This technique uses the original test case order for prioritization and the prioritized cluster order for selection.
    - Tcl-cm-prior (*Tccp*): This technique uses the code metric for prioritization and the prioritized cluster order for selection.

    The previous study [8] used several cluster sizes, but in this study, to simplify the comparisons, we chose cluster sizes of 10 and 20 which showed moderate and the best results, respectively.
- Heuristic (*Trrb*): The heuristic technique uses requirements and their risks to prioritize test cases as described in Section 4.1.1.

4.1.3.2. Dependent Variables

We considered two dependent variables as follows:

- Average Percentage of Fault Detection (APFD): The APFD [18, 41, 50] value represents the average for the percentage of fault detection during the execution of a particular 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

[1] $Tcm = \frac{NBD}{Max(NBD)} + \frac{MCC}{Max(MCC)} + \frac{LOC}{Max(LOC)}$

run on the x-axis. The prioritization techniques are being considered as better techniques when their APFD values are closer to 100, and the technique that obtains the highest APFD value is considered to be the best prioritization technique. Let T be a test suite containing n test cases, and let F be a set of m faults revealed by T. Let TF_i be the first test case in ordering T' of T, which reveals fault i. The APFD value for test suite T' is given by the equation:

$$APFD = 1 - \frac{TF_1 + TF_2 + \cdots + TF_m}{nm} + \frac{1}{2n}$$

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 Figure 4.2. 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 E-C-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%.

- Percentage of Total Risk Severity Weight (PTRSW): PTRSW shows how effective the test suite is for finding more faults in the system's risky components as early as possible. The PTRSW value ranges from 0% to 100%. If we run all test cases, the PTRSW value reaches 100%. If we execute a portion of the test cases by stopping at a certain point, the PTRSW value is under 100% and the higher the value is, the better the technique is. The following equation shows how to calculate PTRSW:

  PTRSW = (TRSW/GTRSW)*100%

  The Total Risk Severity Weight (TRSW) and the Grand Total Risk Severity Weight (GTRSW) are explained in detail in the following section.

Test	Fault									
	1	2	3	4	5	6	7	8	9	10
A	X				X					
B						X	X			
C	X	X	X	X	X	X	X			
D					X					
E								X	X	X

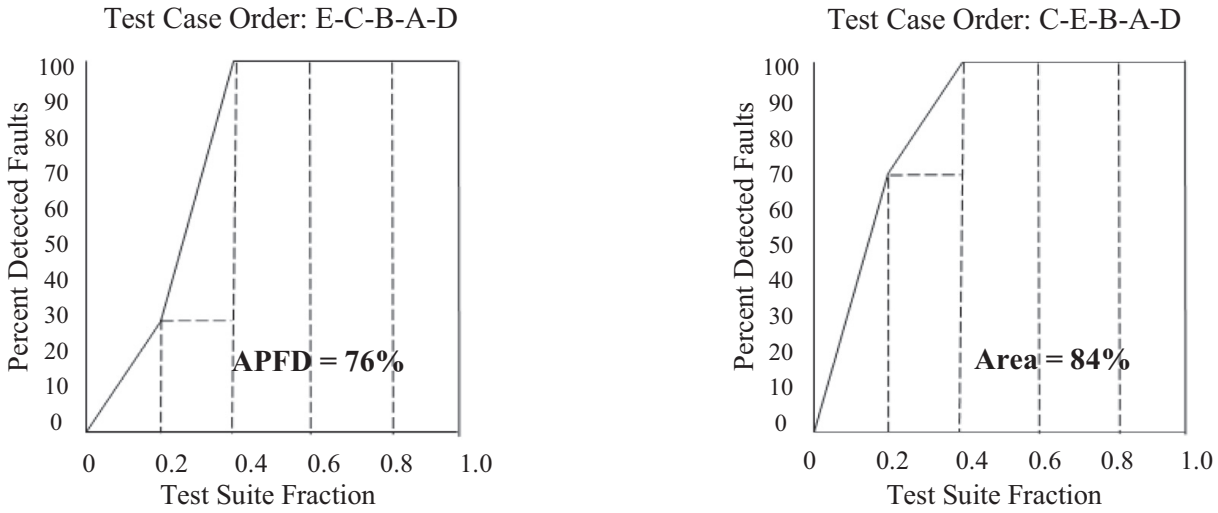

Figure 4.2. APFD Example (Rothermel et al.)

4.1.4. Experimental Setup and Procedure

The requirements risk-based approach requires several types of information, such as requirements-tests mapping information, requirements modification information, risk exposure (RE) and weighted risk exposure (W-RE) values, requirements-classes/methods association information, and fault-test association information.

The test cases and requirements-tests mapping information came with *iTrust*. To obtain requirements modification information, we manually inspected the requirements across all versions. Risks associated with the requirements were estimated and documented as explained in Section 4.1.1.

The RE values for the risk items and the W-RE values for requirements were obtained using the steps explained in Section 4.1.1. To obtain fault data for this experiment, a set of mutation faults created in the previous study [8] was used. We created mutant groups by randomly choosing n mutation faults (between 1 and 10) from those available with that particular version of the program. We repeated this process for each version, and we obtained 13, 12, and 12 mutant groups for v1, v2, and v3, respectively. Because we need to

know the locations for the mutation faults to measure PTRSW, we used the ByteMe mutation analysis tool to identify the classes and methods altered by mutation faults.

To locate the requirements affected by the modification, using the mutation analysis tool, we built a requirements-classes/methods trace file that shows which classes and methods were used to implement a particular requirement. This information, along with the requirements risks, was used to estimate risks for system classes. In this way, a risk caused by a particular mutation fault in a system component was estimated, and the Risk Severity Weight (RSW) value was assigned to the mutation fault. The RSW value ranges between 0 and 10. RSW is 0 if a mutation fault does not cause any risk, and it becomes 10 if a mutation fault causes severe risks in a critical component.

Table 4.9. Example Test Cases, TRSW and Associated Data for Original Test Order-Version 2

Test cases (Orig v2)	Mutations			TRSW	Percentage of Total Risk Severity Weight (PTRSW)
	M1	...	M54		
TC1	3	0	1	4	
:	0	0	0	0	
TC18	0	0	0	0	
:	0	0	0	0	
Sub-total of TRSW after 12.5% test case execution				4	0.83%
:	:	:	:	:	
TC36	0	0	0	0	
Sub-total of TRSW after 25% test case execution				19	3.94%
:	:	:	:	:	
TC71	0	0	0	0	
Sub-total of TRSW after 50% test case execution				225	46.68%
:	:	:	:	:	
TC107	0	0	0	0	
Sub-total of TRSW after 75% test case execution				276	57.26%
:	:	:	:	:	
TC142	0	0	0	0	
Grand total after execution the entire test suite (GTRSW)				482	

A mutation fault is detected by one or more test cases. When multiple test cases detect the same mutation fault, all those test cases are assigned the same RSW value. Also, a single test case can detect multiple mutation faults. In that case, all RSW values for the associated mutation faults are added to obtain the Total Risk Severity Weight (TRSW) for that test case. The TRSW values for test cases are summed to get the Grand Total Risk Severity Weight (GTRSW) for the test suite. Table 4.9 shows sample RSW, TRSW, and GTRSW values calculated for the original test order of *iTrust* Version 2. Columns 2, 3, and 4 contain RSW values of sample test cases. For example, when we execute 12.5%, 25%, 50%, and 75% of the original order of test cases, the PTRSW values are 0.83%, 3.94%, 46.68%, and 57.26%, respectively. As we can

see from the results in the next section, these values are relatively low compared to other techniques, which means the original order of test cases is not able to detect faults in the risky components early.

After collecting all the required data, we performed prioritization techniques. We calculated the APFD and PTRSW values for each test case obtained from the techniques.

4.1.5. Data and Analysis

In this section, we present the results of our study and data analyses for each research question. (We discuss further implications of the data and results in Section 4.1.6.)

4.1.5.1. The effectiveness of requirements risk-based prioritization for improving the rate of fault detection of test cases (RQ1)

Our first research question (RQ1) considers whether a requirements risk-based approach can help improve the effectiveness of test case prioritization. To answer this question, we compare techniques based on the results shown in Table 4.10. The first column in the table lists the control techniques. CS-10 and CS-20 in the cluster-based techniques denote the cluster size that we explained in Section 4.1.3.1. The second column represents the average APFD value of each control technique for version 1, and the third column shows the improvement rates of our requirements risk-based approach over the control techniques. Subsequent columns repeat the same format (APFD values of the controls and the heuristic's improvement rates over the controls) for versions 2 and 3.

The results in Table 4.10 indicate that requirements risk-based prioritization outperformed the first two control techniques (*Torig* and *Tcm*) across all versions; the improvement rates ranged from 24% to 96%. In the case of cluster-based control techniques, the results varied. At a cluster size of 10 (CS-10), the heuristic outperformed the controls for the first two versions, but for version 3, it was not better than the controls. At a cluster size 20 (CS-20), the heuristic did not yield improvement over the controls.

Table 4.10. APFD Comparison And Improvement Over Controls

Control Technique	iTrust-v1		iTrust-v2		iTrust-v3	
	APFD	Improvement over controls (%)	APFD	Improvement over controls (%)	APFD	Improvement over controls (%)
Torig	43.76	37	47.81	27	28.83	96
Tcm	45.77	31	48.80	24	44.84	26
Tcl-orig-prior($CS-10$)	43.59	38	57.28	6	70.60	-20
Tcl-cm-prior($CS-10$)	39.51	52	57.78	5	75.31	-25
Tcl-orig-prior($CS-20$)	72.37	-17	63.42	-4	72.06	-22
Tcl-cm-prior($CS-20$)	75.45	-20	65.33	-7	72.41	-22

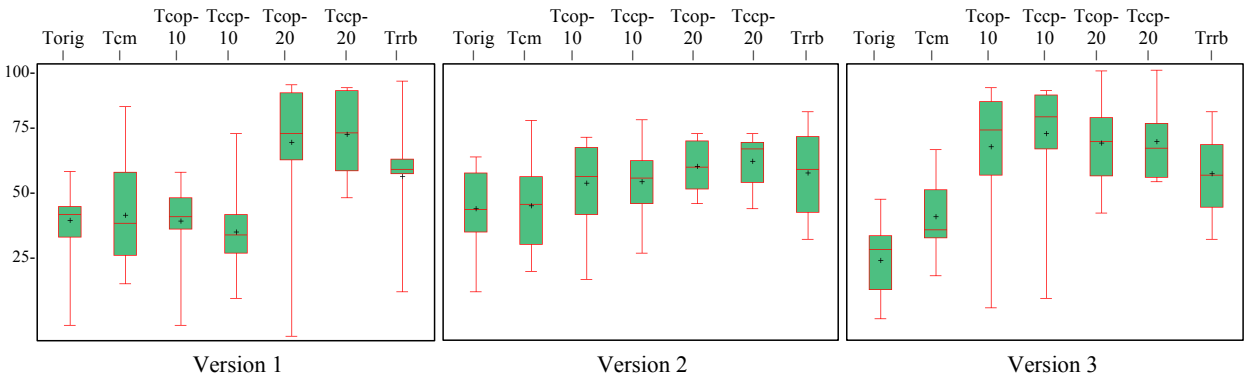

Figure 4.3. APFD Boxplots for All Controls and Heuristic

To visualize our results, we illustrate them in boxplots as shown in Figure 4.3, which presents the APFD values for all techniques and versions. The horizontal axis corresponds to the techniques while the vertical axis represents the APFD values. Each boxplot for version 1 has 13 data points, and for versions 2 and 3, 12 data points are presented.

Examining the boxplots, we observe similar trends from the tables (average values). For versions 1 and 2, the heuristic outperformed the control techniques except for two cases, and for version 3, the heuristic only outperformed two control techniques. The boxplots for version 1 show a wider distribution than the other two versions, and the distance between the minimum and maximum values is very wide for *Tcop-20* and *Trrb*. For version 2, the data distributions across techniques are similar, and they do not show extremely low or high values unlike the results for version 1. For version 3, the control techniques that used clustering with a cluster size of 10 show a wider distribution than other techniques. When we examined the *iTrust* requirements, we found that newly added requirements caused a considerable number of code modifications. The cluster based prioritization technique used code modification information and requirement implementation priority as the primary criteria to prioritize requirement clusters. This factor might have influenced the cluster based approach to produce better results over our approach.

4.1.5.2. The effectiveness of requirements risk-based prioritization for finding faults associated with risky components (RQ2)

With RQ1, we examined whether prioritization that utilizes requirements and their risks can be effective at finding faults earlier. The results are encouraging, but we do not know whether the early detected faults are, indeed, the faults that reside in the risky components. Thus, our second research question (RQ2) considers whether the requirements risk-based approach can be more effective in finding faults associated with risky components early compared to the control.

To investigate this research question, we measured the PTRSW values that we described in Section 4.1.3.2, which show how fast the technique can detect faults in the risky components. For this evaluation, we considered two control techniques: original (*Torig*) that produced lowest APFD value and the cluster-based technique with a cluster size of 20 that produced the best results (*Tccp*) for the RQ1. Table 4.11 shows the results when we foreshorten the test execution process by 12.5%, 25%, 50%, and 75%. This means that, for a 50% cutting ratio, we simulate the effects of having the testing process halted half way through.

The results show that our risk based approach can detect more faults in the risky components early than the controls except for two cases (*Tccp* at 12.5% and 25% execution rates for version 3). In particular, our approach produced relatively high fault detection rates at 50% for version 1 and 75% for version 2. These results indicate that the use of requirements risks during prioritization was effective in locating faults that reveal risks early. Further, even when companies need to cut their testing process short due to their product release schedule, still can identify and fix more important faults under the limited time, budget than otherwise.

Figure 4.4 shows the results graphically. The horizontal axis shows the percentage of test execution, and the vertical axis represents the PTRSW values. This figure includes the results for a 100% test execution rate. As we observed from the Table 4.11, our approach outperforms the controls at all percentage levels for versions 1 and 2, but for version 3, our approach is not better than *Tccp* under the 50% test execution rate. Again, we speculate that using code modification information worked better for version 3 because version 3 underwent a major code modification that included a large number of requirements changes. From this observation, we think that combining our approach with code modification information for versions that go through major changes could improve the outcome.

Table 4.11. Percentage of Total Risk Severity Weight (PTRSW) for Different Test Execution Levels

Version	Method	Percentage of Total Risk Severity Weight (PTRSW) (%)			
		Execution Rate (12.50%)	Execution Rate (25%)	Execution Rate (50%)	Execution Rate (75%)
Version 1	Original Order	0.00	0.00	26.49	70.90
	Req-Cluster	7.46	7.46	30.97	71.00
	Req-Risk	21.64	30.22	74.63	82.00
Version 2	Original Order	0.83	3.94	46.68	57.26
	Req-Cluster	1.00	6.00	56.00	63.00
	Req-Risk	10.00	26.00	77.00	95.00
Version 3	Original Order	0.00	0.00	30.10	35.08
	Req-Cluster	36.00	36.00	66.00	88.00
	Req-Risk	0.00	13.00	68.00	91.00

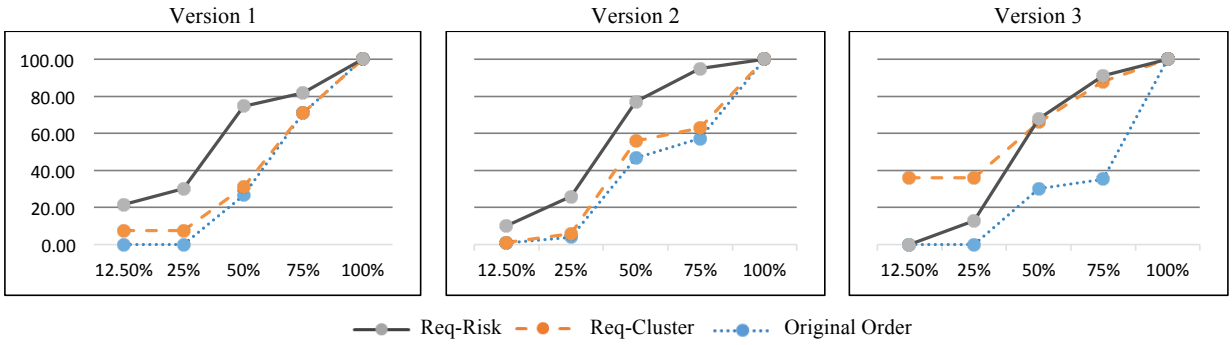

Figure 4.4. PTRSW Comparison Graphs for All Versions of iTrust

4.1.6. Discussion and Implications

From our results, we drew the following observations. First, our results suggest that the use of requirements and their risks can help improve the effectiveness of test case prioritization. The proposed technique performed better than techniques that used original order and source code information, but for the clustering techniques, the results were not consistent (in particular, for version 3). As we discussed in Section 4.1.5, we speculate that the use of prioritization factor and code change pattern in version 3 affected this outcome. The source code of version 3 was considerably affected by requirements changes. For this situation, the use of code modification information (e.g., code metric) would work better in identifying test cases that exercise more error-prone components. This observation indicates that we might need to consider different test case prioritization factors under different circumstances.

Second, our study results strongly support the conclusion that the use of requirements and their risks can help early detection of faults that reside in the risky components. The requirements associated with the critical risk items are assigned high priorities and those requirements are further prioritized depending on the degree they correlate with additional requirement-risks related factors (requirements modification and volatility). Hence, test cases associated with requirements having more risky behaviors get top priorities in the test suite, and this mechanism is the primary reason for the successful early detection of faults that reside in the risky components.

The results of the study provide important implications for software industry. Typically, companies that build safety or mission critical systems spend enormous time and effort in testing because the consequences of system failure often involve high costs and can even be life threatening. Also, the majority of modern software systems are web-based and deployed through the internet. This means that software systems are vulnerable to various security attacks, and thus they require thorough testing. For both cases,

finding critical faults early plays an important role in producing reliable systems more cost-effectively. Further, as we discussed in the previous section, the proposed technique was able to detect more critical faults early under the limited time slots. Thus, using our approach, companies can manage their production schedule more effectively by fixing costly faults as early as possible.

4.1.7. Threats to Validity

4.1.7.1. Construct Validity

The risk identification is subject to human judgment (in our case, graduate students). Therefore, the results can be biased by the personnel's knowledge and experience. Control for this threat can be achieved through applying our approach to the industrial environment with various stakeholders (i.e., software developers, testers, system users, etc.) who can perform such task.

4.1.7.2. Internal Validity

In this study, we assumed that there are no project or process risks associated with the requirements. We only considered the software product risk category when evaluating the RE of risk items. This assumption potentially influences the effectiveness of the technique but can be minimized by considering more product risk items.

4.1.7.3. External Validity

Our experiment used the *iTrust* system, a mid-size, open source application which has requirements documents. This limitation to generalization can be addressed by using different sizes of industrial applications with future studies. However, applying our approach to large industrial applications might not be ideal because large systems could consist of thousands of requirements and thus performing risk estimations with such a large number of requirements would be a very time consuming process. Under such circumstances, we can identify a set of subsystems that have potential risks and apply our approach to those subsystems.

4.1.8. Conclusions

In this research, we proposed a new test case prioritization method, which is based on requirements risks information, and assessed the approach by comparing other existing techniques. Our results indicated that the requirements risk-based prioritization can be effective in detecting faults early and even better in locating faults with the risky components early. With the proposed approach, software companies can manage their testing and release schedules better by providing early feedback to testers and developers so that they can fix the problems as soon as possible.

