Arabian Journal for Science and Engineering (2020) 45:6125–6138
https://doi.org/10.1007/s13369-020-04472-z

RESEARCH ARTICLE-COMPUTER ENGINEERING AND COMPUTER SCIENCE

Risk-Based Test Case Prioritization by Correlating System Methods and Their Associated Risks

Hosney Jahan[1] · Ziliang Feng[1] · S. M. Hasan Mahmud[2]

Received: 22 July 2019 / Accepted: 17 March 2020 / Published online: 2 April 2020

Abstract
Regression testing aims to ensure the quality of a software after modification. However, re-executing the entire test suite in regression testing is a time-consuming, costly, and tedious process that often requires additional budget and time. Thus, limited resources always result in early termination and poor quality software. Test case prioritization aims to improve regression testing by re-scheduling the test cases in a manner that could increase the fault detection rate. Risk-based testing has gained popularity in the area of software testing. However, most of the existing methods compute risk values manually, which makes these methods tiresome, laborious, and slow. In this paper, we propose a semi-automatic risk-based test case prioritization approach based on software modification information and methods (functions) invocation relationship. The objective of this research is to make risk-based testing more systematic and flexible by automating the risk assessment process, and find high-risk faults early. The proposed approach utilizes the requirements modification information, complexity, and size of the methods as the risk indicating factors. We applied an automated procedure to extract these risk factors and compute the risk values of the system methods. The proposed approach is empirically evaluated with two software applications with multiple versions in terms of its fault detection rate, both overall and in risky modules. The experimental results suggest that our proposed approach improves test efficiency by spotting defects early overall and even earlier in the high-risk modules than existing state-of-the-art approaches.

Keywords Software testing · Regression testing · Risk-based testing · Test case prioritization · APFD

1 Introduction

Regression testing is a crucial process which validates the quality of a software after it is being modified. A common practice involves reusing the entire test suite. However, this method becomes infeasible in case of tight schedule and restricted budget. To this end, several systematic methods have been proposed for optimizing regression testing by re-running all or some of the test cases of a modified version. Existing researches can be broadly categorized into three main categories, namely, test case selection [1], minimization [2] and prioritization [3–5]. The prior two techniques minimize the length of the test suite which involves the risk of discarding fault exposing test cases [6]. In contrast, test case prioritization (TCP) rearranges the whole test suite based on some criteria, enabling the testers to run the most significant test cases early in the testing phase, increasing test effectiveness.

Over the past several years, a number of TCP techniques have been proposed and empirically studied to find faults early in a software system and expand the fault detection rate. Existing methods include: history-based [7], coverage-based [8], fault severity-based [9], cost-based [10] etc. Most of these methods order test cases on the basis of previous test execution history and prior knowledge of existing faults, which may not be always available. Furthermore, these approaches do not put much attention on finding defects early in high-

✉ Ziliang Feng
fengziliang@scu.edu.cn

Hosney Jahan
hosney.hstu08@gmail.com

S. M. Hasan Mahmud
hasan.swe@daffodilvarsity.edu.bd

[1] College of Computer Science, Sichuan University, Chengdu 610065, China

[2] School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China

risk modules, though it is of great importance to focus on high-risk modules to compensate resource restriction.

Risk-based testing (RBT), on the other hand, focuses on those portions of a software that exhibit maximum risk. It aims to find the most critical faults in the risky modules as early as possible, with a minimum cost [11]. Thus, it enables the testers to prevent or reduce the possibility of occurring a fault that leads to severe damage. In risk-based testing, the likelihood of occurring an error and the harm caused by that error are used to measure the risk values associated with a certain portion or a module of a system, higher risk value indicating high risk module. However, most of the existing risk-based techniques are manual where the risk values are usually determined by experts based on their judgment, insight knowledge, and various metrics [12]. Considering each system requirement or component manually is a very tiresome and time-consuming process, which hinders the benefits of RBT. Therefore, an automated risk assessment procedure can make RBT more systematic, precise and applicable to practitioners.

Previous studies [13] suggest that combining risk information and requirements may enhance the efficiency of TCP. Several existing risk-based TCP approaches combined system requirements and risk information for computing the risk exposer values of the requirements, showing promising results [14]. Nevertheless, very few researches have investigated system methods (functions) for assessing risk values. A popular TCP technique, namely, total method coverage-based prioritization technique [3,15] suggests that test cases covering more methods have a higher possibility of exposing errors, which is based on the fact that all the methods possess the equal possibility of being faulty. In practice, the likelihood of the methods of being faulty is not always equal. Thus, prioritizing test cases based on only their covered methods may not be very precise. However, if we can assess the risks associated with the methods and use them as a means of identifying important test cases, it may result in better test efficiency by exposing faults early in risky areas. To our best knowledge, very few research [16] have used system methods to assess the risks of a system. However, they used historical data and followed a manual procedure to extract the risk factors. Moreover, their approach is evaluated with only the Average Percentage of Faults Detected (APFD) metric without validating whether it can find errors early in high-risk modules.

To address the aforementioned limitations, we propose a systematic method-level risk-based test case prioritization approach (MR-TCP) that extracts risk values of the system methods and prioritizes test cases on the basis of their computed risks, achieving maximum efficiency. In the context of this paper, risk is described into two extents: failure likelihood or probability, i.e. the possibility of a failure to occur; and failure consequence or impact, i.e., the damage or loss caused by that failure. We employed an automated process to calculate the failure likelihood and failure impact values of the methods, which also reduces the time and subjectivity of manual risk assessment. The failure likelihood is computed based on three risk factors (requirements modification information, complexity, and size of the methods) and the failure impact is computed using the method calling relationship. Based on the obtained failure probability and failure impact values, the risk values of each method have been computed. Here, the estimated risk of a method defines the likelihood of a fault to be occurred by that method. Finally, the risk values of the methods are used to calculate the risk values of the test cases and prioritize them. We compared our approach with other state-of-the-art approaches based on two case studies, where the results of the proposed approach showed better performance in effectiveness and feasibility than others. Besides, it is also able to detect faults quicker in error-prone locations.

In this paper, the following contributions have been made:

- A semi-automatic risk-based test case prioritization approach based on method-level risk factors and invocation relationship within the methods, facilitating method-wise risk estimation.
- An evaluation strategy to automatically derive methods failure probability values based on three risk indicating factors.
- An estimation scheme for automatically deriving failure impact values of the methods based on the method calling relationship and manually allotted impact values of the methods.
- An inclusive comparison with existing approaches from the perspective of fault detection rate both in general and high-risk modules.

The rest of the paper is structured as follows. Section 2 briefly reviews the existing related work. Section 3 presents the proposed risk-based test case prioritization approach. Section 4 presents the experiential evaluation of the proposal, including the research questions. The results and their implications with two software systems are discussed in Sect. 5. Finally, Sect. 6 concludes the paper with a discussion of future work.

2 Related Work

2.1 Test Case Prioritization Approaches

Selecting proper attributes is one of the most crucial parts of test case prioritization. Over the past several years, many researchers inspected several attributes and approaches for prioritizing test cases in order to increase the fault exposure

rate and test effectiveness. Yoo and Harman surveyed various prioritization approaches in [8], where they explored open issues and provided guidelines for further research. Based on previous studies, we roughly categorized the existing approaches into five groups: coverage-based, historical information based, cost-aware based, requirement-based and other criteria based approaches.

Coverage-based approaches. These approaches consider the coverage information of the system under test (SUT), for instance, statement coverage, method coverage [3,17], etc. Jones and Harrold [18] proposed an algorithm for ordering test cases based on the coverage data of modified conditions or decisions. Mei et al. [19] proposed a prioritization approach where they considered the static call graphs of JUnit test cases and the SUT to extract the coverage information. Instead of using dynamic coverage-based methods, they applied estimated coverage data. Though, the dynamic methods showed better fault detection capability, however, their approach might also be effective in situations where coverage data collection is unsuitable or time-consuming. Another coverage-based approach presented by Nardo et al. [20] used a real-world industrial system, together with real faults. They validated the results using four regression testing approaches, namely, prioritization, selection, minimization, and a hybrid approach. The results showed that additional coverage based prioritization techniques gained significantly better fault detection rates than others. However, coverage-based approaches face a common shortcoming that they aim to achieve 100% coverage, which sometimes demand for extra cost and resources than the allocation.

