Software Quality Journal (2020) 28:397–423
https://doi.org/10.1007/s11219-019-09463-4

Enhanced regression testing technique for agile software development and continuous integration strategies

Sadia Ali[1] · Yaser Hafeez[1] · Shariq Hussain[2] · Shunkun Yang[3]

Published online: 13 September 2019

Abstract
To survive in competitive marketplaces, most organizations have adopted agile methodologies to facilitate continuous integration and faster application delivery and rely on regression testing during application development to validate the quality and reliability of the software after changes have been made. Consequently, for large projects with cost and time constraints, it is extremely difficult to determine which test cases to run at the end of each release. In this paper, a test case prioritization and selection approach is proposed to improve the quality of releases. From existing literature, we analyzed prevailing problems and proposed solution relevant to regression testing in agile practices. The proposed approach is based on two phases. First, test cases are prioritized by clustering those test cases that frequently change. In case of a tie, test cases are prioritized based on their respective failure frequencies and coverage criteria. Second, test cases with a higher frequency of failure or coverage criteria are selected. The proposed technique was validated by an empirical study on three industrial subjects. The results show that the method successfully selects an optimal test suite and increases the fault detection rate (i.e., more than 90% in the case of proposed technique and less than 50% in other techniques), which reduces the number of irrelevant test cases and avoids detecting duplicate faults. The results of evaluation metrics illustrate that the proposed technique significantly outperform (i.e., between 91 and 97%) as compared to other existing regression testing techniques (i.e., between 52 and 68%). Therefore, our model enhances the test case prioritization and selection with the ability for earlier and high fault detection. Thus, pruning out irrelevant test cases and redundant faults and enhancing the regression testing process for agile applications.

Keywords Regression testing · Agile methodology · Agile environment · Test case prioritization · Test suite selection · Frequency of change

✉ Shunkun Yang
ysk@buaa.edu.cn

Extended author information available on the last page of the article

1 Introduction

Over time, the world has become increasingly dependent on information technology and software has now become an essential part of much of everyday life. This is also true for businesses in a wide variety of industries, including airlines, media, security agencies, and education. As a result, many methods and strategies have been developed to increase the reliability and efficiency of software development projects in order to satisfy discerning customers (Alkharabsheh et al. 2018; Felderer and Herrmann 2019; Horváth et al. 2019; Zhao et al. 2019).

Many modern development strategies, such as agile methodologies, have been developed to fulfill increasing demands to deliver high-quality software in a short amount of time with limited resources in an extremely competitive environment (Agren et al. 2018; Anand and Dinakaran 2017; Heck and Zaidman 2018). These strategies address frequent changes and releases, team coordination, customer collaboration, continuous releases, etc. Continuous integration (CI) methodologies and frequent releases are now standard practice in many software organizations when it is essential to increase the speed of delivering new products and features (Haghighatkhah et al. 2018; Horváth et al. 2019).

However, the number of changes introduced in each release or iteration to support new or existing features significantly increases the number of bugs in the code (Kandil et al. 2015, 2016; Knauss et al. 2015; Thangiah and Basri 2016). This forces the team to prioritize fixing bugs over regression testing, which compromises the quality of the released software. It is much easier to reduce the regression requirements when there is only a small number of user stories as retesting all test cases after each change for a large number of user stories quickly becomes unworkable (Elbaum et al. 2014; Knauss et al. 2015; Rosero et al. 2017).

Agile practices rely on regression testing (RT), which requires a significant amount of effort and resources during implementation (Hettiarachchi et al. 2016; Thangiah and Basri 2016). Agile approaches utilize recursive implementations and test sequences to reduce the interval between tests (Huang et al. 2012; Kandil et al. 2014). RT is commonly used to verify the quality of software applications after modifications have been made in the development process. It is a maintenance activity that is executed to provide assurance that the variations and/or modifications have not adversely affected the present functionality of the software (Ansari et al. 2016; Spieker et al. 2017). As large test suites require a significant amount of time and expense to execute, various techniques have been proposed to reduce the time and cost of execution, such as test case prioritization (TCP), regression test selection (RTS), test suite minimization (TSM), and test suite augmentation (TSA) (Do 2016; Spieker et al. 2017). RT reuses previous test suites as well as any new test cases that have been added to validate new or modified features (Anderson et al. 2014; Do 2016). The TSM technique is used to prune outdated or obsolete test cases, and regression selection selects a subclass of test suites with which to validate modifications (Kandil et al. 2015; Miranda and Bertolino 2017). TCP efficiently orders the test cases based on specific requirements, such as early fault detection or coverage rate (Kandil et al. 2014; Panichella et al. 2015; Rosero et al. 2016; Rosero et al. 2017). The TSM and RTS methods identify test cases that should be permanently removed from the test suite in order to focus on identifying significant faults in the changed parts of the software (Al-Hajjaji et al. 2019; Ansari et al. 2016; Wang et al. 2019). TSA identifies newly added code and creates new tests to validate the functions of the revised system (Kandil et al. 2016; Miranda and Bertolino 2017; Panichella et al. 2015). TCP is used to organize the complete test suite to facilitate earlier fault detection within a limited time and cost based on

various criteria, such as code coverage, faults, historical information, and requirements (Al-Hajjaji et al. 2019; Azizi and Do 2018; Chen et al. 2018; Flemström et al. 2018; Wang et al. 2019).

A significant amount of research has been conducted to determine how to efficiently prioritize and select test cases to increase the fault detection rate. However, existing RT techniques in the agile environment lack the ability to effectively prioritize and select test cases. The irrelevant and redundant test cases execution results in the identification of redundant faults and repeated re-execution of the test suite.

Therefore, to address these issues, contributions of our research work are as follows:

- In our research, we present a test case prioritization and selection approach for agile strategy in CI context on criteria of frequently changed and failed test cases, to increase the capability of fault detection rate.
- The proposed model offers test case prioritization and selection using frequently change test cases and failed test cases due to CI at every release in agile development strategies. Consequently, we divide our proposed model called CTFF (prioritize and select frequently change test cases and failed frequency) in two phases. The first phase consists of clustering frequently change test cases into multiple clusters. Afterwards, parameters for prioritization are generated based on the highest failure frequency, and if more than one test cases have similar frequency, then test code coverage used as second priority criteria. In the second phase, test cases with highest priority are selected from each cluster for execution to identify maximum faults.
- Therefore, the proposed approach resolves drawback of RT in CI through modern development strategies. The CTFF model evaluated using three different software with different size of test cases developed in an agile environment. From the analysis of results, it is revealed that the CTFF model detects more faults and effective in all cases as compared to random prioritization and other fault-based methods.
- The study provides a roadmap, baseline, and empirical evidence for future research in domain of RT for continuous integrations.

The remainder of this paper is organized as follows. Section 2 describes the related work whereas Section 3 presents the methodology in which CTFF model is introduced. The details of the empirical study are presented in Section 4. Results and discussion are presented in Section 5, and conclusions drawn from this research are presented with future work direction in Section 6.

2 Related work

This section summarized existing literature in regression test prioritization. Existing acknowledged literature shows researchers' probable diverse approaches used to improve the TCP and RTS techniques for regression testing using agile methods.

Studies in the literature on agile RT have suggested that the optimum time for regression testing techniques for agile environments is at the sprint and release level (Anita, and Chauhan, N. 2014; Haghighatkhah et al. 2018; Kandil et al. 2016; Thangiah and Basri 2016). At the sprint level, the authors in Kandil et al. (Kandil et al. 2016) proposed a weighted sprint TCP technique that orders test cases on the basis of three parameters. While at the release level, a

cluster-based release TC_S technique was proposed to group user stories according to the similarities between the modules. Whereas, the selection of test cases was dependent on features identified by the faults in the failed test cases via text mining techniques. However, no prioritization technique was proposed at the release level to prioritize the selected test cases in order to determine which sequence of test cases must be completed to satisfy the testing objective. In addition, no selection technique was proposed at the sprint level to assemble an optimal regression test suite from the entire test suite. The benefit of selection at the sprint level is that this will reduce the number of test cases, which will then reduce the time and effort expended.

Kandil et al. (2015) reduced the number of test cases by evaluating the number of user stories with similar issues that are covered by different test cases. The remaining test cases were then prioritized using weighted agile parameters to increase the detection rate of faults. Knauss et al. (2015) found that executing continuous integration activities in a large software development project is difficult due to structural, social, and practical differences between organizations. It is therefore essential that the test cases be prioritized so that faults can be detected quickly, and bugs can be identified.

Some of the other common regression testing TCP techniques developed by various authors are as follows. Coverage-based TCP uses coverage information to reorder the test cases in order to identify the maximum number of faults (Horváth et al. 2019; Miranda and Bertolino 2018; Spieker et al. 2017). Coverage-based techniques use criteria for prioritization such as the statement, branch, or path coverage (Gupta et al. 2015; Horváth et al. 2019). History-based techniques employ historical data to prioritize the test cases in future sessions, which requires historical information about the test cases to be maintained, such as the execution history of the test cases, the fault detection rate (Abu Hasan et al. 2017; Al-Hajjaji et al. 2019; Aman et al. 2018; Azizi and Do 2018), and other factors (Azizi and Do 2018; Haghighatkhah et al. 2018; Miranda and Bertolino 2018).

Other types of TCP for regression testing include human and probability-based techniques. Lin et al. (2013) improved prioritization in the current version by referring to the results of earlier versions. The experimental results identified similar types of test cases and found that the proposed approach outperformed existing approaches. In addition, the method improved software quality while ignoring test changes and continuously created test cases to cover new features. The conclusion from the above-referenced studies is that methods are needed to more accurately prioritize test cases based on historical information to improve long-term software performance.