In this step, risk assessment required human experts' involvement. Human experts' involvement with the risk estimation process is important, but it makes the estimation process subjective and imprecise. In the next step of our research, we propose a systematic risk estimation approach using a fuzzy expert system to address these limitations.

4.2. Risk-based Test Case Prioritization Using a Fuzzy Expert System

Section 4.1 proposed a new requirements risk-based test case prioritization approach by considering the direct relationship between requirements risks and test cases. We also introduced a new evaluation method to measure how effective test case prioritization approaches were for detecting defects in risky components of software systems.

While our previous research was shown to be promising, the approach required human experts' involvement during the risk estimation process. Human involvement with the risk estimation process is important, but it makes the estimation process subjective and imprecise. To avoid the possible imprecision introduced by human judgment, a more systematic approach should be considered. Often, fuzzy expert systems have been utilized to address such problems because they can provide a mechanism to simulate the judgment and reasoning of experts in a particular field. To date, many researchers have used fuzzy expert systems in different application areas to help with complex decision-making problems, such as the diagnosis of disease [4] and risk estimation in aviation [26]. These studies have shown that fuzzy expert systems can be used to systematically represent human expertise in a particular domain and to deal with imprecision and subjectivity-related issues of the decision-making process while making the decision-making process more effective.

In this research, we propose a systematic risk estimation approach using a fuzzy expert system to address the limitations of our previous approach and this research was published in the journal of Information and Software Technology, 2016 [29]. We also reduced the number of risk items used for the risk estimation and simplified the prioritization approach so that we can perform test case prioritization with less effort. Further, from the results of our previous requirements risk-based approach, we learned that incorporating code information with the requirements could improve the rate of fault detection. Therefore, in this study, we used code information to extract requirements risks with respect to a few risk indicators in addition to the information obtained from requirements specifications written in natural language. Because we use code information, the proposed approach is applied during the testing phase after coding is done. To evaluate our approach, we used one open source application and one industrial application developed in Java.

The results of our study indicate that the systematic, risk-based test case prioritization approach has the capability to find faults earlier compared with other test case prioritization techniques, including our previous requirements risk-based approach described in Section 4.1. Moreover, the new approach is also better at finding more faults earlier in high-risk components than other techniques.

4.2.1. Fuzzy Expert Systems

In this research, we use a fuzzy expert system to derive requirements modification status (RMS) and potential security threats (PST) values. The fuzzy expert system used in this work simulates human expert's reasoning to derive the RMS and PST values for each requirement in a similar way that a human expert would estimate these values using the same input values. Existing empirical studies [27, 65] indicate that fuzzy expert systems can improve the effectiveness of decision making process in many different application areas including regression testing. Moreover, fuzzy expert systems can handle ambiguity, which in turn produce output values much closer to realistic values.

To provide a better understanding about the process of acquiring the RMS and PST values, we summarize the mechanism for a fuzzy expert system used with our approach. A fuzzy expert system is comprised of fuzzy membership functions and rules. It contains four main parts: fuzzification, inference, composition, and defuzzification. Figure 4.5 shows the typical architecture of a fuzzy expert system. The fuzzification process transforms the crisp input into a fuzzy input set. The inference process uses the fuzzy input set to determine the fuzzy output set using rules formulated in the knowledge base and the membership functions. The composition process aggregates all output fuzzy sets into a single fuzzy set. Finally, the defuzzification process calculates a crisp output using the fuzzy set produced by the composition process.

The knowledge base shown in Figure 4.5 contains the selected fuzzy rule set. In a fuzzy expert system, fuzzy rules play a vital role because they are formulated based upon the experts' knowledge about the domain of interest. The fuzzy rule's antecedent defines the fuzzy region of the input space, and the consequent defines the fuzzy region of the output space. Fuzzy rules can not only support multiple input variables, but also multiple output variables. The following equation shows an example of a fuzzy rule.

if x is A and y is B then z is C

where x and y are input variables, and z is the output variable. A is a membership function defined on variable x; B is a membership function defined on variable y, whereas C is a membership function defined on output variable z.

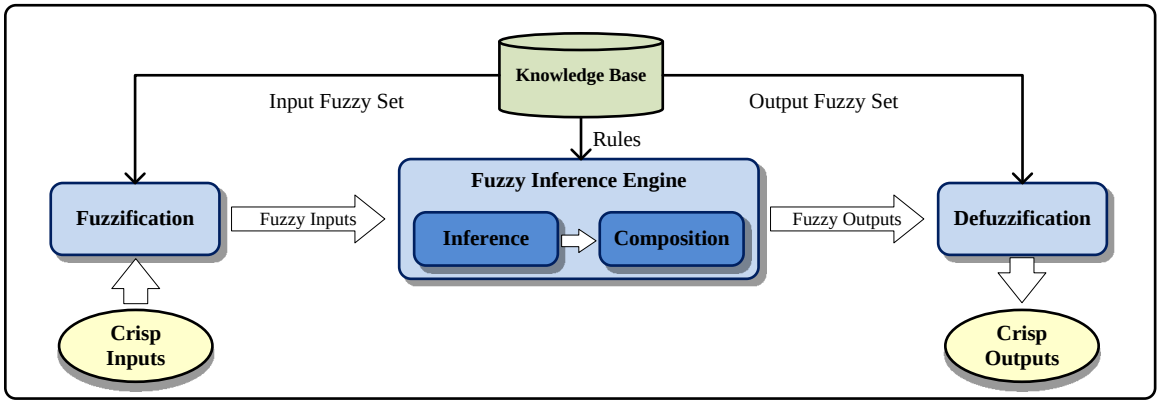

Figure 4.5. Architecture of a fuzzy expert system

4.2.1.1. Fuzzy Set Theory

To understand the fuzzification process, some knowledge of fuzzy set theory is necessary. Fuzzy set theory was first introduced by Zadeh [68] in 1965, and it defines fuzzy sets as an extension of conventional sets. In a conventional set, an element either belongs or does not belong to the set. Equation 4.6 defines a conventional set where $\mu_A(x)$ shows the membership of an element, x, of a conventional set, A.

$$\mu_A(x) = \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{if } x \notin A \end{cases} \tag{4.6}$$

Unlike conventional sets, in fuzzy sets, the membership of elements in the sets can be partially represented. Because fuzzy sets permit partial membership, the degree of membership is determined by using membership functions for unit interval [0, 1]. Hence, the imprecision of input data is handled by obtaining degree of membership values for each membership function defined in the fuzzy expert system. A fuzzy set is defined as shown in Equation 4.7.

$$A = (x, \mu_A(x))|x \in X, \mu(x) : X \rightarrow [0, 1] \tag{4.7}$$

where A is the fuzzy set, μ_A is the membership function, and X is the universe of discourse.

4.2.1.2. Fuzzification

In the fuzzification step, the input variables' values are used to determine the degree to which these values fit into each membership function used by the fuzzy rules. There are several types of membership functions available, such as triangular, trapezoidal, and gaussian. In this research, we utilize the triangular

membership function, which is widely used for fuzzy expert system based research and applications. Various empirical studies [53, 65] indicate that the triangular membership function is easy and simple to apply compared to other membership functions. As an initial investigation of the use of a fuzzy expert system, we chose the triangular membership function due its application simplicity. The triangular membership function is specified in Equation 4.8.

$$\mu_A(X) = \begin{cases} 0, & x < 0 \\ (x-a)/(b-a), & a \leq x \leq b \\ (c-x)/(c-b), & b \leq x \leq c \\ 0, & x > 0 \end{cases} \tag{4.8}$$

where A is the fuzzy set, μ_A is the membership function, X is the universe of discourse, a is the lower limit, b is the modal value, and c is the upper limit.

Table 4.12 shows the membership functions used for the input variables in our experiment.

Table 4.12. Input variable membership functions

Linguistic Value	Triangular Fuzzy Numbers (a,b,c)
Low	(0,0,5)
Medium	(0,5,10)
High	(5,10,10)

4.2.1.3. Inference

In the inference step, the fuzzified inputs (i.e., the degree of appropriate membership functions for input variable values) are applied on each rule antecedent to determine the degree of truth for each rule. The degree of truth is applied to the consequent of each rule, thus each output variable obtains an appropriate membership function. The output variable membership functions defined in our experiment are shown in Table 4.13.

Our fuzzy expert system is comprised of nine rules that represent experts' knowledge and experiences regarding the risks associated with software requirements. A sample set of rules used by the fuzzy inference process to estimate RMS is shown in Table 4.37. With the first rule (R1), the membership functions of two input variables, RML and PRV are low, and the membership function for the output variable, RMS is also low.

Table 4.13. Output variable membership functions

Linguistic Value	Triangular Fuzzy Numbers (a,b,c)
Low	(0,0,5)
Medium	(0,5,10)
High	(5,10,10)

Table 4.14. Fuzzy rules for RMS

R1. If RML is Low and PRV is Low then RMS is Low
R2. If RML is Medium and PRV is Low then RMS is Low
R4. If RML is Medium and PRV is Medium then RMS is Medium
R9. If RML is High and PRV is High then RMS is High

4.2.1.4. Composition

The composition step is to combine all membership functions obtained from each rule for each output variable and to form a single membership function for each output variable.

4.2.1.5. Defuzzification

The last step of fuzzy expert system is defuzzification which is an optional process. The defuzzification process produces crisp output for each output variable. In our experiment, we need exact, crisp output to quantitatively measure the RMS and PST for each requirement. Therefore, in this step, the resulting membership function of the previous step is defuzzified into a single number. There are several methods to perform the defuzzification. In our fuzzy expert system, we follow the Mamdani [42] type fuzzy inference process. Therefore, we use the center of gravity (COG) method which is considered as more accurate and widely used with the Mamdani-type fuzzy expert system. Equation 4.9 shows how to compute the COG that represents the crisp output for a particular output variable.

$$y^* = \frac{\int_a^b \mu_A(y) y dy}{\int_a^b \mu_A(y) dy} \tag{4.9}$$

where $y*$ is the crisp output; $\mu_A(y)$ is the aggregated membership function, A, on interval ab; and y is the output variable.

To illustrate the aforementioned processes, we provide a simple example. Suppose we estimate a requirements modification status for a requirement using two inputs: requirements modification level value of 10 and potential requirement volatility value of 8. The fuzzification process fuzzifies these values to produce the degree of membership for each input membership function. The fuzzy rules defined in the

expert system use the fuzzified input and the inference process produces the degree of memberships for the output variable (requirements modification status). All membership functions obtained from each rule are combined and the resulting membership function is defuzzified at the defuzzification process to obtain the final crisp value of 8.4 as the requirements modification status value for this requirement.

4.2.2. Approach

In this section, we describe the proposed approach. Our new method consists of four main steps.

1. Estimate risks by correlating with requirements
2. Calculate the risk exposure for the requirements
3. Calculate the risk exposure for risk items
4. Prioritize requirements and test cases

Figure 4.6 gives an overview of the proposed approach. The main steps are shown in light blue boxes, and the inputs and outputs for each step are shown in the ovals. The first three steps are used to calculate the requirements priorities. In the last step, test cases are prioritized using the results produced by the first three steps. The following subsections describe each step in detail by using an example that we excerpted from our experimental data which are fully described in Sections 4.2.3 and 4.2.6.

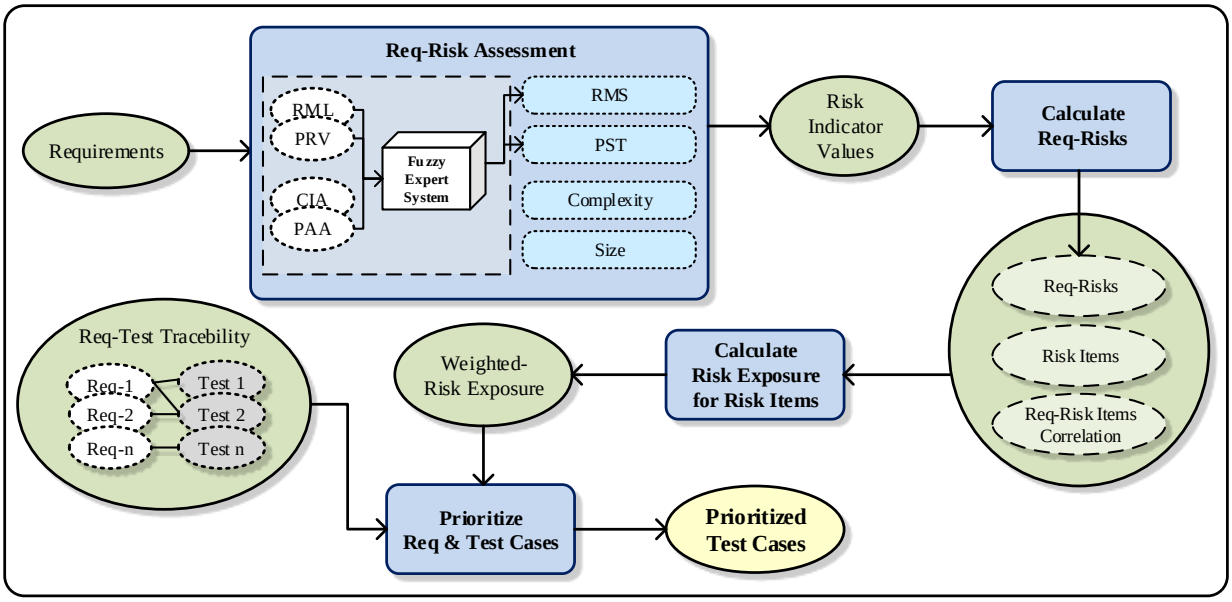

Figure 4.6. Overview of the risk-based approach

4.2.2.1. Estimate Risks by Correlating with the Requirements

In order to perform risk assessment for the requirements, we identify four risk indicators that have been used by previous requirements and risk-based regression testing research [7, 28, 36, 59]: requirements complexity (RC), requirements size (RS), requirements modification status (RMS), and potential security threats (PST). These previous studies indicate that these risk indicators can be effective in finding defects in software systems, thus we focus on these four risk indicators in this study, but we consider to use other risk indicators, such as usage rate [22], as we evaluate our approach in the future. While obtaining the first two risk indicators is straightforward, the last two risk indicators can be subjective, so we utilize a fuzzy expert system to reduce the subjectivity and possible errors made by human judgment. To calculate the first two risk indicators (RC and RS), we used both source code and requirements information, and for others (RMS and PST), we used requirements information. We explain each of these indicators in detail.

4.2.2.1.1. Requirements Complexity (RC)

Requirements that need complex functionalities during implementation tend to introduce more faults. A case study conducted by Amland [7] showed that requirements that need complex functionalities at the coding phase tend to introduce a higher number of defects. In addition, the study indicated that functions with a higher number of faults have a higher McCabe complexity. Therefore, in this research, we used McCabe complexity to measure the requirements complexity (RC). We measured the McCabe complexity values of software functionalities (source code) using Eclipse IDE. The complexity of requirements was determined by examining the relationships between requirements and functionalities. For the specific applications that we used with our experiment, the complexity values ranged from 1 to 12, and we normalized these values into a range from 0 to 10. A value of 0 indicated the lowest complexity, whereas 10 indicated the highest complexity for the requirements. The eighth column of Table 4.36 shows the example for the complexity values used for our experiment.

4.2.2.1.2. Requirements Size (RS)

To measure the requirements size (RS), the size of functions associated with the requirements is used, and it is measured through lines of code (LOCs). Functions with higher LOC numbers tend to contain more defects. Amland's case study [7] shows that the size of the functions could affect the number of faults in a system. In the experiment we performed, requirements size values range from 16 to 508, and these values are normalized into a range from 0 to 10, where a value of 0 indicates the lowest size and 10 indicates the highest size for the requirements. The last column of Table 4.36 shows the example for the RS values.

4.2.2.1.3. Requirements Modification Status (RMS)

To estimate the RMS, two aspects of requirements modifications are considered: requirements modification level (RML) and potential requirements volatility (PRV). These two variables are the inputs for the fuzzy expert system that we develop for this research, and the fuzzy expert system produces RMS using these two input variables. RMS reflects the overall modification status of each requirement. RML represents the degree of a requirement's modification by comparing the same requirement in the previous version. However, manual comparison of requirements can be a subjective and time-consuming process. Therefore, to reduce the amount of time and subjectivity, we developed a program that uses cosine similarity [37] to measure similarities between requirements. The program compares two given requirements and produces the requirements modification level values (RML). This semi-automated approach helps eliminate some mistakes that may occur in the manual requirements-comparison process. Here, requirements modification includes existing requirements change and new requirements addition. The RML values are normalized into a range from 0 to 10, where 10 indicates the highest requirements modification level and 0 indicates no modification. New requirements for a subsequent version are automatically assigned the value of 10 because functionalities associated with the new requirements have a high possibility of introducing new faults to the system. After finishing this process, we examine the final RML values to ensure that the RML values obtained from the automated process reflect the actual modification levels for the requirements.

The requirement's PRV values are used to quantify the possibility of having requirements changes in later versions of the system. PRV values are measured through experts' knowledge and experiences with requirements engineering. Several requirements characteristics, such as functional instability, possible interface changes, and other factors that may influence the requirements modifications, are taken into account to estimate the PRV values. For example, consider a healthcare application. For such an application, requirements that calculate an insurance premium would change whenever the insurance policy changes. Thus, these requirements are assigned a higher PRV value compared to the requirements that handle patients' information. The second and the third columns in Table 4.36 show the example for the RML and PRV values used in our experiment, and the fourth column shows the RMS values provided by the fuzzy expert system which is explained in Section 4.2.1.

4.2.2.1.4. Potential Security Threats (PST)

Today, software security is a major concern for software applications (in particular, for web applications) due to the rapid growth of malicious activities, such as SQL injection, eavesdropping, etc., against software applications. Security flaws for a software application lead to severe consequences unless the security-related issues are identified and properly handled as early as possible. Therefore, in this approach, potential security threats (PST) is used as an indicator of security-related risks that reside in the requirements. Again, the fuzzy expert system is employed with a different rule set to estimate the PST values. The two input variables contain sets of software security objectives that are identified in the software security field [1, 48]. The first input variable contains the major security objectives, such as confidentiality, integrity and availability (CIA), and the second variable consists of secondary security objectives, such as privacy, authentication and accountability (PAA).

To estimate these input variable values for each requirement, we developed a term-extraction tool to find the number of security key words in a particular requirement that were related to the input variables. For example, suppose we identified 50 security key words associated with the first input variable (i.e., CIA) and a particular requirement contains 5 of the 50 keywords, then the CIA variable value of the requirement is 0.1 (5/50). Then, the input variable values obtained from the tool are normalized into a range from 0 to 10. A value of 0 indicates no association with CIA, whereas 10 indicates the highest association. The same technique is used to obtain the PAA values for each requirement. The fifth and sixth columns of Table 4.36 show the CIA and PAA values, respectively, and the seventh column shows the PST values provided by the fuzzy expert system.

