Risk-based Test Case Prioritization Using a Fuzzy Expert System

Charitha Hettiarachchi
North Dakota State U.
charitha.hettiarachc@ndsu.edu

Hyunsook Do
U. of North Texas
hyunsook.do@unt.edu

Byoungju Choi
Ewha Womans U.
bjchoi@ewha.ac.kr

August 25, 2015

Context: The use of system requirements and their risks enables software testers to identify more important test cases that can reveal the faults associated with system components.
Objective: The goal of this research is to make the requirements risk estimation process more systematic and precise by reducing subjectivity using a fuzzy expert system. Further, we provide empirical results that show that our proposed approach can improve the effectiveness of test case prioritization.
Method: In this research, we used requirements modification status, complexity, security, and size of the software requirements as risk indicators and employed a fuzzy expert system to estimate the requirements risks. Further, we employed a semi-automated process to gather the required data for our approach and to make the risk estimation process less subjective.
Results: The results of our study indicated that the prioritized tests based on our new approach can detect faults early, and also the approach can be effective at finding more faults earlier in the high-risk system components compared to the control techniques.
Conclusion: We proposed an enhanced risk-based test case prioritization approach that estimates requirements risks systematically with a fuzzy expert system. With the proposed approach, testers can detect more faults earlier than with other control techniques. Further, the proposed semi-automated, systematic approach can easily be applied to industrial applications and can help improve regression testing effectiveness.

1 Introduction

Software products change over time due to feature updates or user demand changes. When software functionalities change, software engineers need to retest the software to ensure that the changes did not affect software quality. Regression testing is one of the important maintenance activities, but it requires a great deal of time and effort. Often, software companies have pressures with time and budget, so expensive and time-consuming regression testing could be a major burden for them.

To overcome these schedule and cost-related concerns with regression testing, many researchers have proposed various cost-effective regression testing techniques [20, 33, 35]; in particular, test case prioritization techniques have been actively studied because they provide appealing benefits, such as flexibility for testers who need to adjust their testing efforts for the limited time and budget [10, 13, 42, 44]. While the majority of test case prioritization approaches utilize source code information, some researchers have investigated using other software artifacts, such as

system requirements and design documents, produced during early development phases [7, 25, 41]. For instance, Krishnamoorthi and Mary [25] proposed a system-level test case prioritization approach using the information obtained from the requirements specification, such as requirements completeness and implementation complexity. Srikanth et al. [41] also introduced a system-level test case prioritization technique that analyzes and evaluates the requirements in terms of requirement volatility, complexity, customer priority, and fault proneness.

In addition to utilizing requirements information for test case prioritization, some researchers used risk information that can help identify more important test cases that are likely to detect defects associated with the system's risks (e.g., safety or security risks) [43, 48]. The results of previous research work empirically showed that the effectiveness of test case prioritization could be improved by using requirements risks. However, these risk-based test case prioritization techniques did not consider the direct relationship between requirements risks and test cases [48] or only used one type of risk, such as fault information obtained from preceding versions [43]. Further, these studies evaluated the approaches by measuring how fast the reordered test cases detected faults; the approaches utilized risk information to prioritize tests, so they should be evaluated by measuring whether the detected faults are, indeed, from the locations where risks reside in the product. To address these limitations, our previous work [21] proposed a new requirements risk-based test case prioritization approach by considering the direct relationship between requirements risks and the test case. 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 [18]. 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. 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. Moreover, the new approach is also better at finding more faults earlier in high-risk components than other techniques.

The rest of the paper is organized as follows. Section 2 describes the fuzzy expert system used in this research and the related work. Section 3 describes our new prioritization technique in detail. Section 4 describes our experiment including the research questions. Section 5 presents the results and analysis. Section 6 discusses our results and their implications. Section 7 presents the conclusions and discusses future work.

2 Fuzzy Expert Systems and Related Work

In this section, we provide background information on the fuzzy expert system and the existing work related to test case prioritization techniques, mainly focusing on techniques that use requirements, risks, and fuzzy expert systems which are most closely related to our work.

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 [19, 46] 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 1 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 1 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.

