2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)

A Systematic Requirements and Risks-Based Test Case Prioritization Using a Fuzzy Expert System

Charitha Hettiarachchi
School of Computer Science and Information Systems
Northwest Missouri State University
Maryville, USA
charitha@nwmissouri.edu

Hyunsook Do
Department of Computer Science and Engineering
University of North Texas
Denton, USA
hyunsook.do@unt.edu

***Abstract*—The use of risk information can help software engineers identify software components that are likely vulnerable or require extra attention when testing. Some studies have shown that the requirements risk-based approaches can be effective in improving the effectiveness of regression testing techniques. However, the risk estimation processes used in such approaches can be subjective, time-consuming, and costly. In this research, we introduce a fuzzy expert system that emulates human thinking to address the subjectivity related issues in the risk estimation process in a systematic and an efficient way and thus further improve the effectiveness of test case prioritization. Further, the required data for our approach was gathered by employing a semi-automated process that made the risk estimation process less subjective. The empirical results indicate that the new prioritization approach can improve the rate of fault detection over several existing test case prioritization techniques, while reducing threats to subjective risk estimation.**

***Keywords*—Regression testing, requirements risks-based testing, fuzzy expert systems, test case prioritization, software requirements, empirical study**

I. Introduction

As software systems evolve, numerous changes are inevitable. For instance, customer requirements for a system may change, and thus previously validated system components might need to be modified, or new functionalities might need to be added. To ensure the quality of the modified system, regression testing should be performed. However, the cost of performing regression testing can be expensive, in particular, for large and complex systems.

In order to make the regression testing process more efficient and to reduce the cost of regression testing, several techniques, such as test case prioritization, test case selection, and test case reduction [1], [2], [3], have been proposed. In particular, test case prioritization techniques have been proposed and studied by many researchers due to their practical benefits. For most of these techniques, source-code information has been the major software artifact used for implementation; the software products are developed based on the product requirements, so the requirements information could help provide a better understanding about the source of errors. Any defect in the requirements could be potential defects in the implemented system, and by identifying and utilizing requirements risks that could introduce defects into the system, software testers can better manage their activities with a better understanding of the nature of such defects.

The importance of using requirements information while testing has been well recognized by the requirements engineering community [4], [5], and some researchers have begun to use requirements information for regression testing. For instance, Krishnamoorthi and Mary [6] suggested that using information in the requirements specifications such as customer-assigned priority can improve system-level test case prioritization. Srikanth et al. [7] introduced a system-level test case prioritization approach that uses requirements-related factors such as requirements volatility and complexity. In addition to utilizing requirements information, other researchers have utilized information about the risks that reside in the requirements. For instance, Chen et al. [8] proposed a specification-based method for regression testing that analyzes requirements risks for test selection. Yoon et al. [9] used the relationship among requirements risk exposure, risk items, and test cases to evaluate prioritized test cases.

While these 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, they contain some limitations, such as considering a limited number of risk types and not utilizing a direct relationship between requirements risks. In our previous research [10], we proposed a new requirements risk-based test case prioritization approach that considers a direct relationship among requirements, several risk items, and test cases. The experimental results are promising, but our previous approach requires human expert involvement to estimate requirements risks including identifying risk indicators and risk items and this process can be subjective, expensive, and time-consuming. Furthermore, we presented a requirements risk-based test case prioritization approach that uses a fuzzy expert system to make risk estimation process systematic and efficient while using the expert knowledge [11]. The empirical results indicate that the prioritized tests, based on this approach can detect faults early, compared to several test case prioritization techniques. However, this new approach consists of many steps and the fuzzy expert system contributes only to assess two risk indicators which are used to estimate risk of requirements.

978-1-7281-3927-2/19/$31.00 ©2019 IEEE
DOI 10.1109/QRS.2019.00054

To address these limitations, we employed a fuzzy expert system (FES) to make our risk estimation process more systematic, efficient, and cost-effective in using human expert knowledge. Compared to previous FES-based approach [11], this new approach reduces the number of steps required to prioritize tests and also reduces the number of risk indicators. In addition, it uses FES to directly estimate the risks residing in the requirements rather than estimating risks of risk indicators, and the new approach doesn't require risk items in the risk estimation process. To date, researchers in various areas have applied fuzzy expert systems to help solve complex decision-making problems. For example, Adeli and Neshat [12] 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. [14] 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 technique 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 [10].