Historical information-based approaches. These approaches determine several attributes for prioritization by analyzing the former historical records. A historical data-based method is presented by Park et al. [7] where the execution time and test criticality are used as the prioritization factors in order to assess the cost and fault severity for cost-cognizant prioritization. Qu et al. [21] reported a black-box based prioritization approach by utilizing the previous testing history and run-time data for computing test cases relationship matrix. The relationship matrix is then used to foresee the fault detection relationship within the test cases and prioritize them. Srikanth et al. [22] introduced an approach to prioritize system use cases and combined them with the past records to order test cases. Engström et al. [23] employed several attributes such as execution history, static priority, execution cost, age of test cases, etc. to do the same. However, these approaches are not very efficient in cases the previous test history is unavailable.

Cost-aware based approaches. Yoo et al. [24] reported a test case ordering technique, which is able to minimize the required number of pair-wise comparisons by clustering the test cases. Other cost-based approaches [10,25] used previous execution history of the test cases to prioritize new test cases, similar to historical information based approaches. Unlike these approaches, our approach is entirely based on methods risk information, and invocation relationship, which can be achieved instantly, without requiring any historical records.

Requirement-based approaches. In these approaches test cases are prepared and prioritized based on system requirements, which might perform a vital role in discovering critical faults. Krishnamoorthi and Mary [26] suggested a test case prioritization approach for detecting severe faults. They proposed six factors to achieve this goal, namely, customer priority, changes in requirement, implementation complexity, completeness, traceability, and fault impact. All these factors are used to determine the requirement factor value (RFV) and test case weight (TCW), which are later used to prioritize test cases. Srikanth et al. [27] presented a system-level test case ordering technique by using fault proneness and system requirements.

Other criteria based approaches. Elbaum et al. [28] implemented a set of experiments to investigate how the fault revealing rate of the test suites are affected by various factors (i.e. program structure, change characteristics etc.) and evaluated which metric performs as the finest predictor of APFD. Using mutation scores in test case prioritization are also considered as an important piece of information that can expand the fault detection rate of a test suite [29]. However, in the primary testing phase, it is unknown which faults will be detected by which test cases. Therefore, applying mutation scores are not appropriate in practice. In addition to these techniques, several other techniques have been frequently employed for prioritizing test cases, such as machine learning-based approaches [30–32], Bayesian network [33], Ant Colony Optimization [34] and so forth.

In software testing, the goal of the testers'is to find maximum faults by utilizing minimum resources, since there are always time and budget limitations. To do so, they should focus on finding faults in high-risk modules. However, the aforementioned prioritization approaches are typically based on coverage information, historical data and system requirements related features. Moreover, they ignore the risk indicating factors, and thus, are unable to find faults in high-risk modules. In order to address this issue, we propose a risk-based test case prioritization approach which estimates the risk coverage degree of the test cases, where test cases associated with risky modules are executed preferentially to increase the reliability of the system with maximum efficacy.

2.2 Risk-Based Testing Approaches

Erdogan et al. [35] surveyed the area of risk analysis in software testing and categorized risk-based testing approaches into eight types. Focusing on their work, we merge these approaches into four groups and discuss the most relevant ones.

Approaches with focus on the general level. Usually, the reliability and robustness of a system is determined by the integrity of testing, before a software is being released. Using potential risk information of a system helps in early fault detection, thereby enhancing the quality of testing [36]. In this regard, Felderer and Ramler [37] introduced an approach to apply risk-based testing in a software testing process and provided a model for their integration. Felderer and Schieferdecker [38] further proposed a taxonomy of risk-based testing which provides a framework to understand, categorize, assess, and compare risk-based testing methods to assist their selection and adapting for particular intentions. The taxonomy considered risks associated with all the phases of testing and contains three top-level classes-risk drivers, risk assessment, and risk-based testing process.

A risk-based test case prioritization approach is reported by Yoon and Choi [39], where the risk exposer value is considered as the main criterion for assessment, which is determined by the domain experts to define the severity of a risk associated with a fault. The approach is evaluated based on two criteria: 1. the rate of fault detection and 2. the severity of those faults detected by the test suites. However, this approach is solely dependent on the domain experts, which is expensive and time-consuming. Besides, the results could vary depending on different experts. Amland [11] proposed a risk-based testing approach where he defined the risk exposer value as a product of the likelihood of fault occurrence and failure cost. Furthermore, he provided suggestions on the necessity of system requirements in order to achieve better results. Nayak et al. [40] reported a prioritization approach by considering the effect of different factors to investigate how they help in improving the testing procedure.

Model-based risk estimation approaches. Stallbaum and Metzger [41] proposed a requirements metric based automatic risk assessment process by converting the SUT into a set of use case diagrams. Two factors have been considered for risk assessment, i.e., cost and probability, where cost is estimated manually by experts and probability is estimated automatically based on four criteria: volatility, complexity, faultiness, and test intensity. Finally, for each use case diagram, risk has been assessed by multiplying cost and probability. Foidl and Felderer [42] introduced a procedure to integrate quality models with risk-based testing by applying the information and data extracted from an open quality model QuaMoCo in risk-based testing. Lachmann et al. [43] formulated a novel risk-based testing approach for software product lines (SPL) integration testing. For each variant, they automatically computed the failure probabilities and failure impacts for each of its architectural components. Besides, they avoided the computational overhead of generating and analyzing the variants by exploiting their variability, defined as deltas.

Approaches with focus on system requirements. Usually, a software system is developed based on some requirements. Therefore, using requirements information in test case prioritization is beneficial in locating most significant and error-prone test cases. Furthermore, applying potential risk information extracted from program structures as auxiliary measures also help to discover errors in a system. Based on these benefits, Yoon et al. [44] developed a risk-based test case prioritization approach by correlating the relationship between requirements and risk information. Firstly, they identified the product risk items and then calculated risk weight by multiplying the weight of risk likelihood and weight of risk impact. Afterward, the risk exposer values of the risk items have been computed by correlating the risk items and the requirements. Finally, all these information are used to determine the priority of the test cases. Similar to this approach, Hettiarachchi et al. [14] reported a prioritization approach integrating software requirements and risk information, with a slight modification that it computes the weighted risk exposer values (W-RE) to prioritize the requirements. In [45], the same authors extended their prior work by combining two prioritization factors with risk information, and applied on an enterprise-level cloud application to explore whether it can improve the efficacy of test case prioritization. To do so, they grouped the system requirements into requirements categories and computed risk exposure (RE) values for each category, which are later used for prioritizing test cases. The results showed promising improvement over other prioritization techniques. Different to these approaches, we computed risks for system methods instead of requirements to inspect whether it can improve the fault exposure rate in risky areas.

Other criteria-based approaches in test case prioritization. Hettiarachchi et al. [46] presented a semi-automatic risk-based prioritization approach by using Fuzzy expert system. Mainly, four risk indicators have been used, i.e., requirements complexity, size, requirements modification status and Potential security threats, which are extracted from the program source files and then a Fuzzy expert system is applied on the indicators to get the final values. Another novel risk-based test case prioritization approach is formulated by Wang et al. [47] by considering the transmission of information flows between software components. Three risk factors are considered to estimate the risk associated with the system classes-threat, vulnerability, and consequence. By converting the SUT into an information flow-based directed network model, the risk factors are extracted and used to evaluate the risk coverage weight for each class. Finally, an additional greedy algorithm is applied to create a prioritized test case order.