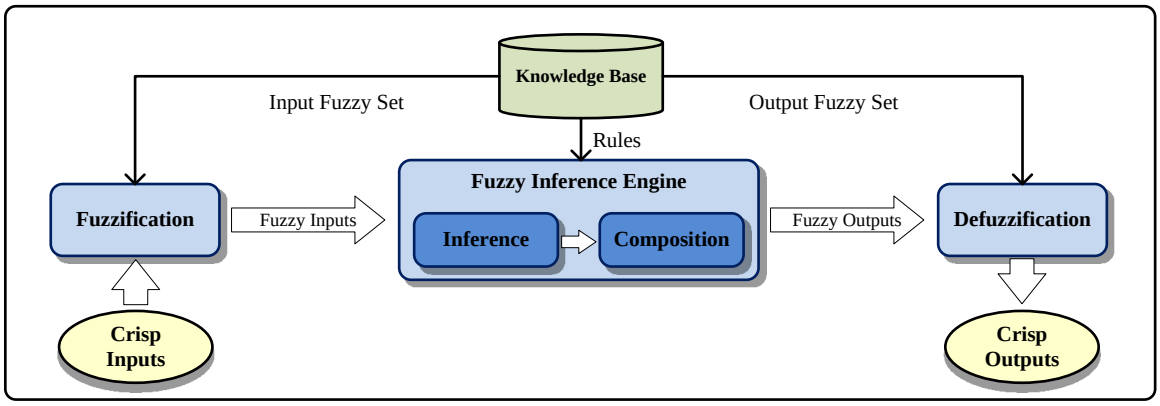

Figure 1: Architecture of a fuzzy expert system

Fuzzy Set Theory. To understand the fuzzification process, some knowledge of fuzzy set theory is necessary. Fuzzy set theory was first introduced by Zadeh [49] 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 1 defines a conventional set where $\mu_A(x)$ shows the membership of an element, x, of a conventional set, A.

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

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 2.

$$A = (x, \mu_A(x))|x \in X, \mu(x) : X \rightarrow [0, 1] \tag{2}$$

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

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 [39, 46] 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 3.

$$\mu_A(X) = \begin{cases} 0, & x < 0 \\ (x-a)/(b-a), & a \leq x \leq b \\ (c-x)/(c-b), & b \leq x \leq c \\ 0, & x > 0 \end{cases} \tag{3}$$

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

Table 1: Input variable membership functions

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

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 2.

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 3. 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 2: Output variable membership functions

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

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.

Table 3: 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 Medium and PRV is High then RMS is High

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 [30] 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 shows how to compute the COG that represents the crisp output for a particular output variable.

$$y^* = \frac{\int_a^b \mu_A(y) y dy}{\int_a^b \mu_A(y) dy} \tag{4}$$

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.

2.2 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 [47], and some of them have been used by several software organizations [28, 42]. The majority of prioritization techniques have used the information obtained from software source code [10, 36, 40]. 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 [31, 40]. Beyond code-based information, other software artifact types, such as software requirements and design information, have also been utilized. For example, Srikanth et al. [41] 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 [25] also proposed a model to prioritize test cases using the requirements specification to improve the rate of severe fault detection. Arafeen and Do [7] 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 [43, 48]. Many risk-based testing techniques have adopted Amland's [6] 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 approach, we also used the risk exposure of both requirements and risk items to prioritize tests. Stallbaum et al. [43] 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. [48] used the relationship among requirements risk exposure, risk items, and test cases to determine the order of test cases. Another paper [27] 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 [17] 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. [14] 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 [51] 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, 23], risk assessment in aviation [18], risk assessment in construction

projects [9], and selecting superior stocks on the stock exchange [15]. 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 [24], and risk analysis for e-commerce development [32]. 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 [39] 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. [46] 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 Proposed 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 2 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 and 5.

3.1 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 [6, 21, 25, 41]: 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 [16], 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.

Requirements Complexity (RC) Requirements that need complex functionalities during implementation tend to introduce more faults. A case study conducted by Amland [6] 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 shows the example for the complexity values used for our experiment.

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 [6] 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.