The remainder of the paper is arranged as follows: Section II describes our new prioritization technique in detail. Section III describes our experiment, including the research question. Section IV presents the results and analysis. Section V discusses our results and their implications. Section VI describes the related work, and Section VII presents conclusions and discusses possible future work.

II. Methodology

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

Fig. 1 gives an overview of the proposed technique. The steps shown in the light-blue boxes represent the main steps of the proposed approach. The items in the ovals show inputs and outputs of each step. 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.

A. Estimate Requirements Risks Using FES

In this subsection, we describe how we used FES to estimate the software requirements' risks. Human experts make decisions by using input values and their knowledge. FES can emulate human expert knowledge and the decision-making process. In the first step (fuzzification), crisp input values are transformed into a fuzzy input set. In the second step (inference), FES uses fuzzy inputs to determine the fuzzy output by means of fuzzy rules and membership functions defined in a knowledge base. In the third process (composition), all output fuzzy sets are aggregated into a single fuzzy set. The last process (defuzzification) produces the crisp outputs. The remainder of this subsection describes these four steps of FES in detail. In general, FES involves four main steps: fuzzification, fuzzy inference, composition, and defuzzification.

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 [4], [15], [16], 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.

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.

Requirements Complexity (RC): We consider the requirements' complexity as the second risk indicator. According to a study conducted by Amland and Garborgsv [17], 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

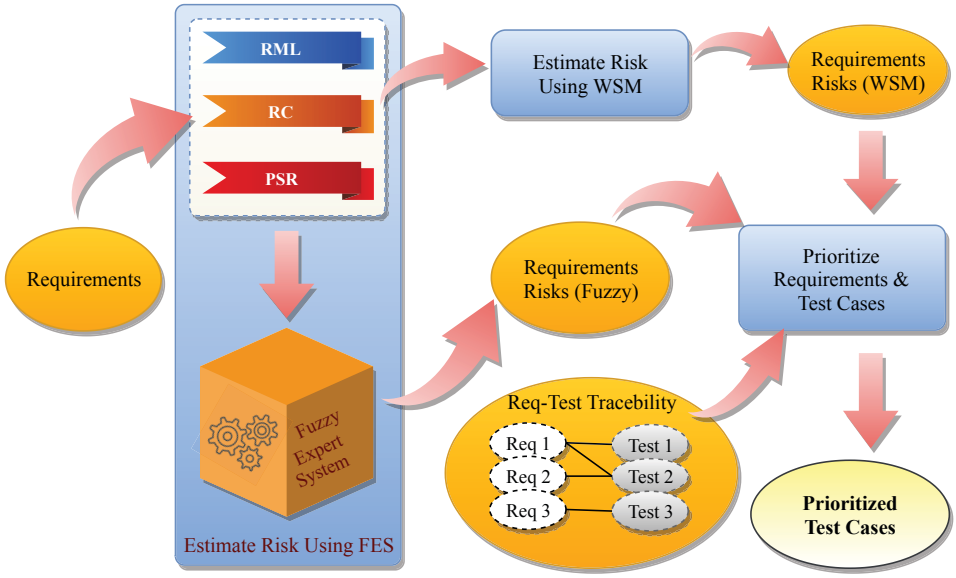

Fig. 1. Overview of Requirements Risk-Based Approach

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.

Potential Security Risks (PSR): Today, security of software plays a very important role in software applications because of numerous malicious activities 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 the term extraction tool. The database consists of six security objectives that are identified in the software security field: confidentiality, integrity, availability, privacy, authentication, and accountability. For each security objective, a set of security keywords has been identified and placed in the database. The keywords table can be updated according to the application domain for better estimation.

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 I 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 I and II. 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 I and Table II, the parameter values are the same for the input and output membership functions.