Table 4.15. Risk indicators and fuzzy input-output values

	Fuzzy input		Fuzzy output	Fuzzy input		Fuzzy output		
Requirement	RML	PRV	RMS	CIA	PAA	PST	Complexity	Size
UC1S1	0	2	4.6	7	6	8.1	1.9	5.8
UC2S1	0	3	4.6	7	9	8.3	1.9	4.3
UC8S1	5	6	5.0	5	7	6.0	1.9	2.8
UC10S1	6	5	5.0	6	8	8.3	1.0	1.6
UC11S1	2	5	5.0	5	7	4.5	2.8	9.4
UC23S3	3	5	5.0	6	6	5.7	1.9	10.0
UC26S3	0	3	4.6	4	5	3.2	2.8	4.9
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
UC34S6	10	8	8.4	5	6	5.2	1.0	1.7

4.2.2.2. Calculate the Risk Exposure for the Requirements

Risk assessment for software requirements is performed by considering the probability of the risk occurrence (risk likelihood) for the risk indicators and the degree of possible damage (risk impact) with each indicator. We consider the four risk indicators explained in the previous step, and the weight for each indicator is determined using the analytic hierarchy process (AHP) that supports pairwise comparisons [51]. The comparison's result values are normalized into a range from 1 to 5, and we obtain the final weight values for each indicator. In Table 4.16, columns two to five show the comparison values of risk indicators; sixth column shows the total; and the last column shows the priority vector (PV) values calculated for each indicator. Then, the PV values obtained from each expert are averaged. (In this example, two experts perform pairwise comparison.) The first column of Table 4.17 shows the risk indicators; the second and third columns show the PV values obtained from each expert; and the fourth column shows the averaged PV values. The last column shows the normalized weight values for the risk indicators.

Table 4.16. Risk indicator comparison

	RMS	Complexity	PST	Size	Total	Priority Vector
First Expert's Comparison						
RMS	0.37	0.37	0.38	0.35	0.37	0.37
Complexity	0.37	0.37	0.38	0.30	0.37	0.36
PST	0.18	0.18	0.19	0.30	0.18	0.21
Size	0.05	0.06	0.03	0.05	0.05	0.04
Second Expert's Comparison						
RMS	0.38	0.32	0.48	0.38	1.56	0.39
Complexity	0.38	0.32	0.24	0.29	1.22	0.31
PST	0.19	0.32	0.24	0.29	1.03	0.26
Size	0.05	0.05	0.04	0.05	0.19	0.05

Table 4.17. Risk indicators and weights

Indicator	PV1	PV2	Average	Weight
Requirements Modification Status	0.37	0.39	0.38	5
Requirements Complexity	0.36	0.31	0.34	4
Potential Security Threats	0.21	0.26	0.24	3
Size	0.04	0.05	0.05	1

Equation 4.10 is used to calculate the risk exposure of a requirement (RE(Req)) by following a common risk-assessment practice used by several researchers (i.e., multiplication of risk likelihood and risk

impact). Each risk indicator value corresponds to the risk likelihood of a requirement, and the weight of the indicator corresponds to the risk impact.

$$RE_{(Req_j)} = \sum_{i=1}^{n} (W_i * R_{ji}) \tag{4.10}$$

where n is the number of indicators; W_i is the weight of the indicator; and RI_{ji} is the risk value of the requirement, Req_j, in terms of indicator i.

The following example shows the risk calculation for the UC1S1 requirement used in our experiment ("The healthcare personnel create a patient as a new user of the system."). The last column of Table 4.18 shows the risks for a sample set of requirements.

$RE_{(UC1S1)}$ = (5*4.6) + (4*1.9) + (3*8.1)+ (1*5.8)= 60.8

The risk values of the requirements RE(Req) range from 0 to 130, where 0 indicates the lowest risk and 130 indicates the highest risk. In this example, the risk of UC1S1 is 60.8, implying an average risk level.

Table 4.18. Risk indicator values and the risk exposure for requirements

Requirement	RMS	Complexity	PST	Size	RE(Req)
	5	4	3	1	
UC1S1	4.6	1.9	8.1	5.8	60.8
UC2S1	4.6	1.9	8.3	4.3	59.8
UC8S1	5.0	1.9	6.0	2.8	53.5
UC10S1	5.0	1.0	8.3	1.6	55.6
UC11S1	5.0	2.8	4.5	9.4	59.2
UC23S3	5.0	1.9	5.7	10.0	59.8
UC26S3	4.6	2.8	3.2	4.9	48.9
⋮	⋮	⋮	⋮	⋮	⋮
UC34S6	8.4	1.0	5.2	1.7	63.1

4.2.2.3. Calculate the Risk Exposure of Risk Items

In the previous subsection, the process for requirements risk exposure estimation was explained. These requirements risk exposure (RE(Req)) values indicate how risky each requirement is from the system requirements' point of view. Although, the (RE(Req)) value of a requirement reflects the risk residing in the requirement, it is not sufficient to obtain information about the association between the requirement and potential defect types of a software system. When a particular requirement is associated with multiple defect types, that requirement has a high tendency to expose more defects. Thus, the association information

between requirements and potential defect types help identify the requirements with a higher defect density which eventually cause common software failures. Therefore, high priorities can be assigned to the test cases that are correlated to such requirements with a higher defect density.

To prioritize the test case using the association information between requirements and potential defect types, we adopted a process defined by Yoon et al. [67]. We improved this process by introducing weights for the risk items (RiIM). The weight values for the risk items were obtained from our previous requirements risk-based research [28], and for this research, we further calibrated the weight values by using the risk exposure values for risk items in Yoon et al.'s approach [67]. The risk items denoted potential defect types for a software system. The process of calculating risk exposure for risk items involved the following activities: (1) Identify Risk Items: The risk items employed in this research were derived from studies that used different applications and standards [9, 30, 63]. The identified risk items are shown in Table 4.19. The input problem was an example for risk items because input data can cause several problems for an operating software system. For instance, input data that were not validated before the execution or input data that were beyond the valid boundary resulted in operational failures for the system or a system crash. (2) Calculate risk exposure values for the risk items (RE(RiIM)): The risk level for a particular risk item was quantified by using the risk exposure values.

To find the risk exposure values for risk items, first, we need to find the probability of fault occurrence by considering the association between risk items and requirements. When we consider a set of requirements, some of them may be associated with a particular risk item. The probability of fault occurrence depends on the number of associated requirements. When a particular risk item is associated with multiple requirements, its probability for fault occurrence will increase. Second, we need to estimate the impact of the risk items. For this purpose, we consider the risk exposure of requirements (RE(Req)). When a particular risk item is associated with requirements that have higher risk exposure, the risk impact of that particular item increases.

In this research, we ignored the Startup/ShutDown risk item which was used for our previous risk-based approach [28] because Startup/ShutDown risk item only associated with a small number of requirements and also did not make a significant impact on the test case prioritization. Additionally, in Yoon et al.'s [67] approach, the Startup/ShutDown risk item indicated a relatively low exposure value. Eliminating one risk item caused an approximate 17% work reduction for this risk items' risk exposure calculation subprocess.

Table 4.19. Software product-risk items

	Risk Item	Abbreviation
i	Input Problem	IP
ii	Output Problem	OP
iii	Calculation	Calc
iv	Interactions	Inac
v	Error Handling	ErHa

Table 4.20 shows a matrix to calculate the risk exposure (RE(RiIM)) values for risk items and the weighted risk exposure (W-RE) values for requirements. The matrix lists requirements, risk exposure for requirements, and a set of risk items. Each risk item ($RiIM_x$) has a severity value (SV_x) that indicates how risky the item is. The severity values are defined in Table 4.21. If a risk item is associated with a certain requirement, the C_{mx} value is 1; otherwise, the value is 0. Again, multiple associations between risk times and requirements can exist. The last row in the matrix shows the RE values for risk items, and the RE values are calculated using Equation 4.11; the last column shows the W-RE values that are calculated by incorporating RE(RiIM) values with the severity values of risk items for each requirement. The final outcome of this step, the W-RE values, is used to prioritize requirements.

Table 4.20. Risk exposures and weighted risk exposure matrix

Requirement	RE(Req)	Risk Items				W-RE
		RiIM1 (SV1)	... ...	RiIMx (SVx)	RiIMn (SVn)	
Req_1	RE(Req_1)	C_{11}		C_{1x}	C_{1n}	W-RE(Req_1)
⋮	⋮	⋮	...	⋮	⋮	⋮
Req_y	RE(Req_y)	C_{y1}	...	C_{yx}	C_{yn}	W-RE(Req_y)
Req_m	RE(Req_m)	C_{m1}	...	C_{mx}	C_{mn}	W-RE(Req_m)
		RE(RiIM1)	...	RE(RiIMx)	RE(RiIMn)	

Table 4.21. Severity of risk items

Severity Value	Description
1	Least critical risk item
2	Slightly critical risk item
3	Moderately critical risk item
4	Very critical risk item
5	Most critical risk item

Equations 4.11 and 4.12 are used to calculate the risk exposure values for risk items and the weighted risk exposure values for requirements.

$$RE_{(RiIM_x)} = \sum_{y=1}^{m} (RE_{(Req_y)} * C_{yx}) \tag{4.11}$$

where m is the number of requirements, $RE_{(Req_y)}$ is the risk exposure of Req_y, and C_{yx} is 1 when requirement Req_y is associated with risk item $RiIM_x$ or 0 otherwise.

$$W - RE_{(Req_y)} = \sum_{x=1}^{n} (RE_{(RiIMx)} * C_{yx} * SV_x) \tag{4.12}$$

where n is the number of risk items for the system; $RE_{(RiIMx)}$ is the risk exposure value of the risk item, $RiIM_x$; C_{yx} indicates the correlation between the requirement, Req_y, and the risk item, $RiIM_x$; and SV_x is the severity value of the risk item, $RiIM_x$.

Table 4.22. Example of risk exposures and weighted risk exposures of iTrust

Requirement	TC ID	RE(Req)	OP	ErHa	Inac	IP	Calc	W-RE
Risk Items Severity Values			4	5	5	2	3	
UC1S1	TC2	60.8	1	1	0	1	0	61384.6
UC2S1	TC5	59.8	1	1	1	1	0	88440.5
UC2S1	TC6	57.2	1	1	1	1	0	88440.5
UC8S1	TC13	53.5	1	1	0	1	1	68855.7
UC10S1	TC19	55.6	1	1	1	1	1	95911.6
UC11S1	TC21	59.2	1	1	1	0	0	79497.2
UC23S3	TC31	59.8	1	1	1	1	1	95911.6
UC26S3	TC33	48.9	1	1	1	0	0	79497.2
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
UC34S6	TC141	63.1	1	1	1	1	0	88440.5
Risk Exposure (RE(RiIM))			6146.0	5571.5	5411.2	4472.0	2490.0	

Table 4.22 shows a sample data set collected from our experiment. In this example, we can see that the output problem risk item is associated with all requirements. Thus, the risk exposure value of the output problem risk item can be calculated using Equation 4.11 as follows:

$RE_{(OP)}$ = (60.8*1) + (59.8*1) + (53.5*1) + (55.6*1) + . . . + (63.1*1) = 6146.0

Because the output problem has a high RE value compared to other risk items, it implies that the output problem is a high risk area for this product.

After calculating the RE values for risk items, we calculate the W-RE values for each requirement by utilizing Equation 4.12. For instance, we obtain the W-RE value for UC1S1 as follows.

W-RE$_{(UC1S1)}$ = (6146.0*1*4) + (5571.5*1*5) + (5411.2*0*5) + (4472.0*1*2) + (2490.0*0*3) = 61384.6

After calculating the W-RE values for each requirement, we prioritize requirements by their W-RE values (in descending order). Table 4.23 shows a portion of our data. From the table, we can see that each requirement has one or more corresponding test cases (The last subsection explains how to map these two.) and that the requirements appear by their W-RE values (in descending order). When multiple requirements have the same W-RE value (e.g., UC33S1 and UC26S2 have the same value, 7250.3, in the table.), we used the RE(Req) values as the second factor to prioritize the requirements. The next section describes, in detail, the prioritization of the test cases.

Table 4.23. Example for a prioritized test suite: iTrust

Requirement	TC-ID	W-RE	Re(Req)
UC33S1	TC117	7250.3	76.5
UC26S2	TC94	7250.3	70.3
UC21	TC30	7090.6	80.2
UC21	TC80	7090.6	70.1
⋮	⋮	⋮	⋮
UC12	TC26	4735.4	80.0
UC30S3	TC100	4236.6	67.2
UC30S1	TC99	4236.6	56.3
UC3S3	TC43	2947.6	33.3
UC24	TC91	2860.2	42.5
UC25	TC92	2860.2	20.8
⋮	⋮	⋮	⋮
UC2S2	TC77	1787.7	66.5

4.2.2.4. Prioritize the Requirements and Test Cases

In this final stage, we prioritize requirements using the W-RE values and then by the the risk exposure of requirements (RE(Req)). To prioritize test cases, we need to know the relationship between the test cases and the requirements. In this research, we use the traceability matrices created by the object programs' (i.e., *iTrust* and *Capstone*) developers. Unavailability of traceability matrices could require more effort to adopt our proposed approach. However, often, the documents that are created in the early software development stages, such as business requirements document and functional specification document, are used when

creating test cases, which can help construct traceability information between requirements and test cases. Further, some industrial tools such as Qmetry [2] and Microsoft Testing Manager [3] can aid establishing traceability.

For most software applications, including our experimental programs, a single requirement is tested with multiple test cases. As a result, test cases associated with the same requirement have the same W-RE value. However, the different functions used to manipulate a particular requirement have different complexity levels and LOCs, hence producing different RE(Req) values. Therefore, after mapping test cases to their corresponding requirements and functions, the test cases with similar W-RE values can be re-prioritized using the RE(Req) values. If test cases still have the same W-RE and RE(Req) values, then we randomly order those test cases.

4.2.3. Empirical Study

To investigate the effectiveness of our new risk-based test case prioritization approach, we designed and performed a controlled experiment. The following subsections describe research questions, the objects of analysis, independent variables, dependent variables and measures, experimental setup and procedure, and threats to validity.

In this study, we investigate the following research questions:

RQ1: Can systematic risk-based test case prioritization improve the rate of fault detection for test suites?

RQ2: Can systematic risk-based test case prioritization find more faults in the risky components early?

4.2.3.1. Objects of Analysis

In order to evaluate the new approach, we utilized two applications: one open source and one industrial application developed for graduate-student project. *iTrust* is the open source application used for this experiment. The *iTrust* program is a patient-centric electronic health record system which was developed by the Realsearch Research Group at North Carolina State University. In this experiment, four versions of the *iTrust* application (versions 0, 1, 2, and 3) were used. We considered the functional requirements of each version, and the test cases were used to test the system functionalities associated with the system requirements. For *iTrust*, all the test cases used in the experiment were developed by the *iTrust* system developers. The industrial application, *Capstone*, was developed by computer science graduate students at North Dakota State University in collaboration with a local software company. *Capstone* is an online application that is used to automate the company's examination procedure.

The information about the system components for all versions of the two applications used in this experiment is shown in Table 4.24. For each system, version 0 (base version) is not listed in the table because regression testing starts with the second version. However, the information obtained from version 0 is utilized to obtain the mutants for version 1.

Table 4.24. Experiment objects and associated data

Object	Version	Requirements	Test Cases	Size (KLOCs)	Mutation Faults	Mutation Groups
iTrust	V1	91	122	24.42	54	13
	V2	105	142	25.93	71	12
	V3	108	157	26.70	75	12
Capstone	V1	21	42	6.82	118	23

4.2.4. Variables and Measures

4.2.4.1. Independent Variable

The test case prioritization technique is the independent variable manipulated in this study. We consider seven control techniques, including a requirements risk-based technique and one heuristic prioritization technique, as follows:

- Control Techniques
  - Original (*Torig*): The object program provides the testing scripts. *Torig* executes test cases in the order in which they are available in the original testing script.
  - Total statement coverage (*Tsc*): This technique prioritizes test cases based on the total number of statements exercised by test cases.
  - Code metric (*Tcm*): This technique uses a code metric that we defined in our previous study [8]. The code metric is calculated using three types of information obtained from the source code, Line of Code (LOC), Nested Block Depth (NBD), and McCabe Cyclomatic Complexity (MCC), which are considered good predictors for finding error-prone modules [52, 69].[2]
  - Requirements-based clustering: We consider another set of techniques proposed by Arafeen and Do [8] as control techniques that follow the requirements-based clustering approach. Because all requirements are not equally important to clients, requirements clustering is important to prioritize the requirements based on their importance to the client so that the tester can pay more attention to the test cases associated with high-priority requirements. With the previous ap-

[2] $Tcm = \frac{NBD}{Max(NBD)} + \frac{MCC}{Max(MCC)} + \frac{LOC}{Max(LOC)}$

proach, test cases are clustered based on requirements and test case association. Within clusters, test cases are prioritized using code metric information or use the original test case order. Thus, all techniques are categorized into two broad categories based on the original test case order and the code metric test case order. In this research, we only considered the code metric based category which produced relatively better results compared to original order category. Then, the clusters are ordered in three ways: original cluster order, random cluster order, and prioritized cluster order. Clusters are prioritized using both requirements commitment (i.e., the priority of requirements to be implemented by the developers) and code modification information. After ordering the test cases and the clusters, test cases are selected from the clusters in a round-robin fashion for the prioritization. The three requirements-based clustering techniques are as follows:

  * Tcl-cm-orig (*Tcco*): It uses the code metric based test case order within clusters, and the clusters are ordered according to the original order of the clusters.
  * Tcl-cm-rand (*Tccr*): This technique uses the code metric based test case order within clusters, and the clusters are ordered randomly.
  * Tcl-cm-prior (*Tccp*): This technique uses the code metric based test case order within clusters, and the clusters are ordered according to requirements commitment and code modification information.

  This requirements cluster-based approach [8] used five different cluster sizes for the *iTrust* application. In this study, we only considered cluster sizes 10 and 20 which produced moderate and the best results, respectively. In the case of *Capstone*, the only cluster size used in requirements cluster-based approach [8] was cluster size 6.

- Requirements risk-based (*Trrb*): This technique prioritizes test cases based on the risks residing in the requirements and the association between a system's requirements and potential defect types.

- Heuristic (*Tfrrb*): The proposed technique uses a fuzzy expert system to estimate requirements risks and prioritizes the test cases as described in Section 4.2.2.

4.2.4.2. Dependent Variables

In this experiment, we considered the two dependent variables described in Section 4.1.3.2, Average Percentage of Fault Detection (APFD) and Percentage of Total Risk Severity Weight (PTRSW).

4.2.5. Experimental Setup and Procedure

In this study, for each requirement, we estimated the risk exposure using a fuzzy expert system and then estimated the weighted risk exposure (W-RE) values for the requirements as described in Section 4.2.2. We prioritized the requirements based on the W-RE values and then by the risk exposure of requirements (RE(Req)) values of the requirements. To obtain the prioritized test cases, we used the requirements-test mapping information which was provided by the object programs' developers.

To empirically evaluate the proposed approach, we also need fault data. Due to the absence of the faults with the applications, we used a set of mutation faults created for each object program during previous research [8]. The second-last column in Table 4.24 lists the total number of mutation faults for each program version. In actual testing scenarios, however, programs do not typically contain this many faults. Thus, to reflect more realistic situations, our previous study [17] introduced *mutant groups* which were formed by randomly selecting mutants from the pools of mutants created for each version, ranging from 1 to 10 mutants per group. For the *iTrust* application, 13, 12, and 12 mutation groups were created for version 1, 2, and 3, respectively, whereas 23 mutation groups were created for *Capstone* version 1.

To perform the risk estimation process, we followed the steps described in Section 4.2.2. As a human expert, one graduate student who has several years of software industry experience performed the risk estimation process. To obtain the requirements modification level (RML); confidentiality, integrity, and availability (CIA) values; and privacy, authentication, and accountability (PAA) values, the student followed the process explained in Section 4.2.2. When the final RML, CIA, and PAA values were obtained, those values were reviewed by the student to validate the inaccuracy. On average, only 3% of the requirements needed slight adjustments for their RML values, and less than 2% of the requirements required minor adjustments for the CIA and PAA values. The potential requirements volatility (PRV) values for the requirements were estimated based on the expert's knowledge and experience. The weight values used for the risk indicators were determined through the priority vector (PV) values of analytic hierarchy process (AHP) that was performed by two experts (graduate students). We averaged and normalized the priority vector values obtained from each expert for every indicator to obtain the final weight values for risk indicators. Test cases were prioritized using W-RE and RE(Req), and with test cases that had the same W-RE and RE(Req) values, those test cases were randomly prioritized. For all the applications and versions used for this experiment, on average, only 3% of the test cases required random orders.