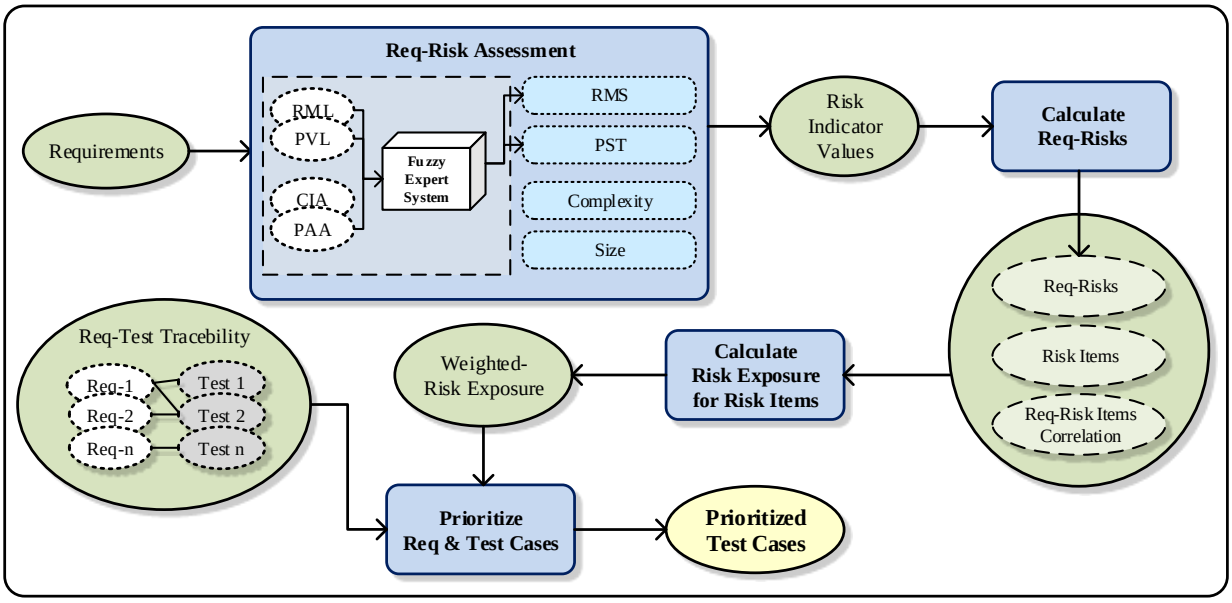

Figure 2: Overview of the risk-based approach

The last column of Table 4 shows the example for the RS values.

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 [26] 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 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 2.1.

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, 34]. 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 show the CIA and PAA values, respectively, and the seventh column shows the PST values provided by the fuzzy expert system.

Table 4: 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

3.2 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 [37]. 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 5, 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 6 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.

Equation 5 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

Table 5: 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 6: 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

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

$$RE_{(Req_j)} = \sum_{i=1}^{n} (W_i * R_{ji}) \tag{5}$$

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 7 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.

3.3 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

Table 7: 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

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. [48]. 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 [21], 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 [48]. 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 [8, 22, 45]. The identified risk items are shown in Table 8. 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 [21] 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 [48] 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 8: 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 9: Risk exposures and weighted risk exposure matrix

		Risk Items				
Requirement	RE(Req)	RiIM1 (SV1)	... ...	RiIMx (SVx)	RiIMn (SVn)	W-RE
Req_1	$\text{RE}(\text{Req}_1)$	C_{11}		C_{1x}	C_{1n}	$\text{W-RE}(\text{Req}_1)$
⋮	⋮	⋮	...	⋮	⋮	⋮
Req_y	$\text{RE}(\text{Req}_y)$	C_{y1}	...	C_{yx}	C_{yn}	$\text{W-RE}(\text{Req}_y)$
Req_m	$\text{RE}(\text{Req}_m)$	C_{m1}	...	C_{mx}	C_{mn}	$\text{W-RE}(\text{Req}_m)$
		RE(RiIM1)	...	RE(RiIMx)	RE(RiIMn)	

Table 9 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 10. 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 6; 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.

Equations 6 and 7 are used to calculate the risk exposure values for risk items and the weighted risk exposure values for requirements.

$$RE_{(RiIM_x)} = \sum_{y=1}^{m} (RE_{(Req_y)} * C_{yx}) \tag{6}$$

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.

$$W - RE_{(Req_y)} = \sum_{x=1}^{n} (RE_{(RiIMx)} * C_{yx} * SV_x) \quad (7)$$

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 10: 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

Table 11: 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 11 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 6 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 7. For instance, we obtain the W-RE value for UC1S1 as follows.