Similar to our approach, Fu et al. [16] proposed a test case prioritization algorithm where they manually estimated risk values of program methods based on program changes and method invocation relationship. The risk values are then used

to determine the fault revealing capability of the test cases, and an ILP technique is applied to prioritize them. Unlike this approach, we used different risk factors and followed an automated procedure to extract them from the source files.

All these approaches mentioned above illustrate the benefits of risk analysis in software testing, especially requirement-based risk assessment approaches have shown promising results. The success of these approaches motivated us to apply method-level risk factors in our research, and investigate whether it can extend system dependability with maximum efficiency. However, these requirement-based prioritization approaches have some limitations, such as, most of them utilize only one kind of risk, i.e., the previous history of system faults, which may not be always available; and perform manual computation of risk values, making them time-consuming and expensive. Further, most of these approaches are validated with only one metric to show how fast the prioritized test cases can find faults. Since these approaches applied risk information for prioritizing test cases, they should be validated by indicating whether the revealed faults are from risk-prone areas. To overcome these limitations, we propose a semi-automatic method-level risk-based test case prioritization approach that aims to automate the risk assessment process, making it more supple and precise. We applied some novel method-level risk factors and the method calling relationship to assess the risk values of the methods, without demanding any previous execution history. Furthermore, we validated our approach based on two metrics: 1. Overall fault detection rate, and 2. Ability to detect faults in risky modules. Here, the computation of the risk values separately for each method (instead of considering that all the system methods exhibit the equal possibility of being faulty [3]) enabled us to find faults quicker in high-risk modules, with a minimum cost.

3 Proposed Prioritization Approach

In this section, we present our proposed risk-based test case prioritization approach. The proposed approach introduces a completely automatic procedure to compute the failure probabilities and a semi-automatic procedure to compute the failure impact values, which are applied for prioritizing test cases. Figure 1 provides an overview of the proposed approach. In the initial step, we automatically performed a textual analysis of the program's source files and extracted the changed requirements, size, and complexity of each method. These risk factors are then used to compute the failure likelihood of the methods. Next, a novel technique has been employed for calculating the methods failure impact values. To do so, we manually assigned method impact values to each of the methods and combined these impact values along with the method invocation relationship. Afterward, the failure likelihood and failure impact values are used to compute the risk values of the methods. Furthermore, we evaluated the risk values of the test cases based on the methods risk values and method-test case correlation. Finally, the prioritized test suite is generated according to their computed risk values.

3.1 Risk Indicating Factors Collection

In this paper, we applied three risk factors to compute the risk values of the methods, which are: changed requirements (CR), methods complexity (MC), and methods size (MS). The factors CR and MC are applied in our previous research [48] to measure the changed requirements and complexity of the test cases, where they showed promising results. In [48], an artificial neural network (ANN) is trained based on these factors along with some other factors, and finally, it automatically assigned priorities to new test cases. Different from this approach, here, we used these factors to compute the risks associated with the methods. The other factor, i.e., MS have been previously used by other researchers in requirement-based risk analysis [14,46,47], where they applied it for computing risk exposer values of system requirements. However, here, we applied MS for computing the risk value of the methods. All of these factors are extracted automatically. In the next subsections, we present details about these factors and discuss associated procedures to obtain them from the program source files.

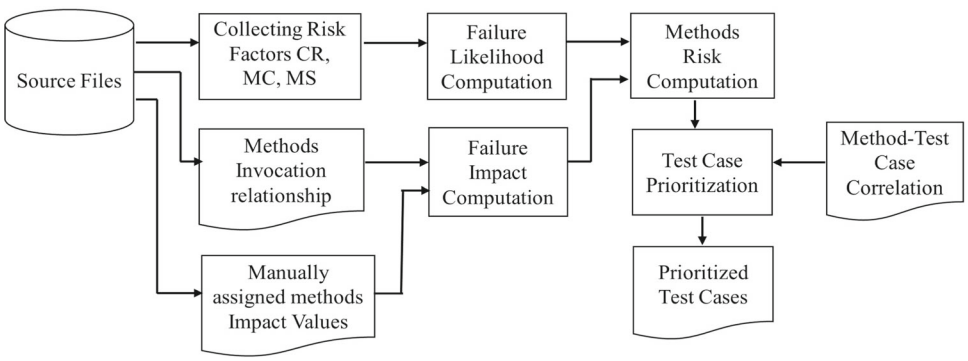

Fig. 1 Overview of the test case prioritization approach

3.1.1 Changed Requirements (CR)

This factor is based on the number of changed requirements within a method. In a software system, faults usually arise from the modification of the software, i.e., changes in its requirements. Hence, using the number of modified requirements in a method could be a good indicator to reveal faults. Moreover, the degree of modification also influences the risk value of a method [16]. Therefore, a method covering more modified requirements is more likely to expose high-risk faults and vice versa.

In order to obtain this factor, we performed a comparative analysis between the main version and the modified version of the SUT. During the comparison, the modified requirements are identified and counted for each method of the modified version. To get the requirements in the source code, we adopted the process applied by Lin et al. [2], where they expressed the system requirements in terms of branch coverage, and defined a true case and a false case for each if-statement (requirements). That means, each if or else statement in a method represents a system requirement. We normalized all the obtained CR values within the range of 0 and 1 by dividing each value with the highest value of CR for any method amongst all the methods to get the final outcome, where, 0 indicates the lowest number of modified requirements and 1 indicates the highest number of modified requirements in a method, as shown in Eq. (1).

$$\mathrm{CR}_i = \frac{\mathrm{NR}_i}{\mathrm{Max}_{\forall i}\{\mathrm{NR}_i\}} \tag{1}$$

where NR_i represents the number of modified requirements in method M_i. Suppose, the method M_1 contains 3 modified requirements and M_5 contains the highest modified requirements which is 7. Therefore, CR_1 is equal to (3/7) or 0.43.

3.1.2 Methods Complexity (MC)

The complexity of a method is computed based on the total number of requirements in that method. A method usually contains one or more requirements. However, containing many requirements makes a method complex, and thus, methods with high complexities may misguide the developers in producing wrong products, which may further affect the risks related to the methods. Therefore, methods containing many requirements or high complexities are considered as more complex and vice versa.

To obtain the complexity information of the methods, the source code of the modified version of the SUT has been thoroughly examined. The complexity of the methods is computed by counting the number of requirements covered by each method. To do so, as mentioned in Sect. 3.1.1, we followed the procedure applied by Lin et al. [2], where each if or else statement in the SUT denotes a requirement. Each MC value is then normalized within the range of 0 and 1, where a value of 0 indicates the least complex method, and 1 indicates the most complex method as shown in Eq. (2).

$$\mathrm{MC}_i = \frac{\mathrm{NC}_i}{\mathrm{Max}_{\forall i}\{\mathrm{NC}_i\}} \tag{2}$$

where NC_i represents the number of conditions or requirements in method M_i.

3.1.3 Methods Size (MS)

To compute this factor, the lines of code (LOC) for each method has been counted. Previous studies showed that the number of defects in a system could be affected by the size of the methods [11,46]. Therefore, methods having higher LOC values are more likely to contain faults. We normalized the MS values within the range of 0 and 1, where 0 indicates the lowest sized method, and 1 indicates the highest sized method as shown in Eq. (3).

$$\mathrm{MS}_i = \frac{\mathrm{LOC}_i}{\mathrm{Max}_{\forall i}\{\mathrm{LOC}_i\}} \tag{3}$$

where LOC_i represents the lines of codes in method M_i.

3.2 Methods Failure Likelihood Computation (MFL)

Previous studies revealed that the activities performed during the development phase of a software strongly influence the failure likelihood of a risk event [41]. Based on this concept, we used the risk factors discussed in the previous section to compute the failure likelihood of the methods, since all of them are extracted from the source files. We measured the failure likelihood (MFL_i) by multiplying the three risk factors. Mathematically, for method M_i, MFL_i can be computed as:

$$\mathrm{MFL}_i = \mathrm{CR}_i \times \mathrm{MC}_i \times \mathrm{MS}_i \tag{4}$$

3.3 Methods Failure Impact Computation (MFI)

In order to compute the failure impact values of the methods, two factors have been considered: 1. method calling relationship, and 2. methods impact values. The method calling relationship echoes the functional dependency of a software. If a method contains faults, then the faults may also affect those methods that are directly or indirectly dependent on that method. Consequently, a method invoked by a number of other methods may frequently be executed during the execution of a test case and faults in that method may promptly be occurred. Therefore, we believe that a large invoked degree of a method is more likely to easily find faults in that method.