To calculate the PTRSW values for each test case prioritization technique, we need know the classes and methods modified by the mutation faults. Therefore, we used a mutation analysis tool, *ByteME* (Bytecode Mutation Engine, a tool from the software-artifact infrastructure repository (SIR) [18]), to locate these altered classes/methods and developed the faults-classes/methods trace files for each version of the object programs. Further, we used the requirements-classes/methods trace files to identify the association relationship between requirements and classes/methods. We estimated the risks of classes/methods caused by mutation faults, risk severity weight (RSW), using the information provided by *ByteME* and the requirements risks. The RSW values estimated for classes/methods were assigned to the associated mutation faults. The RSW values ranged from 0 to 10. The RSW value was 0 if there was no risk caused by a mutation fault for a class/method, whereas 10 indicated the highest risk caused by a mutation fault on a critical system component.

The faults-tests traceability matrix is used to identify the relationship between mutation faults and test cases. One mutation fault can be detected by one or more test cases. In that case, the same RSW value is assigned to all test cases associated with the same mutation fault. On the other hand, one test case can detect several mutation faults, and all RSW values for the detected mutation faults are added together to obtain the total risk severity weight (TRSW) for that test case. The sum of the TRSW values of all test cases produces the grand total risk severity weight (GTRSW) for the test suite.

When developing software applications, software companies often face time and budget constraints, and typically, the companies cut back on testing activities to ensure a timely release for their product. When testing activities are cut short, faults will slip through testing. If testing techniques can detect riskier defects earlier, then the companies could reduce potential severe consequences. To examine this situation, we consider four different test execution rates as shown in Table 4.25. For example, the STRSW values for the original test order of *iTrust* version 2 are 4 and 19 for test execution rates of 12.5% and 25%, respectively. Table 4.25 shows sample RSW, TRSW, STRSW, GTRSW, and PTRSW values for the original test case order of *iTrust* version 2. The first column shows the test cases; columns 2, 3, and 4 show the sample set of RSW values for the test cases; column 5 shows TRSW and STRSW; and the last column shows the PTRSW values for different test execution rates. For example, the STRSW for the 50% test execution rate is 225, and the PTRSW is 46.68%. When we compare these data with the results of other techniques shown in Table 4.29, these values are relatively low, meaning that the original order of test cases is unable to detect faults in the risky components effectively.

After gathering all required data, we obtained the prioritized test suite by following the proposed approach. We calculated the APFD and PTRSW values for the prioritized test suite.

Table 4.25. Example test cases, PTRSW, and associated data for iTrust original test order-version 2

Test cases (Orig v2)	RSW of Mutations			TRSW	Percentage of Total Risk Severity Weight (PTRSW)
	M1	...	M71		
TC7	3	0	1	4	
⋮	⋮	⋮	⋮	⋮	
TC18	0	0	0	0	
⋮	⋮	⋮	⋮	⋮	
Sub-total of TRSW (STRSW) after 12.5% test case execution				4	0.83%
⋮	⋮	⋮	⋮	⋮	
TC20	0	1	8	10	
Sub-total of TRSW (STRSW) after 25% test case execution				19	3.94%
⋮	⋮	⋮	⋮	⋮	
TC53	0	2	2	5	
Sub-total of TRSW (STRSW) after 50% test case execution				225	46.68%
⋮	⋮	⋮	⋮	⋮	
TC106	0	9	8	42	
Sub-total of TRSW (STRSW) after 75% test case execution				276	57.26%
⋮	⋮	⋮	⋮	⋮	
TC136	0	8	8	24	
⋮	⋮	⋮	⋮	⋮	
Grand total after execution the entire test suite (GTRSW)				482	

4.2.6. Data and Analysis

In this section, we present the results of our study and the data analyses for each research question. We discuss further implications of the data and results in Section 4.1.6.

4.2.6.1. The effectiveness of risk-based prioritization with a fuzzy expert system for improving the fault detection rate of test cases (RQ1)

In our first research question (RQ1), we consider whether the risk-based approach that incorporates a fuzzy expert system can help improve the effectiveness of test case prioritization. To answer this question, we compared techniques based on the results shown in Tables 4.26, 4.27, and 4.28.

Table 4.26 shows the APFD values for our heuristic technique (*Tfrrb*). Columns 2 to 4 show the APFD values for *iTrust* versions 1, 2, and 3, respectively, and the last column shows the APFD value for the *Capstone* application. The first column of Table 4.27 shows the control techniques. CS10 and CS20 indicate the two cluster sizes. All APFD values shown in Tables 4.27 and 4.28 are averaged values for the

13, 12, and 12 data points of *iTrust* versions 1, 2, and 3, respectively, and the 23 data points for *Capstone* version 1. Table 4.27 and 4.28 also show the improvement rates for our proposed approach over the control techniques.

Table 4.26. Heuristic APFD: iTrust and Capstone

	iTrust			Capstone
Version	V1	V2	V3	V1
APFD (Tfrrb)	63	65.78	79.44	67.86

The results in Table 4.27 indicate that our approach (*Tfrrb*) outperformed the original (*Torig*), statement coverage (*Tsc*), code metric (*Tcm*), and our previous requirements risk-based approach (*Trrb*) across all versions. The improvement rates ranged from 5% to 175.83%. However, when we compared our approach to the cluster-based approaches, the trend varied across versions. For version 1, the heuristic outperformed the control techniques with a cluster size of 10 (The improvement rates ranged from 59.45% to 63.81%.), but it was not better than the techniques with a cluster size of 20. In the case of version 2, the results were reversed. The heuristic performed slightly better than the control techniques with a cluster size of 20, and it performed slightly worse than the control with a cluster size of 10. In the case of version 3, the heuristic outperformed all cluster-based control techniques.

Table 4.27. APFD comparison and improvement over controls: iTrust

Control Technique	iTrust - V1		iTrust - V2		iTrust - V3	
	APFD	Improvement Over Control (%)	APFD	Improvement Over Control (%)	APFD	Improvement Over Control (%)
Torig	43.70	44.16	47.80	37.62	28.80	175.83
Tsc	53.02	18.82	56.75	15.91	69.26	14.70
Tcm	45.80	37.55	48.80	34.81	44.84	77.18
Trrb	60.00	5.00	60.60	8.55	56.00	41.86
Tcco -CS10	38.46	63.81	68.26	-3.63	68.48	16.00
Tccr -CS10	38.88	62.04	68.30	-3.69	64.54	23.09
Tccp -CS10	39.51	59.45	57.78	13.85	75.31	5.48
Tcco -CS20	65.58	-3.93	62.67	4.96	66.69	19.12
Tccr-CS20	66.22	-4.86	63.65	3.35	67.68	17.38
Tccp-CS20	75.45	-16.50	65.33	0.69	72.41	9.71

The results for *Capstone* shown in Table 4.28 indicate that the proposed approach outperformed all control techniques except for the *Tccp* technique. The heuristic and *Tccp* produce the same results. For this application, the improvement rates range from 0.00% to 55.26%.

Table 4.28. APFD comparison and improvement over controls: Capstone

Control Technique	Capstone - V1	
	APFD	Improvement Over Control (%)
Torig	43.70	55.26
Tsc	55.19	22.96
Tcm	67.80	0.07
Trrb	62.88	7.90
Tcco	61.19	10.89
Tccr	61.47	10.37
Tccp	67.86	0.00

To show our results visually, we present them in boxplots. Figure 4.7 presents the boxplots that show APFD values for the control techniques and heuristic for all *iTrust* and *Capstone* versions. For *iTrust* version 1, each boxplot has 13 data points, and for versions 2 and 3, each has 12 data points. In the case of *Capstone*, each boxplot has 23 data points. Each subfigure for *iTrust* contains boxplots for ten prioritization techniques; the first nine boxplots present data for the control techniques, and the last one presents the heuristic technique. The last subfigure, *Capstone*, contains boxplots for seven prioritization techniques; the first seven boxplots present data for the control techniques, and the last one presents the heuristic technique.

When we examine the boxplots for *iTrust*, in version 1, the results for requirements risk-based approaches (*Trrb* and *Tfrrb*) show a wider distribution of data points than the other two versions. The results with other techniques for all versions show similar data-distribution patterns except for a couple cases. For version 3, the control techniques that used clustering with a cluster size of 10 show a wider distribution than other techniques. Overall, the heuristic shows better results compared to the controls across all versions of *iTrust* except for a few cases (clustering-based approaches for version 2). In the case of *Capstone*, all techniques show a similar data-distribution pattern, and the heuristic produces the best median value (indicated with a line in the box) compared to the control techniques and the best average value (indicated with a diamond) except for one case (*Tcm*).

4.2.6.2. The effectiveness of risk-based prioritization with a fuzzy expert system to find faults in risky components (RQ2)

For the first research question, we consider whether the risk-based approach that incorporates a fuzzy expert system can find faults earlier. The results are encouraging, but we do not know whether the early detected faults are, indeed, the faults that reside in the risky components. Thus, our second research question (RQ2) considers whether the risk-based approach with a fuzzy expert system can be more effective

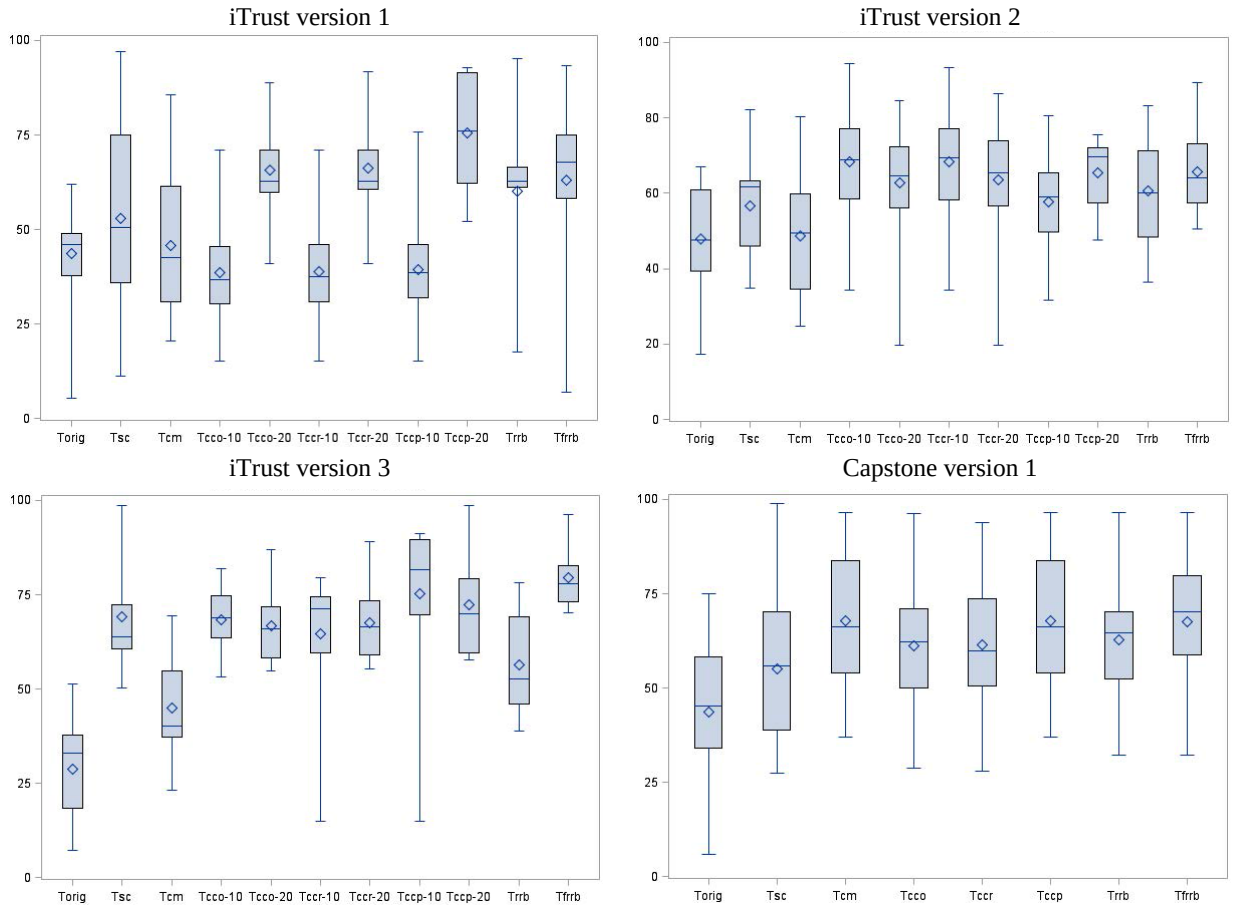

Figure 4.7. APFD boxplots for all controls and the heuristic: iTrust and Capstone

at finding faults associated with risky components early compared to the control techniques. To evaluate RQ2, we consider five control techniques: *Torig*, *Tsc*, *Tcm*, *Trrb*, and a clustering technique that produced the best results for RQ1 (i.e., *Tccp-CS20* for *iTrust* and *Tccp* for *Capstone*).

To address this research question, we measured the percentage of the total risk severity weight (PTRSW) values that we described in Section 4.1.3.2, showing how fast the technique can detect faults in the risky components. Tables 4.29 and 4.30 show the PTRSW values for all versions of *iTrust* and *Capstone*, respectively. The first column of Tables 4.29 and 4.30 shows the version number, and the second column categorizes the control techniques and heuristic. The third column shows the prioritization techniques. The subsequent columns show the PTRSW values when we foreshorten the test execution process by the specified percentage for each column. For example, the sixth column shows the PTRSW values when 50% of the test cases are executed, meaning that, for a 50% cutting ratio, we simulate the effects of having the testing process halted halfway through.

Table 4.29. Percentage of total risk severity weight (PTRSW) for different test execution levels: iTrust

Version	Technique		Percentage of Total Risk Severity Weight (PTRSW) for Different Test Execution Levels: *iTrust*			
			Execution Rate 12.5%	Execution Rate 25%	Execution Rate 50%	Execution Rate 75.0%
V1	Controls	Torig	0.00	0.00	26.49	70.90
		Tsc	12.31	26.07	70.51	76.49
		Tcm	1.87	5.22	29.85	52.24
		Tccp-CS20	7.46	7.46	30.97	71.27
		Trrb	21.64	30.22	67.91	82.09
	Heuristic	Tfrrb	21.64	26.12	70.52	79.48
V2	Controls	Torig	0.83	3.94	46.68	57.26
		Tsc	42.53	43.36	58.51	95.13
		Tcm	11.00	26.97	67.63	74.90
		Tccp-CS20	0.83	17.22	67.43	74.48
		Trrb	10.37	26.35	68.46	95.44
	Heuristic	Tfrrb	42.74	43.78	73.24	95.44
V3	Controls	Torig	0.00	0.85	25.32	29.36
		Tsc	46.98	46.98	47.44	99.07
		Tcm	0.43	10.43	19.36	65.32
		Tccp-CS20	45.53	45.53	70.85	90.00
		Trrb	0.00	18.94	66.38	86.38
	Heuristic	Tfrrb	49.79	70.64	97.66	99.57

Table 4.30. Percentage of total risk severity weight (PTRSW) for different test execution levels: Capstone

Version	Technique		Percentage of Total Risk Severity Weight (PTRSW) for Different Test Execution Levels: *Capstone*			
			Execution Rate 12.5%	Execution Rate 25%	Execution Rate 50%	Execution Rate 75%
V1	Controls	Torig	11.51	16.43	54.26	75.30
		Tsc	24.15	37.36	54.68	77.45
		Tcm	24.33	39.35	66.04	83.67
		Tccp	11.20	23.55	52.59	79.75
		Trrb	22.87	37.10	58.03	84.67
	Heuristic	Tfrrb	24.49	40.08	66.77	85.45

The results showed that our approach can detect more faults in the risky components earlier than the controls except for a few cases (*Trrb* in version 1 for *iTrust*). For version 1, the heuristic (*Tfrrb*) and the requirements risk-based approach (*Trrb*) showed very similar results, but for versions 2 and 3, the trend changed. For version 2, the heuristic produced relatively high fault detection rates when test execution rates were low, and for version 3, the differences between these two techniques were more outstanding. For instance, at a 25% execution rate, the heuristic produced 70.64%, but *Trrb* produced only 18.94%. In the case of *Capstone* (Table 4.30), our approach outperformed all control techniques. These results indicated that using requirements risks with a fuzzy expert system during prioritization was effective in locating faults that reveal risks early. Further, even when companies need to cut their testing process short due to their product release schedule, they can still identify and fix more important faults with the limited time and budget than otherwise.

Figure 4.8 presents the results graphically. The first three subgraphs show the results for *iTrust*, and the last graph shows the PTRSW comparison for *Capstone*. As we observed from Tables 4.29 and 4.30, our approach outperforms all control techniques for version 1 except for one technique (*Trrb*), and for versions 2 and 3, our approach outperforms the controls at all test execution rates. The trend for version 3's results is different from other versions. The results of the techniques for version 3 vary widely across test execution rates. In particular, the clustering-based approach performs better than all other control techniques. In the case of *Capstone*, the heuristic outperforms all control techniques across all test execution rates. Similar to version 1 of *iTrust*, the heuristic and the requirements risk-based approach (*Trrb*) produce very similar results.

