A Novel Risk-Based Testing Framework for Distributed Agile Software Development

Esha Khanna
Department of Computer Engineering
J. C. Bose University of Science and Technology YMCA, India
Department of Computer Science and Technology Manav Rachna University, India
eshakhanna30@gmail.com

Rashmi Popli
Department of Computer Engineering
J. C. Bose University of Science and Technology, YMCA, India
rashmipopli@gmail.com

Naresh Chauhan
Department of Computer Engineering
J. C. Bose University of Science and Technology, YMCA, India
nareshchauhan19@gmail.com

Abstract: *Distributed Agile Software Development (DASD) is one of the most commonly adopted lifecycle methodologies in the IT industry. It combines the speed benefits and adaptability of agile development with the cost-effectiveness of distributed development. In DASD, software is built iteratively in sprints, allowing for changes to be introduced in the later stages of development. However, such changes can affect the functionality of previously implemented features, highlighting the critical need for regression testing. In distributed agile environments, constraints on resources, time, and communication make it essential to prioritize tasks to maximise the efficiency of testing efforts. This work presents a novel risk-based Test Case Prioritization (TCP) approach for a distributed agile environment that aims to prioritize test cases based on the risk values associated with features by correlating sprint features with their corresponding test cases. The risk value of a feature is evaluated by considering modified requirements, feature complexity, and interdependencies. The goal is to identify and test high-risk features early in the distributed agile development cycle, thereby uncovering critical defects earlier. The proposed work is evaluated using an empirical study. The results show that the proposed technique has outperformed the existing state-of-the-art techniques.*

Keywords: *Test case prioritization, risk-based testing, distributed agile software development.*

Received April 18, 2025; accepted July 31, 2025
https://doi.org/10.34028/iajit/22/6/14

1. Introduction

In a distributed agile environment, where teams are geographically dispersed and development cycles are fast-paced, the need for regression testing becomes paramount [27]. However, in distributed agile setups, resources, time, and communication can be limited, necessitating prioritization to optimise testing efforts [30]. Prioritizing regression test cases allows the teams to focus on critical functionalities first, ensuring that high-impact areas are thoroughly tested within the constraints of distributed collaboration [6]. This approach not only maximises test coverage but also minimises the risk of regression issues slipping into production [21].

Over the past several years, various Test Case Prioritization (TCP) techniques have been proposed and empirically studied to identify faults at early stages, thereby increasing the fault detection rate [5, 13, 21, 26, 31, 34, 35, 42]. Existing methods include history-based [16], coverage-based [26], fault severity-based [25], and requirement-based [14, 37], among others. Some work combines the existing methods to attain better results [25]. However, these techniques typically prioritize test cases according to past test execution data and existing fault information, although such data may not always be accessible. Additionally, these methods often neglect the crucial task of identifying defects early in high-risk modules, despite the significant importance of concentrating on such areas to offset resource limitations.

On the contrary, Risk-Based Testing (RBT) directs attention to the segments of software posing the greatest risk [15]. Its objective is the early identification of the most critical faults within the risky modules, while keeping costs to a minimum. This approach empowers testers to prevent or mitigate the occurrence of faults that could result in significant damage [3, 17, 40]. However, the existing risk-based TCP techniques do not consider all the risk factors associated with the DASD environment.

This work proposes a novel risk-based TCP approach that correlates the test cases with features in a given sprint. This approach also takes into account the 10 risk categories identified for the DASD environment [19].

The goals of the paper are as follows-

- To propose a method to calculate risk values associated with features in a sprint.
- To propose a novel RBT framework by correlating test cases with features and assigning higher priority to test cases covering high-risk features.
- To carry out an empirical study to evaluate the efficiency of the framework.
- To compare the results of the study with state-of-the-

art techniques.

The paper is organised as follows. Section 2 presents the background, section 3 presents the proposed model, section 4 presents the empirical study, and section 5 concludes.

2. Background

Distributed Agile Software Development (DASD) has emerged as a prominent Software Development Life Cycle (SDLC) model after the COVID-19 pandemic. DASD combines the agility of Agile Software Development (ASD) with the speed benefits of Distributed Software Development (DSD) [2, 20]. ASD methodology is centred around the principles and practices of agile frameworks, which focus on iterative development, collaboration between cross-functional teams, flexibility in responding to change, and continuous improvement [38]. In an agile development scenario, the software is developed in sprints [7]. A sprint is a small timeframe (usually of 2-4 weeks) in which the agile team focuses on releasing an increment. In the scrum framework, the agile team consists of the product owner, scrum master and a development team including testers, designers and User Experience (UX) specialists [23]. The product owner is responsible for prioritising the backlog on the basis of feedback from users and the development team. In a sprint, there is a collection of ceremonies that take place to mark the milestone of a sprint. The first meeting is called sprint planning, which is led by the scrum master. In this meeting, the team decides the goal of the sprint. A sprint backlog is prepared by adding the stories from the product backlog, which reflects the goal of the sprint. After the release increment is decided, the team starts working on the user stories of a sprint. A short meeting is carried out daily during the sprint, which helps every team member to be aligned with the sprint goal. At the end of the sprint, a sprint review meeting is carried out to discuss the completion of the goal of the sprint. The product owner then takes a call to release the increment. The product owner then reprioritizes the product backlog based on the current sprint [38].