We automatically obtained the invoked degree of the methods by analyzing the source files. For the second factor, the impact values of the methods are manually assigned, which defines the importance of a method to the system. To do so, we can analyze the importance of the requirements contained by each method or use an expert's opinion, similar to the traditional risk-based approaches. The impact values are defined within a scale of 1 to 5, where, 1 indicates a less important method with the lowest impact and 1 indicates a very important method with the highest impact.

After obtaining the method calling relationship and assigning failure impact values, we computed the final failure impact values of the methods. The computation of MFI is based on the static relationship of the methods. The computation formula is as follow:

$$\mathrm{MFI}_i = \frac{\sum_{j=1}^{x} \mathrm{MI}_{ij}}{|\mathrm{NMC}_i| \times \mathrm{Max}_{\forall i}\{\mathrm{MI}_i\}} \tag{5}$$

where $\sum_{j=1}^{x} \mathrm{MI}_{ij}$ represents the sum of the impact values of all the methods by which method M_i is invoked, including the impact value of M_i itself; NMC_i represents the total number of methods by which method M_i is called, including itself; and $\mathrm{Max}_{\forall i}\{\mathrm{MI}_i\}$ is the highest impact value of any method amongst all the methods. We normalized the results within the range of 0 and 1.

For example, suppose, there are 5 methods in the SUT i.e., M_1, M_2, M_3, M_4 and M_5, whose impact values are 2, 1, 3, 2 and 4, respectively, and M_1 is invoked by M_2 and M_3. Therefore, the failure impact value of M_1 is,

$$\mathrm{MFI}_1 = \frac{\mathrm{MI}_1 + \mathrm{MI}_2 + \mathrm{MI}_3}{|\{M_1, M_2, M_3\}| * \mathrm{MI}_5} = \frac{2+1+3}{3*4} = 0.5.$$

Moreover, for methods which are not called by any other methods, we set $\sum_{j=1}^{x} \mathrm{MI}_{ij}$ to its own impact value and NMC_i to 1.

3.4 Method's Risk Value Computation

In previous studies [3], test cases have been prioritized by considering that all the system components have equal possibility of being defective and test cases covering more components were denoted with the highest priority. However, the faulty possibilities of system components are different in practice, and thus it seems inaccurate to order test cases based on only their covered components. Instead of considering the same possibility for all system components, MR-TCP individually calculates the risk values of the methods, which are later used for ordering test cases. Therefore, employing an assessed risk value for each test case measures its contribution in detecting high-risk failures.

The risk value of a method indicates the probability of occurring the faults residing within that method. Higher risk values designate higher probability of a method to contain errors and more easily the errors can be spotted. Thus, risk information related to system methods could provide a way to identify important test cases that can expose high-risk faults. Faster identification and correction of such faults expedite the regression testing process correspondingly. In this paper, we compute the risk value of a method by multiplying the failure likelihood and failure impact value of that method, as shown in Eq. (6).

$$\mathrm{MRV}_i = \mathrm{MFL}_i \times \mathrm{MFI}_i \tag{6}$$

3.5 Test Case Prioritization

In this final stage, we compute the risk value of the test cases based on the sum of the risk values of all the methods covered by them. To do so, we require the correlation between the methods and the test cases. To extract the correlation between the methods and test cases, we performed a textual analysis of the source files of the test cases and the SUT. We separated the methods of the SUT by using the keyword *public void*, and used the methods names as strings. Afterwards, for each test case $t_i \in T$, all the method names covered by t_i are recorded. However, if the correlation information is difficult to extract, documents such as functional specification and software requirements which have been used while creating the test cases can help to construct the correlation between the methods and the test cases [46]. Inaccuracy of such correlation may result in inaccurate computation of risk values of the test cases, which may further lead to negligence of a severe fault in a high risk module. Furthermore, some industrial or online tools such as EMMA [49] can also be used to establish the correlations. The risk value of each test case can be computed as follows:

$$T_k = \sum_{i=1}^{y} \mathrm{MRV}_{ik} \tag{7}$$

Here, the indices k represents the kth test case, and y represents the number of methods covered by the kth test case.

After computing the risk values of the test cases we prioritized them in descending order based on those values. The test cases of a system may represent different requirements by covering the same methods. As a result, test cases covering same methods will also have same risk values. In such a scenario, we randomly prioritized those test cases. Moreover, according to Eqs. (4) and (6), methods containing no changed requirements (i.e., CR = 0) will obtain a risk value of 0 ($\mathrm{MRV}_i = 0$) correspondingly, i.e., it has very less chance to contain faults. Accordingly, test cases which contain only

these methods whose risk value is 0 ($MRV_i = 0$) possess less possibility to reveal faults as well. Therefore, we positioned those test cases at the end of the test suite.

4 Empirical Study

To investigate the efficiency of MR-TCP, we design and perform a set of empirical studies. The following sections include the research questions, experimental subjects, evaluation metrics, evaluation methodology and threats to validity.

4.1 Research Questions

Based on the context of our research, we formulated the following two research questions:

RQ1 Can MR-TCP increase the fault detection rate in comparison with other state-of-the-art approaches?

RQ2 Can MR-TCP detect faults early in high risk modules than existing state-of-the-art approaches?

4.2 Experimental Subjects

In order to evaluate the proposed approach we used two software applications with multiple versions: credit card approval system (CCAS) and registration-verifier application (RVA), which are also used in our previous study [48]. We have collected CCAS and RVA from literature [50,51]. CCAS accepts or rejects a credit card approval application based on several input combinations. We implemented three versions of the subject system, each having a number of faults. Version 1, version 2 and version 3 comprehends 4, 4 and 5 faults, respectively. It contains a total of 48 requirements, and 401, 417 and 432 test cases for the three subsequent versions. The second case study, RVS, maintains and manages student's records and validates their registration based on the Iranian Universities Bachelor-Students Registration Policies. Two different versions have been implemented where version 1 contains 7 faults and version 2 contains 9 faults. It contains 70 requirements, and 504 and 542 test cases for version 1 and 2, respectively.

Both these case studies and related test cases are implemented in java. In order to generate the test cases, we followed the control flow graph (CFG) based test case generation approach proposed by Amman and Offutt in [17]. The test cases are designed and coded with the junit test framework, in method-level structure. The generated versions of the case studies include code modification to seed errors in them, based on common mutation operators as stated in the literature [50,51]. The risk level of the seeded faults can be considered as medium risky.

4.3 Evaluation Metrics

To evaluate the effectiveness of MR-TCP, we considered two metrics:

Average Percentage of Faults Detected (APFD) the APFD metric, proposed by Elbaum et al. [3], measures how fast the errors of a system are spotted in a test suite. It calculates the average fault detection rate within the range of 0 and 1, where higher values represent better fault detection rate and vice versa. The computation formula of APFD is as follows:

$$\text{APFD} = 1 - \frac{\text{TF}_1 + \text{TF}_2 + \cdots + \text{TF}_m}{n \times m} + \frac{1}{2n} \tag{8}$$

where n denotes the total number of test cases in the test suite, m is the number of faults in the SUT and TF_f denotes the location of the test case that finds fault f in the test suite.