Wang and Zeng (2014) employed a prioritization model to add flexibility to the test case prioritization process based on a multi-dimensional equation for sorting test cases for earlier execution. They also found that the performance of dynamic test case prioritization strategies could be improved by considering weighted probability distributions. However, a limitation is that this technique ignores test changes and historical fault information and fails to maintain a repository for future regression tests. Historical information with code coverage is more effective in terms of fault detection than approaches in which only test cases covering changed lines were executed (Gupta et al. 2015).The use of code-coverage information in prioritization techniques improves maximum fault detection but overlooks the effectiveness of fault rate detection and does not consider the requirements of regression testing.

Miranda and Bertolino (2017) proposed a hybrid approach for software reuse that improved the fault detection rate and reduced the size of the test suite. However, they did not consider coverage information pertaining to the reuse of test cases in regression testing. Silva et al. (2016)

presented a hybrid approach for TCP and RTS, which was based on the relationship between system components, was found to solve problems effectively with high quality. However, it did not maintain a repository for historical information and did not consider the fault information of test cases when performing regression testing on new versions of the software. Consequently, large number of test cases were generated which degraded the fault detection ability.

The prioritization approach by Wang and Zeng (2016) used historical data and prioritized requirements to determine the initial test case priority. Even though the performance of this approach was acceptable, redundant faults were detected during fault divisions to requirement property. The coverage information and change in test cases were not used for regression testing. In addition, while multi-criteria were used for prioritization, the frequency of test changes was ignored during prioritization. Abu Hasan et al. (2017) presented dissimilarity clustering TCP using historical information to identify maximum faults in less time.

Elbaum et al. (2014) proposed an algorithm that increased the effectiveness of continuous integration. The initial pre-submit phase applied RTS techniques to select test suites for the specific modules to be tested. In the post-submit phase, dependent and changed modules were tested and prioritization techniques utilized to sequence test cases to increase the likelihood of earlier fault detection. Both phases employed novel techniques and proposed less expensive algorithms. However, the proposed selection and prioritization techniques did not account for the capability and variability of the computing infrastructure and only considered a limited dataset. The problem with that approach is that different datasets are necessary in order to properly understand the RTS limitations when the size of the code and the number of changes increase. Other factors, such as the fault severity and importance of the user stories, are also significant in agile environments.

Kandil et al. (2014) proposed an RTS approach that analyzed the historical relationship between test case failures and code changes when determining the optimal test suite. However, this approach employed a historical RTS technique that required the maintenance of repositories to store the historical data, which comes at a high cost, and this technique was not automated. To date, this approach has only been used in two scenarios requiring a restructuring of the test suite and rearranging of the effort across the test scope.

Anita and Chauhan (Anita,, and Chauhan, N. 2014) proposed a method of test selection that used a weighted undirected graph of user stories based on the average path length and value constraints. Test cases were selected based on their relevance to the specific source and destination user stories. The advantage of this technique was that it selected an optimal set of user stories to ensure high levels of quality and action. However, it was not automated and test cases were selected only based on optimized user stories, and without any prioritization of the test cases when determining the order of execution. This is problematic as other factors should also be considered when selecting test cases in an agile environment.

Azizi and Do (2018) proposed TCP-based collaborative filtering recommender system using change historical information in dynamic environment for decision-making process. They observed that multi-criteria can improve the effectiveness of TCP and need to increase fault rate with new item additions in an intelligent way. Haghighatkhah et al. (2018) proposed RT for fault detection in a continuous integration environment. Availability of failure history data is an important criteria, but only improves effectiveness to a certain extent while history-based diversity is more effective but has disadvantage of high execution time. Ouriques et al. (2018) compared different existing TCP techniques in context of model-based testing using replicated study to investigate influence of test case size on the efficiency of fault detection rate ability.

Al-Hajjaji et al. (2019) proposed similarity-based TCP technique for product line with diverse feature interaction coverage. The study analyzed the effectiveness in both real and seeded fault detection, after the evaluation on three different applications of distinctive feature size. Horváth et al. (2019) investigated the impact of code coverage-based Java language tools on TCP and TSM, and found that coverage information is useful to highlight number of line code covered by each test case for optimization during RT. Whereas, Wang et al. (2019) for embedded systems, proposed location-based TCP using gravitation law for high reliability after modification. Shin et al. (2018) defined multi-objective TCP method for uncertainty prediction in cyber physical systems.

Other existing studies leveraged the TCP and RTS techniques to effectively identify the maximum number of fault as soon as possible with a reduce number of test cases (Azizi and Do 2018; Flemström et al. 2018; Haghighatkhah et al. 2018; Noor and Hemmati 2015; Spieker et al. 2017; Wang and Zeng 2016). However, most of these prioritization techniques only considered code coverage criteria and ignored information pertaining to the tests that frequently changed, which limited the ability to reduce the number of test cases and identify the maximum number of faults (Azizi and Do 2018; Fischer et al. 2018; Flemström et al. 2018; Lu et al. 2016; Ma et al. 2019; Noor and Hemmati 2015). Another limitation is that code coverage criterion alone is not sufficient to identify a maximum number of faults, and multi-criteria are necessary for optimal fault detection (Lachmann et al. 2015; Ouni et al. 2017; Shin et al. 2018).

TCP techniques have also been used along with other criteria to sort test cases for maximum fault detection based on changes in the coverage information, fault rate, and historical information (Aman et al. 2018; del Sagrado and del Águila 2018; Horváth et al. 2019; Magalhães et al. 2017; Mahali and Mohapatra 2018), and a few studies also considered test changes (Almasri et al. 2017; Azizi and Do 2018; Chen et al. 2018; Lu et al. 2016). However, there is still a need to improve TCP techniques in order to locate latent bugs (Alkharabsheh et al. 2018; Azizi and Do 2018; Felderer and Herrmann 2019; Miranda and Bertolino 2017; Özdağoğlu and Kavuncubaşı 2019; Wang et al. 2019). It would also be extremely beneficial to identify a technique that is able to detect the maximum number of faults in the shortest possible time.

A summary of the existing techniques based on respective factors and their limitations like low fault detection ability, considering only code criteria, continuous integration, etc., is provided in Table 1. From literature, we identified that the main reasons for proposing different regression testing techniques are to increase fault detection ability with reduction of redundant faults and irrelevant test cases using different criteria like coverage and historical information. Consequently, these studies still lack in improving the rate of fault detection due to some factors. These factors are fault detection ability (FDA), code coverage (CC), tests changes (TC), fault rate (FR), multi-criteria (MC), irrelevant test cases (ITC), and continuous integration (CI) as described in Table 1 with reasons described in existing literature. These factors are identified in multiple existing studies and are relevant to scope and context of our work, and provide the basis for comparison of results.

As explained above, most existing RT techniques are not intended for use in agile development, while there is room for improvement in those that do improve the quality of modern development strategies for RT to provide maximum and earlier fault identification. In this research, a test case prioritization and selection model is proposed for agile methodologies in a CI context to identify frequently changed and failed test cases in order to increase the fault detection rate. This model employs test case prioritization and selection by clustering frequently changed test cases, produced by CI at every release.

The proposed model, called the CTFF model, prioritizes and selects frequently changed test cases and the corresponding failure frequencies. The process consists of two phases. The first

Table 1 Limitations in existing techniques

Factors	Limitations	References
FDA	Significantly declines due to an unnecessary reduction in the size of test cases	Azizi and Do 2018; Fischer et al. 2018; Ma and Provost 2017; Mahali and Mohapatra 2018; Ouriques et al. 2018; Wang and Zeng 2016
CC	In some cases, only code coverage not significantly identifies maximum faults.	Abu Hasan et al. 2017; Haghighatkhah et al. 2018; Horváth et al. 2019; Knauss et al. 2015; Lu et al. 2016; Noor and Hemmati 2015; Silva et al. 2016
TC	Ignores the tests changes	(Aman et al. 2018; Azizi and Do 2018; Lu et al. 2016
FR	When a new test case added, or old test case changed, detecting a fault is not the same as any of the previously failing test cases.	Azizi and Do 2018; Fischer et al. 2018; Noor and Hemmati 2015; Wang and Zeng 2016
MC	Multi-criteria are optimal for prioritization, but limited studies were found in contrast to this.	Lachmann et al. 2015; Noor and Hemmati 2015; Ouni et al. 2017; Shin et al. 2018
ITC	Usually select and execute irrelevant test cases	Magalhães et al. 2017; Silva et al. 2016; Souto and d'Amorim 2017)
CI	Requires optimum technique at release level and while continue integration	Kandil et al. 2016; Noor and Hemmati 2015; Ouriques et al. 2018

phase clusters redundant frequently changed test cases into multiple clusters and then generates prioritization parameters based on the highest failure frequency. If multiple test cases have similar failure frequencies, then test code coverage is used as the second priority criterion. In the second phase, test cases are selected for execution from the individual clusters to identify the maximum number of faults. In this way, the proposed model addresses a key limitation of RT in CI.

Thus, a key objective of the CTFF model is to enhance the RT process for agile applications by increasing the fault detection rate while excluding irrelevant and/or redundant tests and considering the test cases that frequently change and/or fail.

3 Methodology

In this section, we present the proposed model and describe its internal processes and working. In the CTFF model, applied hybrid techniques of regression testing to mitigate the shortcomings of existing techniques, i.e., irrelevant test case selection and redundant faults. The test case change history is used for clustering to reduce the size of test cases, frequently failed test cases for prioritization of test cases with code coverage to remove ambiguity in case of same priority, and then select highest priority test cases from each cluster to identify maximum faults as quickly as possible.