During the next sprint, feedback from the users and development team is incorporated into the first release. This may include the addition of new functionality or modification to the existing feature. The changes incorporated in the released version endanger the correct working of the existing features, that has been inherited from the previous version. This calls for retesting the next increment with all the test suites of the first and next increment to verify that new changes have not altered the working of previously inherited modules. This testing is known as regression testing [27, 11]. As the agile methodology accepts the changes during the lifecycle of a product, regression testing has to be carried out multiple times in each sprint in Scrum. Re-executing all the test cases of the regression suite is a time-consuming task; therefore, the test cases are reordered such that the most important test cases are executed before the others. TCP is a type of regression testing that reorders the test cases on the basis of some factors [21, 26, 27]. These factors include code coverage, branch coverage, method coverage, history-based, and risk-based [13, 31, 35].

A test case covering maximum program code maximises the likelihood of fault detection. Coverage-based techniques order the test cases of a test suite in such a way that a test case having more coverage is assigned a higher priority than a test case covering a lower priority. Figure 1 presents different coverage-based techniques, including statement coverage, branch coverage, function coverage, and fault index prioritization [21, 36, 42].

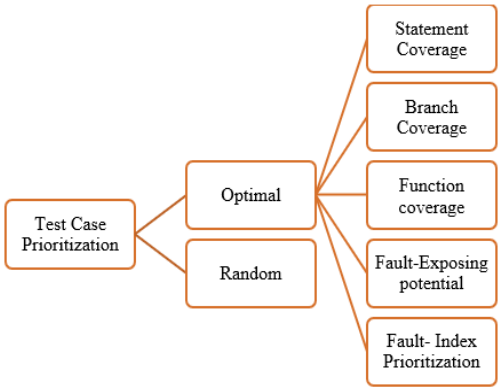

Figure 1. Coverage-based TCP techniques.

In the literature, several authors have proposed and empirically evaluated regression testing techniques. In the work by Singh *et al*. [33], a novel test case reduction method was proposed, combined with a Support-based Whale Optimisation Algorithm (SWOA) for DASD. However, the limitation of test case reduction is that it may exclude potentially critical test cases, leading to reduced test coverage.

Fault-aware TCP has been used in recent works. Sugave *et al*. [39] proposed Fault-aware TCP in software testing using Jaya archimedes optimisation algorithm. Garg and Shekha [10] proposed a method that combines the Fault Sensitivity Index (FSI) with a ranking-based Genetic Algorithm to prioritize test cases. Gupta *et al*. [12] aimed to enhance fault detection rates and coverage efficacy in both unit and integration testing phases, based on Shuffled Frog-Leaping Algorithm (SFLA). However, the limitation of fault awareness techniques is that existing Fault awareness information may not always be available.

Kumar and Saxena [22] proposed cost cost-based technique for TCP using the Hungarian algorithm. However, the Cost associated with a particular test case may not always be available. The method assumes static test cases and may not adapt well to environments where test cases frequently change or evolve.

Junaid *et al*. [18] investigated the application of history-based TCP techniques in regression testing. It identifies the problem of equal priority assignments in

history-based TCP and proposes random sorting as a potential solution. However, the prioritization technique is static and doesn't adapt dynamically to new failures or evolving test conditions. This makes it less suitable for DASD projects.

Blockchain has also been used for TCP [9, 29]. However, lots of power is required for mining blocks, which leads to high energy consumption. Moreover, data modification in blocks is a major concern.

Traditional TCP techniques often rely on historical test execution data and known fault information. However, these sources of data might not always be available.

RBT focuses on prioritizing segments of the software that pose the highest risk. Its goal is to identify critical faults within these high-risk areas as early as possible, minimising costs while preventing or mitigating significant damage [15].

Despite its advantages, current RBT techniques often fail to account for risk factors specific to the DASD environment. This work proposes a novel risk-based TCP method by correlating the test cases with the risk values of features, considering the risk categories of DASD. In the existing literature, several feature prioritization techniques are available, including RICE (reach, impact, confidence, effort), MoSCoW (Must have, should have, could have, won't have), Eisenhower matrix and Weighted Shortest Job First (WSJF) [41]. However, most of these methods are subjective. This work presents a novel approach to finding the risk values of features based on modified requirements, feature complexity, and interdependency.

3. Proposed Model

In order to efficiently prioritize the test cases in a sprint, the risk values of features in the sprint are evaluated. The risk associated with sprint features is calculated based on failure probability and failure impact. Failure probability is derived from two components, i.e., changed requirements and feature complexity. Failure impact is calculated on the basis of the dependency of features. After assigning risk values to the feature, the correlation between the feature and the test case is obtained. This ultimately guides in prioritising the cases. Figure 2 presents the proposed risk-based TCP model.

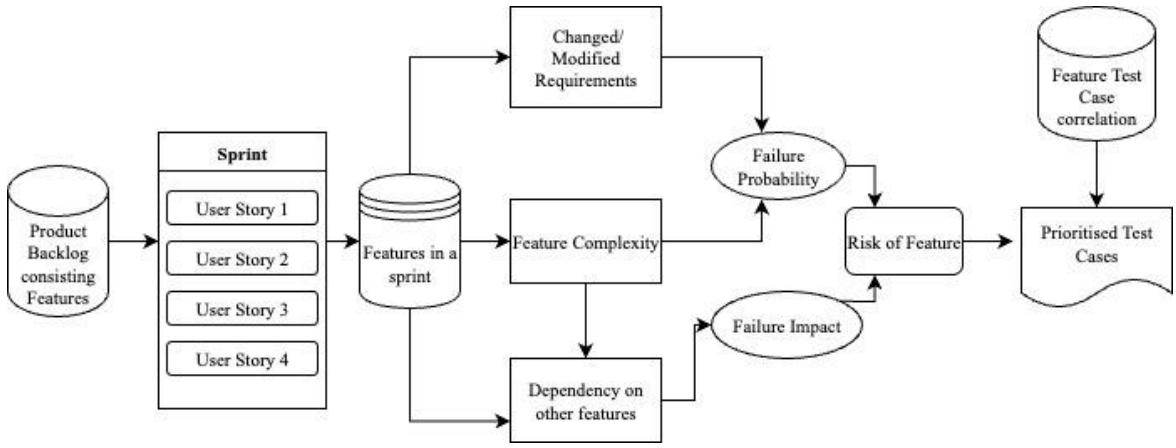

Figure 2. Risk-based TCP for DASD.