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 [18]. 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

TABLE I
INPUT AND OUTPUT VARIABLES FOR THREE MEMBERSHIP FUNCTIONS

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

TABLE II
INPUT AND OUTPUT VARIABLES FOR FOUR MEMBERSHIP FUNCTIONS

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

the rule's determined degree of truth and derives the output membership function.

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 III illustrates the input and output variable values for each requirement. The first column of Table III 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.

TABLE III
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

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 results. Several studies have shown that requirements modifications can result in a defective system [7], [19]. The declaration of the fuzzy rules for our FES is based on this knowledge, and we give high priority to requirements modification, medium priority to requirements 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 [15], [19], [20].

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 IV.

TABLE IV
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

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, 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.

B. 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 1 as a secondary prioritization criterion.

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

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

The three risk indicators described in the previous step were used to estimate the requirements' risks in terms of WSM. To determine the weight for each indicator, we employed the analytic hierarchy process (AHP), which is based on a pairwise comparison of risk indicators [21], [22]. Two experts who have several years of experience in the software industry performed the comparison. We averaged the priority vector (PV) values obtained from each expert to get the final PV values for each indicator. In Table V, 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 VI, the risk indicators are shown in the first column, PV values obtained from two experts are shown in the second and third columns, and the averaged PV values are shown in the fourth column. 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 [15], and we observe that similar risk indicators have almost the same values.

TABLE V
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 VI
RISK INDICATORS AND WEIGHTS

Indicator	PV1	PV2	Average	Weight
Requirements Modification Level	0.47	0.54	0.51	5
Requirements 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 1 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 VII 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 VII
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

C. 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.

III. 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?

A. Objects of Analysis

In order to evaluate our new approach, we utilize three student-authored applications: two open source applications and one capstone project. The *iTrust* [23] program is the open source application used for this experiment. The Realsearch Research Group at North Carolina State University developed this patient centric, electronic health-record system. We used four versions of the *iTrust* system (versions 0, 1, 2, and 3) in this experiment. 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 VIII. 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.

TABLE VIII
EXPERIMENT OBJECT AND ASSOCIATED DATA

Object	Ver.	Size (KLOCs)	Req.	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

B. Variables and Measures

Independent Variable: In this experiment, we manipulate the test case prioritization technique as the independent variable. 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 [24]. 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). These metrics are considered good predictors for discovering software components which tend toward errors [25], [26].[1]
  - 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 [10].
  - 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 in terms of three weighted risk indicators, and the overall risk for that requirement is calculated by employing the weighted sum model. Section II 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 II.

Dependent Variable and Measures: 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 [27].

To understand the APFD measure, consider the following example with five test cases and ten faults, as shown in Fig. 2. When the test cases are executed in the order E-C-B-A-D, 30% of the faults can be detected by executing test case E (i.e., 20% of the tests). For the same order, 100% of the faults can be detected after executing test case C (i.e., 40% of the tests). On the other hand, the C-E-B-A-D order can detect 70% of the faults with only 20% of the test cases and 100% of the faults with only 40% of the tests. The area covered by the solid line represents the APFD value. A higher APFD value means higher fault detection. In this example, the APFD values of the C-E-B-A-D and E-C-B-A-D test orders are 84% and 76%, respectively. Therefore, C-E-B-A-D is a better test order than E-C-B-A-D.

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

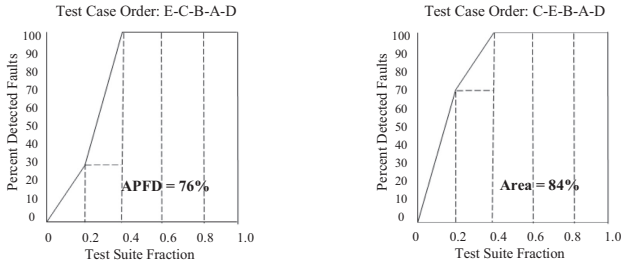

Fig. 2. APFD Example (Rothermel et al.)

C. Experimental Setup and Procedure

For each requirement in this study, we estimated the risks using a fuzzy expert system as explained in Section II. We used three input variables (RML, RC, and PSR) for FES to obtain

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

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 II. 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 integrated development environment (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 [24]. 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).

D. Threats to Validity

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.