3.1 Proposed CTFF model

A workflow of the proposed CTFF model is shown in Fig. 1, along with major phases in the CTFF model. The motivation behind building such a model is to improve the implementation of RT in agile development scenarios to ensure software quality at the end of every feature or product release. The model is based on a hybrid of TCP and

RTS and considers test case coverage instead of code coverage criteria during prioritization to select the most frequently changed and failed test cases. To better accommodate large numbers of test cases, the cases are clustered according to their change frequency. If two clusters have similar change frequencies, then they are prioritized based on their frequency of failure and coverage criteria. The underlying premise is that if there are changes in the test cases, then the likelihood of faults increases, which then affects the functionality of the software.

In the following, the main phases of the CTFF model are briefly described:

1. The first phase is the test case extraction. This phase consists of the following stages.

- User stories—In the agile environment, user stories are used to define the functionality that should be provided by the system. A user story explains what the user expects from the system and represents the business value that an agile cross functional team should add to the product in a sprint. In this step, user stories are stored in a file for the selected sprint. Each user story can have multiple related test cases.
- Test suite for current release/iteration—The user stories associated with the release are captured in order to extract all associated test cases.

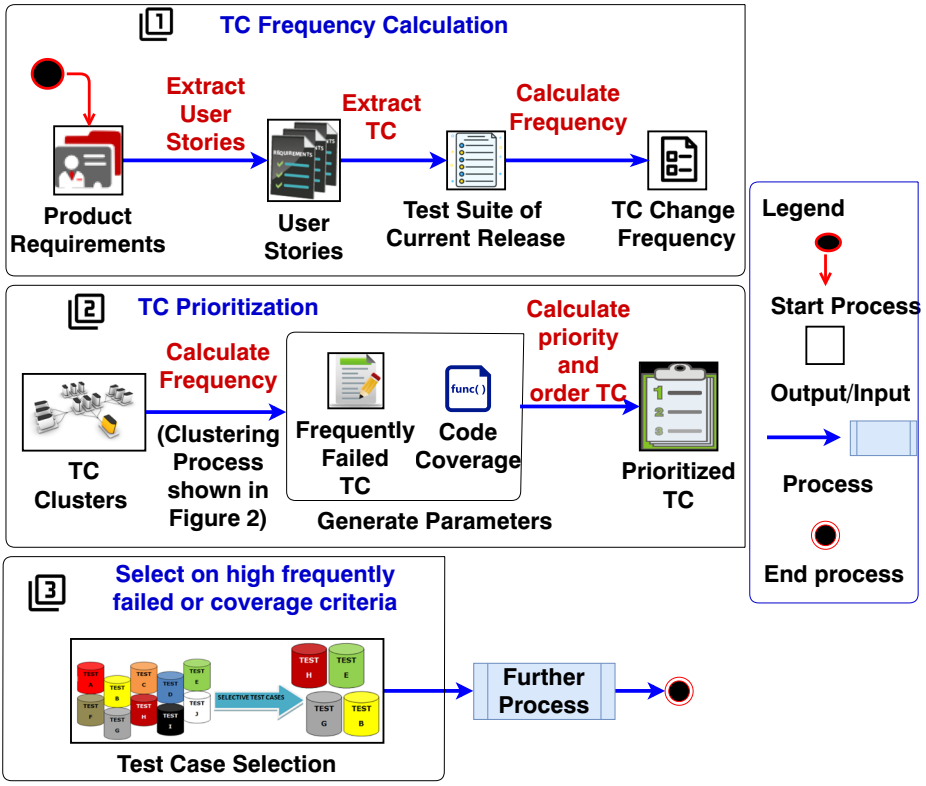

Fig. 1 CTFF model

2. The test case clustering process includes the following steps.

- Identify frequently changed test cases—Identify test cases that frequently change across different releases of software user stories. The frequency of change is computed by taking an average of the changes in similar test cases in each version of modified test cases. The frequency of changes can be calculated as follows:

$$T_f = \left[{}^{(\Sigma T_c)} \Big/ {}_{n} \right] T_f = \left[{}^{(\Sigma T_c)} \Big/ {}_{n} \right] \quad (1)$$

where $T_f$is the change frequency of the test cases, T_c is the total number of test cases changes in each release, and n is the number of times the software was modified.

- Cluster—In the cluster module, test cases are clustered into groups with similar frequencies of change. In other words, test cases with similar change frequencies are grouped in the same cluster. The CTFF algorithm employs a semi-supervised clustering technique called semi-supervised K-means, which combined semi-supervised nonlinear dimensionality reduction and K-means (Abu Hasan et al. 2017; Arafeen and Do 2013; Gultepe and Makrehchi 2018; Ni et al. 2017). In the K-means clustering process, the total number of clusters depends on k, which is the number of elements used to group tests into clusters. This process is illustrated in Fig. 2; which start from extracting data information using k value to cluster data based on similarity. At the end, we get different clusters of unstructured large data to reduce complexity and ambiguity.

  The distance between the test case frequency and the centroid of a cluster is based on the Euclidean distance (ED) as per Eq. (2) (Arafeen and Do 2013; Gultepe and Makrehchi 2018; Ni et al. 2017) where d_{cf} is the ED of test cases t_c and t_f, m is the number of instances, and x is the instance feature value. Clustering was performed using the SPSS tool.

$$d_{cf} = \sqrt{\left(\sum mk{-}1 (X_{ck} - X_{\mathrm{f}k} \right)^2} \quad (2)$$

3. After the test cases have been extracted and clustered, prioritization and selection are performed.

- Generate parameters—First, metrics are generated for use when assigning priorities, the most important of which is the number of frequently failed test cases. The error description is captured from previous test executions and is then used to prioritize test system functionality as follows.

$$F_p = \frac{(\sum TF_i)}{n} \quad (3)$$

where TF_i is the total number of times the test cases failed, and n is the total number of failed test cases. Code coverage criteria are used for prioritization if there are multiple clusters with similar failed frequency percentages. In this case, the percentage of test cases covered is used instead, which is defined as the total number of code lines covered by each test case.

4. Prioritize test cases—In this step, the test clusters are ordered and prioritized to ensure that the most important cases are selected first.

5. Selection—Finally, the test suite is formalized to include all selected test cases from among all of the clusters of large test cases that were identified as having the highest failed frequency or highest coverage.

4 Empirical study

In this section, we investigate the effectiveness of the CTFF model described in Section 3. The efficacy of the CTFF model is assessed by means of a practical assessment focused on the following two research questions (RQs):

RQ1: Does the CTFF able to increase the fault detection rate during RT using clustering frequently changed test cases and previous fault information?

In this RQ, we analyzed that either CTFF model increases the number of faults exposed or not. As most existing studies in the literature did not cluster frequently changed test cases, nor did they initialize the RT process based on test case changes.

RQ2: Is the proposed CTFF technique more effective and efficient as compared to random prioritization and fault-based techniques?

To investigate effectiveness of the CTFF performance in terms of Average Percent of Faults Detected (APFD) and f-measure for test suite prioritization and selection as compared to existing random prioritization and fault-based procedures.

RQ3: Is performance of CTFF technique affected by the size of the test cases particular to changes?

The objective of RQ3 to investigate the impact of selected test case sizes on the efficiency of CTFF fault detection ability as compared to other techniques and to reduce irrelevancy in TC selection and redundant fault detection in test prioritization.

4.1 Industrial systems

Three cases have been selected for performance evaluation of CTFF model. The selection of three different cases was based on scope, domain, and backgrounds. One of the cases is the open-source system, i.e., iTrust and other cases are from real-world industrial application; due to confidentiality reasons, links to the involved companies were omitted and their names were replaced by alphabetical numbering.

Case A (CA) iTrust is a patient centric electronic health record system. It was developed by the Realsearch research group team at North Carolina State University for patients to electronically record health-related information (Arafeen and Do 2013; Hettiarachchi et al. 2016). An iTrust industrial system was used when estimating the performance of the CTFF method. The iTrust system is an open-source application. In terms of the corresponding test suite and test case change information, appropriate test cases (TC) were developed, and other artifacts were generated, such as a traceability matrix, requirement specification, and requirement-modification history, by the developers of iTrust.

Case B (CB) is a web-based IT services application that describes a large information system (we obtained this application from a GR Solutions Private Ltd. company Islamabad). The application was written in Java language and different technical relevant information about construction-based projects. The company providing Case C (CC) is a customer relationship management (CRM) (Azizi and Do 2018) system

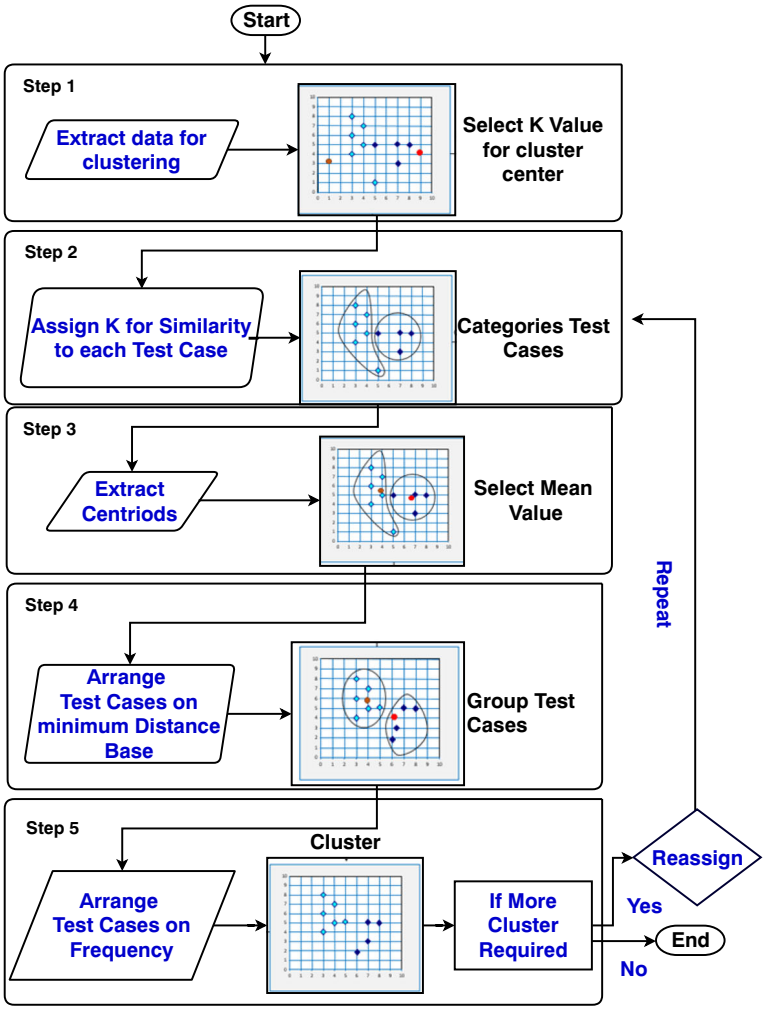

Fig. 2 Clustering of test cases

written in Java language to provide users a customized environment to create their own shapes and diagrams without to writing code. Table 2 shows prioritization criteria for representative applications. The data contains already verified use cases in which no real faults exit so used mutant faults.