Steps of the proposed model are explained as follows.

- *Step* 1: Extract features from current sprint user stories.

Several tools, including Jira, Azure DevOps, typically offer various features for organising, prioritising, and tracking user stories and associated features within the sprint backlog, helping teams effectively manage their Agile development process.

- *Step* 2: Derive the failure probability and failure impact.

The risk value of a feature is calculated as a product of failure likelihood (probability) and failure impact. We calculate failure likelihood based on two factors: changed requirements and feature complexity.

- Failure probability=Changed Requirements+Feature Complexity.
- Changed/Modified requirements: this factor is determined by the number of changes or modifications made to the requirements within a feature. In software systems, faults often emerge because of changes to the software, particularly changes in requirements. Thus, tracking the number of modified requirements within a feature can serve as an effective indicator for identifying potential faults [1]. This factor is obtained by summing the number of user stories of the current sprint associated with a feature. The values are normalized within the range 0-1 by dividing each obtained value by the highest value of changed requirements. The need for normalisation is required to scale all the factors in the same range.

$$CR_i = \frac{NSi}{Max\ \forall i\{NSi\}} \quad (1)$$

Where NS_i represents the number of user stories of

feature F_i.

- Feature cmplexity: the feature complexity is computed based on the effort required and risk values associated with the user story. Effort in Agile methodology is expressed through story points, which are a relative measure of the complexity, effort, and uncertainty associated with implementing a user story. Efforts required to complete a feature can be calculated by obtaining the sum of story points of all the user stories associated with that feature in the sprint [36]. The effort is then normalized within the range 0-1 by dividing each value obtained by the highest value of effort.

$$E_i = \frac{NSPi}{Max\forall i\{NSPi\}} \quad (2)$$

Where NSP_i represents the number of story points of feature F_i.

The complexity of the feature also depends on the risk severity of the user story associated with a feature. The risk severity of a user story can be obtained based on 10 risk factors identified in distributed agile environment. 10 risk categories are represented as objective statement risks, design risks, testing risks, coding risks, release and deployment risks, project management risks, communication risks, technology-based risks, external stakeholder risks and group awareness risks [20]. This task is accomplished using a Hierarchical Fuzzy Inference System (HFIS) [32]. The model is created by dividing the risk categories into two classes, i.e. SDLC risks and other risks. Objective statement, design, coding, testing and release and deployment risks come under the SDLC class. Project Management, communication, external stakeholders and technology-based risks are kept in the other risks category. The output of these two Fuzzy Inference Systems (FIS) is then fed to a third FIS, which gives us the total risk severity of the user story. The number of rules using this HFIS drastically reduces to $3^5+3^5+5^2$, i.e., 511.

The textual risks associated with user stories are converted to numerical representation by using the Term Frequency-Inverse Document Frequency (TF-IDF) technique. Then, k-means clustering is applied to create clusters for each risk category. Next, the fuzzy c-means algorithm is applied to find the membership values of fuzzy sets for each cluster. The rule base, fuzzy sets and the presence percentage of risk categories are fed as input to HFIS. The output of the system is a risk value associated with a user story, which is then defuzzified and is used for Backlog prioritization. The overall architecture is presented in Figure 3.

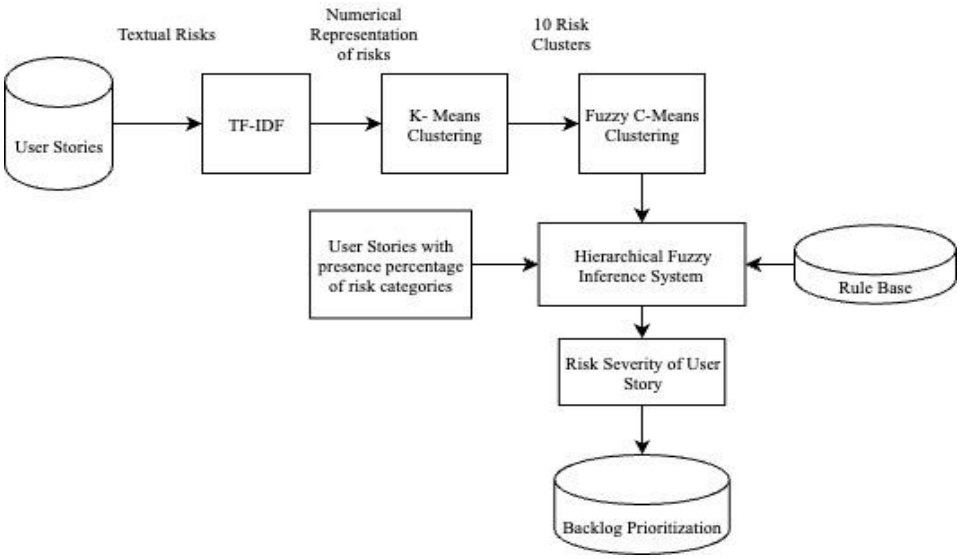

Figure 3. Hierarchical fuzzy inference-based risk management system for user story.