Human expertise is required for setting the fuzzy rules for this approach. The results could be affected by human expertise because human knowledge and experience are subjective factors. This threat can be reduced by using experts who have application domain knowledge and experience.

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.

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.

The applications used in this research are industrial (*Capstone*) and open source applications (*PasswordSafe* & *iTrust*). The size of these three real-world object programs range from small to medium. Therefore, applications in the same context may expect similar results. However, evaluating this approach within more applications, different contexts, and larger applications is required to better understand the effectiveness of the approach and to generalize our findings.

IV. Data and Analysis

In this section, we present and analyze our study's results. We summarize the data in Table IX. 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 IX shows the results for the *iTrust*, *PasswordSafe*, and *Capstone* applications. In Table IX, the first column shows the object programs, the second column shows version numbers, columns three to six show the APFD values for the control

TABLE IX
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

techniques, and the rest of the columns show the heuristics' APFD values.

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 (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 cases. 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.

Analysis of results for PasswordSafe: The results for *PasswordSafe* (Table IX) 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.

Analysis of results for Capstone: The *Capstone* results (Table IX) 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.

Further Analysis: To visualize our results, we illustrate them in boxplots. Fig. 3, 4, and 5 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).

V. 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 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*. Code metric (Tcm) control technique outperformed the heuristics of proposed approach. 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. The FES used in this research and the semi-automated process estimated the risks residing in the requirements without major human involvement. Compared to the previous approach [11], this new approach simplified the risk estimation process by removing a major step, reduced the number of risk indicators, and eliminated the risk items from the risk estimation process. These changes can reduce human effort by approximately 40% to 50% and will also lead to almost the same percentage of cost and time savings. In this experiment, we used small and mid-sized 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).

VI. Related Work

In this section, we discuss prior test case prioritization related work performed by other researchers. We mainly consider existing work focus on techniques using requirements, risks, and fuzzy expert systems. Further, we discuss the related work that is relevant when using fuzzy expert systems for software engineering.

To date, the majority of test case prioritization techniques have utilized source code information [28], [29], [30], but recently some researchers have investigated the use of requirements information. For example, Srikanth et al. [7] proposed a system-level, value-driven test case prioritization approach to improve the detection rate for severe failures by using four requirement-based prioritization factors: requirements volatility, implementation complexity, customer priority, and fault proneness. Krishnamoorthi and Mary [6] also proposed a requirement coverage-based technique that prioritizes system-level test cases and improves the effectiveness of test case prioritization. Arafeen and Do [24] proposed a requirements clustering approach that divides the requirements into a num-