Test case change history data is also required and we derived this data from version base of each respective application. The performance test of the CTFF model is based on three releases of each application. A flow chart of this empirical study is shown in Fig. 3; which starts from extracting user stories and recording requirement tasks. Then, extract TCs for clustering and end after selection of optimal set of TCs to execute from large bundle of TCs. The other techniques, i.e., random prioritization (RP) and fault-based (FB) used for effectiveness measurement. The RP prioritize test cases simply with random priority assignment to each test case for execution, and randomly assign to all application test cases for fault

Table 2 Prioritization criteria

Object	Use cases	No. of versions	Size (LOC)	No. of faults	No. of test cases
CA	543	4	3432	21	940
CB	23,018	7	266,456	35	1234
CC	14,541	5	145,987	30	1320

detection. While in FB, historical information about test cases is used, i.e., previous failure rate for priority and selection of test cases.

The extracted requirement specification after conversion into user stories or task cards is shown in Fig. 4, which depict list of some user stories for every release and complete system. A snapshot of test cases that were extracted based on the user stories or tasks cards for current release with complete execution steps are shown in Table 3. The test file consists of different test cases according to each user's story with steps of test case execution and verification at the end of every release.

Note that the test cases were extracted for every release of the system. Then, the average number of changes in the test cases was computed using Eq. (1). For example, $TC-1$ changes 4 times and a total number of releases or modifications are 6; the change frequency is 0.66 times. All those test cases with similar change frequency in a single cluster and test cases with different change frequency placed in different clusters. Therefore, for CA, five clusters were extracted out of 143 TCs in one release.

The similar change frequencies of the test cases were clustered using the SPSS 19 tool. Figure 5 depicts the number of clusters and the number of cases in each cluster

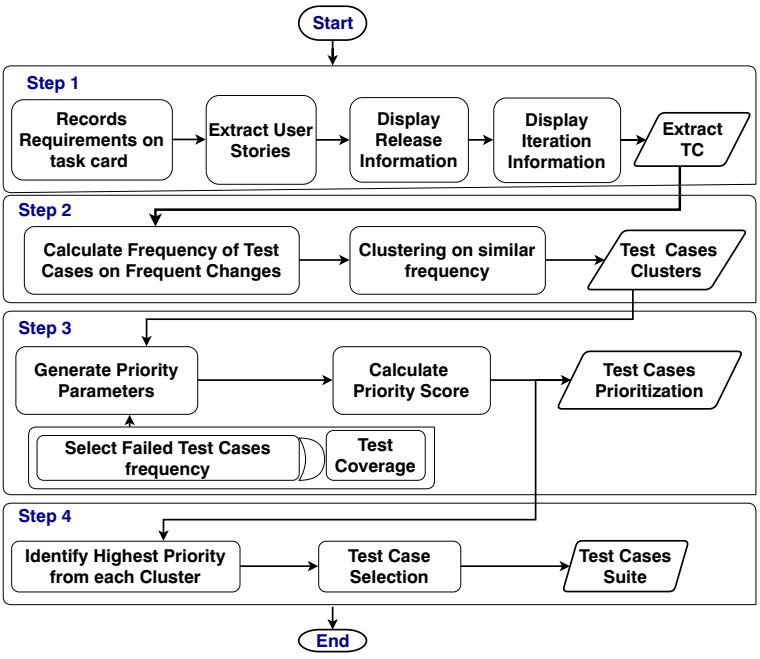

Fig. 3 Case study flow chart

for two releases of CA, while Fig. 6 illustrates the centroids of the clusters to explain that less frequently change TCs placed in cluster one, i.e., C_1 and more frequently change TCs placed in C_5. Parameters were then generated based on the frequently failed tests as the first priority criteria and test coverage as the second. Therefore, for iTrust application, we depict the hierarchy of the clusters after classifying the test cases based on the number of changes, as shown in Fig. 7, where the *x*-axis indicates the details of the test cases and the *y*-axis indicates the numbers of clusters. From Fig. 7, it is clear that five numbers of clusters were created on the base of similarity of change frequency of TCs. For example, in release 1 of 143 test cases included then using SPSS tool (for Eq. (2) calculation) to identify clusters of TCs and in cluster one, i.e., C_1 include $TC-64$, $TC-57$, …, n and same in all clusters based on changed frequency.

4.2 Mutant faults

Mutant faults were introduced into the iTrust code to properly validate fault detection. Mutation testing is a method used to assess the completeness of a test suite by implanting seeded errors into the application. This is used to determine whether the test suite can distinguish these changes at a significantly lower cost than hand-seeded faults. Different mutation operators are utilized, such as arithmetic administrator change (e.g., the addition (+) operator is supplanted with a (−), (*) or (/)), Logical Connector Change (e.g., the AND connector with an OR or XOR connector), Relational Operator Change (e.g., the (>=) operator is replaced with (<=), (==), (!=)), Access Flag Change (e.g., this operator changes a private access flag to a free access flag), Overriding Variable Deletion (erases a declaration of overriding factors), Overriding Variable Insertion (embeds factors from a parent class into the child class), Overriding Method Deletion (erases a declaration of an overriding method in a subclass so that the overridden method is referenced), and Argument Order Change (changes the order of arguments in a method invocation, if there is more than one argument) (Arafeen and Do 2013; Hettiarachchi et al. 2016).

In actual testing scenarios, programs do not typically contain as many faults as these numbers of mutants. Thus, we introduced mutant faults, which were formed by randomly selecting mutants from the pools of mutants created for each version to measure effectiveness of the CTFF model.

SRS	User_Stories_Id	User_Stories_Title
1	US-SR-iTrust-001-01-01	Add Entry Action
2	US-SR-iTrust-001-01-02	Delete Entry Action
3	US-SR-iTrust-001-01-03	Edit Entry Action
4	US-SR-iTrust-001-01-04	Edit Office Visit Base Action
5	US-SR-iTrust-001-01-05	Office Visit Base Action
6	US-SR-iTrust-001-01-06	Patient Base Action

Fig. 4 User stories file

Table 3 Test cases file

Sr. #	User story ID	Test case ID	Test case description	Steps
1. UC-SR-iTrust-001-01 Add Entry Action				
1.	UC-SR-iTrust-001-01-01	UC-SR-iTrust-001-01-01-TC-01	iTrust-Application access	Provide URL in the web browser to access iTrust application
2.	UC-SR-iTrust-001-01-02	UC-SR-iTrust-001-01-01-TC-02	Data entry tab availability to an authorized user	Provide URL in the web browser to access iTrust application Enter user ID of authorized internal user in the login field Enter password Click Login button
2. UC-SR-iTrust-LMS-001-02- Delete Entry Action				
3.	UC-SR-iTrust-001-02-01	UC- SR-iTrust-001-02-01-TC-01	Delete data entry	Provide URL in the web browser to access iTrust application Enter user ID of authorized internal user in the login field Enter password Click Login button Select Menu as MIS Click Delete button

Fig. 5 Clusters

Number of Cases in each Cluster

Cluster	1	21.000
	2	29.000
	3	18.000
	4	61.000
	5	14.000
Valid		143.000
Missing		.000

4.3 Evaluation metrics

The following metrics are used in the evaluation of the proposed approach.

4.3.1 APFD

This metric is used to compute the fault detection rate over the entire prioritized test suite (Azizi and Do 2018; Fischer et al. 2018; Li et al. 2018; Magalhães et al. 2017). The higher the value of the fault detection rate, the earlier the maximum number of faults will be detected during regression testing. This metric can be computed as follows.

$$\mathrm{APFD} = 1 - \frac{(\mathrm{TF}_1 + \mathrm{TF}_2 + \ldots + \mathrm{TF}_m)}{(mn)} + \frac{1}{2n} \tag{4}$$

where TF_i is the number of first test cases in order of execution, m is the total number of faults identified in the program, and n is the number of test cases.

4.3.2 Precision (P)

The P measure indicates the accuracy with which test cases were selected to be rerun (Kandil et al. 2016; Ni et al. 2017; Rosero et al. 2017), and can be computed as follows.

$$P = \frac{|\mathrm{T}'f|}{|\mathrm{T}'f + \mathrm{T}'r|} \tag{5}$$

where $T'f$ represents the set of selected test cases that revealed faults, and $T'r$ represents the test cases that did not reveal faults.