Risk severity of a feature can be calculated by the summation of risk severity of all user stories associated with a feature and then normalizing the obtained value in the range 0-1 by dividing each value by the highest value of risk severity.

$$RFi = \frac{RUSi}{Max\forall i\{RUSi\}} \quad (3)$$

Where $RUSi$ represents the risk of the user story associated with feature Fi.

Feature complexity is calculated by the summation of effort and risk severity of the feature. The value is then divided by 2 to scale between the range 0-1.

$$FC_i\ (E_i + RF_i)/2 \quad (4)$$

Where FC_i represents the feature complexity of feature F_i.

Failure probability is calculated by the summation of changed requirement and feature complexity. The obtained value is then divided by 2 to scale between 0 to 1.

$$FPi = (CRi + FCi)/2 \quad (5)$$

Where FP_i is the Failure Probability of Feature F_i.

- Failure impact: the failure impact value of a feature is determined by considering the functional

dependencies within the software. If a feature contains faults, these faults may propagate to other features that are either directly or indirectly dependent on it. As a result, a feature that is invoked by multiple other features is likely to be executed more often during testing, increasing the likelihood of faults occurring in that feature. Therefore, a feature with a high invocation degree is more prone to faults. To calculate the impact, we measure the number of outgoing dependencies (outdegree) of the feature.

$$FI_i = \frac{NFCi}{Max\forall i\{NFCi\}} \quad (6)$$

Where NFC_i is the total number of outdegrees of a feature F_i.

- *Step* 3: Calculate the risk of a feature.

According to Bohem, risk is calculated as a product of failure occurrence and failure impact. Therefore, feature risk is calculated by taking the product of failure probability in Equation (5) and failure impact Equation (6) [4].

$$FR_i = FP_i * FI_i \quad (7)$$

- *Step* 4: Prioritize the test cases by test case-feature correlation.

The test case is prioritized based on the value obtained by the summation of risk values of all the features covered by the test case. To extract a correlation between features and test case, we perform textual analysis of acceptance criteria from the user story. Several tools like xRay (for Jira) and qTest (for DevOps) are available to trace the correlation between test cases and features [28]. Therefore, prioritized value of a test case can be derived using the following Equation (8).

$$T_k = \sum_{i=1}^{n} FRik \quad (8)$$

In this context, *k* denotes the test case, and *n* refers to the number of features covered by the *k*-th test case. After calculating the risk values for each test case, we prioritized them in descending order according to those values. Since test cases in a system may correspond to different requirements but cover the same features, test cases that cover the same features will have identical risk values. In such cases, we randomly prioritized these test cases.

4. Empirical Study

The efficiency of the proposed model is evaluated with an Empirical study. Research questions, experimental subjects, evaluation matrix and threats to validity are presented in this section.

4.1. Research Questions

- **RQ1:** Can the proposed model prioritize the test cases?
- **RQ2:** Can the proposed model efficiently prioritize the test cases associated with a user story in comparison with other state- of- art approaches?

4.2. Experimental Subjects

A dataset has been collected from an IT company that employs a DASD methodology, with teams located across India, Canada, and Ukraine. Additionally, the team members within each country are geographically spread across various locations. The distribution of team members across the three countries is outlined in Table 1. The company utilises several tools to facilitate their Agile processes: Azure DevOps for defining workflows and managing both product and sprint backlogs, and *Slack* for team communication. The data set comprises 1400 user stories with 11 attributes as presented in Table 2. The data set is confidential.

Table 1. Details of teams.

Members	Team 1	Team 2	Team 3
Developers	8	7	8
Quality analyst	5	5	4
Scrum master	1	1	1
Team product owner	2	1	1
UI/ UX	2	1	2

Table 2. Data set attributes.

ID	Columns	Data type
1	User Story ID	Int
2	User story title	String
3	Sprint ID	Int
4	Acceptance criteria	String
5	Estimation	Float
6	Dependency	String
7	Priority	Int
8	Risk Involved	String
9	Probability of occurrence	Int
10	Potential Impact	Int
11	Level of Urgency	Int

The dataset includes information on various stories completed in different sprints, along with their associated story points. The proposed work has been implemented on 320 user stories consisting of 24 features. The sample association of user story and features is presented in Table 3.

Table 3. Features associated with user stories.

S.No.	Feature number
User Story 1	1
User Story 2	1
User Story 3	1
User Story 4	1
User Story 5	2
User Story 6	2
User Story 7	2
User Story 8	3
User Story 9	3
User Story 10	4
User Story 11	4
User Story 12	4

Initially, K-means clustering is applied on the dataset to cluster the user stories into 10 risk categories [19]. To

apply k-means clustering, the textual risk tags are converted to a vector space representation using TF-IDF. Figure 4 presents the implementation in Python. Then, for each obtained cluster of risk category, fuzzy parameters are obtained by applying the fuzzy-c-means algorithm. The fuzz.cluster.cmeans function from the scikit-fuzzy library is used to apply the algorithm.