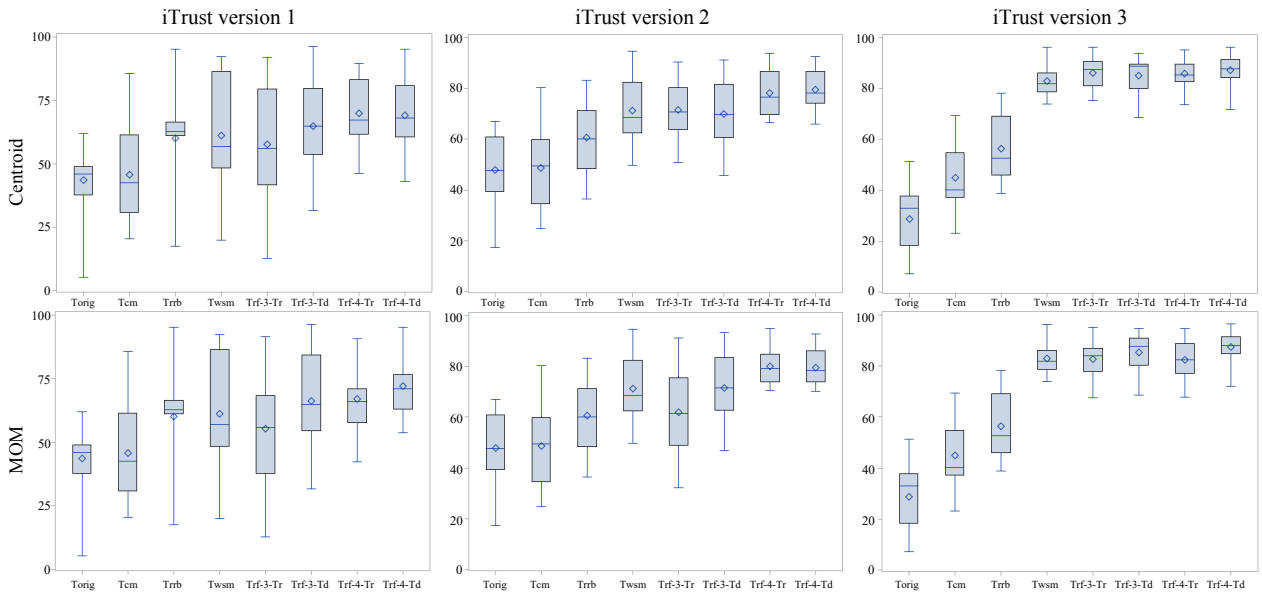

Fig. 3. APFD boxplots for all controls and the heuristic: iTrust

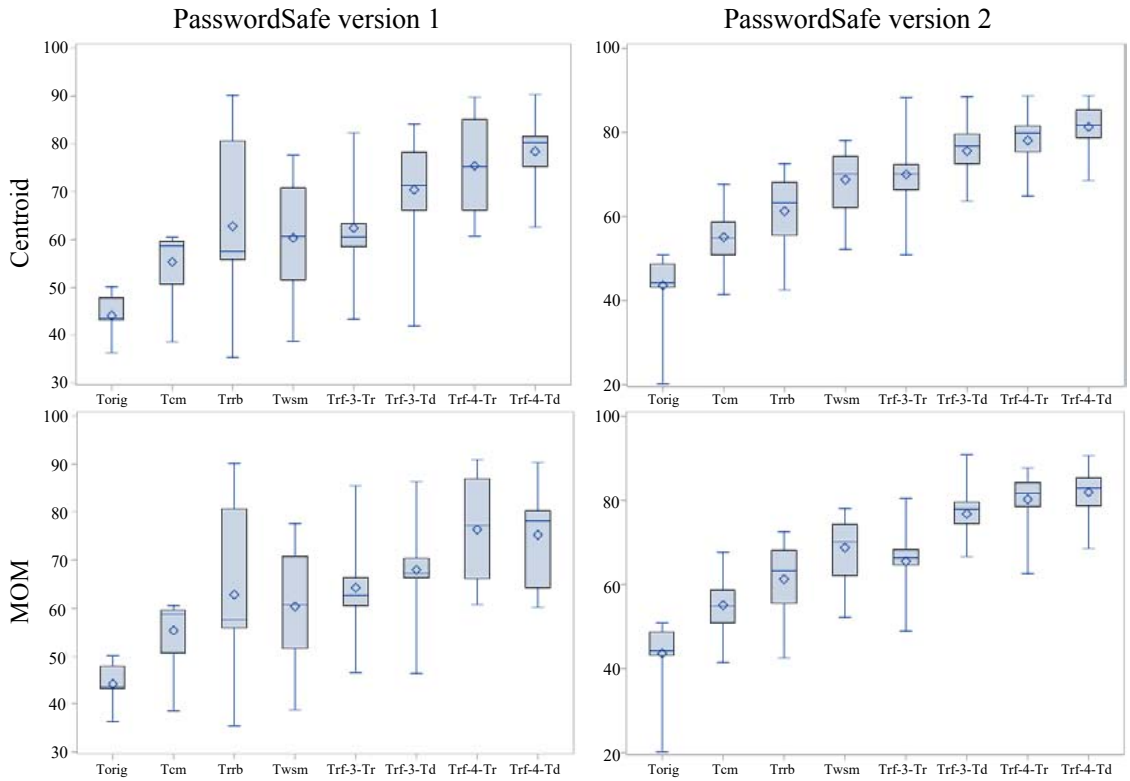

Fig. 4. APFD boxplots for all controls and the heuristic: PassowrdSafe

ber of clusters by considering the requirements' similarity, which is determined through a text-mining technique, and the authors prioritized the test cases utilizing the relationship between the requirement and test cases.