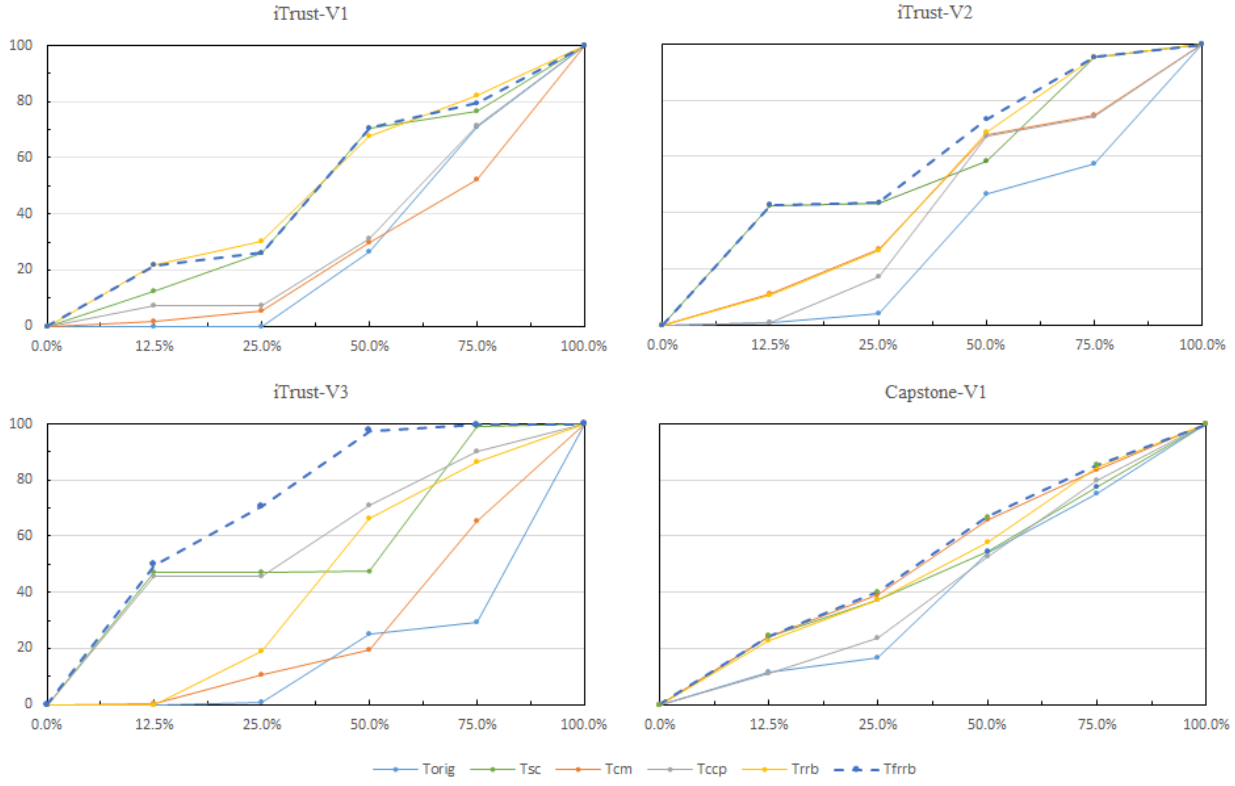

Figure 4.8. PTRSW comparison graphs for all versions of iTrust and Capstone

4.2.7. Discussion and Implications

In this section, we present more insight about the findings of our research and possible implications. Through the results we obtained with our study, we draw the following observations. First, the proposed systematic, risk-based approach is capable of outdoing the original test order, code-metric-based approach

and our previous requirements risk-based approach for all versions of the *iTrust* application. Cluster-based approaches with a cluster size of 20 outperform our proposed approach for version 1 while two cluster-based approaches with a cluster size of 10 outperform our approach for version 2. However, our proposed approach outperforms all cluster-based approaches for version 3. The source code of version 3 is significantly affected by the requirements modifications for version 3. In our previous requirements risk-based approach [28], we did not get better results for version 3 compared to cluster-based approaches because we only used one code-related risk factor, line of code (LOC), to estimate requirements risk. However, in this research, we use both LOC and McCabe Complexity, and we speculate that using more source code data to extract risk information has affected the better results obtained for version 3 that underwent major code modifications. In the case of *Capstone*, our approach outperforms all control techniques while producing the same result as the *Tccp* approach. Overall, the proposed approach produces very effective results across all versions of the *iTrust* and *Capstone* programs in spite of imprecise, inconsistent, and complex conditions that may occur when extracting the risk information from software requirements. Using a fuzzy expert system contributes to handling such circumstances successfully and facilitates making more realistic risk estimations. Further, we emulate expert thinking in the risk estimation procedure by using our fuzzy expert system and minimize subjectivity in the risk estimation procedure. Additionally, we employ a semi-automated process to assess the risk factors for our risk estimation procedure. Hence, having a systematic risk estimation procedure is the most possible reason for getting better results with all versions of every object program.

Second, the results indicated that our proposed approach has the ability to detect more faults early in risky components of software applications. In particular, for the third version of *iTrust*, our proposed approach detected a significantly higher number of faults in risky components at low test execution rates compared to the first and the second versions. Again, the requirements modifications in version 3 and their effect on the source code are a possible reason for this outcome. In the case of *Capstone*, our new approach produced the best results over other control techniques. In the proposed approach, requirements were primarily prioritized by considering their direct relationship with critical risk items, and then, the requirements were further prioritized using requirements risks which were estimated through a less subjective, fuzzy expert system based approach. Because the test cases are prioritized using the association between requirements and test cases, the top test cases in the prioritized test suite were able to detect faults in the high-risk components early. Therefore, in this research, considering the direct relationship between critical risk items and test cases was the major reason for the early detection of more faults in high-risk components.

The results of this research indicate very important implications for the software industry. Modern software systems which are developed in today's software industry are very complex and are vulnerable to malicious attacks; frequent changes are inevitable to maintain the systems' quality. Software systems which are intended to be available online (web-based) and the systems which are considered as mission critical are even more complex, undergo frequent changes, and have much bigger concerns for their security-related issues. The proposed approach pays much attention to these factors in terms of fault detection. Hence, by adopting our proposed approach, software development companies can detect faults in their modern applications within a short time frame. Furthermore, if a company happens to shorten their regression testing process due to any constrains (time, budget, etc.), they can still detect more faults in their applications, including faults in high-risk components, with the limited resources. In particular, early detection of more defects in critical systems is very important because such defects can eventually lead to severe failures, such as life-threatening conditions or huge financial loss. Therefore, using our proposed approach, companies can develop their software systems with more confidence and cost-effectiveness while meeting their tight production deadlines.

4.2.8. Threats to Validity

This section describes the internal, external, and construct threats to the validity of this study, and the approaches we used to limit their effects.

4.2.8.1. Construct Validity

Estimating the potential requirements volatility (PRV) for requirements and determining the correlation between requirements and risks items needed human involvement. Because human judgment is based on several factors, such as human experts' knowledge and experiences, the results could vary from person to person. We formulated and validated the rules of our fuzzy expert system based on our experiences and knowledge, but a fuzzy expert system with fewer or more rules could be developed and potentially change the results.

4.2.8.2. Internal Validity

In this study, we estimated the risks residing in the functional requirements in terms of product risks and did not consider other types of software risks, such as project and process risks. This threat can be minimized by adding more appropriate risk indicators in the requirements risk estimation process. We estimated the complexity and the size of a particular requirement using the McCabe Cyclomatic Complexity (MCC) and Lines of Code (LOC) for the class/method utilized to implement that particular requirement.

There are other alternatives available to measure the complexity and the size, and the results could be affected by the choice of different alternatives. However, much of the previous research has shown that MCC and LOC are good indicators to measure the complexity and size with ease.

Moreover, the crisp output obtained from the fuzzy expert system may vary due to several factors, such as the number of input/output membership functions, the types of waveforms, and the defuzzification methods. The fuzzy expert system used for this research is based on three triangular membership functions for both input and output variables because many previous fuzzy expert systems used a similar configuration successfully. We utilized the center of gravity (COG) method for the defuzzification because it is a widely used defuzzification method and is also considered as accurate. However, further studies can be performed to investigate the effectiveness of the proposed approach with different configurations.

4.2.8.3. External Validity

The object programs we used in this research are a small, industrial application (*Capstone*) and a mid-size, open source application (*iTrust*). Therefore, our findings cannot be interpreted in the context of large applications. This limitation can be addressed by applying our approach to large, open source and industrial applications.

4.2.9. Conclusions

In this research, we presented a systematic risk estimation approach using a fuzzy expert system which can minimize the subjectivity, imprecision, and inconsistency issues confronted by the requirements risks estimation process. We empirically evaluated the new approach using two Java applications with multiple versions. The results of this study demonstrated that our new systematic, risk-based approach can detect faults earlier and is even better at finding faults in the risky components earlier than the control techniques. With the proposed approach, software companies can manage their testing and release schedules better by providing early feedback to testers and developers so that the development team can fix the problems as soon as possible.

Thus far, we applied requirements risk-based test prioritization on small industrial application and mid-size open source software system. In the next step, we have proposed another requirements risk-based and requirements-based prioritization scheme that practitioners can implement with minimum effort on very large enterprise level software systems.

4.3. Requirements Based Test Prioritization Using Risk Factors: An Industrial Study

In a prior research, Srikanth et al. [59] introduced a Prioritization of Requirements for Test (PORT) 1.0 scheme where they showed the efficacy of TCP at the system level by considering four factors for each requirement. 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) for each requirement. Test cases that map to requirements with higher priority were ordered earlier for execution. They 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 [55, 56]. From this prior study, with extensive sensitivity analysis, they learned that CP was the biggest contributor to improving the rate of fault detection [56]. 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 research is to present PORT 2.0 where we apply only a set of factors towards prioritization [3]. In the prior study [59], 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 test. 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. 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 this research we show the application of these prioritization techniques on a very large 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

[3]The material in this subsection was co-authored by H. Srikanth, C. Hettiarachchi, and H. Do. C. Hettiarachchi had primary responsibility for conducting the empirical studies of requirements risk-based prioritization approaches. C. Hettiarachchi also drafted and revised all versions of this chapter. H. Srikanth had primary responsibility for conducting the empirical studies of requirements-based prioritization approaches and for providing object program data. H. Do served as the senior adviser, proofreader and checked the analysis conducted by H. Srikanth and C. Hettiarachchi.

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 research 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.

4.3.1. Approach

In this section, we describe two proposed requirements-based approaches: PORT 2.0 that applies two factors CP and FP, towards prioritizing test cases, and a risk-based prioritization technique that prioritizes based on risks associated with requirements categories. In this work, we consider three PORT-based prioritization approaches and two risk-based approaches. The following subsections describe them in detail.

4.3.1.1. PORT-Based Prioritization

Our prior contribution to the test case prioritization problem was the development of PORT 1.0 [59, 55, 56]. 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 [55, 56]. In our prior work, we validated PORT 1.0 on four industrial projects and results are discussed [55]. 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 [56]. In this research, we extend the research by further investigating CP towards 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 [55, 56].

- Customer Priority (CP): provides a measure of how important a particular requirement is to the customer. According to Moisiadis, approximately 45% of the software functions are never used, 19%

are rarely used, and only 36% of the software functions are always used [44]. A focus on customer requirements for development has been shown to improve customer-perceived value and satisfaction [11, 12, 34]. 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 [46].

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 towards 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.

4.3.1.2. Risk-Based Prioritization

Risk-based approaches used in software testing typically focus on risks associated with software requirements [9, 33]. Amland [6, 7] defined risk exposure as a product of probability of fault occurrence and the cost if the fault occurs in the production. In this research, we considered 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 [6, 7] 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 Equation 4.13.

$$RE_i = RL_i * RI_i \tag{4.13}$$

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 Equation 4.14.

$$RL(Cat)_i = T_i/TT_i \tag{4.14}$$

where $RL(Cat)i$ is the risk likelihood of requirements category i and Ti and TTi 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 (RI_{BC}). 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 RI_{BC} 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 un-subscription 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 the Equation 4.15.

$$RI_{FP}(Cat)_i = FF_i/TFF_i \tag{4.15}$$

where $RI_{FP}(Cat)_i$ 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 the Equation 4.15 as follows.

$RI_{FP}(Cat)_A = 30/150 = 0.2$

After calculating RL and RI values, the risk exposure values for both business-criticality and the fault proneness based approaches are calculated using the Equation 4.13. When calculating the risk exposure of a requirements category, both approaches use the same risk likelihood value that is calculated by using Equation 4.14. 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.

$RE_{BC}(Cat)_A = 0.10 * 5 = 0.50$

$RE_{FP}(Cat)_A = 0.10 * 0.2 = 0.02$

Finally, these risk exposure values of requirements categories are used to prioritize requirements categories. One risk-based prioritization technique is based on the RE_{BC} values while the other risk-based prioritization technique is based on the RE_{FP} values. Thus, two distinct sets of prioritized requirements categories are obtained by prioritizing the requirements categories using RE_{BC} and RE_{FP} 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 RE_{BC} 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.

4.3.2. 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?

4.3.2.1. Object of Analysis

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.

4.3.3. Variables and Measures

In this section, we describe independent and dependent variables.

4.3.3.1. 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 4.3.1.1
  - 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

4.3.3.2. Dependent Variable

In this research, our dependent variable is Average Percentage of Fault Detection (APFD) described in Section 4.1.3.2.

4.3.4. 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 4.31 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 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 to 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 4.31 below shows the CP for each of the requirements category. 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 FP 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 FP 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 FP weight (0.3) and add it to its CP score (5) 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 4.3.1.2.
8. Risk Impact (RI_{BC}): 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 4.3.1.2.
9. Risk Impact (RI_{FP}): the risk impact estimated by means of field failures data.
10. Risk Exposure (RE): the quantitative measurement of risks reside in a requirements category. The multiplication of RL and RI of a particular requirements category produces the RE of the category as shown in Equation 4.13 in Section 4.3.1.2. The tenth column contains the business-criticality based, risk exposure values (RE_{BC}) of each requirements category.
11. Risk Exposure (RE_{FP}): the fault proneness based, risk exposure value of a requirements category.

Table 4.31. Experimental data set: SaaS

Requirements Categories	Test Cases	Defect Ratio	FP	CP	FP+CP	RL	RI_{BC}	RI_{FP}	RE_{BC}	RE_{FP}
A	302	0.19	5	5	5.0	0.169	5	0.427	0.84263	0.07196
B	79	0.27	2	3	2.7	0.044	3	0.067	0.13225	0.00297
C	56	0.00	0	2	1.4	0.031	2	0.005	0.06250	0.00016
D	128	0.00	2	5	4.1	0.071	5	0.090	0.35714	0.00642
E	30	0.00	1	4	3.1	0.017	4	0.034	0.06696	0.00056
F	191	0.16	3	5	4.4	0.107	5	0.180	0.53292	0.01916
G	22	0.00	1	3	2.4	0.012	3	0.045	0.03683	0.00055
H	13	0.00	1	3	2.4	0.007	3	0.022	0.02176	0.00016
I	246	0.01	1	2	1.7	0.137	2	0.034	0.27455	0.00463
J	45	0.00	2	4	3.4	0.025	4	0.067	0.10045	0.00169
K	185	0.00	1	3	2.4	0.103	3	0.011	0.30971	0.00116
L	49	0.00	0	0	0.0	0.027	1	0.005	0.02734	0.00014
M	50	0.32	0	2	1.4	0.028	2	0.005	0.05580	0.00014
N	33	0.12	1	4	3.1	0.018	4	0.011	0.07366	0.00021
O	363	0.02	1	4	3.1	0.203	4	0.011	0.81027	0.00228
Average			1.40	3.27						
Weight			0.3	0.7						

Table 4.31 presents the different requirements categories, the total number of test cases of each category, the defect ratio, FP, CP, FP+CP, RL, RI_{BC}, RI_{FP}, RE_{BC}, and RE_{FP}. 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 RI_{BC} is 5, the RI_{FP} is 0.427, the RE_{BC} is 0.84263 and the RE_{FP} is 0.07196.

PORT and risk-based approaches require two different experimental setups. The following two subsections (4.3.4.1 and 4.3.4.2) explain the experimental setup for each of the approaches in detail.

4.3.4.1. 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) 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 to 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 4.31.

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.

4.3.4.2. 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 4.3.1.2. The seventh column of Table 4.31 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 4.31 shows the business-criticality based risk impact (RI_{BC}) values of all requirements categories. The ninth column of the table shows RI_{FP} values of all requirement categories. The resultant risk impact values range from 0.005 to 0.427.

After completing the estimations of RL and RIs, risk exposure (RE) values for each requirements category are calculated by multiplying RL of a category by its RI using the Equation 4.13 explained in Section 4.3.1.2. The tenth column of Table 4.31 shows the business-criticality based risk exposure values (RE_{BC}) and the last column shows the fault proneness based, risk exposure values (RE_{FP}) of all requirements categories. In this experiment, the risk exposure values of business-criticality based approach (RE_{BC}) range from 0.02176 to 0.84263 whereas fault proneness based, risk exposure (RE_{FP}) 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 RE_{BC} and RE_{FP}, 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.

4.3.5. Data and Analysis

In this section, we summarize the results of our study. We use APFD metric to measure the effectiveness of the prioritization techniques. The results of our test case prioritization techniques are shown in Table 4.32. 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.

Table 4.32. Test case prioritization results: SaaS

	Average of 30 Random Samples	PORT			Risk-Based	
	(Tran)	(Tp-cp)	(Tp-fp)	(Tp-fp-cp)	(Tr-bc)	(Tr-fp)
APFD Values	51	69	80	72	71	77

Now, we briefly discuss our results for each of our research questions and we discuss further implications of the results in Section 4.3.6.

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

Table 4.32 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 4.32 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 FP and CP (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 Figure 4.9, 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 towards applying the techniques.

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

Table 4.32 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).

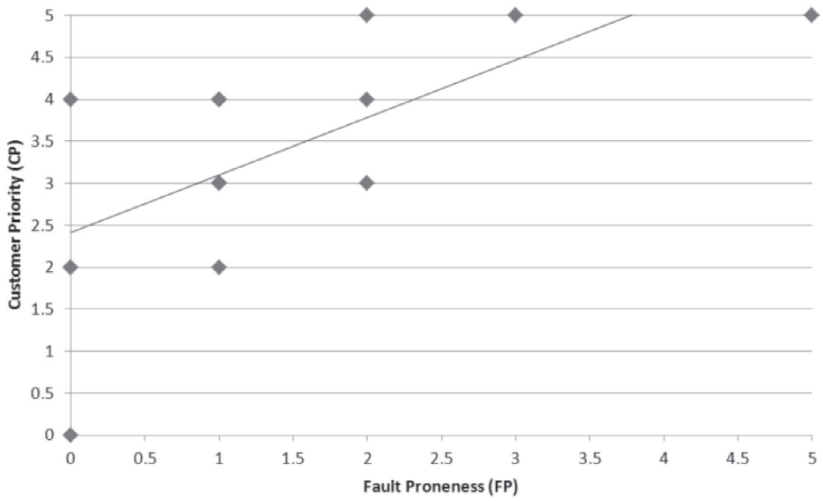

Figure 4.9. Correlation between CP and FP (H. Srikanth et al.)

4.3.6. Discussion and Implications

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 [28]. 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 [28], 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.

4.3.7. Threats to Validity

4.3.7.1. 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 (RI_{BC}) 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.

4.3.7.2. 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.

4.3.7.3. 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 [24, 64]. 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).

4.3.8. 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 research, 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, FP and CP 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 demonstrated that risk-based approaches can be effective with non-coverage based prioritization schemes.

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.

4.4. A Systematic Approach to Test Case Prioritization Using a Fuzzy Expert System

Our previous approaches that use requirements including risks have improved regression testing techniques and have advanced our understanding about the role of requirements in improving software quality. In our previous research described in Section 4.1, we proposed a new requirements risk–based test case prioritization approach that considers a direct relationship among requirements, several risks items, and the test cases. The experimental results are promising, but our previous approach requires human expert involvement to estimate requirements risks in terms of risk indicators and risk items, a process that can be subjective, expensive, and time consuming.

To address these limitations, we employed a fuzzy expert system (FES) to make our risk estimation process more systematic, efficient, and cost–effective using human expert knowledge. To date, researchers in various areas have applied fuzzy expert systems to help solve complex decision-making problems. For example, Adeli and Neshat [4] applied fuzzy expert systems to diagnose heart disease. Carr and Tah [13] used fuzzy expert systems to assess the risks of construction projects. However, very few researchers have applied fuzzy expert systems to the software testing domain. For instance, Xu et al. [65] used fuzzy expert systems for test case selection in regression testing.

In this work, we investigate whether the use of a fuzzy expert system can help improve the effectiveness of test case prioritization techniques that use requirements and risks. The semi-automated approach used in this research simplified the lengthy risk-assessment process and also made the process less subjective and more efficient. The proposed approach was evaluated by using three applications (two open source and one industrial). The study results indicated that our test case prioritization techniques that use requirements and risks with a fuzzy expert system can improve the rate of fault detection compared to several other control techniques, including our previous requirements risk–based technique described in Section 4.1.