Final Cluster Centers

	Cluster				
	1	2	3	4	5
Tf	.20	.40	.60	.80	1.00

Fig. 6 Clusters centroids

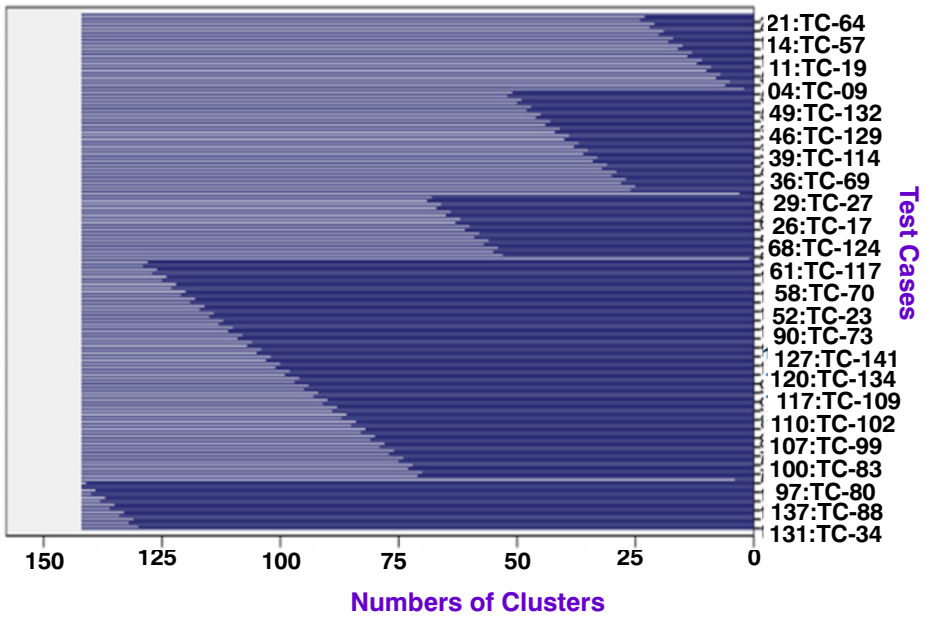

Fig. 7 Hierarchical view of clusters

4.3.3 Recall (R)

The R measure indicates the percentage of selected test cases relative to all failed test cases (Kandil et al. 2016; Ni et al. 2017; Rosero et al. 2017) and can be computed as follows.

$$R = \frac{|\mathrm{T}'f|}{|\mathrm{T}'f + \mathrm{T}'n|}, \tag{6}$$

where $T'n$ represents the set of tests that were not selected and did not fail. A value close to one indicates a high level of accuracy.

4.3.4 F-measure (F)

This measure is a combination of both P and R, which indicates the overall efficiency of the optimal test cases selection process (Kandil et al. 2016; Ni et al. 2017; Rosero et al. 2017). It can be computed as follows.

$$F = \frac{(2 \times P \times R)}{(P + R)} \tag{7}$$

5 Results and discussion

The results of the practical assessment with respect to RQs are as follows.

RQ1: The results of evaluation confirmed that the approach adopted in the CTFF model to cluster frequently changed test cases during process initialization significantly increased the fault detection rate. When the CTFF procedure was applied to the CA,

CB, and CC datasets, test cases with high priority were identified. Identical scenarios were used to assess the performance of random and fault-based ordering techniques for the purpose of comparison. In CTFF technique after applying clustering, we extracted different clusters with similar change frequency of different TC size in all three cases. Then, calculated failed frequency by dividing total number of times failed TC over total time changes (using Eq. (3)). In the next step of CTFF technique, after the selection of Frequently Failed Test Cases (FFTC) from each cluster for CA, CB, and CC, we prioritized TCs in order to highest Failed Test Cases Frequency (FTCF). In case of a tie in FTCF, we used Test Cases Coverage (TCC) as 2nd priority criteria for tie breaking. The detail of priority and frequency criteria of selected TCs are listed in Table 4 of all techniques.

The order of TCs used by various techniques are listed in Table 5 for all selected cases and depicted different TCs for execution to detect maximum faults in first execution. The faults detected by each method are depicted in Fig. 8 with a comparison of other methods in three cases. The *x*-axis indicates the various TC executed while *y*-axis highlights a number of faults detected for that test case.

The results show that the CTFF technique detected almost 100, 98, and 99% of the faults in CA, CB, and CC respectively after first execution of TCs. While the performance of other methods on the first run identified only 20, 30, and 40% for all cases (fault-based) and 40, 20, and 50% for all three cases (random prioritization) of the faults, respectively. Hence, Fig. 8 with box plots explained that most of faults detected earlier in all cases, i.e., CA, CB, and CC using CTFF technique as comparison to RP and FB. In RP and FB required many executions for detecting faults which increases time and cost of development.

RQ2: We investigated the effectiveness of the CTFF with comparison to other methods for all cases using APFD evaluation metric of Eq. (4). The results indicate that APFD for the CTFF model is at higher rate as compared to those of existing methods (i.e., RP and FB) and is shown in Fig. 9. In all three cases, CTFF has higher APFD value which means that CTFF's prioritization criteria is key component and have a higher ability to identify faults than other methods in all three cases.

Table 4 Prioritization criteria

Sr. #	FFTC			FTCF			TCC		
	CA	CB	CC	CA (%)	CB (%)	CC (%)	CA (%)	CB (%)	CC (%)
1	TC-25	TC-08	TC-35	90	85	89	29.5	29.5	29.5
2	TC-27	TC-22	TC-17	90	83	85	10.0	10.0	10.0
3	TC-30	TC-13	TC-20	80	85	85	30.0	30.0	30.0
4	TC-31	TC-33	TC-28	70	60	70	29.0	29.0	29.0
5	TC-18	TC-15	TC-12	60	50	60	25.0	25.0	25.0
6	TC-10	TC-30	TC-31	60	40	50	20.0	20.0	20.0
7	TC-29	TC-29	TC-29	50	40	40	17.1	17.1	17.1
8	TC-13	TC-13	TC-13	40	30	40	20.0	20.0	20.0
9	TC-05	TC-05	TC-05	40	60	30	18.7	18.7	18.7
10	TC-14	TC-14	TC-14	30	40	40	20.0	20.0	20.0
11	TC-01	TC-01	TC-01	20	30	30	18.5	18.5	18.5
12	TC-33	TC-33	TC-33	20	20	20	18.1	18.1	18.1
13	TC-08	TC-08	TC-08	10	20	20	17.3	17.3	17.3

Table 5 Prioritized test suites

Sr. #	Technique		Test cases
1	CTFF approach	CA	{TC-25, TC-30, TC-31, TC-18, TC-29, TC-13, TC-14, TC-01, TC-33, TC-08}
		CB	{TC08, TC22, TC13, TC33, TC15, TC30, TC39, TC43, TC03, TC24, TC21, TC53, TC18}
		CC	{TC35, TC17, TC20, TC28, TC13, TC03, TC19, TC33, TC15, TC12 TC61, TC23, TC18}
2	Random prioritization	CA	{TC-01, TC-03, TC-05, TC-07, TC-09, TC-11, TC-13, TC-15, TC-17, TC-19}
		CB	{TC-05, TC-13, TC-25, TC-37, TC-49, TC-51, TC-63, TC-75, TC-87, TC-99}
		CC	{TC-15, TC-23, TC-25, TC-33, TC-35, TC-43, TC-45, TC-53, TC-55, TC-63}
3	Fault-based	CA	{TC-29, TC-19, TC-18, TC-17, TC-16, TC-15, TC-14, TC-13, TC-12, TC-11}
		CB	{TC-25, TC-03, TC-35, TC-13, TC-15, TC-34, TC-65, TC-23, TC-75, TC-43}
		CC	{TC-11, TC-12, TC-21, TC-03, TC-35, TC-14, TC-65, TC-35, TC-25, TC-63}

The results of the performance analysis are shown in Table 6 and Figs. 10, where it can be seen that CTFF approach successfully identified an optimal set of test cases and increased the fault detection capability. As shown, the precision, recall, and *F* measure values using Eqs. 5, 6, and 7 respectively are calculated for the CTFF efficiency analysis. The results indicate that test cases selected using CTFF have higher efficiency (i.e. 0.95, 0.92, and 0.96 points *F* measure for CA, CB, and CC respectively) than those of random prioritization (i.e., 0.56, 0.69, and 0.52 points *F* measure for CA, CB, and CC respectively) and fault-based methods (i.e., 0.68, 0.67, and 0.67 points *F* measure for CA, CB, and CC respectively) in CA, CB, and CC. Figure 10 a, b, and c for CA, CB, and CC respectively show that CTFF is more efficient than other methods. Consequently, results proved that selected test suite in CTFF for CA, CB, and CC is optimal set of TCs with a higher degree of fault detection. Whereas, using RP and FB techniques for CA, CB, and CC demonstrate less degree of fault detection ability.