In addition to utilizing requirement information, other researchers investigated how to use the risks in software systems to improve the testing process [31], [32], [33]. Stallbaum and Metzger [34] introduced a new, automated risk-assessment approach that used requirement metrics and illustrated the approach's benefit for risk-based and requirements-based testing.

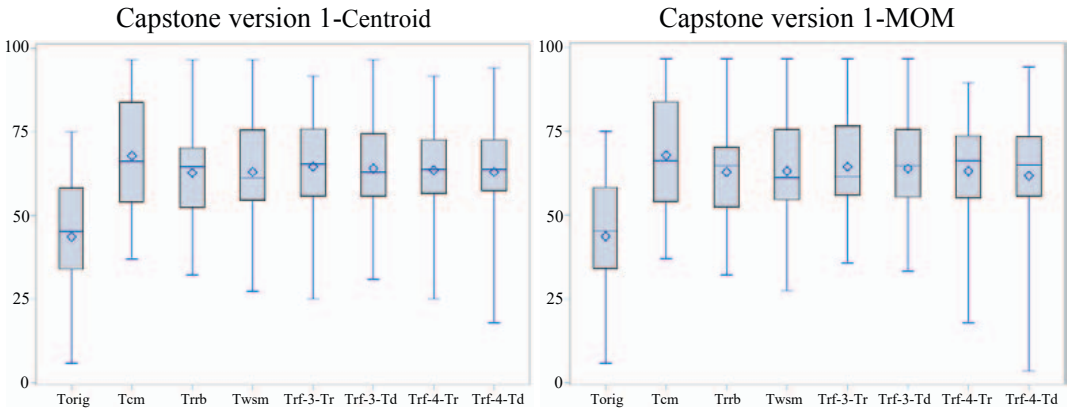

Fig. 5. APFD boxplots for all controls and the heuristic: Capstone

Li et al. [35] proposed a value-based software engineering framework to make the software testing process effective and efficient by considering the quality-related risk of new system features as one of the main prioritization factors. Further, several other researchers investigated how software systems' risks can be utilized to prioritize the test cases and to improve the fault detection rate [8], [9], [36]. For instance, Stallbaum et al. [36] proposed a technique called RiteDAP (risk-based test case derivation and prioritization) that can automatically generate test case scenarios from activity diagrams and prioritize test cases using the estimated risks in the system activities. Chen et al. [8] presented a specification-based regression testing approach that prioritizes test cases based on the risk-exposure values obtained through risk analysis with fault-history information. Also, some researchers considered the risks along with the requirements in the field of test case prioritization. Yoon et al. [9] used the relationship among requirements risk exposure, risk items, and test cases to determine the order for the test cases.

Fuzzy expert systems related research is another area pertinent to our study. Fuzzy expert systems have been utilized in many different areas, such as medical diagnosis [12], [37], [38], risk assessment [39], and system controlling [40], to address uncertainties, imprecision, and subjectivity with the decision-making process. For example, Neshat et al. [38] designed a fuzzy expert system to diagnose liver disorders. Carr and Tah [13] proposed a technique that uses fuzzy expert systems to assess the risks for construction projects. In recent times, the software engineering field has used fuzzy expert systems in many different applications, such as software-effort prediction [41], software cost estimation [42] and a risk analysis of e-commerce development [43]. For instance, Ahmed et al. [41] proposed an approach to obtain accurate cost and schedule estimation by handling uncertainty and imprecision issues that may occur during the early stages of software development; their approach was aided by a fuzzy expert system. More recently, fuzzy expert systems were applied to the regression testing area. For instance, Schwartz and Do [44] developed a fuzzy expert system to improve a multiple-criteria decision-making process that tries to identify the most cost-effective regression testing technique for a particular software version by addressing several limitations of the existing, adaptive regression testing strategies. Hettiarachchi and Do [11] introduced a requirements risk-based test case prioritization approach using a fuzzy expert system. This approach used the fuzzy expert system only to perform the risk assessment of two risk indicators. Xu et al. [45] proposed a fuzzy expert system to select test cases effectively while solving the inaccuracy and subjectivity related to the test selection process. In this work, we used a fuzzy expert system to systematically estimate requirements risks and to improve the test case prioritization process.