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from google.colab import files

df = pd.read_excel('data.xlsx')
print(df.head())
keywords = ["dependency", "Dependent", "Automation", "Rework", "High priority"]
df['extracted_keywords'] = df['Risks'].apply(lambda x: [word for word in manual_keywords if word in x])

# Vectorization using TF-IDF
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(df['Risks'])

# Clustering using K-Means
num_clusters = 3  # You can adjust the number of clusters
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
kmeans.fit(tfidf_matrix)

# Assign cluster labels to the DataFrame
df['cluster'] = kmeans.labels_

# Output the DataFrame with clusters and extracted keywords
print(df[['Risks', 'extracted_keywords', 'cluster']])
```

Figure 4. Implementation.

Table 4. Membership values of fuzzy sets for each risk category.

S. No	Risk Category	Low	Medium	High
1	Objective statement	(0,0,0.2,0.3)	(0.2,0.3,0.6,0.7)	(0.6,0.7,1,1)
2	Design risk	(0,0,0.15,0.25)	(0.2,0.3,0.37,0.45)	(0.4, 0.6,1,1)
3	Coding risk	(0,0,0.15,0.25)	(0.2,0.27,0.37,0.45)	(0.4, 0.6,1,1)
4	Testing risk	(0,0,0.2,0.3)	(0.25,0.4,0.6,0.7)	(0.65,0.75,1,1)
5	Release and deployment	(0,0,0.1,0.2)	(0.15,0.24,0.32,0.4)	(0.35,0.45,1,1)
6	Project management	(0,0,15,0.3)	(0.2,0.35,0.55,0.7)	(0.6,0.75,1,1)
7	Communication risk	(0,0,0.15,25)	(0.2,0.3,0.39,0.45)	(0.4,0.5,1,1)
8	External stakeholder	(0,0,0.11,0.2)	(0.15,0.2,0.32,0.4)	(0.35,0.45,1,1)
9	Group awareness	(0,0,0.2,0.3)	(0.25,0.4,0.56,0.7)	(0.65,0.75,1,1)
10	Technology based	(0,0,0.1,0.2)	(0.15,0.25,0.37,50)	(0.45,0.55,1,1)

Table 5. Rule base for FIS1.

R_ID	Objective statement risks (1-3)	Design risk (1-3)	Coding (1-3)	Testing (1-3)	Release and deployment (1-3)	SDLC risk severity (1-5)
1	1	1	3	1	3	3
2	3	1	3	3	1	5
3	1	1	1	1	3	4
4	2	2	1	1	3	5
5	3	1	2	2	1	5
6	3	1	1	1	1	5
7	2	1	3	2	1	3
8	2	3	1	2	3	5
9	2	1	3	3	2	4
10	3	2	3	3	2	5

Table 6. Rule base for FIS2.

R_ID	Project management risk (1-3)	Communication risk (1-3)	External stakeholder (1-3)	Group awareness (1-3)	Technology based (1-3)	Risk Severity 2 (1-5)
1	2	3	3	1	3	3
2	1	2	1	1	1	1
3	3	2	3	2	1	3
4	2	3	2	2	3	3
5	1	2	3	3	2	2
6	3	3	2	2	2	2
7	2	1	3	3	1	2
8	3	2	1	2	1	2
9	2	3	3	1	3	2
10	2	3	1	3	2	1

Table 7. Rule base for FIS3.

Risk ID	SDLC risk severity (1-5)	Other risks severity (1-5)	Total risk severity (1-5)
1	3	3	3
2	5	1	5
3	4	3	4
4	5	3	5
5	5	2	5
6	5	2	5
7	3	2	3
8	5	2	5
9	4	2	4
10	5	1	5

After applying fuzzy-c-means clustering, the overlapping sets are obtained for each risk category, representing linguistic values low, medium and high. The membership values of these fuzzy sets for each risk category are represented in Table 4. The rule base of rules is created with the help of domain experts from the IT industry. The sample rule base for FIS1, FIS2 and FIS3 is represented in Tables 5, 6, and 7, respectively.

The effort is calculated based on the story points obtained from the dataset. Then the feature complexity is calculated using Equations (2) to (4).

A total of 24 features are tagged with the given set of user stories. The risk value associated with these features is obtained by extracting the values of changed requirements, feature complexity and dependency of Features from the SUT. The obtained values are normalized, and feature risk is calculated using Equations (1) to (7). The obtained values for the 4 features are presented in Table 8.

Table 8. Feature risk.

Features	Changed requirements	Feature complexity	Dependency	Risk of feature
Feature 1	1	1	0.5	1
Feature 2	0.75	0.5	1	1.25
Feature 3	0.5	0.68	0	0
Feature 4	0.75	0.68	1	1.43

4.3. Evaluation Matrix

The Average Percentage of Faults Detected (APFD) metric, introduced by Elbaum *et al.* [8], evaluates the speed at which faults in a system are identified within a test suite. It calculates the average fault detection rate on a scale from 0 to 1, where higher values indicate a better rate of fault detection, and lower values suggest poorer detection performance.

The computation formula of *APFD* is as follows:

$$APFD = 1 - \frac{TF1 + TF2 + \cdots . TFm}{n * m} + \frac{1}{2n} \qquad (9)$$

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

4.4. Results

To verify the correct functionality of features, 150 test cases have been taken in a suite. These test cases are then correlated to the features. Table 9 represents the sample correlation for 15 test cases. The test cases of the suite are prioritized based on original order, reverse order, random order, feature coverage based and proposed feature risk-based techniques. The different prioritization algorithms are compared based on the average percentage of fault detection metric. The results are presented in Table 10.

Table 9. Correlation of test cases and features.

Test Cases	Feature 1	Feature 2	Feature 3
TC1	✓	✓	
TC2	✓	✓	
TC3	✓	✓	
TC4	✓	✓	
TC5	✓		
TC6		✓	
TC7		✓	
TC8		✓	
TC9	✓	✓	
TC10			✓
TC11			✓
TC12			✓
TC13		✓	
TC14		✓	
TC15		✓	

Table 10. Average percentage of faults detected.

TCP	APFD
Original Order	0.71
Reverse Order	0.86
Random order	0.84
Feature coverage based	0.89
Proposed technique	0.95

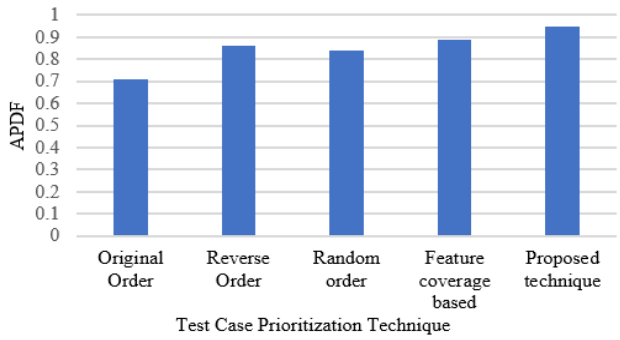