Percentage of Total Risk Severity Weight (PTRSW) PTRSW, defined by Hettiarachchi et al. [46], measures the efficiency of a test suite regarding early identification of faults in high-risk modules of a particular system. The PTRSW value range from 0 and 1. A prioritization technique is considered as effective in detecting more errors in risky modules if it achieves a higher PTRSW value with a lower test case execution rate. Eq. (9) presents the computation formula of PTRSW.

$$\text{PTRSW} = \frac{\text{STRSW}}{\text{GTRSW}} \tag{9}$$

where STRSW represents the sub-total of the total risk severity weight, and GTRSW represents the grand-total of the total risk severity weight, which are described in the next section.

4.4 Evaluation Methodology

In order to perform the risk assessment process, we require failure likelihood and failure impact of each method. To obtain these values, the steps defined in Sect. 3 have been followed. The risk indicating factors: CR, MC and MS, and the method calling relationship are extracted automatically by analyzing the source files of the SUT. The impact values of the methods are assigned manually based on the previous testing experience of the authors.

To evaluate the effectiveness of our proposed MR-TCP approach and address the research questions, we compared our approach with five other test case prioritization approaches: 1. original order (OO-TCP), 2. random order (RO-TCP), 3. reverse order (REO-TCP), 4. ANN approach (ANN-TCP) [48] and 5. total method coverage approach (TMC-TCP) [3]. For OO-TCP, the test cases are executed in their original order. For RO-TCP, ten test suites have been created by randomly prioritizing the test cases and executed

Table 1 Sample computation process of PTRSW and related data for version 2 of RVA applying MR-TCP approach

Test cases	RSW				TRSW	PTRSW (%)
	F1	F2	…	F9		
TC54	0.23	0.044	0	0.23	0.50	
…	…	…	…	…	…	
TC65	0.23	0.044	0	0	0.27	
STRSW after executing 25% of the test cases					46.33	28.73
…	…	…	…	…	…	
TC201	0	0.044	0.8	0	1.07	
STRSW after executing 50% of the test cases					105.15	65.12
…	…	…	…	…	…	
TC235	0	0	0.12	0.23	1.55	
STRSW after executing 75% of the test cases					152.76	94.3
GTRSW after executing the whole test suite					161.25	

based on each prioritized order. For REO-TCP, the test suite is executed in the reverse order. In ANN-TCP [48], an artificial neural network is trained based on three factors (number of modified modules, number of modified requirements and test cases complexity) for each specific version of the SUT. After training, the trained ANN automatically assigned priorities to new test cases and based on the predicted order, the test suite has been executed. Lastly, since our approach is method-level, for better comparison, we used total method coverage approach (TMC-TCP) [3]. To obtain this prioritized order, we used the method-test case correlation information extracted in Sect. 3.5, where the test cases are prioritized based on their covered methods, and executed the test cases according to the obtained order. Finally, to answer the research question RQ1, the APFD value is computed for all the test suites of the prioritization approaches. For RO-TCP, the APFD value is computed for all the ten test suites, and the average value is taken as the final result.

To address the research question RQ2, we calculated the PTRSW values for all the prioritization approaches following the similar procedure defined in literature [46]. In order to compute the PTRSW values, we require the information about those methods which have been modified or contain faults. Since faults may arise due to the changes in the requirements, we used the factor CR to identify which methods have been modified in the SUT and created a fault-method trace file for all the versions of the SUTs. Afterward, the risk values of the methods are assigned to their associated mutation faults. Here, we used the risk values of the methods computed in Sect. 3.4, which can also be stated as risk severity weight (RSW) of the methods. The RSW ranges within 0 and 1, where, 0 indicates no risk fault for a method and 1 indicates high-risk fault for a critical module.

A test case may detect one or several faults. Therefore, test cases detecting same faults are assigned with same RSW value. However, if a test case detects several faults, all the RSW values of the identified faults are added together to get the total risk severity weight (TRSW). The sum of TRSW values of all the test cases represents the grand total risk severity weight (GTRSW) of the test suite. We computed the sub-total of risk severity weight (STRSW) values for three execution rates, i.e., 25%, 50%, and 75%. Table 1 presents an example of the computation process and the results of PTRSW for the MR-TCP approach of RVA version 2.

All the simulations are conducted on a personal computer with the following specifications: Intel Core i5 6400 CPU @ 2.70GHz (4 CPUs), 8 GB RAM, and 1 TB HDD, with Windows 8.1 and the compiler platform was Eclipse 4.7.

5 Results and Discussion

In this section, we present and discuss the results of the case studies according to the research questions defined in Sect. 4.1.

5.1 Overall Fault Detection Capability of MR-TCP (RQ1)

In order to answer the first research question, we evaluated the fault detection capability of MR-TCP by comparing its results with some of the most applied prioritization approaches. Tables 2 and 3 presents the comparison results of the prioritization approaches in terms of APFD metric for CCAS and RVA, respectively. The first columns of the tables represent the version number of the subject systems; the second columns present the prioritization approaches; the third columns denote the APFD results of the prioritization approaches; and the last columns show the improvement of MR-TCP over other approaches.

Table 2 APFD comparison results of prioritization approaches for CCAS

Versions	Approaches	APFD	Improvement
V1	OO-TCP	0.68	0.23
	RO-TCP	0.58	0.33
	REO-TCP	0.82	0.09
	ANN-TCP	0.85	0.06
	TMC-TCP	0.91	0
	MR-TCP	0.91	–
V2	OO-TCP	0.54	0.44
	RO-TCP	0.62	0.36
	REO-TCP	0.76	0.22
	ANN-TCP	0.96	0.02
	TMC-TCP	0.96	0.02
	MR-TCP	0.98	–
V3	OO-TCP	0.57	0.33
	RO-TCP	0.53	0.37
	REO-TCP	0.61	0.29
	ANN-TCP	0.80	0.10
	TMC-TCP	0.79	0.11
	MR-TCP	0.90	–

Table 3 APFD comparison results of prioritization approaches for RVA

Versions	Approaches	APFD	Improvement
V1	OO-TCP	0.56	0.34
	RO-TCP	0.54	0.36
	REO-TCP	0.74	0.16
	ANN-TCP	0.82	0.08
	TMC-TCP	0.81	0.09
	MR-TCP	0.90	–
V2	OO-TCP	0.67	0.13
	RO-TCP	0.59	0.21
	REO-TCP	0.87	−0.07
	ANN-TCP	0.80	0
	TMC-TCP	0.77	0.03
	MR-TCP	0.80	–

The results in Table 2 show that the proposed MR-TCP approach achieved higher APFD value than the original (OO-TCP), random (RO-TCP), reverse (REO-TCP), and our previous ANN approach (ANN-TCP), for all the three versions of CCAS, which evidently implies the superiority of MR-TCP. Specially, compared to OO-TCP and RO-TCP, MR-TCP achieved substantial improvement (more than 30% in most cases) on APFD. However, in comparison with the total method coverage approach (TMC-TCP), MR-TCP achieved same APFD value (0.91) for version 1 and slightly larger value (0.98) for version 2, but easily outperformed TMC-TCP for version 3 by achieving an improvement rate of 11%.