$W\text{-}RE_{(UC1S1)}$ = (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 12 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 12: 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

3.4 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 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.

4.1 Research Questions

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 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 13. 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 13: 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

4.3 Variables and Measures

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 [7]. 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 [38, 50].[1]
  - Requirements-based clustering: We consider another set of techniques proposed by Arafeen and Do [7] 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 approach, 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

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

the clusters are ordered according to requirements commitment and code modification information.

This requirements cluster-based approach [7] 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 [7] 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 3.

Dependent Variable and Measures

We considered two dependent variables:

- Average Percentage of Fault Detection (APFD):
  The APFD [12, 29] value represents the average for the percentage of fault detection during the execution of a particular test suite. APFD values range from 0 to 100. 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.
- Percentage of Total Risk Severity Weight (PTRSW):
  PTRSW is used to measure the effectiveness of test suites in terms of finding more faults for risky components in a particular system as early as possible. The PTRSW values range from 0% to 100%. Equation 8 shows how to calculate the PTRSW value. When a test suite can detect all faults for risky components in a system at a particular test case execution rate, then the PTRSW becomes 100% at that particular execution rate. A test case prioritization technique can be considered as effective in finding more faults for risky components if the test suite produced by that technique is capable of achieving a higher PTRSW value for a lower test case execution rate. For example, if the PTRSW value of a test suite produced by prioritization technique A is 100% when half of the test cases are executed (i.e., 50% test execution rate), whereas the prioritized test suite for technique B produces a 70% PTRSW value for the same 50% test execution rate, then technique A is considered the most effective technique to find more faults in the risky components.

$$PTRSW = (STRSW/GTRSW) * 100\% \tag{8}$$

The sub-total of the total risk severity weight (STRSW) and the grand total of total risk severity weight (GTRSW) are explained, in detail, in the next subsection.

4.4 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 3. 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 [7]. The second-last column in Table 13 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 [11] 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 3. 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 3. 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) [12]), 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 14. 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 14 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 18, 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.

4.5 Threats to Validity

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

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

Table 14: 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	

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.

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.

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.

5 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 6.

5.1 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 15, 16, and 17.

Table 15 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 16 shows the control techniques. CS10 and CS20 indicate the two cluster sizes. All APFD values shown in Tables 16 and 17 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 16 and 17 also show the improvement rates for our proposed approach over the control techniques.

The results in Table 16 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.

The results for *Capstone* shown in Table 17 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%.

To show our results visually, we present them in boxplots. Figure 3 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.

Table 15: Heuristic APFD: iTrust and Capstone

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

Table 16: 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

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*).

5.2 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 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*,