4.4.1. Approach

In this section, we describe our requirements risk–based test case prioritization approach, which uses the fuzzy expert system (FES) to estimate the risks in software requirements. The new approach has three major steps:

1. Estimate requirements risks using FES
2. Estimate requirements risks using the weighted sum model (WSM)
3. Prioritize requirements and test cases

Figure 4.10 gives an overview of the proposed technique. The main steps of the approach are shown in light-blue boxes, while the inputs and outputs for each step are shown in the ovals. The first step estimates the risk values for each requirement using a fuzzy expert system. The second step estimates the risk values for each requirement using the weighted sum model. In the last step, requirements are prioritized by using the results produced from the first two steps, and then test cases are prioritized with mapping information between requirements and test cases. The following subsections describe each step in detail.

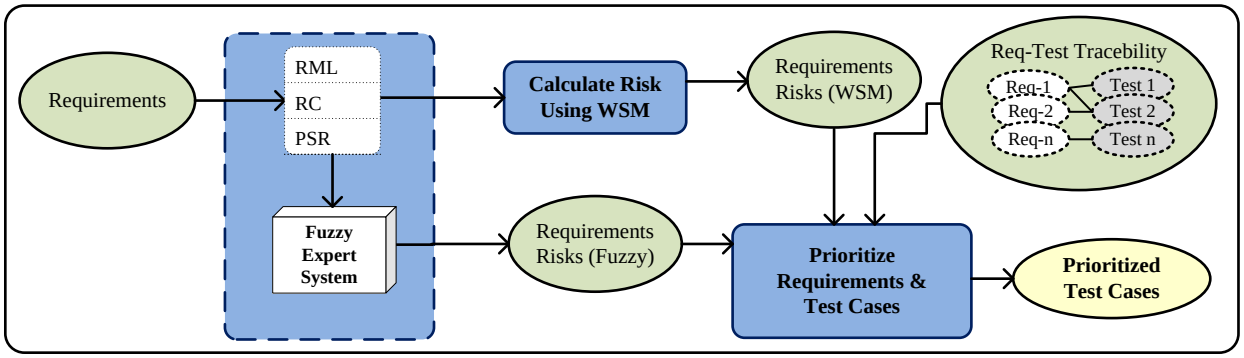

Figure 4.10. Overview of Requirements Risk-Based Approach

4.4.1.1. Estimate Requirements Risks Using FES

In this subsection, we describe how we used fuzzy expert system to estimate the software requirements' risks. In general, FES involves four main steps: fuzzification, fuzzy inference, composition, and defuzzification.

4.4.1.1.1. Fuzzification

FES requires values for the input variables to perform the fuzzification process. In this step, based on our experience and exploring the existing literature [7, 9, 63] we identified three risk indicators that can

help locate software system faults: requirements modification level (RML), requirements complexity (RC), and the potential security risks (PSR) of the requirements. These risks indicators are the input variables for FES.

4.4.1.1.1.1. Requirements Modification Level (RML)

During a software system's life cycle, requirements modifications are inevitable, and the modified requirements are likely to introduce defects. Thus, we consider the requirements modification level (RML) as a risk indicator. RML indicates the degree of modification for a requirement that has been changed from the previous version.

To support the RML estimation process, we developed a requirements comparison tool using the Python programming language. The comparison tool reads a set of requirements from two consecutive versions and illustrates the differences (by the modification percentage) between the two corresponding requirements. The percentage is normalized into a range from 0 to 10, where 0 indicates no modifications and 10 indicates the highest modification level. New requirements are assigned the highest value of 10. Because the tool may not detect minor text changes in the requirements, we performed a quick review of the RML values obtained with the comparison tool in order to be certain that these RML values properly reflect the requirements modifications.

4.4.1.1.1.2. Requirements Complexity (RC)

We consider the requirements' complexity as the second risk indicator. According to a study conducted by Amland [6], requirements for complex functionalities tend to introduce more faults during implementation, and functions with a higher number of faults also have a higher McCabe complexity. Therefore, to measure the complexity of the requirements, we used the McCabe complexity of the function that implements a requirement. Note that, in our experiment, traceability matrices that came with the applications were used to identify the software functions associated with the requirements. The obtained McCabe complexity values are normalized into a range from 0 to 10. A value of 0 indicates the lowest complexity, whereas 10 indicates the highest complexity for a requirement.

4.4.1.1.1.3. Potential Security Risks (PSR)

Software security is a major concern for software applications due to the rapid growth of malicious activities against software applications such as SQL injection, eavesdropping, etc. A software application's security flaws will lead to severe consequences unless software security-related issues are identified and properly handled as early as possible. Thus, as the third risk indicator, we used potential security risks

(PSR) that reside in the requirements. To estimate the PSR value for each requirement, we used a term extraction tool. For a particular requirement, the tool can find the number of security keywords, which are predefined in a database. For example, if we have 100 security keywords in the database and a particular requirement contains 20 of the 100 keywords, then the PSR value is calculated by dividing 20 by 100. Likewise, with the tool, PSR values for all requirements are calculated, and the resultant PSR values are then normalized into a range from 0 to 10. A value of 0 indicates that there are no risks related to security, whereas a 10 indicates the highest security risks.

The security keywords database is a part of a term extraction tool. When developing the database, we consider six security objectives that are identified in the software security field: confidentiality, integrity, availability, privacy, authentication, and accountability. For each security objective, we identify a set of security keywords and place them in the database. The first column of Table 4.33 shows the security objectives, and the second column shows a sample set of keywords for each objective. Terms in the first row such as "system", "patient", "record", and "display", are examples of security keywords about confidentiality.

Table 4.33. Security objectives and keywords

Security objective	keywords
Confidentiality	system, provide, ability, patient, result, exam, capture, datum, record, send, display, confidential, data, medication, information, list, requirement, status, consuming, order, complete
Integrity	ability, exam, send, capture, result, store, consuming, patient, pass, click, pick-list, status, application, element, create, generate
Availability	run, availability, retain, time, year, nurse, destroy, application, legally, recent, retention, care, maximum, real-time, information, period, destruction, record, historical
Privacy	consent, patient, person, phi, disclosure, purpose, privacy, directive, require, organization, law, authorization, information, connect, disclose, healthcare, inform, monitor, jurisdiction, collect
Authentication	authentication, login, mac2002, username, user, authenticate, identify, cash, identity, password, waitlist, log, registration, list2012, uniquely, credentials, valid
Accountability	provide, exam, result, send, consuming, click, pass, patient, capture, pick-list, application, audit, status, record

Once we obtain the values for each input variable, the fuzzification process determines the degree of membership of each input value for the membership functions defined in FES. RML, RC, and PSR are the input variables for FES, and each input variable is scaled from 0 to 10.

In this work, we want to compare how different membership functions and different wave types for those membership functions affect the experiment's overall results. Thus, we use fuzzy expert systems with four different membership functions: three triangular membership functions, three trapezoidal membership functions, four triangular membership functions, and four trapezoidal membership functions. The

triangular membership functions are defined by three parameters (a, b, c), where *a* is the lower bound, *b* is the mean value, and *c* is the upper bound. Table 4.34 shows these parameter values for three membership functions. The trapezoidal membership functions are defined by four parameters (a,b,c,d), where *a* is the lower limit, *b* is the lower support limit, *c* is the upper support limit, and *d* is upper limit. Three membership functions have three linguistic values, and four membership functions have four linguistic values as shown in Tables 4.34 and 4.35. These linguistic values categorize the input and output variables into the number of input and output sets. For instance, low, medium, and high linguistic values represents the risk levels of the requirements, from low to high, for the three membership functions' output variables. In this work, as shown in Table 4.34 and Table 4.35, the parameter values are the same for the input and output membership functions. The graphs for the membership functions for input and output variables are shown in Figure 4.11 and Figure 4.12.

Table 4.34. Three membership functions for input and output variables

Linguistic Value	Triangular Fuzzy Numbers (a,b,c)	Trapezoid Fuzzy Numbers (a,b,c,d)
Low	0,0,5	0,0,2,4
Medium	0,5,10	2,4,6,8
High	5,10,10	6,8,10,10

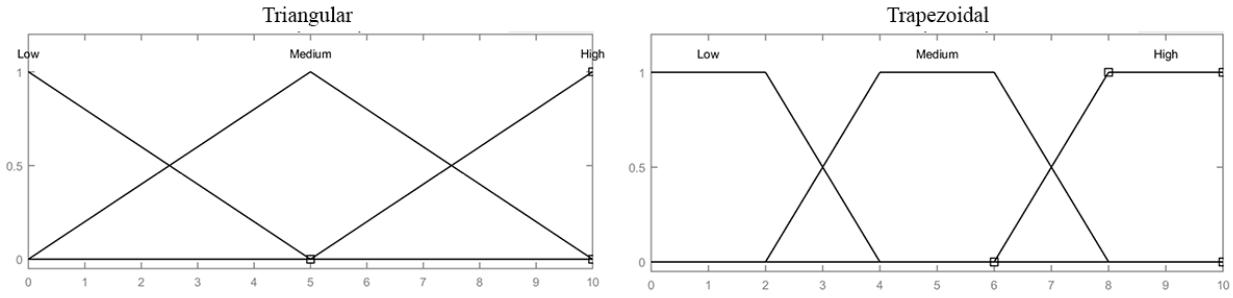

Figure 4.11. Three membership functions of input and output variables

4.4.1.1.2. Fuzzy Inference

The fuzzy inference process takes the fuzzified input from the fuzzification process and determines the fuzzy output set using the Mamdani fuzzy inference system [42]. The fuzzified inputs are applied to the antecedent of each rule in order to determine the degree of truth for each rule defined in FES. The consequent of each rule takes the rule's determined degree of truth and derives the output membership function.

Table 4.35. Four membership functions for input and output variables

Linguistic Value	Triangular Fuzzy Numbers (a,b,c)	Trapezoid Fuzzy Numbers (a,b,c,d)
Low	0,0,3.3	0,0,1,3
Moderate	0,3.3,6.6	1,3,4,6
High	3.3,6.6,10	4,6,7,9
Very High	6.6,10,10	7,9,10,10

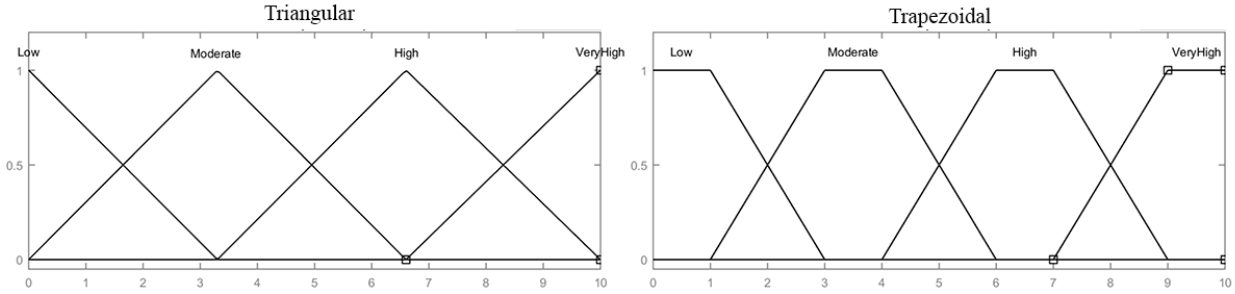

Figure 4.12. Four membership functions of input and output variables

The output membership functions represent different levels of risks for the requirements. For instance, the three values (low, medium, and high) for the three output membership functions categorize the requirement risks into three risk categories, from low risk to high risk, according to the rules defined in FES. Because our parameter values for the output membership functions range from 0 to 10, our requirement risk values (i.e., output values of FES) also range from 0 to 10. Table 4.36 illustrates the input and output variable values for each requirement. The first column of Table 4.36 shows a sample set of requirements for the *iTrust* application. The second, third, and fourth columns show the input variable values for RML, RC, and PSR, respectively. The last column shows the output variable values of requirements (RR[Fuzzy]). Because the rule set defined in the FES is the core part of the FES, we explain, in detail how we define the rules for our FES in the next section.

4.4.1.1.3. Fuzzy Rules

The fuzzy rule set is the main component of a fuzzy expert system, and it represents the expert knowledge. The rules are developed using the knowledge acquired from different sources such as existing literature, human expertise and experiences, and historical data. To construct the rules for our FES in order to prioritize test cases based on requirements risks, we need to know how the risk factors are associated with the potential faults. To gain such knowledge, we used the aforementioned knowledge-gathering approaches and also consulted experts to obtain their opinions to calibrate our rule set for better and more accurate

Table 4.36. Risk indicators and fuzzy input-output values

Requirement	RML	RC	PSR	RR(Fuzzy)
UC1S1	0.0	1.9	10.0	4.0
UC3S4	0.0	2.8	4.0	4.5
UC6S1	0.0	2.8	4.6	4.5
UC10S2	7.9	1.9	2.8	4.9
UC11S1	6.5	2.8	6.0	5.3
UC23S3	3.0	1.9	4.6	4.6
UC26S3	0.0	2.8	1.0	3.6
:	:	:	:	:
UC34S5	10.0	2.8	6.4	5.8

results. Several studies have shown that requirement modifications can result in a defective system [56, 59]. The declaration of the fuzzy rules for our FES is based on this knowledge, and we give high priority to requirement modification, medium priority to requirement complexity, and relatively low priority to software security. Priority for each of these criteria was determined based on the software engineer's experience and knowledge from the existing literature. For instance, some research shows that requirements modification is a primary source of defects for software systems [7, 40, 59].

In our FES, there are three input variables and one output variable. For the experimentation, we consider three and four membership functions in our approach. When we use three membership functions, 81 different rules can be constructed, but after eliminating meaningless rules, only 27 rules are selected. Suppose we have the following rules in the original rule set:

Rule 1: If RML is H and RC is H and PSR is H then Risk is H.

Rule 2: If RML is H and RC is H and PSR is H then Risk is M.

Rule 3: If RML is H and RC is H and PSR is H then Risk is L.

Even though we can construct three different rules, we only select one rule (Rule 1) because choosing multiple rules with different output memberships for the same set of input variables that have the same input membership functions is meaningless. When we use four membership functions, we construct 256 rules, but only 64 rules are utilized with FES. A sample subset of our rules with three membership functions is shown in Table 4.37.

4.4.1.1.4. Defuzzification

The defuzzification process provides crisp output for each requirement and represents the overall risk level for a requirement. We use the crisp values to prioritize requirements. The higher the crisp value,

Table 4.37. Fuzzy rules for requirement risk-based testing

R1. If RML is H and RC is H and PSR is H then Risk is H
R2. If RML is H and RC is H and PSR is M then Risk is H
R6. If RML is H and RC is M and PSR is L then Risk is H
R8. If RML is H and RC is L and PSR is M then Risk is M
R12. If RML is M and RC is H and PSR is L then Risk is M
R15. If RML is M and RC is L and PSR is M then Risk is M

the higher the requirement's priority. We use two different defuzzification methods, centroid and middle of maximum (MOM), which are considered more accurate and are more widely accepted than other methods.

4.4.1.2. Estimate Requirements' Risks Using the Weighted Sum Model (WSM)

In the previous step, we obtained the fuzzy risk values, RR(Fuzzy), for each requirement, and several requirements may have the same value; therefore, several requirements could have the same priority level. In that case, we calculate the risks for each requirement using the weighted sum model (WSM) shown in Equation 4.16 as a secondary prioritization criterion.

$$RR(WSM)_{(Req_j)} = \sum_{i=1}^{n} (W_i * R_{ji}) \tag{4.16}$$

where n is the number of risk indicators, W_i is the weight of the risk indicator, and I_{ji} is the risk level of the requirement, Req_j, for risk indicator i.

We considered the three risk indicators explained in previous step, and the weight for each indicator was determined by using the analytic hierarchy process (AHP), which is based on a pairwise comparison of risk indicators. Two experts performed the comparison, and we averaged the priority vector (PV) values obtained from each expert to get the final PV values for each indicator. In Table 4.38, columns two to four show the comparison values for the risk indicators, the fifth column shows the total, and the last column shows the PV values.

In Table 4.39, the first column shows the risk indicator, the second and third columns show the PV values obtained from two experts, and the fourth column shows the averaged PV values. These values are normalized to a scale from 1 to 5. Normalized PV values are the weights for the risk indicators and are shown in the last column. Further, we compared the weight values obtained from the AHP process with the risk weight values of a widely used risk estimation approach [7], and we observe that similar risk indicators have almost the same values.

Table 4.38. Risk indicator comparison

	RML	Complexity	PSR	Total	Priority Vector
First Expert's Comparison					
RML	0.44	0.40	0.57	1.42	0.47
Complexity	0.44	0.40	0.29	1.13	0.38
PSR	0.11	0.20	0.14	0.45	0.15
Second Expert's Comparison					
RML	0.55	0.57	0.50	1.62	0.54
Complexity	0.27	0.29	0.33	0.89	0.30
PSR	0.18	0.14	0.17	0.49	0.16

Table 4.39. Risk indicators and weights

Indicator	PV1	PV2	Average	Weight
Requirement Modification Level	0.47	0.54	0.51	5
Requirement Complexity	0.38	0.30	0.34	3
Potential Security Risk	0.15	0.16	0.16	1

After determining the weights, we applied the input variable values for the requirements to Equation 4.16 and obtained the risk values for each requirement. The following example shows how to calculate the risk value for the UC1S1 requirement ("The healthcare personnel creates a patient as a new user of the system.") using WSM. The first column of Table 4.40 shows a sample set of requirements, the second to fourth columns show the input values for the risk indicators, and the last column shows the risk value that is calculated for each requirement using WSM.

RR(WSM)$_{(UC1S1)}$ = (5*0) + (3*1.9) + (1*10) = 15.7

Table 4.40. Requirements risks: WSM

Requirement	RML	RC	PSR	RR(WSM)
UC1S1	0.0	1.9	10.0	15.7
UC3S4	0.0	2.8	4.0	12.4
UC6S1	0.0	2.8	4.6	13.0
UC10S2	7.9	1.9	2.8	48.0
UC11S1	6.5	2.8	6.0	46.9
UC23S3	3.0	1.9	4.6	25.3
UC26S3	0.0	2.8	1.0	9.4
:	:	:	:	:
UC34S5	10.0	2.8	6.4	64.8

4.4.1.3. Prioritize Requirements and Test Cases

In this final stage, we prioritize the requirements using the fuzzy risks values for the requirements RR(Fuzzy), and then using the RR(WSM) values when the requirements have the same RR(Fuzzy) values. If the requirements still have the same priority, then we randomly order those requirements. For all the applications and versions used in this experiment, only a few tests (less than 5% of tests per version) required random prioritization. To prioritize test cases, we needed to determine the relationship between test cases and requirements. Therefore, we used the traceability matrices developed by the object programs' developers to generate the prioritized test cases.

4.4.2. Empirical Study

In this study, we investigate the following research question:

RQ: Does requirements risk–based, test case prioritization based on a fuzzy expert system improve the rate of fault detection with regression testing?

4.4.2.1. Objects of Analysis

Table 4.41. Experiment object and associated data

Object	Version	Size (KLOCs)	Requirements	Test Cases	Mutation Faults	Mutation Groups
iTrust	v1	24.42	91	122	54	13
	v2	25.93	105	142	71	12
	v3	26.70	108	157	75	12
PasswordSafe	v1	17.51	26	56	72	14
	v2	18.40	32	70	77	15
Capstone	v1	6.82	21	42	118	23

In order to evaluate our new approach, we utilize three applications: two open source applications and one capstone project. The *iTrust* [31] program is the open source application used for this experiment. It is a patient centric, electronic health-record system that was developed by the Realsearch Research Group at North Carolina State University. In this experiment, four versions of the *iTrust* system (versions 0, 1, 2, and 3) are used. We consider the functional requirements of the three object programs, and the test cases are used to check the system functionalities that are associated with system requirements. All the test cases used for the experiment were developed by the *iTrust* system developers.