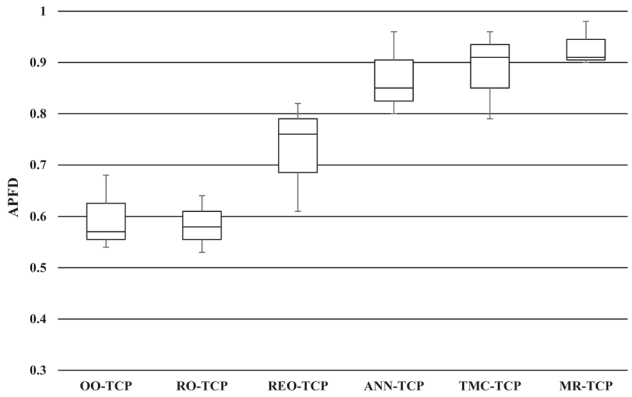

Fig. 2 Box plot of the APFD results of case study 1

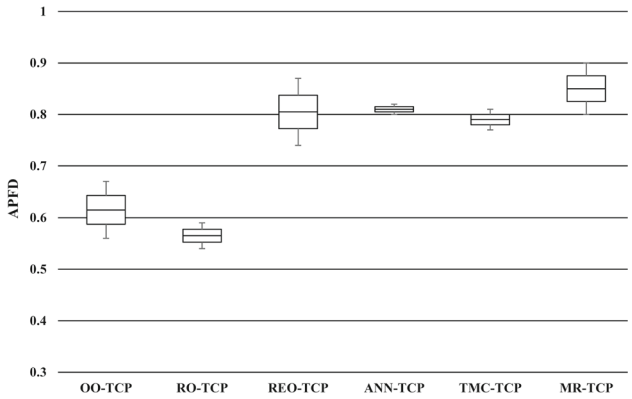

Fig. 3 Box plot of the APFD results of case study 2

Table 3 presents the comparison results for the second case study, RVA. For version 1, MR-TCP outperformed all the other approaches by achieving APFD value of 0.90. However, for version 2, REO-TCP obtained highest APFD (0.87); ANN-TCP obtained the same value (0.80) as MR-TCP, and the remaining approaches were outperformed by the proposed approach. Although, MR-TCP did not perform very well in case of version 2, but, overall, it achieved the highest average APFD value of 0.85.

Further, we represent the APFD results of the prioritization approaches visually in terms of two boxplots. Figure 2 shows the results of CCAS where MR-TCP and TMC-TCP conjugally achieved the highest median APFD value of 0.91, with the pick values of 0.98 and 0.96, respectively. Although, the ANN-TCP and REO-TCP obtained decent median APFD values of 0.85 and 0.76, respectively, still they are outweighed by MR-TCP.

The results of the second case study, shown in Fig. 3, reveals that MR-TCP attained the highest median APFD value of 0.85. Here, TMC-TCP, ANN-TCP and REO-TCP obtained almost similar median APFD values with 0.79, 0.81, and 0.80, respectively. Moreover, MR-TCP considerably outperforms the OO-TCP and RO-TCP approaches for both

case studies. Hence, MR-TCP offers significant improvement regarding efficiency compared to the other TCP techniques.

From the tables and boxplots, we can see that TMC-TCP attained good APFD values. However, for both the case studies, i.e., CCAS and RVA, the average APFD of MR-TCP (0.93 and 0.85, respectively) is higher than that of TMC-TCP (0.88 and 0.76, respectively). This is because, test cases covering a method do not necessarily mean that they can also cover the faulty requirements in that method. MR-TCP, on the other hand, uses the risks associated with the methods while prioritizing test cases, which is estimated by directly considering the modified requirements of each method (Sect. 3.1.1). Thus, MR-TCP increases the testing efficacy correspondingly.

Moreover, we further evaluate the time effectiveness of the proposed approach by computing the percentage of executed test cases required for discovering all the faults residing within SUT. To do so, a comparison is drawn between all the six prioritization approaches. The comparison results are presented via bar graphs in Figs. 4 and 5 for CCAS and RVA, respectively. The bar graphs are plotted to show the prioritization approaches in *x*-axis and the percentage of executed test cases to detect all the faults in *y*-axis. The results denote that the number of test cases required for revealing the faults are less for the proposed approach in comparison with the other approaches. From Fig. 4, it can be perceived that our proposed MR-TCP is able to reveal all the errors by executing only 23% of the total test cases whereas OO-TCP, RO-TCP, REO-TCP, ANN-TCP and TMC-TCP needs 75%, 61%, 36%, 53% and 32%, respectively, for CCAS. Similarly, for RVA, it is noticed from Fig. 5 that MR-TCP requires only 53% of the total test cases to detect all the faults, whereas OO-TCP, RO-TCP, REO-TCP, ANN-TCP, and TMC-TCP require 91%, 68%, 59%, 55% and 61%, respectively. For both case studies our proposed MR-TCP reveals errors earlier than the other five approaches, by executing minimum test cases. Thus, the required testing time will be minimized correspondingly, enhancing the time effectiveness.

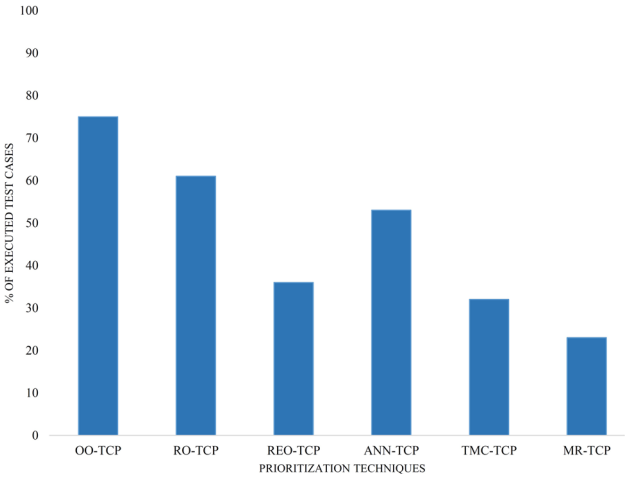

Fig. 4 Bargraph representing the percentage of executed test cases required to reveal all the faults for different approaches for CCAS

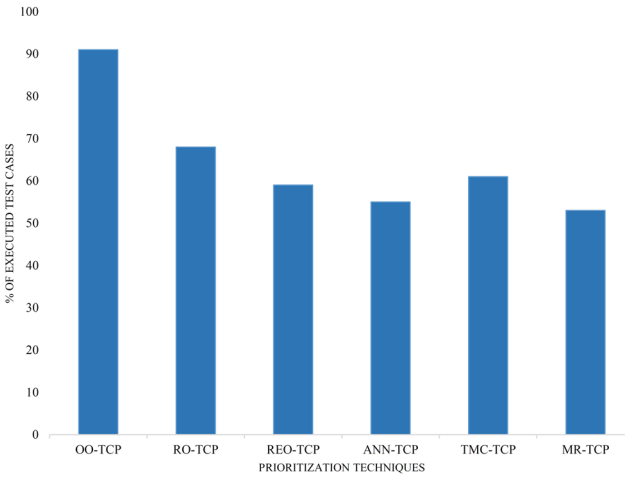

Fig. 5 Bargraph representing the percentage of executed test cases required to reveal all the faults for different approaches for RVA

5.2 Fault Detection Capability of MR-TCP in High Risk Modules (RQ2)

In the first research question, we investigated whether our proposed approach is able to detect faults early. Though the results are inspiring, however they do not indicate that the discovered faults are from risky modules. In the research question RQ2, we evaluated the effectiveness of MR-TCP in terms of detecting faults early in high risk modules, and compared the results with the same existing prioritization approaches as Sect. 5.1.

In order to address this question, the PTRSW values are estimated (Sect. 4.3) for both the case studies. Tables 4 and 5 present the comparison summary of different prioritization approaches in terms of the PTRSW values for all the versions of CCAS and RVA, respectively. The first columns of the tables represent the version numbers, the second columns present the prioritization approaches, and the subsequent columns present the PTRSW values for different execution rates.