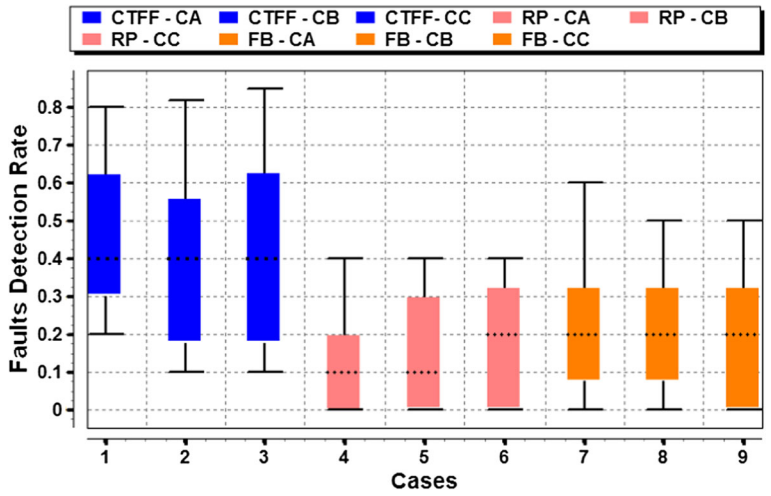

Fig. 8 Comparison of the techniques

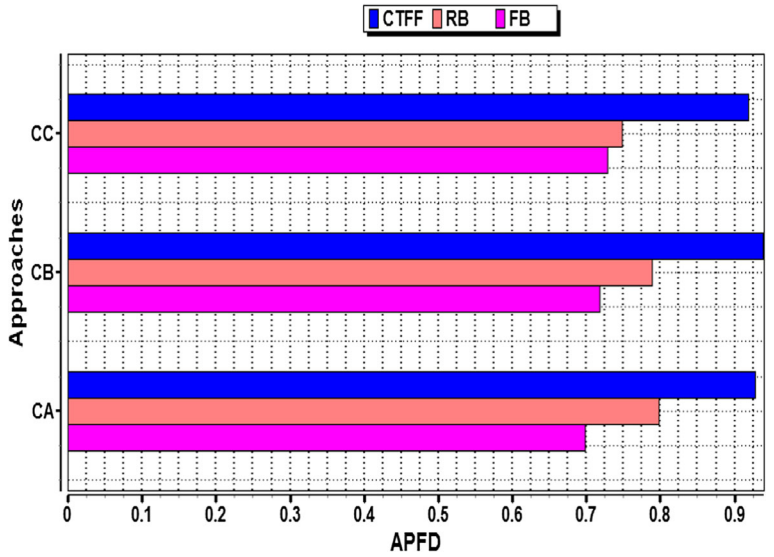

Fig. 9 APFD of all techniques

RQ3: In this, evaluated impact of size of failed test case selection for execution on irrelevancy and redundant faults using CTFF in all three cases. Further, compared results with other techniques to provide more evidence about size variation sensitivity to irrelevancy and redundancy. However, size variation for test suite execution was not the same for each technique in all cases. The CTFF is less sensitive to test suite size for prioritization and selection as compared to RP and FB. For test suite size variation reduction, we divided total number failed TCs selected and prioritized over total TCs failed. For large code coverage, we divide number of lines of code covered by failed selected TCs for execution over total number of lines of code covered by each TC. The results demonstrate that CTFF technique has no issue of irrelevant TC selection and redundant faults after prioritization as shown in Fig. 11. The *y*-axis depict the relative percentage of TC to size variation for all three techniques on *x*-axis. Whereas, results for FB and RP indicate that these are more affected and have an issue of irrelevant TC selection for execution with redundant fault detection after the execution of TC.

Table 6 Evaluation metric analysis

Type	Cases	CTFF	Random prioritization	Fault-based
Precision	CA	0.92	0.48	0.68
Recall	CA	1	0.70	0.70
F measure	CA	0.958333	0.569492	0.689855
Precision	CB	0.90	0.46	0.65
Recall	CB	0.99	0.72	0.7
F measure	CB	0.928125	0.69000	0.67407
Precision	CC	0.92	0.45	0.63
Recall	CC	1	0.68	0.71
F measure	CC	0.964439	0.523076	0.6767

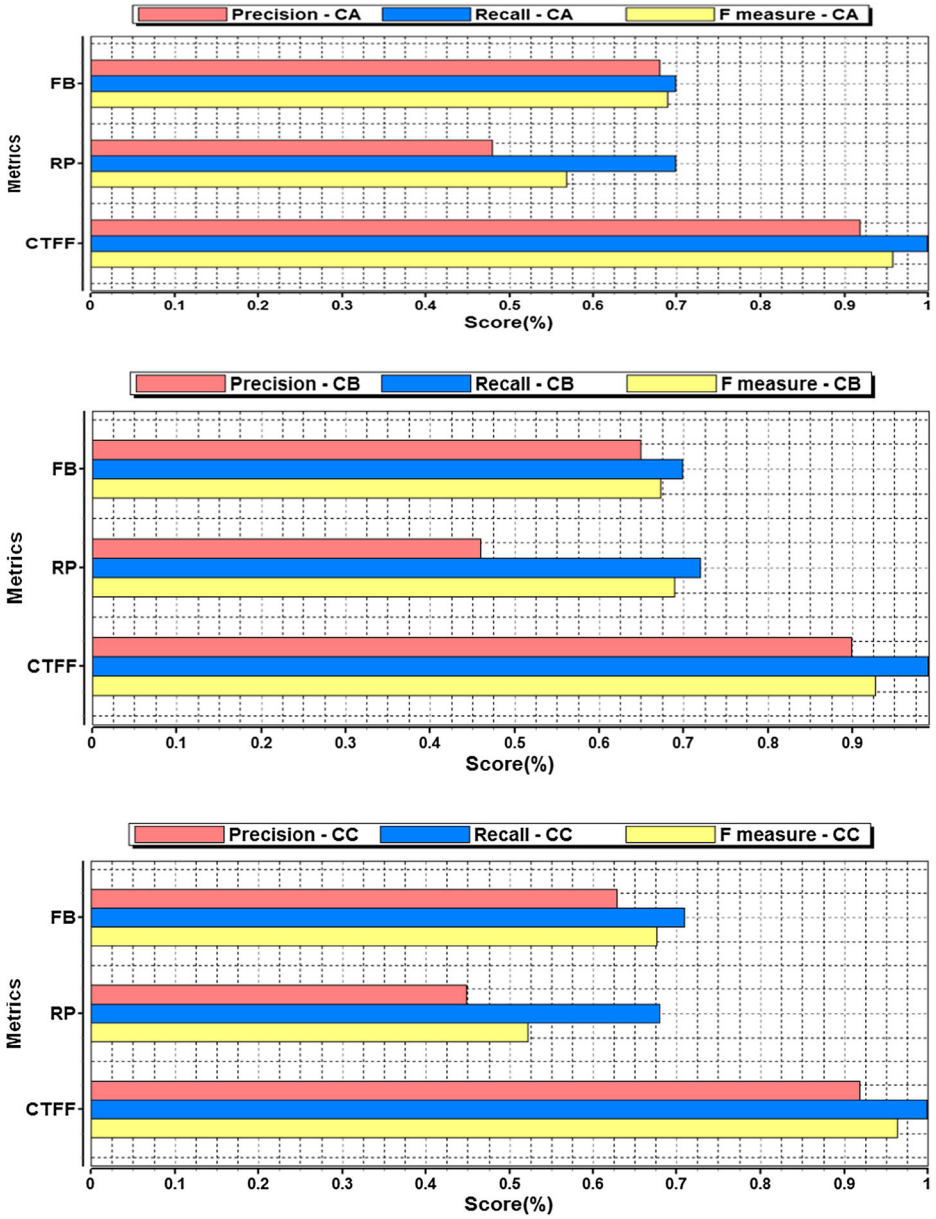

Fig. 10 Performance analysis for CA, CB and CC

5.1 Threats to validity

As is typical when evaluating case studies, several threats arise that dispute the theoretical rationality of the results, which necessitates the repetition of the research to approve or refute decisions. The core threats can be categorized as follows (Felderer and Herrmann 2019; Miranda and Bertolino 2018; Ouriques et al. 2018):

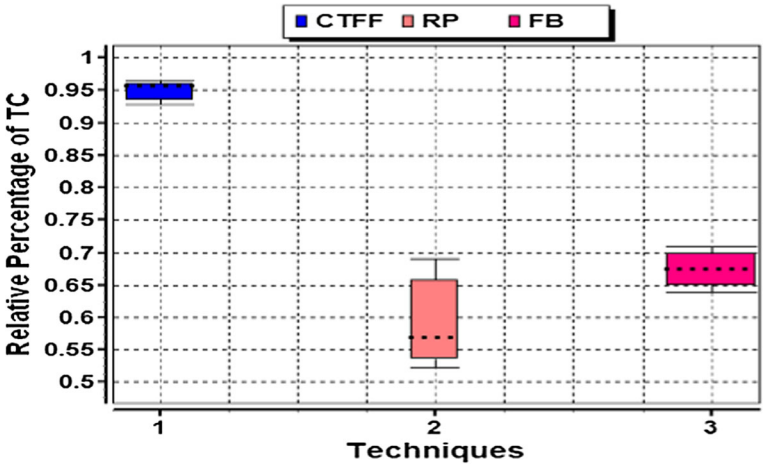

Fig. 11 Variation analysis

- Internal TV related to factors regarding the arrangement of requirements. To address this threat, mitigation steps must be adopted to avoid using diverse criteria for ordering and selection. Therefore, we used test case change frequency and failed frequency to identify higher fault detection rate ability to reduce irrelevancy and redundancy in test selection and fault detection respectively. Experimental results proved that CTFF has improved fault detection ability and detect maximum faults as earlier as possible.
- Construct TV considers the connections between the various concepts and reflections. This requires the use of evaluation metrics to assess the validity of the diverse practices in the CTFF model. Therefore, we used APFD metric for prioritization and F measure for selection to estimate the effectiveness of CTFF as compared to other techniques.
- Conclusion TV relates to the associations between action and consequence. This can be mitigated via a rigorous practical assessment of the various decisions employed in CTFF authentication. Then, a case study can be used to defend the decisions via a qualitative analysis to reduce the bias. All authors participated in evaluation for data collection and counteractions on analysis of results.
- External TV relates to the generality of the verdicts in real industrial projects. This enhances the validity of the conclusions and fosters more investigation in the relevant domains by replicating the results of the research in diverse situations. Therefore, to avoid this TV, we used three different subjects as a case study for evaluation so that results can be validated in diverse domains.

6 Conclusions and future work

This study presents an approach to resolve challenges in regression testing when supporting continuous integration activities in modern development strategies. Several techniques that have been proposed in the literature failed to increase fault detection and identification rate as they did not exclude redundant faults or irrelevant test cases during execution. In fact, most existing techniques rely on code coverage or historical information when selecting and ranking test cases, which ignores faulty test cases.