PasswordSafe is the second application used in the experiment. *PasswordSafe* is a JAVA based, open source password management program that helps to manage multiple passwords easily and securely

on Linux, Mac, and Windows operating systems. Three versions of the *PasswordSafe* program (versions 0, 1, and 2) are utilized with our experiment. The requirements documentation, traceability matrices, and some test cases were developed by graduate students of North Dakota State University in order to increase the test coverage while most of test cases were developed by the developers of the program.

Capstone is the third application used in the experiment. This program was developed by computer science graduate students at North Dakota State University to facilitate online examination procedures. Two versions of the *Capstone* program (versions 0 and 1) are utilized with our experiment, with v0 being considered the base version for the regression testing. The data for the three applications used during this experiment is shown in Table 4.41. For each system, v0 (the base version) is not listed in the table because regression testing starts with the second version. However, we utilize the information from v0 to obtain the mutants for v1.

4.4.3. Variables and Measures

4.4.3.1. Independent Variable

The test case prioritization technique is the independent variable manipulated in this study. We consider four control techniques, including two requirements risk–based techniques and one heuristic prioritization technique, as follows:

- Control Techniques
  - Original (*Torig*): The object program provides the testing scripts. *Torig* executes test cases in the order in which they are available in the original testing script.
  - Code metric (*Tcm*): This technique uses a code metric that we defined in our previous study [8]. The code metric is calculated using three types of information obtained from source code—Line of Code (LOC), Nested Block Depth (NBD), and McCabe Cyclomatic Complexity (MCC)—which are considered good predictors for finding error-prone modules [52, 69].[4]
  - Requirements risk–based technique (*Trrb*): This technique is our previous risk–based technique which prioritizes the test cases based on the risks in the requirements as well as the association between requirements and a system's potential defect types [28].
  - Requirements risk–based technique with WSM (*Trrb-wsm*): This technique prioritizes the test cases based on the risks residing in the requirements. A requirement's risk levels are estimated

[4] $Tcm = \frac{NBD}{Max(NBD)} + \frac{MCC}{Max(MCC)} + \frac{LOC}{Max(LOC)}$

in terms of three weighted risk indicators, and the overall risk for that requirement is calculated by employing the weighted sum model. Section 4.4.1 describes the WSM–based technique in detail.

- Heuristic (*Trrb-fes*): The heuristic technique prioritizes test cases based on the requirements risk that is primarily estimated using the fuzzy expert system. This proposed approach is explained, in detail in Section 4.4.1.

4.4.3.2. Dependent Variable

We considered one dependent variable, average percentage of fault detection (APFD). The APFD value represents the average for the percentage of fault detection while executing a particular test suite. APFD values range from 0 to 100. Test case prioritization techniques are evaluated based on the APFD values so that techniques that obtain higher APFD values (closer to 100) are considered better practices compared to techniques with lower APFD values.

4.4.4. Experimental Setup and Procedure

For each requirement in this study, we estimated the risks using a fuzzy expert system as explained in Section 4.4.1. We used three input variables (RML, RC, and PSR) for FES to obtain the crisp output that represents the requirements' overall risk level. To obtain the input variable values for each requirement, we used a tool-assisted approach as explained in Section 4.4.1. A graduate student who has several years of software industry experience built a Python program with cosine similarity to measure the RML values. Another "R" language–based term-extraction program was developed to measure PSR. To obtain the McCabe Complexity, Eclipse IDE was used. The input variable values obtained from this semi-automated process were reviewed by the graduate student to validate accuracy. For RML values, 3% of the requirements needed few modifications for their RML values, whereas about 2% of the requirements required slight adjustment to their PSR values. For example, the semi-automated process may not adequately evaluate the RML value for a numerical modification, which might be considered a significant modification (e.g., format change of data records from version 5.0 to 6.0). Therefore, in such circumstances, a manual inspection is required. In this experiment, two sets of fuzzy rules were used. For three and four membership functions, the graduate student (as a human expert) developed twenty-seven and sixty-four rules, respectively. The fuzzy expert system was built in MATLAB and obtained crisp output by combining different membership functions, wave types, and defuzzification methods. Therefore, for each version of every object program, we obtained eight

sets of crisp outputs. The crisp output produced by the fuzzy expert system was the first criterion for test case prioritization. If several requirements had the same crisp values, then we used a secondary criterion that estimated the requirements risk using WSM. The same input variable values obtained for FES were used as input for WSM.

The weight values for WSM were obtained using the AHP process as explained in Section II-B. Two graduate students performed the AHP process as human experts, and the priority vector values obtained from each expert for every risk indicator were averaged and normalized to obtain the final weighted values for the risk indicators. To obtain the prioritized test cases, we used the requirement and test case mapping information that was provided by the object programs' developers.

We needed fault data to empirically evaluate the proposed approach, so we used a set of mutation faults that were created for our previous study [8]. From this set of mutations, we created several mutation groups for each version by randomly selecting n mutation faults per group, where n ranges from 1 to 10. We repeated this process across all versions of each object program used for this experiment to obtain the mutation fault groups. The *iTrust* application used thirteen, twelve, and twelve mutation groups for versions 1, 2, and 3, respectively, the *PasswordSafe* application used fourteen and fifteen mutation groups for versions 1 and 2, and the *Capstone* application used twenty-three mutation groups for version 1. The Eclipse IDE was used to obtain the code metric data, Line of Code (LOC), McCabe Cyclomatic Complexity (MCC), and Nested Block Depth (NBD), which are required for the code-metric control technique (Tcm).

4.4.5. Data and Analysis

In this section, we present and analyze our study's results. We summarize the data in Table 4.42. All values shown in the table are average APFD values. The *iTrust* program has thirteen, twelve, and twelve data points for v1, v2, and v3, respectively, the *PasswordSafe* program has fourteen and fifteen data points for v1 and v2, and *Capstone* has twenty-three data points for v1.

Table 4.42 shows the results for the *iTrust*, *PasswordSafe*, and *Capstone* applications. In Table 4.42, the first column shows the object programs, the second column shows version numbers, columns three to six show the APFD values for the control techniques, and the rest of the columns show the heuristics' APFD values.

4.4.5.1. Analysis of results for iTrust

The results for *iTrust* show that all eight heuristics outperformed *Torig* across all versions. The improvement rates ranged from 26.58% to 204%. For version 1, *Trrb-fes-4-Td* with Middle of Maximum

Table 4.42. iTrust, PasswordSafe, and Capstone Results: APFD

Object	Version	Control				Heuristic							
						Three Membership Function				Four Membership Function			
		Torig	Tcm	Trrb	Trrb-wsm	Triangular		Trapezoidal		Triangular		Trapezoidal	
						Centroid	MOM	Centroid	MOM	Centroid	MOM	Centroid	MOM
iTrust	V1	43.7	45.8	60	61.2	57.9	55.3	65.0	66.2	70.0	67.1	69.2	72.0
	V2	47.8	48.8	60.6	71.3	71.5	61.9	69.9	71.6	78.2	80.0	79.5	79.5
	V3	28.8	44.8	56	83.1	86.3	82.8	85.2	85.4	85.8	82.5	87.2	87.6
PasswordSafe	V1	44.2	55.3	62.8	60.4	62.4	64.2	70.4	67.9	75.4	76.4	78.5	75.2
	V2	43.7	55.1	61.4	68.8	70.0	65.6	75.6	76.8	78.1	80.3	81.2	81.8
Capstone	V1	43.7	67.8	62.9	62.9	64.5	64.4	64.0	63.8	63.5	63.1	64.4	63.8

(MOM) produced the best result (64.74%), while *Trrb-fes-4-Tr* and *Trrb-fes-4-Td* with MOM produced the best results for version 2 (67.41%) and version 3 (204.16%), respectively. Note that *4-Td* represents four trapezoidal membership function, whereas *3-Tr* represents three triangular membership function.

When we compared the heuristics with *Tcm*, we observed similar trends. All heuristics outperformed *Tcm*, but the improvement rates were slightly lower than *Torig* (from 20.78% to 95.59%). For version 1, *Trrb-fes-4-Td* with MOM produced the best result (57.19%), while *Trrb-fes-4-Tr* and *Trrb-fes-4-Td* with MOM produced the best results for version 2 (63.98%) and version 3 (95.59%), respectively.

In the cases of *Trrb* and *Trrb-wsm*, the trends changed slightly. For *Trrb*, the improvement rates ranged from -7.81% to 56.48%. For version 1, *Trrb-fes-4-Td* with MOM produced the best result (19.99%), while *Trrb-fes-4-Tr* and *Trrbf-4-Td* with MOM produced the best results for version 2 (32.05%) and version 3 (56.48%), respectively. However, only in version 1, *Trrb-fes-3-Tr* with centroid and MOM did not perform better than the control technique *Trrb*.

The improvement rates for *Trrb-wsm* range from -9.56% to 17.71%. Similar to other control techniques, the best results are produced by the heuristics for each version as follows: for version 1, *Trrb-fes-4-Td* with MOM deffuzification (17.71%); for version 2, *Trrb-fes-4-Tr* with MOM (12.21%); and for version 3, *Trrb-fes-4-Td* with MOM (5.51%). For some cases, the heuristics did not perform better than *Trrb-wsm* (e.g., *Trrb-fes-3-Tr* with centroid and MOM for version 1, *Trrb-fes-3-Td* with centroid and *Trrb-fes-3-Tr* with MOM for version 2, and *Trrb-fes-3-Tr* with MOM and *Trrb-fes-4-Tr* with MOM for version 3).

Further, we compared the two heuristic groups (three and four membership functions). The results show that four membership functions produced better results over three membership functions except for two case. When we compared another two heuristic groups (triangular and trapezoidal membership functions), in most cases, techniques that used trapezoidal membership functions produced better results than the techniques that utilized triangular membership functions.

4.4.5.2. Analysis of results for PasswordSafe

The results for *PasswordSafe* (Table 4.42) show that all eight heuristics outperformed *Torig* for all two versions. The improvement rates ranged from 41.21% to 87.22%. For version 1, *Trrb-fes-4-Td* with centroid produced the best result (77.58%), while *Trrb-fes-4-Td* with MOM produced the best results for version 2 (87.22%).

Similar to *Torig*, all heuristics outperformed *Tcm*. However, the improvement rates were slightly lower than *Torig* (from 12.76% to 48.45%). For version 1, *Trrb-fes-4-Td* with centroid produced the best result (41.80%), while *Trrb-fes-4-Td* with MOM produced the best results for version 2 (48.45%).

For *Trrb*, the improvement rates ranged from -0.69% to 33.31%. For version 1, *Trrb-fes-4-Td* with centroid produced the best result (24.88%), while *Trrbf-4-Td* with MOM produced the best results for version 2 (33.31%). Only *Trrb-fes-3-Tr* with centroid did not perform better than the control technique *Trrb* in version 1.

The improvement rates for *Trrb-wsm* ranged from -4.75% to 29.95%. For version 1, *Trrb-fes-4-Td* with centroid deffuzification (29.95%) produced the best result, while *Trrb-fes-4-Td* with MOM (18.85%) produced the best results for version 2. For only one case (*Trrb-fes-3-Tr* with MOM), the heuristic did not perform better than *Trrb-wsm*.

When we compared the three and four membership functions heuristic groups, heuristics with four membership functions produced better results over three membership functions in all cases. When we compared the triangular and trapezoidal membership functions heuristic groups, heuristics with trapezoidal membership function produced better results over three triangular membership function except for one case.

4.4.5.3. Analysis of results for Capstone

The *Capstone* results (Table 4.42) show that all heuristics outperformed three of the four control techniques, *Torig*, *Trrb*, and *Trrb-wsm*, but not *Tcm*. *Trrb-fes-3-Tr* with centroid produced the best improvement rates. The improvement rates are 47.59%, 2.54%, and 2.54% over *Torig*, *Trrb*, and *Trrb-wsm*, respectively.

When we compared the two heuristic groups (three and four membership functions), the results were very similar for both groups. The triangular and trapezoidal membership functions heuristic groups also produced similar results.

4.4.5.4. Further Analysis

To visualize our results, we illustrate them in boxplots. Figures 4.13, 4.14, and 4.15 present the boxplots that show APFD values for the control techniques and heuristics for all *iTrust*, *PassowrdSafe*, and *Capstone* versions, respectively. In the boxplot figures, *Twsm* and *Trf* represent *Trrb-wsm* and *Trrb-fes*, respectively. Versions 1, 2, and 3 of *iTrust* have thirteen, twelve, and twelve data points, versions 1 and 2 of *PassowrdSafe* have fourteen and fifteen data points, and *Capstone* has twenty-three data points. Each subfigure contains boxplots for eight prioritization techniques; the first four boxplots present data for the control techniques, and the last four boxplots present the heuristic techniques. Subfigures for the top row represent heuristics with centroid defuzzification while the bottom row represents the heuristics with MOM defuzzification.

When we examine the boxplots for *iTrust*, in version 1, the results for the requirements risk–based approaches (*Trrb* and *Trrb-wsm*) and *Trrb-fes-3-Tr* show a wider distribution of data points compared to the other two versions. The results using other techniques for all versions show similar data distribution patterns. For version 3, all heuristics, *Trrb*, and *Twsm* show consistent improvement over other control techniques. When we examine the boxplots for *PassowrdSafe*, *Tcm*, *Twsm*, and *Trrb-fes-4-Tr* show a wider distribution of data points in version 1. In the case of *Capstone*, all techniques show a similar data distribution pattern. All heuristics and control techniques except *Torig* produce similar median (indicated with a line in the box) and mean (indicated with a diamond).

4.4.6. Discussion and Implications

In this section, we provide more insight about our study's findings and the possible implications for our new approach in the context of the software industry.

Our experimental results show that using requirements risks and a fuzzy expert system can improve the effectiveness of test case prioritization. A fuzzy expert system helps reduce the risk estimation process's subjectivity and addresses the problems of imprecision and uncertainty during the risk estimation process. Further, our semi-automatic approach improves the accuracy of risk estimation.

When we examined the results, we observed that the third version of *iTrust* produced the highest fault detection rates for all heuristics compared to other versions of *iTrust*. By examining the source code of *iTrust*'s third version, we found that the new requirements added to this version affected a considerable amount of source code. This modification could be a possible reason why better results were obtained for