Table 17: 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

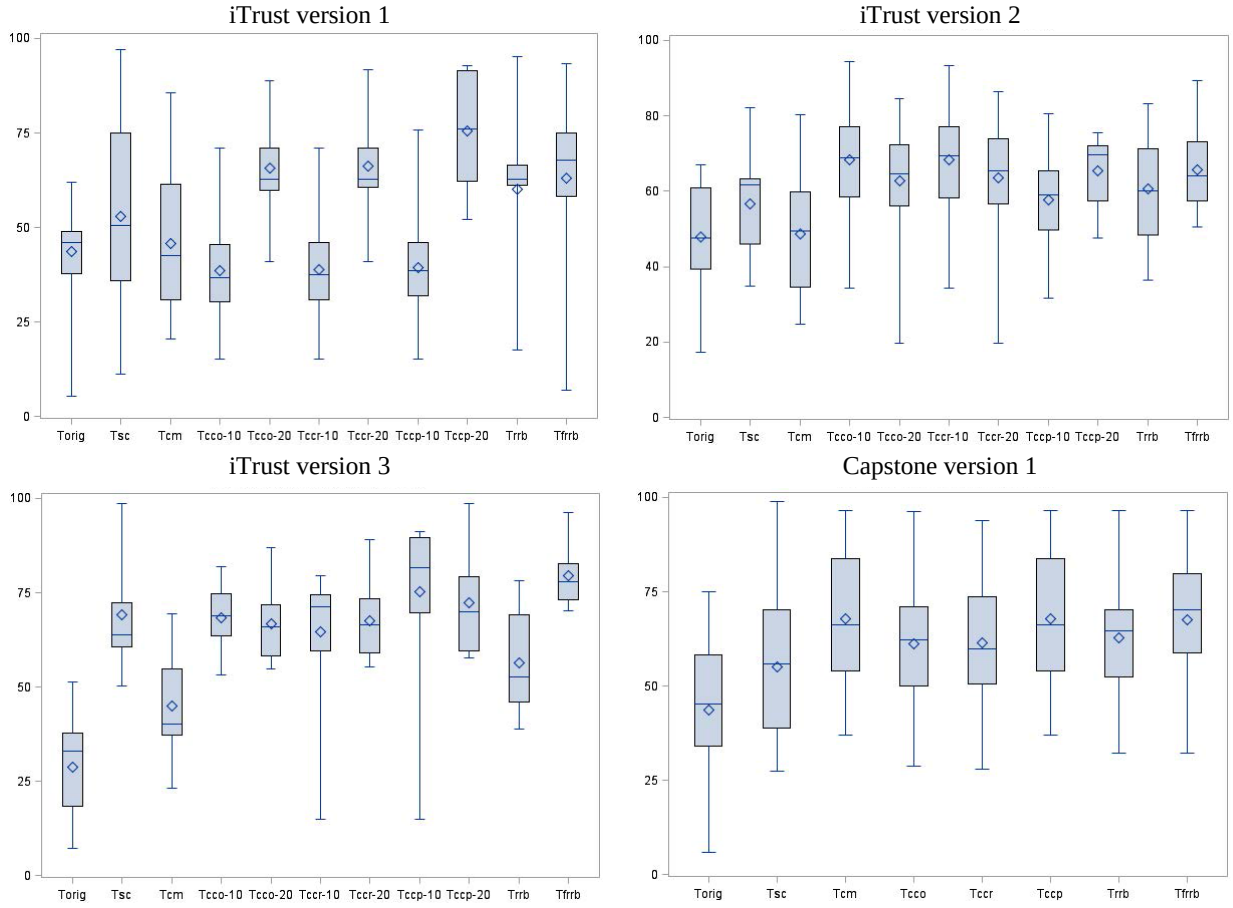

Figure 3: APFD boxplots for all controls and the heuristic: iTrust and Capstone

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 4.3, showing how fast the technique can detect faults in the risky components. Tables 18 and 19 show the PTRSW values for all versions of *iTrust* and *Capstone*, respectively. The first column of Tables 18 and 19 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.

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 19), 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.

Table 18: 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

Figure 4 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 18 and 19, 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