Figure 5. Comparison with existing technique.

As in Figure 5 the results show that the proposed technique outperforms the other prioritization techniques with an APFD value of 0.95.

Table 11 presents a comparison of the proposed work with existing techniques based on APDF.

Table 11. Comparison with existing work.

Authors	Year	TCP technique	APFD
Junaid *et al*. [18]	2024	History based	0.87
Garg and Shekha [10]	2024	Fault Indexed based multiple objective	0.88
Li [24]	2021	Fault severity based	0.91
Gupta and Mahapatra [12]	2023	Feature coverage based	0.87
Sugave *et al*. [39]	2025	Fault aware Jaya Archimedes optimization algorithm	0.93
Kumar and Saxena [22]	2025	Coverage based	0.82
Zhou *et al*. [43]	2021	Random TCP	0.86
Proposed technique		Feature risk severity based	0.95

4.5. Threats to Validity

This section presents the threats to validity of the proposed approach.

4.5.1. Internal Validity

In this study, we have examined feature risks based on 10 risk factors associated with DASD. We acknowledge that the emergence of new, uncertain risks could potentially affect the assessed severity of a feature. Further, for each risk category, we have assigned the membership values of low, medium and high. Altering these membership values could influence the dependent variable of risk severity. However, to address this potential validity threat, a pivot study has been carried out to aim to minimize any such validity threats.

4.5.2. External Validity

The subject systems employed for empirical studies in this work are of medium size. However, it is acknowledged that larger industrial projects may produce varied results. To address these limitations, we plan to conduct further experiments involving bigger industrial projects, which aligns with one of our future research directions.

4.6. Limitations and Future Work

The data of 1400 user stories from a real-world IT project are taken, which is adequate for the study's objectives. Further, the work is evaluated on 320 user stories of an IT project. A total of 10 risk categories related to a distributed agile environment have been included. Future research could broaden the scope of the proposed method by examining its application in a variety of DASD projects, spanning different industries and project scales. This would provide valuable insights into the method's adaptability and scalability in real-world environments, enabling a deeper understanding of its performance across diverse conditions. Additionally, the method's potential for generalization to larger projects and its applicability to additional risk categories will be explored. To further validate its effectiveness, user feedback and expert opinions, gathered through surveys or interviews with project managers, developers, and other stakeholders, could be incorporated into the risk assessment process. This would help ensure the method's practicality, strengthen its reliability, and enhance its usefulness for practitioners, ultimately making it more robust and versatile for real-world applications.

5. Conclusions

TCP is essential in DASD, wherein the software is developed in sprints. TCP reorders the test cases if the SUT in a way that the most important test cases are executed before others. However, the existing TCP techniques use the past test execution data and existing fault information, which may not always be available. RBT, focuses on identifying and testing the risky modules, so that most risky features are tested within a limited time and resources. However, the existing risk-based TCP methods do not take into account the risks involved in a distributed agile environment

This study introduces a novel approach to risk-based test case prioritisation in distributed agile environments, focusing on prioritizing test cases according to the risk levels associated with various features by linking sprint features to their respective test cases. The approach offers a new perspective on evaluating the risk of features by considering factors such as modified requirements, feature complexity, and interdependencies. The aim is to identify and test high-risk features early in the distributed agile development process, helping to uncover critical defects sooner.

While risks have traditionally been used to prioritize test cases, applying risks related to DASD for feature prioritization is a relatively new concept. As DASD grows in popularity, strategies for improving TCP in such settings are becoming increasingly important. Associating risk values with features and correlating them with test cases can lead to more efficient RBT.

The experiments indicate that the proposed approach achieves a TCP with an APFD of 0.95. Ultimately, the findings of the proposed work offer valuable insights and will stimulate future discussion and advancement of knowledge in the field.

References

[1] Alsaadi B. and Saeedi K., "Data-Driven Effort Estimation Techniques of Agile User Stories: A Systematic Literature Review," *Artificial Intelligence Review*, vol. 22, pp. 5485-5516, 2022. https://link.springer.com/article/10.1007/s10462-021-10132-x