VII. Conclusions and Future Work

In this paper, 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. We believe that the semi-automated approach used in this research can help improve software companies' regression testing activities in terms of time and cost.

The results indicated that the technique with the trapezoidal wave type and four membership functions produces better results than the other techniques. Therefore, our future work will further explore whether we can observe a similar trend by applying our approach to a wide variety of applications. We will also investigate the effects of different types of fuzzy expert systems (e.g., type-II), different wave types, and different numbers of membership functions on requirements risk-based regression testing. In this research, we consider three risk indicators: RML, RC, and PSR. However, more domain-

specific risk indicators may produce better risk estimations and thus yield better results. We will also consider the use of different linear and nonlinear risk indicators, which could possibly reveal more risks that reside in the requirements, further improving risk-based testing.

Acknowledgment

This work was supported in part by NSF CAREER Award CCF-1564238 to the University of North Texas.

REFERENCES

[1] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE TSE*, vol. 28, no. 2, pp. 159–182, Feb. 2002.

[2] A. Srivastava and J. Thiagarajan, " Effectively prioritizing tests in development environment," in *Proceedings of the International Symposium on Software Testing and Analysis*, Jul. 2002, pp. 97–106.

[3] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritisation: A survey," *JSTVR*, vol. 22, no. 2, pp. 67–120, Mar. 2012.

[4] J. Bach, "Risk and requirements-based testing," *IEEE Computer*, vol. 32, no. 6, pp. 113–114, 1999.

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

[6] R. Krishnamoorthi and S. Mary, "Factor oriented requirement coverage based system test case prioritization of new and regression test cases," *Information and Software Technology*, vol. 51, no. 4, pp. 799–808, 2009.

[7] H. Srikanth, S. Banerjee, L. Williams, and J. Osborne, "Towards the prioritization of system test cases," *Software Testing, Verification and Reliability*, vol. 24, no. 4, pp. 320–337, 2013.

[8] Y. Chen, R. Probert, and D. Sims, "Specification-based regression test selection with risk analysis," in *Proceedings of the 2002 Conference of the Centre for Advanced Studies on Collaborative research*, Sep. 2002, pp. 1–14.

[9] 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*, vol. 5, no. 10, pp. 823–835, 2012.

[10] C. Hettiarachchi, H. Do, and B. Choi, "Effective regression testing using requirements and risks," in *IEEE Eighth International Conference on Software Security and Reliability*, Jun. 2014, pp. 157–166.

[11] C. Hettiarachchi, H. Do, and B. Choi, "Risk-based test case prioritization using a fuzzy expert system," *Journal of Information and Software Technology*, vol. 69, pp. 1–15, 2016.

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

[13] V. Carr and J. Tah, "A fuzzy approach to constuction project risk assessment and analysis: construction project risk management system," *Advances in Engineering Software*, vol. 32, no. 10, pp. 847–857, 2001.

[14] Z. Xu, K. Gao, and T. Khoshgoftaar, "Application of fuzzy expert system in test case selection for system regression test," in *IEEE International Conference on Information Reuse and Integration*, Aug. 2005, pp. 120–125.

[15] S. Amland, "Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study," *Journal of Systems and Software*, vol. 53, no. 3, pp. 287–295, 2000.

[16] D. Wallace and D. Kuhn, "Failure Modes in Medical Device Software: An Analysis of 15 Years of Recall Data," *Reliability, Quality and Safety Engineering*, vol. 8, no. 4, pp. 301–311, 2001.

[17] S. Amland and H. Garborgsv, "Risk based testing and metrics," *5th International Conference EuroSTAR '99*, pp. 1–20, 1999.

[18] E. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Man-Machine Studies*, vol. 7, no. 1, pp. 1–13, 1975.

[19] H. Srikanth, L. Williams, and J. Osborne, "System test case prioritization of new and regression test cases," in *ESE*, Aug. 2005, pp. 64–73.

[20] Y. Malaiya and J. Denton, "Requirements volatility and defect density," in *10th International Symposium on Software Reliability Engineering*, Nov. 1999, pp. 285–294.