The results in Table 4 clearly show that our proposed MR-TCP approach is able to reveal more high-risk faults than the existing approaches and has obtained the highest PTRSW value for almost all test execution rates. TMC-TCP and MR-TCP achieved almost the same PTRSW values for 25% and 50% execution rates for version 1. In version 2, TMC-TCP achieved slightly bigger value than MR-TCP for 25% execution rate. However, MR-TCP managed to produce relatively higher values than TMC-TCP for 75% execution rate for version 1 and version 2. Similarly, for version 3, MR-TCP obtained higher PTRSW values than TMC-TCP for all

Table 4 PTRSW for different execution rates of test cases for CCAS

Versions	Approaches	Execution rate 25%	Execution rate 50%	Execution rate 75%
V1	OO-TCP	6.77	32.95	65.73
	RO-TCP	12.24	38.19	55.12
	REO-TCP	7.57	32.64	48.32
	ANN-TCP	27.41	54.36	74.82
	TMC-TCP	32.7	67.38	91.37
	MR-TCP	32.7	67.36	92.43
V2	OO-TCP	0.56	21.58	70.50
	RO-TCP	10.32	32.21	61.60
	REO-TCP	1.9	19.14	45.54
	ANN-TCP	46.04	60.01	83.88
	TMC-TCP	55.53	82.47	98.06
	MR-TCP	53.97	83.28	98.63
V3	OO-TCP	1.50	25.51	70.0
	RO-TCP	15.92	40.54	59.54
	REO-TCP	2.6	15.60	45.29
	ANN-TCP	39.07	66.06	85.29
	TMC-TCP	49.17	80.13	95.46
	MR-TCP	54.70	84.40	97.39

Table 5 PTRSW for different execution rates of test cases for RVA

Versions	Approaches	Execution rate 25%	Execution rate 50%	Execution rate 75%
V1	OO-TCP	6.8	23.11	40.97
	RO-TCP	10.26	28.19	51.56
	REO-TCP	1.52	17.66	33.72
	ANN-TCP	24.39	59.34	83.40
	TMC-TCP	34.94	67.84	96.46
	MR-TCP	39.72	68.43	97.29
V2	OO-TCP	4.34	14.34	38.33
	RO-TCP	5.24	12.55	40.33
	REO-TCP	8.5	34.8	71.27
	ANN-TCP	33.98	64.83	90.84
	TMC-TCP	31.68	62.63	94.63
	MR-TCP	28.73	65.12	94.73

the three execution rates. Moreover, our approach considerably outperformed other prioritization approaches for all execution rates in all three versions. Though, our previous ANN-TCP approach obtained good PTRSW values, but it is still outperformed by MR-TCP.

Likewise, the results of Table 5 show that our proposed MR-TCP outperformed almost all the prioritization approaches for all execution rates except for TMC-TCP (25% execution rate for version 1) and ANN-TCP (25% execution rate for version 2). These results implies that using risk information of the methods is effective for faster identification of risky faults. Therefore, we claim that the proposed approach is fairly applicable and feasible.

5.3 Threats to Validity

This section presents the threats to the validity of the proposed approach, and the methods we employed to limit their effects.

5.3.1 Internal Validity

In this paper, we used three risk indicating factors, i.e., the number of changed requirements, size, and complexity of the methods for risk estimation. There are other choices available and selection of different factors may affect the results. However, previous research works have proved these factors as good measures for risk assessment. Extraction of these

factors may exhibit potential errors. To limit this threat, the outcomes of some simple java programs are validated manually. Moreover, in this paper, we computed the risk values associated with the methods of the SUTs, considering only the product risks. However, there are several other types of risks, i.e., process and project risks, which are not considered in this research. Applying additional risk indicating factors in the risk assessment process can reduce this threat.

5.3.2 External Validity

The subject systems used in this research for empirical studies are medium-sized, while, big industrial projects may lead to altered results. Another concern includes the usage of mutation faults, which can hamper the generalizability of the outcomes. However, we performed several experiments by comparing different approaches, which presents the first signs that our proposed approach works adequately fine. Furthermore, The versions of the case studies are created by modifying only the existing requirements and adding no new requirements. However, in real world, software maintenance may include addition of new requirements and deletion of obsolete requirements. We aim to address these limitations by performing further experiments with bigger industrial projects with actual faults, which is also one of our future directions.

5.3.3 Construct Validity

This threat implicates the measurement metrics applied in the empirical studies. The metrics selected in this research to measure the rate of fault detection (APFD) and early finding faults in high-risk modules (PTRSW) have alternative metrics, usage of which may alter the outcomes. However, we choose these metrics based on their inclinations in previous studies. Moreover, the computation of Methods Failure Impact (MFI) requires human involvement, i.e., expert's knowledge and judgment. Therefore, the results may differ from person to person.

6 Conclusion

In this paper, we presented a test case prioritization approach by systematically estimating the risks associated with a system to improve the fault detection rate in risky modules. While providing method impact values manually, our approach is able to calculate failure probabilities and failure impact values for each method automatically. Thus, the proposed approach facilitates risk-based testing with justified effort compared to existing manual approaches. Moreover, the proposed approach is empirically validated with two case studies with multiple versions. The results of the empirical studies demonstrate that our proposed risk-based approach is capable of detecting faults early and even earlier in risky modules compared to the existing prioritization techniques.

In future, we plan to evaluate the effectiveness of the proposed approach with large scale industrial projects with real faults. Furthermore, in addition to size, complexity and number of changed requirements of the methods, we plan to apply other risk factors (such as time overhead, usage rate, etc.) to improve the risk assessment even more.

References

1. Gligoric, M.; Negara, S.; Legunsen, O.; Marinov, D.: An empirical evaluation and comparison of manual and automated test selection. In: ASE'14 Proceedings of 29th ACM/IEEE International Conference on Automated Software Engineering, Vasteras, Sweden, pp. 361–372 (2014)
2. Lin, C.T.; Tang, K.W.; Kapfhammer, G.M.: Test suite reduction methods that decrease regression testing costs by identifying irreplaceable tests. Inf. Softw. Technol. **56**(10), 1322–1344 (2014)
3. Elbaum, S.; Malishevsky, A.G.; Rothermel, G.: Test case prioritization: a family of empirical studies. IEEE Trans. Softw. Eng. **28**(2), 159–182 (2002)
4. Khatibsyarbini, M.; Isa, M.A.; Jawawi, D.N.A.; Tumeng, R.: Test case prioritization approaches in regression testing: a systematic literature review. Inf. Softw. Technol. **93**, 74–93 (2018)
5. Tahat, L.; Korel, B.; Koutsogiannakis, G.; Almasri, N.: State-based models in regression test suite prioritization. Softw. Qual. J. **25**, 703–742 (2017)
6. Catal, C.; Mishra, D.: Test case prioritization: a systematic mapping study. Softw. Qual. J. **21**(3), 445–478 (2013)
7. Park, H.; Ryu, H.; Baik, J.: Historical value-based approach for cost-cognizant test case prioritization to improve the effectiveness of regression testing. In: Proceedings of 2nd Second International Conference on Secure System Integration and Reliability Improvement (SSIRI 2008), Yokohama, Japan, pp. 39–46 (2008)
8. Yoo, S.; Harman, M.: Regression testing minimization, selection and prioritization: a survey. Softw. Test. Verif. Reliab. **22**(2), 67–120 (2012)
9. Yu, Y.T.; Lau, M.F.: Fault-based test suite prioritization for specification-based testing. Inf. Softw. Technol. **54**(2), 179–202 (2012)
10. Elbaum, S.; Malishevsky, A.; Rothermel, G.: Incorporating varying test costs and fault severities into test case prioritization. In: Proceedings of IEEE 23rd International Conference on Software Engineering (ICSE), Toronto, Ontario, Canada, pp. 329–338 (2001)
11. Amland, S.: Risk-based testing: risk analysis fundamentals and metrics for software testing including a financial application case study. J. Syst. Softw. **53**, 287–295 (2000)
12. Felderer, M.; Haisjackl, C.; Pekar, V.; Breu, R.: An exploratory study on risk estimation in risk-based testing approaches. In: Winkler, D., Biffl, S., Bergsmann, J. (eds.) Software Quality. Software and Systems Quality in Distributed and Mobile Environments. SWQD 2015. Lecture Notes in Business Information Processing, vol. 200, pp. 32–43. Springer, Cham (2015)
13. Stallbaum, H.; Metzger, A.; Pohl, K.: An automated technique for risk-based test case generation and prioritization. In: Proceedings of the 3rd International Workshop on Automation of Software Test (AST '08), Leipzig, Germany, pp. 67–70 (2008)
14. Hettiarachchi, C.; Do, H.; Choi, B.: Effective regression testing using requirements and risks. In: Proceedings of 8th International