[2] Alzoubi Y. and Mishra A., "Enterprise Architecture Contribution in Distributed Agile Software Development," *Systems Engineering*, vol. 27, pp. 570-584, 2024.

https://incose.onlinelibrary.wiley.com/doi/full/10.1002/sys.21739

[3] Aziz M. and Choi J., "Prioritization of Risks in Agile Software Projects Through an Analytic Hierarchy Process Approach," *Procedia Computer Science*, vol. 233, pp. 713-722, 2024. https://doi.org/10.1016/j.procs.2024.03.260

[4] Boehm B., "Software Risk Management: Principles and Practices," *IEEE Software, vol. 8*, no. 1, pp. 32-41, 1991. https://ieeexplore.ieee.org/document/62930

[5] Catal C. and Mishra D., "Test Case Prioritization: A Systematic Mapping Study," *Software Quality Journal*, vol. 21, no. 3, pp. 445-478, 2012. https://doi.org/10.1007/s11219-012-9181-z

[6] Chi J., Qu Y., Zheng Q., Yang Z., and et al., "Relation-based Test Case Prioritization for Regression Testing," *Journal of Systems and Software*, vol. 163, pp. 110539, 2020. https://doi.org/10.1016/j.jss.2020.110539

[7] Dingsøyr T., Nerur S., Balijepally V., and Moe N., "A Decade of Agile Methodologies: Towards Explaining Agile Software Development," *Journal of Systems and Software*, vol. 85, no. 6, pp. 1213-1221, 2012. https://doi.org/10.1016/j.jss.2012.02.033

[8] Elbaum S., Malishevsky A., and Rothermel G., "Test Case Prioritization: A Family of Empirical Studies," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159-182, 2002. https://ieeexplore.ieee.org/document/988497

[9] Farooq U., Kalim Z., Qureshi J., Rasheed S., and Abid A., "A Blockchain-based Framework for Distributed Agile Software Development," *IEEE Access*, vol. 10, pp. 17977-17995, 2022. https://ieeexplore.ieee.org/document/9694597

[10] Garg V. and Shekha S., "Fault Sensitivity Index-based Multi-Objective Test Case Prioritization," *Journal of Electrical Engineering*, vol. 75, no. 2, pp. 151-160, 2024. https://reference-global.com/article/10.2478/jee-2024-0018

[11] Gladston A., Nehemiah K., Narayanasamy P., and Kannan A., "Test Case Prioritisation for Regression Testing Using Immune Operator," *The International Arab Journal of Information Technology*, vol. 13, no. 6, pp. 31-37, 2016. https://www.iajit.org/PDF/Vol.%2013,%20No.%206/5585.pdf

[12] Gupta A. and Mahapatra R., "Test Case Prioritization in Unit and Integration Testing: A Shuffled-Frog-Leaping Approach," *Computers, Materials and Continua*, vol. 74, no. 3, pp. 5369-5387, 2023. https://www.techscience.com/cmc/v74n3/50872/html

[13] Hao D., Zhang L., and Mei H., "Test-Case Prioritization: Achievements and Challenges," *Frontiers in Computational Science*, vol. 10, pp. 769-777, 2016. https://doi.org/10.1007/s11704-016-6112-3

[14] Hasnain M., Pasha M., Ghani I., and Jeong S., "Functional Requirement-based Test Case Prioritization in Regression Testing: A Systematic Literature Review," *SN Computer Science*, vol. 2, no. 421, 2021. https://link.springer.com/article/10.1007/s42979-021-00821-3

[15] Hettiarachchi C., Do H., and Choi B., "Risk-based Test Case Prioritization Using a Fuzzy Expert System," *Information and Software Technology*, vol. 69, pp. 1-15, 2016. https://doi.org/10.1016/j.infsof.2015.08.008

[16] Huang Y., Peng K., and Huang C., "A History-based Cost-Cognizant Test Case Prioritization Technique in Regression Testing," *Journal of Systems and Software*, vol. 85, no. 3, pp. 626-637, 2012. https://doi.org/10.1016/j.jss.2011.09.063

[17] Jahan H., Feng Z., and Mahmud S., "Risk-based Test Case Prioritization by Correlating System Methods and their Associated Risks," *Arabian Journal for Science and Engineering*, vol. 45, pp. 6125-6138, 2020. https://link.springer.com/article/10.1007/s13369-020-04472-z

[18] Junaid H., Jawawi D., and Ahmad J., "An Exploratory Study of History-based Test Case Prioritization Techniques on Different Datasets," *Baghdad Science Journal*, vol. 21, no. 2, pp. 609-621, 2024. DOI:10.21123/bsj.2024.9604

[19] Khanna E., Popli R., and Chauhan N., "Identification and Classification of Risk Factors in Distributed Agile Software Development," *Journal of Web Engineering*, vol. 21, no. 6, pp. 1831-1851, 2022. https://ieeexplore.ieee.org/document/10246944

[20] Khanna E., Popli R., and Chauhan N., *Agile Software Development*, Wiley AI, 2023. https://ieeexplore.ieee.org/document/10953243

[21] Khatibsyarbini M., Isa M., Jawawi D., and Tumeng R., "Test Case Prioritization Approaches in Regression Testing: A Systematic Literature Review," *Information and Software Technology*, vol. 93, pp. 74-93, 2018. https://doi.org/10.1016/j.infsof.2017.08.014

[22] Kumar S. and Saxena V., "Optimization and Prioritization of Test Cases through the Hungarian Algorithm," *Journal of Advances in Mathematics and Computer Science*, vol. 40, no. 3, pp. 61-72, 2025. https://journaljamcs.com/index.php/JAMCS/article/view/1978

[23] Lawong D. and Akanfe O., "Overcoming Team Challenges in Project Management: The Scrum Framework," *Organizational Dynamics*, vol. 54, no. 1, pp. 101073, 2025. https://doi.org/10.1016/j.orgdyn.2024.101073