[21] T. L. Saaty, "Decision making with the analytic hierarchy process," *International Journal of Services Sciences*, vol. 1, no. 1, pp. 83–98, 2008.

[22] T. L. Saaty, *The Analytic Hierarchy Process*. New York, NY: McGraw-Hill, 1980.

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

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

[25] N. Schneidewind and H.-M. Hoffman, "An experiment in software error data collection and analysis," *IEEE TSE*, vol. 5, no. 3, pp. 276–286, May 1979.

[26] W. Zage and D. Zage, "Evaluating design metrics on large-scale software," *IEEE TSE*, vol. 10, pp. 75–81, 1993.

[27] A. Malishevsky, G. Rothermel, and S. Elbaum, "Modeling the cost-benefits tradeoffs for regression testing techniques," in *ICSM*, Oct. 2002, pp. 204–213.

[28] 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*, vol. 26, no. 5, pp. 593–617, Sep. 2010.

[29] G. Rothermel, R. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE TSE*, vol. 27, no. 10, pp. 929–948, Oct. 2001.

[30] M. Sherriff, M. Lake, and L. Williams, "Prioritization of regression tests using singular value decomposition with empirical change records," in *ISSRE*, Nov. 2007, pp. 81–90.

[31] 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*, vol. 16, no. 5, pp. 627–642, 2014.

[32] M. Felderer and I. Schieferdecker, "A taxonomy of risk-based testing," *International Journal on Software Tools for Technology Transfer*, vol. 16, no. 5, pp. 559–568, 2014.

[33] M. Felderer, C. Haisjackl, V. Pekar, and R. Breu, "A risk assessment framework for software testing," in *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications*, 2014, pp. 292–308.

[34] H. Stallbaum and A. Metzger, "Employing requirements metrics for automating early risk assessment," in *Proceedings of MeReP07*, 2007, pp. 1–12.

[35] Q. Li, Y. Yang, M. Li, Q. Wang, B. Boehm, and C. Hu, "Improving software testing process: feature prioritization to make winners of success-critical stakeholders," *Journal of Software: Evolution and Process*, vol. 24, no. 7, pp. 783–801, 2012.

[36] 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*, May 2008, pp. 67–70.

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

[38] M. Neshat, M. Yaghobi, M. Naghibi, and A. Esmaelzadeh, "Fuzzy expert system design for diagnosis of liver disorders," *International Symposium on Knowledge Acquisition and Modeling, IEEE* , pp. 252–256, 2008.

[39] M. Hadjimichael, "A fuzzy expert system for aviation risk assessment," *Expert Systems with Applications: An International Journal*, vol. 36, no. 3, pp. 6512–6519, 2009.

[40] S. Soyguder and H. Alli, "Predicting of fan speed for energy saving in hvac system based on adaptive network based fuzzy inference system," *Expert Systems with Applications*, vol. 36, no. 4, pp. 8631–8638, 2009.

[41] M. Ahmed, M. Saliu, and J. AlGhamdi, "Adaptive fuzzy logic-based framework for software development effort prediction," *Information and Software Technology*, vol. 47, no. 1, pp. 31–48, 2005.

[42] M. Kazemifard, A. Zaeri, N. Ghasem-Aghaee, M. Nematbakhsh, and F. Mardukhi, "Fuzzy emotional cocomo ii software cost estimation (fecsce) using multi-agent systems," *Applied Soft Computing*, vol. 11, no. 2, pp. 2260–2270, 2011.

[43] E. Ngai and F. Wat, "Fuzzy decision support system for risk analysis in e-commerce development," *Decision Support Systems*, vol. 40, no. 2, pp. 235–255, 2005.

[44] A. Schwartz and H. Do, "A fuzzy expert system for cost-effective regression testing strategies," in *29th IEEE International Conference on Software Maintenance (ICSM)*, Sep. 2013, pp. 1–10.

[45] Z. Xu, K. Gao, T. Khoshgoftaar, and N. Seliya, "System regression test planning with a fuzzy expert system," *Information Sciences*, vol. 259, pp. 532–543, 2014.