To address the limitations in existing methods, we have proposed the CTFF model that ranks and selects test cases by first clustering the test cases that frequently change. In the case of a tie, test cases are prioritized based on the number of frequently failed test cases and coverage criteria. Thus, CTFF improves the regression testing for agile software projects specifically and provides significant implication for a software organization. For the implementation of proposed techniques, we investigated three software application, each application based on different versions.

The outcome of the evaluation shows the following:

- The proposed technique significantly improved fault detection rate (i.e., more than 90%) at earlier stages as compared to other techniques (less than 50% in RP and FB techniques).
- The results of evaluation metrics illustrate that the proposed technique significantly outperforms (i.e., between 91 and 97%) as compared to other existing techniques (i.e., between 52 and 68%) to avoid irrelevancy and redundancy of test cases and faults respectively.
- The results also described that frequently change and failed test case criteria in regression testing is significant for fault detection, reduces irrelevancy in test case selection, redundancy in faults, and reduces the need of maintaining large historical information.

Furthermore, the empirical evaluation results show that the CTFF model has a high fault detection rate as well as the ability to identify the maximum number of faults rapidly. Additionally, a limitation of our study is that we presented different case studies for evaluation based on already collected datasets and results are not statistically verified which are usually not adopted in a case study. Therefore, to mitigate this limitation, we need experimental evaluation to analyze data using statistical analysis for reliability.

In future work, we are planning to extend the CTFF model to resolve regression testing constraints in component-based software and product line engineering applications. We are also planning to investigate additional research questions for controlled experiment-based evaluation for future research. For instance, we may find a correlation among different metrics to guide quality engineers and researchers for selection of the best suitable regression testing techniques in different scenarios and environment for optimal reliability analysis.

Funding information The work reported in this paper was supported by the National Natural Science Foundation of China (Grant No. 61672080).

Compliance with ethical standards

Conflict of interest The authors declare that there is no conflict of interest.

References

Abu Hasan, M., Abdur Rahman, M., & Saeed Siddik, M. (2017). Test case prioritization based on dissimilarity clustering using historical data analysis. *Information, Communication and Computing Technology, 750*, 269–281. https://doi.org/10.1007/978-981-10-6544-6_25.

Agren, S. M., Knauss, E., Heldal, R., Pelliccione, P., Malmqvist, G., & Boden, J. (2018). The manager perspective on requirements impact on automotive systems development speed. *2018 IEEE 26th International Requirements Engineering Conference (RE)*, 17–28. https://doi.org/10.1109/RE.2018.00-55.

Al-Hajjaji, M., Thüm, T., Lochau, M., Meinicke, J., & Saake, G. (2019). Effective product-line testing using similarity-based product prioritization. *Software and Systems Modeling, 18*(1), 499–521. https://doi.org/10.1007/s10270-016-0569-2.

Alkharabsheh, K., Crespo, Y., Manso, E., & Taboada, J. A. (2018). Software design smell detection: a systematic mapping study. *Software Quality Journal*. https://doi.org/10.1007/s11219-018-9424-8.

Almasri, N., Tahat, L., & Korel, B. (2017). Toward automatically quantifying the impact of a change in systems. *Software Quality Journal, 25*(3), 601–640. https://doi.org/10.1007/s11219-016-9316-8.

Aman, H., Nakano, T., Ogasawara, H., & Kawahara, M. (2018). A topic model and test history-based test case recommendation method for regression testing. 2018 IEEE international conference on software testing, verification and validation workshops (ICSTW), 392–397. https://doi.org/10.1109/ICSTW.2018.00079.

Anand, R. V., & Dinakaran, M. (2017). Handling stakeholder conflict by agile requirement prioritization using Apriori technique. *Computers and Electrical Engineering, 61*, 126–136. https://doi.org/10.1016/j.compeleceng.2017.06.022.

Anderson, J., Salem, S., & Do, H. (2014). Improving the effectiveness of test suite through mining historical data. *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*, 142–151. https://doi.org/10.1145/2597073.2597084.

Anita, & Chauhan, N. (2014). A regression test selection technique by optimizing user stories in an agile environment. *IEEE International Advance Computing Conference (IACC), 2014*, 1454–1458. https://doi.org/10.1109/IAdCC.2014.6779540.

Ansari, A., Khan, A., Khan, A., & Mukadam, K. (2016). Optimized regression test using test case prioritization. *Procedia Computer Science, 79*, 152–160. https://doi.org/10.1016/j.procs.2016.03.020.

Arafeen, M. J., & Do, H. (2013). Test case prioritization using requirements-based clustering. 2013 IEEE sixth international conference on software testing, verification and validation, 312–321. https://doi.org/10.1109/ICST.2013.12.

Azizi, M., & Do, H. (2018). A collaborative filtering recommender system for test case prioritization in web applications. *Proceedings of the 33rd Annual ACM Symposium on Applied Computing - SAC '18*, 1560–1567. https://doi.org/10.1145/3167132.3167299.

Chen, J., Zhu, L., Chen, T. Y., Towey, D., Kuo, F.-C., Huang, R., & Guo, Y. (2018). Test case prioritization for object-oriented software: an adaptive random sequence approach based on clustering. *Journal of Systems and Software, 135*, 107–125. https://doi.org/10.1016/j.jss.2017.09.031.

del Sagrado, J., & del Águila, I. M. (2018). Stability prediction of the software requirements specification. *Software Quality Journal, 26*(2), 585–605. https://doi.org/10.1007/s11219-017-9362-x.

Do, H. (2016). Recent advances in regression testing techniques. In *Advances in Computers* (Vol. 103, pp. 53–77). https://doi.org/10.1016/bs.adcom.2016.04.004.

Elbaum, S., Rothermel, G., & Penix, J. (2014). Techniques for improving regression testing in continuous integration development environments. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014*, 235–245. https://doi.org/10.1145/2635868.2635910.

Felderer, M., & Herrmann, A. (2019). Comprehensibility of system models during test design: a controlled experiment comparing UML activity diagrams and state machines. *Software Quality Journal, 27*(1), 125–147. https://doi.org/10.1007/s11219-018-9407-9.

Fischer, S., Lopez-Herrejon, R. E., & Egyed, A. (2018). Towards a fault-detection benchmark for evaluating software product line testing approaches. *Proceedings of the 33rd Annual ACM Symposium on Applied Computing - SAC '18*, 2034–2041. https://doi.org/10.1145/3167132.3167350.

Flemström, D., Potena, P., Sundmark, D., Afzal, W., & Bohlin, M. (2018). Similarity-based prioritization of test case automation. *Software Quality Journal, 26*(4), 1421–1449. https://doi.org/10.1007/s11219-017-9401-7.

Gultepe, E., & Makrehchi, M. (2018). Improving clustering performance using independent component analysis and unsupervised feature learning. *Human-centric Computing and Information Sciences, 8*(1), 25. https://doi.org/10.1186/s13673-018-0148-3.

Gupta, A., Mishra, N., Tripathi, A., Vardhan, M., & Kushwaha, D. S. (2015). An improved history-based test prioritization technique using code coverage. In *Advanced Computer and Communication Engineering Technology* (Vol. 315, pp. 437–448). https://doi.org/10.1007/978-3-319-07674-4_43.

Haghighatkhah, A., Mäntylä, M., Oivo, M., & Kuvaja, P. (2018). Test prioritization in continuous integration environments. *Journal of Systems and Software, 146*, 80–98. https://doi.org/10.1016/j.jss.2018.08.061.

Heck, P., & Zaidman, A. (2018). A systematic literature review on quality criteria for agile requirements specifications. *Software Quality Journal, 26*(1), 127–160. https://doi.org/10.1007/s11219-016-9336-4.

Hettiarachchi, C., Do, H., & Choi, B. (2016). Risk-based test case prioritization using a fuzzy expert system. *Information and Software Technology, 69*, 1–15. https://doi.org/10.1016/j.infsof.2015.08.008.

Horváth, F., Gergely, T., Beszédes, Á., Tengeri, D., Balogh, G., & Gyimóthy, T. (2019). Code coverage differences of Java bytecode and source code instrumentation tools. *Software Quality Journal, 27*(1), 79–123. https://doi.org/10.1007/s11219-017-9389-z.

Huang, Y.-C., Peng, K.-L., & Huang, C.-Y. (2012). A history-based cost-cognizant test case prioritization technique in regression testing. *Journal of Systems and Software, 85*(3), 626–637. https://doi.org/10.1016/j.jss.2011.09.063.

Kandil, P., Moussa, S., & Badr, N. (2014). Regression testing approach for large-scale systems. *IEEE International Symposium on Software Reliability Engineering Workshops, 2014*, 132–133. https://doi.org/10.1109/ISSREW.2014.96.

Kandil, P., Moussa, S., & Badr, N. (2015). A methodology for regression testing reduction and prioritization of agile releases. 2015 5th international conference on Information & Communication Technology and accessibility (ICTA), 1–6. https://doi.org/10.1109/ICTA.2015.7426903.

Kandil, P., Moussa, S., & Badr, N. (2016). Cluster-based test cases prioritization and selection technique for agile regression testing: cluster-based technique for agile regression testing. *Journal of Software: Evolution and Process, 29*(6), e1794. https://doi.org/10.1002/smr.1794.

Knauss, E., Staron, M., Meding, W., Soder, O., Nilsson, A., & Castell, M. (2015). Supporting continuous integration by code-churn based test selection. *2015 IEEE/ACM 2nd International Workshop on Rapid Continuous Software Engineering*, 19–25. https://doi.org/10.1109/RCoSE.2015.11.