Table 19: 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

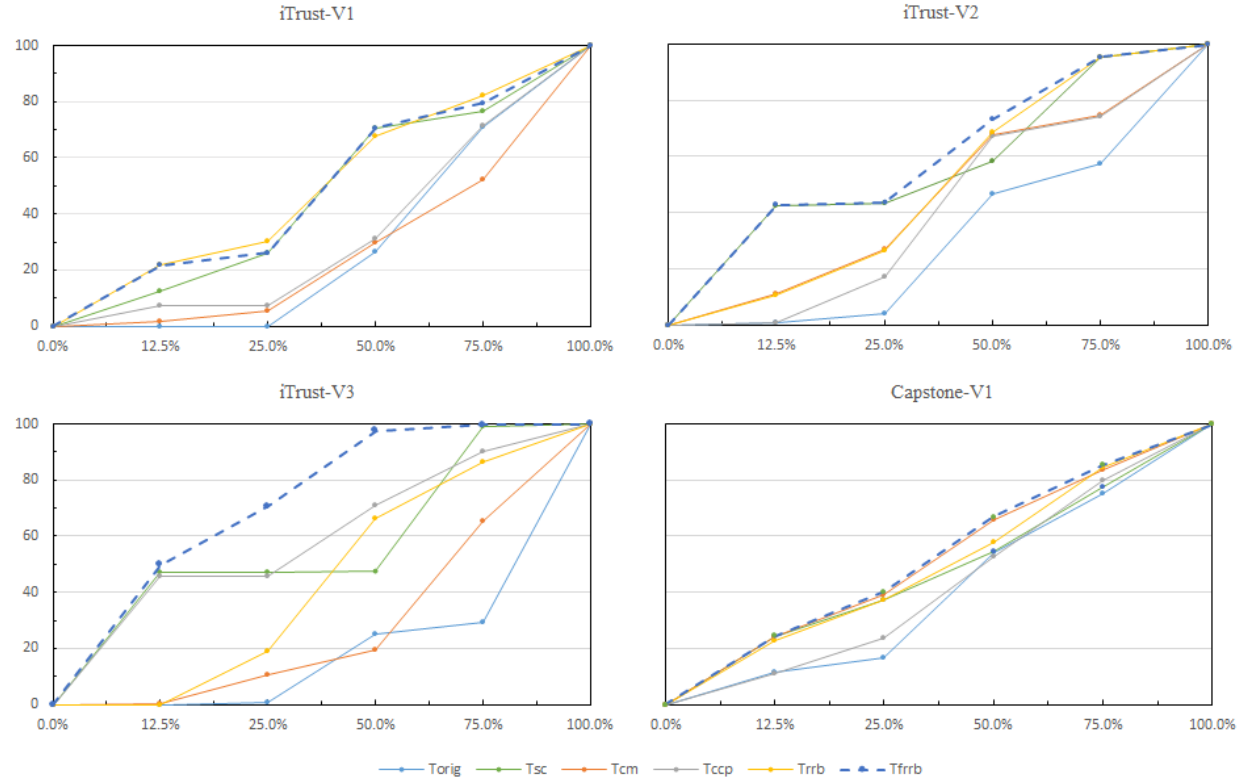

Figure 4: PTRSW comparison graphs for all versions of iTrust and Capstone

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.

6 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 [21], 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.

7 Conclusions and Future Work

In this paper, 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.

While we addressed the limitations of our previous approach, as discussed in Section 4.5, there are still some limitations that need to be overcome. For example, determining the relationships between requirements and risk items is done by human experts, but this process can be improved by reducing human involvement through semi-automated approaches (e.g., semantic analysis of natural language). Another limitation involves the choices of membership functions and the defuzzification method. We considered three triangular membership functions for both the input and output variables and used the centroid defuzzification method to defuzzify output variables, but the choice of these functions and methods could affect our results. Therefore, we plan to conduct more studies by considering different wave types (i.e., trapezoidal, gaussian, etc.), different numbers of membership functions, and different defuzzification methods to see how these variations affect the requirements risk estimation and final results. Further, in this work, we only used four risk indicators, but we plan to use other risk indicators, such as usage rate. We also plan to investigate the use of other fuzzy technologies, such as the fuzzy clustering approach, to improve risk-based regression testing approaches, and plan to evaluate these approaches considering their efficiency and effectiveness.

Acknowledgments

This work was supported, in part, by NSF CAREER Award CCF-1149389 to University of North Texas. This work was also funded by the MSIP (Ministry of Science, ICT, and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2014-H0301-14-1023) supervised by NIPA (National IT Industry Promotion Agency)

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/dd3807632014.

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

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

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] G. Erdogan, Y. Li, R. Runde, F. Seehusen, and K. Stlen. 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.

[15] 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.

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

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

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

[19] 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.

[20] M. J. Harrold, D. Rosenblum, G. Rothermel, and E. Weyuker. Empirical studies of a prediction model for regression test selection. *IEEE TSE*, 27(3):248–263, March 2001.

[21] 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.

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

[23] 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.

[24] 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):22602270, 2011.

[25] R. Krishnamoorthi and S.A. Sahaaya Arul 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.

[26] 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.

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

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

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

[30] 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.

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

[32] 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.

[33] J. Offutt, J. Pan, and J. M. Voas. Procedures for reducing the size of coverage-based test sets. In *Proc. Int'l. Conf. Testing Comp. Softw.*, pages 111–123, June 1995.

[34] 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.

[35] G. Rothermel and M. J. Harrold. Analyzing regression test selection techniques. *IEEE TSE*, 22(8):529–551, August 1996.

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

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

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

[39] 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.

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

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

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

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

[48] 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.

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

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

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