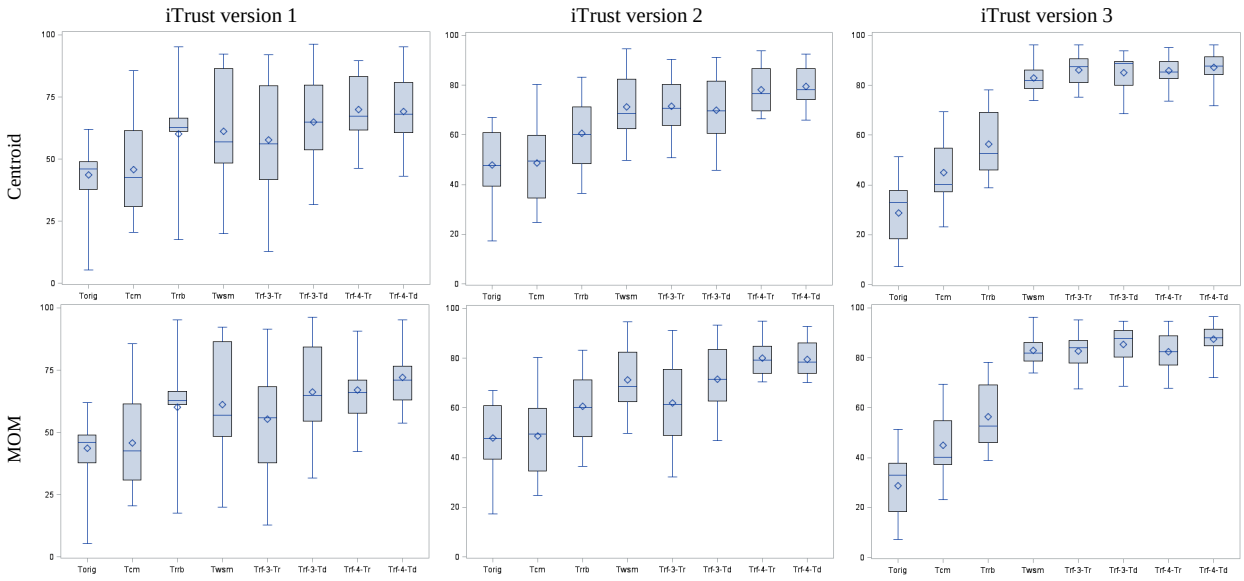

Figure 4.13. APFD boxplots for all controls and the heuristic: iTrust

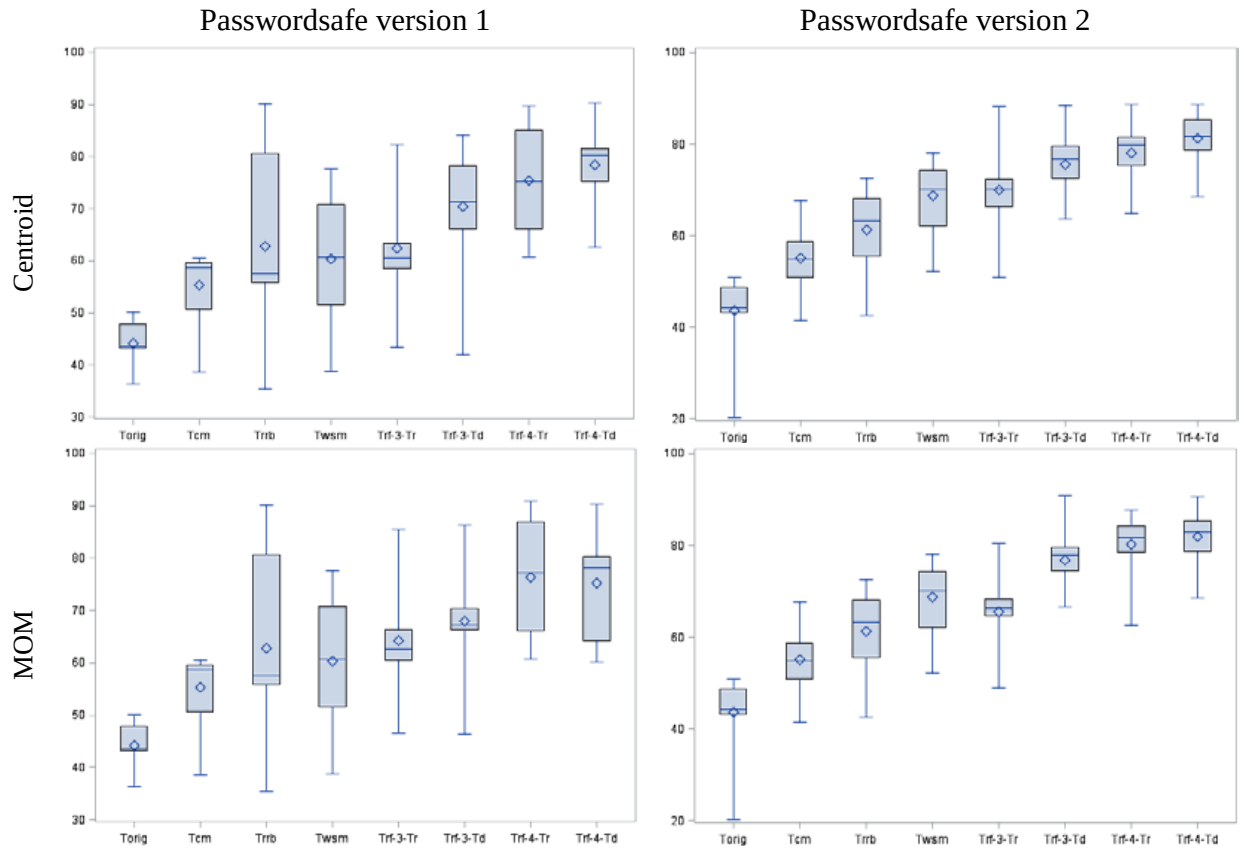

Figure 4.14. APFD boxplots for all controls and the heuristic: PassowrdSafe

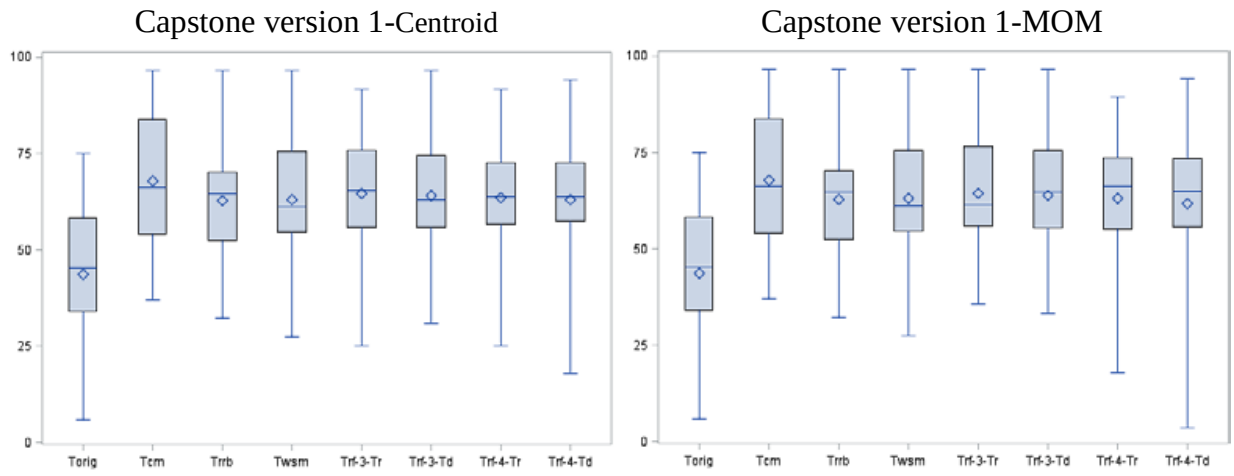

Figure 4.15. APFD boxplots for all controls and the heuristic: Capstone

this version, because the proposed approach uses risk indicators based on both the requirements and code information for risk estimation. For the *PasswordSafe* program, the second version produced better results than the first version. The requirements added to the second version had considerably changed the source code of the program and also the requirements had been updated to reflect the modifications of the source code. We believe that these reasons affected the results we obtained for *PasswordSafe* program. In the case of *Capstone*, the improvement rates for fault detection are low compared to *iTrust* and *PasswordSafe*. We speculate that *Capstone*'s less descriptive requirements affected this outcome. For example, a requirement with a simple description such as "The system shall allow admin to create exam" does not provide sufficient information about the requirement's security implications or modifications, thus making it difficult for the semi-automated process to extract risk information accurately.

This research has several important implications for the software industry. By addressing issues related to subjectivity and by reducing a lengthy risk-estimation process, our new approach can save a considerable amount of time and also lower the cost of regression testing. In this experiment, we used small and mid-size applications, but the new approach can easily be applied to larger industrial applications because the semi-automated process and the fuzzy expert system make the overall risk-estimation process simple and efficient. Further, by building knowledge about risk estimation over time, practitioners might not need to have expert knowledge about risk estimation when they apply this approach. In addition, our approach makes it easy to manage the software development process, because our approach maintains holistic relationships among requirements, test cases, and other software artifact information (e.g., requirements and source code modification information).

4.4.7. Threats to Validity

4.4.7.1. Construct Validity

The risk indicators (input variables) used for this study could affect our results. Many different requirements risk indicators are available. We considered three requirements risk indicators based on the results of our previous experiments and also by considering common risk indicators in the literature.

4.4.7.2. Internal Validity

We performed our study by considering different combinations of three and four membership functions, triangular and trapezoidal membership functions, and two different defuzzification methods. Different numbers of membership functions, shapes of membership functions, and defuzzification methods could be chosen. Changing these factors could produce different results. Therefore, the effect of different factors and combinations on the proposed approach can be further investigated by considering various factors and combinations.

Further, we obtained the weights of the risk indicators for WSM using the AHP process which is based on human judgment. To minimize subjectivity, we obtained comparison values from two experts who have years of software industry experience. Additionally, we compared our risk-indicator weight values with the weights for similar risk indicators available in the literature.

4.4.7.3. External Validity

The security keywords database developed to estimate the PSR values can limit the external validity of our results. The current set of database keywords is not representative of all security concerns. However, we tried to reduce this threat by using common threats that were identified during a thorough examination of the security literature.

4.4.8. Conclusions

In this research, we presented a new requirements risk–based test case prioritization technique that utilizes a fuzzy expert system for systematic risk estimation, and we empirically evaluated the new approach's effectiveness. The proposed approach addressed several issues such as subjectivity, uncertainty, and imprecision that may be encountered in the risk estimation process using a fuzzy expert system that is equipped with expert domain knowledge.

The experimental results obtained from both industrial and open source applications indicated that using a fuzzy expert system can improve the test case prioritization's effectiveness. The results also indicated

that the technique with the trapezoidal wave type and four membership functions produces better results than the other techniques. We believe that the semi–automated approach proposed in this research can help improve software companies' regression testing activities in terms of time and cost.

5. CONCLUSIONS AND FUTURE WORK

In this dissertation, the following steps have been performed to achieve our research goals.

- We investigated the effectiveness of requirements risk–based technique in test prioritization
- We have devised a new evaluation technique to measure test prioritization techniques' effectiveness in detecting more faults in risky components
- We expanded our previous study by addressing its limitations
- We performed an empirical study to investigate the effectiveness of requirements risk-based test prioritization on a very large industrial system
- We developed a prioritization technique using an expert system directly and shown empirically that the technique is very effective

Our results showed that the use of requirements and their risks can help improve the effectiveness of test case prioritization. The results also indicated that the proposed requirements risk-based prioritization approaches are better in detecting more faults in risky components of software systems compared to several control techniques used in this dissertation. The boxplots used in this dissertation clearly visualized the distribution of APFD values for all heuristics and controls for all versions of each object program used for our research work. In our experiments, we have not performed a formal analysis because the applications have too small number of data points to apply a formal analysis. However, in our future research, we will perform a formal statistical analysis by collecting sufficient data points.

5.1. Lessons Learned

Through this research work we identified direct relationship between requirements risks and test cases and overall our approaches improved test prioritization. However, in some cases heuristics were not better than controls. From the results of this research work, we learned that the fuzzy expert systems support not only to reduce subjectivity, imprecision, and uncertainty related issues of risk estimation process but also help to improve the effectiveness of test prioritization. Furthermore, the results suggested that our prioritization approaches are capable of producing better results if software requirements are more descriptive.

5.2. Merit and Impact of This Research

This research is expected to provide theoretical and practical benefits of requirements risk-based test prioritization to both researchers and practitioners who seek more effective and efficient test prioritization techniques in regression testing. First, this research provides guidance and comprehension of how various requirements risk-based test prioritization techniques perform on various types of software systems.

Second, this research provides empirical studies comparing requirements risk-based testing with several different types of test prioritization techniques which are prevalent in software industry. This information will be useful for software industry and researchers to adopt suitable test prioritization techniques which are more practical and easier to apply in their working environments.

5.3. Future Directions

In our future research, we will further investigate less subjective, efficient, and effective test prioritization approaches for regression testing.

We plan to propose another requirements risk-based test prioritization approach which is based on clustering techniques. In this study, we will divide the requirements into a number of clusters by considering the similarities of requirements risks which are determined through text-mining techniques. We will conduct empirical studies to evaluate the effectiveness of this technique using both industrial and open source application systems.

REFERENCES

[1] Minimum Security Requirements for Federal Information and Information Systems . http://csrc.nist.gov/publications/fips/fips200/FIPS-200-final-march.pdf, 2006.

[2] QMetry Test Management. http://www.qmetry.com, 2014.

[3] Quick Start Guide for Manual Testing using Microsoft Test Manager. https://msdn.microsoft.com/en-us/library/vstudio/dd380763%28v=vs.110%29.aspx, 2014.

[4] A. Adeli and M. Neshat. A fuzzy expert system for heart disease diagnosis. *International MultiConference of Engineers and Computer Scientists (IMECS)*, 1:1–7, 2010.

[5] M.A. Ahmed, M.O. Saliu, and J. AlGhamdi. Adaptive fuzzy logic-based framework for software development effort prediction. *Information and Software Technology*, 47(1):31–48, 2005.

[6] S. Amland. Risk based testing and metrics: Risk analysis fundamentals and metrics for software testing. In *5th International Conference EuroSTAR*, page 1–20, November 1999.

[7] S. Amland. Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study. *Journal of Systems and Software*, 53(3):287–295, 2000.

[8] M.J. Arafeen and H. Do. Test case prioritization using requirements-based clustering. *International Conference of Software Testing, Verification and Validation (ICST)*, March 2013.

[9] J. Bach. Risk and requirements-based testing. *IEEE Computer*, 32(6):113–114, 1999.

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

[11] B. Boehm. Value-based software engineering. *Software Engineering Notes*, 28(2), March 2003.

[12] B. Boehm and L. Huang. Value-based software engineering: a case study. *IEEE Computer*, 36(3):33–41, 2003.

[13] V. Carr and J.H.M. Tah. A fuzzy approach to constuction project risk assessment and analysis: construction project risk management system. *Advances in Engineering Software*, 32(10-11):847–857, 2001.

[14] Y. Chen, R.L. Probert, and D.P. Sims. Specification-based regression test selection with risk analysis. In *Proceedings of the 2002 Conference of the Centre for Advanced Studies on Collaborative research*, September 2002.

[15] R. Craig and S. Jaskiel. *Systematic Software Testing*. Artech House Publishers, Boston, MA, first edition, 2002.

[16] H. Do, S. Mirarab, L. Tahvildari, and G. Rothermel. The effects of time constraints on test case prioritization: A series of controlled experiments. *IEEE TSE*, 26(5), September 2010.

[17] H. Do and G. Rothermel. An empirical study of regression testing techniques incorporating context and lifecycle factors and improved cost-benefit models. In *Proceedings of the ACM SIGSOFT Symposium on Foundations of Software Engineering*, November 2006.

[18] H. Do and G. Rothermel. On the use of mutation faults in empirical assessments of test case prioritization techniques. *IEEE Transactions on Software Engineering*, 32(9):733–752, September 2006.

[19] S. Elbaum, A. G. Malishevsky, and G. Rothermel. Test case prioritization: A family of empirical studies. *IEEE TSE*, 28(2):159–182, February 2002.

[20] G. Erdogan, Y. Li, R. Runde, F. Seehusen, and K. Stølen. Approaches for the combined use of risk analysis and testing: a systematic literature review. *International Journal on Software Tools for Technology Transfer*, 16(5):627–642, 2014.

[21] M. Fasanghari and G.A. Montazer. Design and implementation of fuzzy expert system for tehran stock exchange portfolio recommendation. *Expert Systems with Applications*, 37(9):6138–6147, 2010.

[22] M. Felderer and R. Ramler. Integrating risk-based testing in industrial test processes. *Software Quality Journal*, 22(3):543–575, 2014.

[23] M. Felderer and I. Schieferdecker. A taxonomy of risk-based testing. *International Journal on Software Tools for Technology Transfer*, 16(5):559–568, 2014.

[24] S. Ghaisas, P. Rose, M. Daneva, K. Sikkel, and R.J. Wieringa. Generalizing by similarity: Lessons learnt from industrial case studies. In *1st International Workshop on Conducting Empirical Studies in Industry*, page 37–42, 2013.

[25] T.L. Graves. Predicting fault incidence using software change history. *IEEE TSE*, 26:653–661, July 2000.

[26] M. Hadjimichale. A fuzzy expert system for aviation risk assessment. *Expert Systems with Applications*, 3:6512–6519, 2009.

[27] V. Hajipour, A. Kazemi, and S. M. Mousavi. A fuzzy expert system to increase accuracy and precision in measurement system analysis. *Measurement*, 46(8):2770–2780, 2013.

[28] C.S. Hettiarachchi, H. Do, and B Choi. Effective regression testing using requirements and risks. In *Eighth International Conference on Software Security and Reliability*, pages 157–166, June 2014.

[29] C.S. Hettiarachchi, H. Do, and B. Choi. Risk-based test case prioritization using a fuzzy expert system. *Information and Software Technology*, 69:1–15, 2016.

[30] IEEE. *IEEE Guide to Classification for Software Anomalies*. Std 1044.1-1995. Institute of Electrical and Electronics Engineers, Inc, 1996.

[31] iTrust Wiki. http://agile.csc.ncsu.edu/iTrust/wiki/doku.php, 2008.

[32] M.A. Kadhim, M.A. Alam, and H. Kaur. Design and implementation of fuzzy expert system for back pain diagnosis. *International Journal of Innovative Technology and Creative Engineering*, 1:16–22, 2011.

[33] C. Kaner, J. Bach, and B. Pettichord. *Lessons learned in software testing: a context-driven approach*. Wiley, New York, 2002.

[34] J. Karlsson and K. Ryan. A cost-value approach for prioritizing requirements. *IEEE Software*, 14(5):67–74, 1997.

[35] M. Kazemifard, A. Zaeri, N. Ghasem-Aghaee, M.A. Nematbakhsh, and F. Mardukhi. Fuzzy emotional cocomo ii software cost estimation (fecsce) using multi-agent systems. *Applied Soft Computing*, 11(2):2260–2270, 2011.

[36] R. Krishnamoorthi and S.A. Mary. Factor oriented requirement coverage based system test case prioritization of new and regression test cases. *Information and Software Technology*, 51(4):799–808, 2009.

[37] G. Kumaran and J. Allan. Text classification and named entities for new event detection. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 297–304, July 2004.

[38] Q. Li, Y. Yang, M. Li, Q. Wang, B. W. Boehm, and C. Hu. Improving software testing process: feature prioritization to make winners of success critical stakeholders. *Journal of Software: Evolution and Process*, 24(7):783–801, 2012.

[39] M. J. Harrold and A. Orso. Retesting software during development and maintenance. In *ICSM: Frontiers of Software Maintenance*, pages 88–108, September 2008.

[40] Y.K. Malaiya and J. Denton. Requirements volatility and defect density. In *10th International Symposium on Software Reliability Engineering*, pages 285–294, November 1999.

[41] A. Malishevsky, G. Rothermel, and S. Elbaum. Modeling the cost-benefits tradeoffs for regression testing techniques. In *ICSM*, pages 204–213, October 2002.

[42] E.H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7(1):1–13, 1975.

[43] S. Mirarab and L. Tahvildari. A prioritization approach for software test cases on Baysian Networks. In *FASE*, pages 276–290, March 2007.

[44] F. Moisiadis. Prioritizing use cases and scenarios. In *37th International Conference on Technology of OO Languages and Systems*, page 108–119, 2000.

[45] E.W.T. Ngai and F.K.T. Wat. Fuzzy decision support system for risk analysis in e-commerce development. *Decision Support Systems*, 40(2):235–255, 2005.

[46] T.J. Ostrand, E.J. Weyuker, and R.M. Bell. Where the bugs are. *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis*, 29(4):86–96, 2004.

[47] X. Qu, M. Cohen, and G. Rothermel. Configuration-aware regression testing: An empirical study of sampling and prioritization. In *ISSTA*, pages 75–86, July 2008.

[48] M. Riaz, J. King, J. Slankas, and L. Williams. Hidden in plain sight: Automatically identifying security requirements from natural language artifacts. In *22nd IEEE International Conference on Requirements Engineering Conference (RE)*, pages 183–192, August 2014.

[49] G. Rothermel, R. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. *IEEE TSE*, 27(10):929–948, October 2001.

[50] G. Rothermel, R.H. Untch, C. Chu, and M. J. Harrold. Test case prioritization: An empirical study. In *IEEE International Conference on Software Maintenance*, pages 179–188, August 1999.

[51] T. L. Saaty. *The Analytic Hierarchy Process*. McGraw-Hill, 1980.

[52] N.F. Schneidewind and H.M. Hoffman. An experiment in software error data collection and analysis. *IEEE TSE*, 5(3):276–286, May 1979.

[53] A. Schwartz and H. Do. A fuzzy expert system for cost-effective regression testing strategies. In *29th IEEE International Conference on Software Maintenance (ICSM)*, pages 1–10, September 2013.

[54] M. Sherriff, M. Lake, and L. Williams. Prioritization of regression tests using singular value decomposition with empirical change records. In *ISSRE*, pages 81–90, November 2007.

[55] H. Srikanth and S. Banerjee. Controversy corner: Improving test efficiency through system test prioritization. *Journal of Systems and Software*, 85(5):1176–1187, 2012.

[56] H. Srikanth, S. Banerjee, L. Williams, and J. Osborne. Towards the prioritization of system test cases. *Software Testing, Verification and Reliability*, 24(4):320–337, 2014.

[57] H. Srikanth, C.S. Hettiarachchi, and H. Do. Requirements based test prioritization using risk factors: An industrial study. *Information and Software Technology*, 69:71–83, 2016.

[58] H. Srikanth and L. Williams. On the economics of requirements-based test case prioritization. In *EDSER*, pages 1–3, May 2005.

[59] H. Srikanth, L. Williams, and J. Osborne. System test case prioritization of new and regression test cases. In *ESE*, pages 64–73, August 2005.

[60] A. Srivastava and J. Thiagarajan. Effectively prioritizing tests in development environment. In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 97–106, July 2002.

[61] H. Stallbaum, A. Metzger, and K. Pohl. An Automated Technique for Risk-based Test Case Generation and Prioritization. In *Proceedings of the 3rd International Workshop on Automation of Software Test*, pages 67–70, May 2008.

[62] A. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos. Time-aware test suite prioritization. In *Proceedings of the International Conference on Software Testing and Analysis*, pages 1–12, July 2006.

[63] D.R. Wallace and D.R. Kuhn. Failure Modes in Medical Device Software: An Analysis of 15 Years of Recall Data. *Reliability, Quality and Safety Engineering*, 8(4):301–311, 2001.

[64] R. Wieringa and M. Daneva. Six strategies for generalizing software engineering theories. *Science of Computer Programming*, pages 136–152, 2014.

[65] Z. Xu, K Gao, and T.M. Khoshgoftaar. Application of fuzzy expert system in test case selection for system regression test. In *IEEE International Conference on Information Reuse and Integration*, pages 120–125, August 2005.

[66] S. Yoo and M. Harman. Regression testing minimization, selection and prioritisation: A survey. *JSTVR*, pages 67–120, March 2010.

[67] M. Yoon, E. Lee, M. Song, and B. Choi. A test case prioritization through correlation of requirement and risk. *Journal of Software Engineering and Applications*, 5(10):823–835, 2012.

[68] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

[69] W.M. Zage and D.M. Zage. Evaluating design metrics on large-scale software. *IEEE TSE*, 10:75–81, 1993.

[70] P. Zech. Risk–based security testing in cloud computing environments. In *Fourth IEEE International Conference on Software Testing, Verification and Validation*, pages 411–414, March 2011.