Conference on Software Security and Reliability (SERE), San Francisco, CA, USA, pp. 157–166 (2014)
15. Rothermel, G.; Untch, R.H.; Chengyun, C.; Harrold, M.J.: Prioritizing test cases for regression testing. IEEE Trans. Softw. Eng. **27**(10), 929–948 (2001)
16. Fu, W.; Yu, H.; Fan, G.; Ji, X.; Pei, X.: A regression test case prioritization algorithm based on program changes and method invocation relationship. In: Proceedings of 24th Asia-Pacific Software Engineering Conference (APSEC), Nanjing, China, pp. 169–178 (2017)
17. Ammann, P.; Offutt, J.: Introduction to Software Testing. Cambridge University Press, Cambridge (2008)
18. Jones, J.A.; Harrold, M.J.: Test-suite reduction and prioritization for modified condition/decision coverage. IEEE Trans. Softw. Eng. **29**(3), 195–209 (2003)
19. Mei, H.; Hao, D.; Zhang, L.; Zhang, L.; Zhou, J.; Rothermel, G.: A static approach to prioritizing JUnit test cases. IEEE Trans. Softw. Eng. **38**(6), 1258–1275 (2012)
20. Nardo, D.D.; Alshahwan, N.; Briand, L.; Labiche, Y.: Coverage-based regression test case selection, minimization and prioritization: a case study on an industrial system. Softw. Test. Verif. Reliab. **25**, 371–396 (2015)
21. Qu, B.; Nie, C.; Xu, B.; Zhang, X.: Test case prioritization for black box testing. In: Proceedings of IEEE 31st Annual International Computer Software Applications Conference (COMPSAC), Beijing, China, pp. 465-474 (2007)
22. Srikanth, H.; Cashman, M.; Cohen, M.B.: Test case prioritization of build acceptance tests for an enterprise cloud application: an industrial case study. J. Syst. Softw. **119**, 122–135 (2016)
23. Engström, E.; Runeson, P.; Ljung, A.: Improving regression testing transparency and efficiency with history-based prioritization - an industrial case study. In: Proceedings of 4th IEEE International Conference on Software Testing, Verification and Validation, Berlin, Germany, pp. 367-376 (2011)
24. Yoo, S.; Harman, M.; Tonella, P.; Susi, A.: Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge. In: Proceedings of 18th International Symposium on Software Testing and Analysis (ISSTA'09), Chicago, IL, USA, pp. 201-212 (2009)
25. Schwartz, A.; Do, H.: Cost-effective regression testing through adaptive test prioritization strategies. J. Syst. Softw. **115**, 61–81 (2016)
26. Krishnamoorthi, R.; Mary, S.A.S.A.: Factor oriented requirement coverage based system test case prioritization of new and regression test cases. Inf. Softw. Technol. **51**(4), 799–808 (2009)
27. Srikanth, H.; Williams, L.; Osborne, J.: System test case prioritization of new and regression test cases. In: International Symposium on Empirical Software Engineering (ISESE 2005), Noosa Heads, Australia, pp. 64–73 (2005)
28. Elbaum, S.; Gable, D.; Rothermel, G.: Understanding and measuring the sources of variation in the prioritization of regression test suites. In: Proceedings of 7th International Software Metrics Symposium, London, UK, pp. 169–179 (2002)
29. Hou, S.S.; Zhang, L.; Xie, T.; Mei, H.; Sun, J.S.: Applying interface-contract mutation in regression testing of component-based software. In: Proceedings of 23rd IEEE International Conference on Software Maintenance, Paris, France, pp. 174–183 (2007)
30. Lachmann, R.; Schulze, S.; Nieke, M.; Schaefer, I.: System-level test case prioritization using machine learning. In: Proceedings of IEEE 15th International Conference on Machine Learning Applications (ICMLA), Anaheim, CA, USA, pp. 361–368 (2016)
31. Wang, F.; Yang, S.C.; Yang, Y.L.: Regression testing based on neural networks and program slicing techniques. In: Wang, Y., Li, T. (eds.) Practical Applications of Intelligent Systems. Advances in Intelligent and Soft Computing, vol. 124, pp. 409–418. Springer, Berlin (2011)
32. Perini, A.; Susi, A.; Avesani, P.: A machine learning approach to software requirements prioritization. IEEE Trans. Softw. Eng. **39**(4), 145–461 (2013)
33. Mirarab, S.; Tahvildari, L.: An empirical study on Bayesian network-based approach for test case prioritization. In: Proceedings of 1st International Conference on Software Testing, Verification, and Validation, Lillehammer, Norway, pp. 278–287 (2008)
34. Gao, D.; Guo, X.; Zhao, L.: Test case prioritization for regression testing based on ant colony optimization. In: Proceedings of 6th IEEE International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, pp. 275–279 (2015)
35. Erdogan, G.; Li, Y.; Kobro, R.; Seehusen, F.; Stølen, K.: Approaches for the combined use of risk analysis and testing: a systematic literature review. Int. J. Softw. Tools Technol. Transf. **16**(5), 627–642 (2014)
36. Xie, X.; Chen, T.Y.; Kuo, F.: A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. ACM Trans. Softw. Eng. Methodol. **22**(4), 1–40 (2013)
37. Felderer, M.; Ramler, R.: Integrating risk-based testing in industrial test processes. Softw. Qual. J. **22**(3), 543–575 (2014)
38. Felderer, M.; Schieferdecker, I.: A taxonomy of risk-based testing. Int. J. Softw. Tools Technol. Transf. **16**(5), 559–568 (2014)
39. Yoon, H.; Choi, B.: A test case prioritization based on degree of risk exposure and its empirical study. Int. J. Softw. Eng. Knowl. Eng. **21**(2), 191–209 (2011)
40. Nayak, S.; Kumar, C.; Tripathi, S.: Enhancing efficiency of the test case prioritization technique by improving the rate of fault detection. Arab. J. Sci. Eng. **42**(8), 3307–3323 (2017)
41. Stallbaum, H.; Metzger, A.: Employing requirements metrics for automating early risk assessment. In: Proceedings of MeReP07, Palma de Mallorca, Spain, pp. 1–12 (2007)
42. Foidl, H.; Felderer, M.: Integrating software quality models into risk-based testing. Softw. Qual. J. **26**(2), 809–847 (2016)
43. Lachmann, R.; Beddig, S.; Lity, S.; Schulze, S.; Schaefer, I.: Risk-based integration testing of software product lines, In: VAMOS'17 Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems, Eindhoven, Netherlands, pp. 52–59 (2017)
44. Yoon, M.; Lee, E.; Song, M.; Choi, B.: A test case prioritization through correlation of requirement and risk. J. Softw. Eng. Appl. **5**(10), 823–835 (2012)
45. Srikanth, H.; Hettiarachchi, C.; Do, H.: Requirements based test prioritization using risk factors: an industrial study. Inf. Softw. Technol. **69**, 71–83 (2016)
46. Hettiarachchi, C.; Do, H.; Choi, B.: Risk-based test case prioritization using a fuzzy expert system. Inf. Softw. Technol. **69**, 1–15 (2016)
47. Wang, Y.; Zhu, Z.; Yang, B.; Guo, F.; Yu, H.: Using reliability risk analysis to prioritize test cases. J. Syst. Softw. **139**, 14–31 (2018)
48. Jahan, H.; Feng, Z.; Mahmud, S.M.H.; Dong, P.: Version specific test case prioritization approach based on artificial neural network. J. Intell. Fuzzy Syst. **36**(6), 6181–6194 (2019)
49. Roubtsov, V.: EMMA: a free Java code coverage tool (2005). http://emma.sourceforge.net/index.html. Accessed 20 Mar 2019
50. Vanmali, M.; Last, M.; Kandel, A.: Using a neural network in the software testing process. Int. J. Intell. Syst. **17**(1), 45–62 (2002)
51. Shahamiri, S.R.; Wan-Kadir, W.M.N.; Ibrahim, S.; Hashim, S.Z.M.: Artificial neural networks as multi-networks automated test oracle. Autom. Softw. Eng. **19**, 303–334 (2012)