Lachmann, R., Lity, S., Lischke, S., Beddig, S., Schulze, S., & Schaefer, I. (2015). Delta-oriented test case prioritization for integration testing of software product lines. Proceedings of the 19th international conference on software product line - SPLC '15, 81–90. https://doi.org/10.1145/2791060.2791073.

Li, X., Wong, W. E., Gao, R., Hu, L., & Hosono, S. (2018). Genetic algorithm-based test generation for software product line with the integration of fault localization techniques. *Empirical Software Engineering, 23*(1), 1–51. https://doi.org/10.1007/s10664-016-9494-9.

Lin, C.-T., Chen, C.-D., Tsai, C.-S., & Kapfhammer, G. M. (2013). History-based test case prioritization with software version awareness 2013 18th international conference on engineering of complex computer systems, 171–172. https://doi.org/10.1109/ICECCS.2013.33.

Lu, Y., Lou, Y., Cheng, S., Zhang, L., Hao, D., Zhou, Y., & Zhang, L. (2016). How does regression test prioritization perform in real-world software evolution? *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*, 535–546. https://doi.org/10.1145/2884781.2884874.

Ma, C., & Provost, J. (2017). A model-based testing framework with reduced set of test cases for programmable controllers. *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, 944–949. https://doi.org/10.1109/COASE.2017.8256225.

Ma, T., Ali, S., Yue, T., & Elaasar, M. (2019). Testing self-healing cyber-physical systems under uncertainty: a fragility-oriented approach. *Software Quality Journal, 27*(2), 615–649. https://doi.org/10.1007/s11219-018-9437-3.

Magalhães, C., Andrade, J., Perrusi, L., & Mota, A. (2017). Evaluating an automatic text-based test case selection using a non-instrumented code coverage analysis. *Proceedings of the 2nd Brazilian Symposium on Systematic and Automated Software Testing - SAST*, 1–9. https://doi.org/10.1145/3128473.3128478.

Mahali, P., & Mohapatra, D. P. (2018). Model based test case prioritization using UML behavioural diagrams and association rule mining. *International Journal of Systems Assurance Engineering and Management, 9*(5), 1063–1079. https://doi.org/10.1007/s13198-018-0736-7.

Miranda, B., & Bertolino, A. (2017). Scope-aided test prioritization, selection and minimization for software reuse. *Journal of Systems and Software, 131*, 528–549. https://doi.org/10.1016/j.jss.2016.06.058.

Miranda, B., & Bertolino, A. (2018). An assessment of operational coverage as both an adequacy and a selection criterion for operational profile based testing. *Software Quality Journal, 26*(4), 1571–1594. https://doi.org/10.1007/s11219-017-9388-0.

Ni, C., Liu, W.-S., Chen, X., Gu, Q., Chen, D.-X., & Huang, Q.-G. (2017). A cluster based feature selection method for cross-project software defect prediction. *Journal of Computer Science and Technology, 32*(6), 1090–1107. https://doi.org/10.1007/s11390-017-1785-0.

Noor, T. B., & Hemmati, H. (2015). A similarity-based approach for test case prioritization using historical failure data. *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, 58–68. https://doi.org/10.1109/ISSRE.2015.7381799.

Ouni, A., Kessentini, M., ÓCinnéide, M., Sahraoui, H., Deb, K., & Inoue, K. (2017). MORE: a multi-objective refactoring recommendation approach to introducing design patterns and fixing code smells: MORE: a multi-objective refactoring recommendation approach. *Journal of Software: Evolution and Process, 29*(5), e1843. https://doi.org/10.1002/smr.1843.

Ouriques, J. F. S., Cartaxo, E. G., & Machado, P. D. L. (2018). Test case prioritization techniques for model-based testing: a replicated study. *Software Quality Journal, 26*(4), 1451–1482. https://doi.org/10.1007/s11219-017-9398-y.

Özdağoğlu, G., & Kavuncubaşı, E. (2019). Monitoring the software bug-fixing process through the process mining approach. Journal of Software: Evolution and Process, e2162. https://doi.org/10.1002/smr.2162.

Panichella, A., Oliveto, R., Penta, M. D., & De Lucia, A. (2015). Improving multi-objective test case selection by injecting diversity in genetic algorithms. *IEEE Transactions on Software Engineering, 41*(4), 358–383. https://doi.org/10.1109/TSE.2014.2364175.

Rosero, R. H., Gómez, O. S., & Rodríguez, G. (2016). 15 years of software regression testing techniques — a survey. *International Journal of Software Engineering and Knowledge Engineering, 26*(05), 675–689. https://doi.org/10.1142/S0218194016300013.

Rosero, R. H., Gomez, O. S., & Rodriguez, G. (2017). Regression testing of database applications under an incremental software development setting. *IEEE Access, 5*, 18419–18428. https://doi.org/10.1109/ACCESS.2017.2749502.

Shin, S. Y., Nejati, S., Sabetzadeh, M., Briand, L. C., & Zimmer, F. (2018). Test case prioritization for acceptance testing of cyber physical systems: a multi-objective search-based approach. *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis - ISSTA 2018*, 49–60. https://doi.org/10.1145/3213846.3213852.

Silva, D., Rabelo, R., Campanha, M., Neto, P. S., Oliveira, P. A., & Britto, R. (2016). A hybrid approach for test case prioritization and selection. *IEEE Congress on Evolutionary Computation (CEC), 2016*, 4508–4515. https://doi.org/10.1109/CEC.2016.7744363.

Souto, S., & d'Amorim, M. (2017). Time-space efficient regression testing for configurable systems. *ArXiv: 1702.03457 [Cs]*. Retrieved from http://arxiv.org/abs/1702.03457

Spieker, H., Gotlieb, A., Marijan, D., & Mossige, M. (2017). Reinforcement learning for automatic test case prioritization and selection in continuous integration. *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis - ISSTA 2017*, 12–22. https://doi.org/10.1145/3092703.3092709.

Thangiah, M., & Basri, S. (2016). A preliminary analysis of various testing techniques in agile development - a systematic literature review. 2016 3rd international conference on computer and information sciences (ICCOINS), 600–605. https://doi.org/10.1109/ICCOINS.2016.7783283.

Wang, X., & Zeng, H. (2014). Dynamic test case prioritization based on multi-objective. 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 1–6. https://doi.org/10.1109/SNPD.2014.6888744.

Wang, X., & Zeng, H. (2016). History-based dynamic test case prioritization for requirement properties in regression testing. *Proceedings of the International Workshop on Continuous Software Evolution and Delivery - CSED '16*, 41–47. https://doi.org/10.1145/2896941.2896949.

Wang, X., Zeng, H., Gao, H., Miao, H., & Lin, W. (2019). Location-based test case prioritization for software embedded in mobile devices using the law of gravitation. *Mobile Information Systems, 2019*, 1–14. https://doi.org/10.1155/2019/9083956.

Zhao, D., Lin, H., Ran, L., Han, M., Tian, J., Lu, L., & Xiang, J. (2019). CVSkSA: cross-architecture vulnerability search in firmware based on kNN-SVM and attributed control flow graph. *Software Quality Journal*. https://doi.org/10.1007/s11219-018-9435-5.

Sadia Ali is a PhD student at the University Institute of Information Technology (UIIT), PMAS Arid Agriculture University, Rawalpindi, Pakistan, under the supervision of Dr. Yaser Hafeez (UIIT). She received her Master of Science in Computer Science (MSCS) from PMAS Arid Agriculture University, Rawalpindi, Pakistan, in 2017. Her current research interests mainly focus on developing requirements engineering and testing solutions for component-based systems.

Yaser Hafeez is an Associate Professor at the University Institute of Information Technology, PMAS Arid Agriculture University, Rawalpindi, Pakistan. He holds a PhD degree in Software Engineering. His research interests include requirements engineering process, global software engineering, agile software development practices, software quality engineering, and computing engineering.

Shariq Hussain received the master's degree in computer science from PMAS Arid Agriculture University, Rawalpindi, Pakistan, in 2007, and the Ph.D. degree in Applied Computer Technology from the University of Science and Technology Beijing, Beijing, China, in 2014. Since 2014, he has been with the Department of Software Engineering, Foundation University Islamabad, Rawalpindi Campus, where he is currently an Assistant Professor. His main research interests include web services, QoS in web services, web service testing, IoT, context awareness, and e-learning. He served as Workshop/Special Session Co-chair for IEEE International Conference on Internet of People 2018 (IoP 2018) and publicity chair for IEEE International Conference on Internet of People (IoP 2015). He is also serving as Guest Editor for SN Applied Sciences (Special Collection: IOT in Mobile, Big Data Analytics and Cloud Computing) and editorial board member of Journal of Next Generation Information Technology (JNIT) - AICIT, Rep. of S. Korea.

Shunkun Yang received his B.S., M.S., and Ph.D. degrees from the School of Reliability and Systems Engineering at Beihang University in 2000, 2003, and 2011, respectively. He is an associate research professor at Beihang University since 2016. He is also an associate research scientist of Columbia University through September 2014 to September 2015. His main research interests are reliability, testing and diagnosis for embedded software, CPS, IoT, Intelligent manufacturing, etc.

Affiliations

Sadia Ali[1] · Yaser Hafeez[1] · Shariq Hussain[2] · Shunkun Yang[3]

Sadia Ali
sadiaalief@gmail.com

Yaser Hafeez
yasir@uaar.edu.pk

Shariq Hussain
shariq@fui.edu.pk

[1] University Institute of Information Technology, Pir Mehr Ali Shah Arid Agriculture University, Rawalpindi, Pakistan

[2] Department of Software Engineering, Foundation University Islamabad, Rawalpindi Campus, Rawalpindi, Pakistan

[3] School of Reliability and Systems Engineering, Beihang University, Beijing, China