[24] Li Y., "A Fault Prediction and Cause Identification Approach in Complex Industrial Processes Based on Deep Learning," *Computational Intelligence and Neuroscience*, vol. 2021, pp. 1-13, 2021. https://onlinelibrary.wiley.com/doi/10.1155/2021/6612342

[25] Mahdieh M., Mirian-Hosseinabadi S., Etemadi K., Nosrati A., and Jalali S., "Incorporating Fault-Proneness Estimations into Coverage-based Test Case Prioritization Methods," *Information and Software Technology*, vol. 121, pp. 106269, 2020. https://doi.org/10.1016/j.infsof.2020.106269

[26] Mukherjee R. and Patnaik K., "A Survey on Different Approaches for Software Test Case Prioritization," *Journal of King Saud University-Computer and Information Sciences*, vol. 33, no. 9, pp. 1041-1054, 2021. https://doi.org/10.1016/j.jksuci.2018.09.005

[27] Parida S., Rath D., and Mishra D., *Meta Heuristic Techniques in Software Engineering and its Applications*, Springer, 2022. https://doi.org/10.1007/978-3-031-11713-8_16

[28] Pecorelli F., Palomba F., and De Lucia A., "The Relation of Test-Related Factors to Software Quality: A Case Study on Apache Systems," *Empirical Software Engineering, vol. 26*, no. 6, pp. 1-42, 2021. https://doi.org/10.1007/s10664-020-09891-y

[29] Qureshi J., Farooq M., Khelifi A., and Atal Z., "Harnessing the Potential of Blockchain in ChainAgilePlus Framework for the Improvement of Distributed Scrum of Scrums Agile Software Development," *IEEE Access*, vol. 12, pp. 105724-105743, 2024. https://ieeexplore.ieee.org/document/10595117

[30] Sakhrawi Z. and Labidi T., "Test Case Selection and Prioritisation Approach for Automated Regression Testing Using Ontology and COSMIC Measurement," *Automated Software Engineering*, vol. 31, no. 2, pp. 1453, 2024. https://doi.org/10.1007/s10515-024-00447-8

[31] Saraswat P., Singhal A., and Bansal A., *Software Engineering*, Springer, 2019. https://doi.org/10.1007/978-981-10-8848-3_48

[32] Shivanandam S. and Deepa S., *Principles of Soft Computing*, Wiley, 2011. https://ebooks.wileyindia.com/home/product-details/282517;seoMode=true

[33] Singh M., Chauhan N., and Popli R., "Test Case Reduction and SWOA Optimization for Distributed Agile Software Development Using Regression Testing," *Multimedia Tools and Applications, vol. 84*, pp. 7065-7090, 2025. https://link.springer.com/article/10.1007/s11042-024-19148-1

[34] Singh Y., "Systematic Literature Review on Regression Test Prioritization Techniques: Difference between Literature Review and Systematic Literature," *Informatica*, vol. 36, no. 4, pp. 379-408, 2012. https://www.informatica.si/index.php/informatica/article/view/420/424

[35] Singhal S., Jatana N., Suri B., Misra S., and Fernandez-Sanz L., "Systematic Literature Review on Test Case Selection and Prioritization: A Tertiary Study," *Applied Sciences*, vol. 11, no. 24, pp. 1-34, 2021. https://www.mdpi.com/2076-3417/11/24/1212

[36] Sommerville I. and Sawyer P., *Requirements Engineering: A Good Practice Guide*, John Wiley and Sons, 1997. https://www.wiley.com/en-us/Requirements+Engineering%3A+A+Good+Practice+Guide-p-9780471974444

[37] Srikanth H., Hettiarachchi C., and Do H., "Requirements Based Test Prioritization Using Risk Factors: An Industrial Study," *Information and Software Technology*, vol. 69, pp. 71-83, 2016. https://doi.org/10.1016/j.infsof.2015.09.002

[38] Stray V., Hoda R., Paasivaara M., Lenarduzzi V., and Mendez D., "Theories in Agile Software Development: Past, Present, and Future Introduction to the XP 2020 Special Section," *Information and Software Technology*, vol. 152, pp. 107058, 2022. https://doi.org/10.1016/j.infsof.2022.107058

[39] Sugave S., Kulkarni Y, Jagdale B., and Gutte V., "Fault-Aware Test Case Prioritization in Software Testing Using Jaya Archimedes Optimization Algorithm," *Journal of Electronic Testing*, vol. 41, pp. 41-60, 2025. https://link.springer.com/article/10.1007/s10836-025-06157-7

[40] Wang Y., Zhu Z., Yang B., Guo F., and Yu H., "Using Reliability Risk Analysis to Prioritize Test Cases," *Journal of Systems and Software*, vol. 139, pp. 14-31, 2018. https://doi.org/10.1016/j.jss.2018.01.033

[41] Webber R., *Unlocking Agile's Missed Potential*, Wiley-IEEE Press, 2022. https://onlinelibrary.wiley.com/doi/10.1002/9781119849117.ch5

[42] Yoo S. and Harman M., "Regression Testing Minimisation, Selection and Prioritization: A Survey," *Test Verification and Reliability*, vol. 22, no. 2, pp. 1-60, 2007. https://www.cse.chalmers.se/~feldt/advice/yoo_2010_regression_testing_survey.pdf

[43] Zhou Z., Liu C., Chen T., Tse T., and Susilo W., "Beating Random Test Case Prioritization," *IEEE Transactions on Reliability*, vol. 70, no. 2, pp. 654-675, 2021. https://ieeexplore.ieee.org/document/9118977

Esha Khanna is currently pursuing Ph. D. in Computer Engineering from J. C. Bose University of Science and Technology, YMCA, Faridabad, India. She holds a Master of Technology degree and a Bachelor of Technology degree in Computer Science and Engineering. She is working as an Assistant Professor at Manav Rachna University, India. She has 11 years of teaching experience. Her main research area focuses on Machine Learning, Software Testing, and Distributed Agile Software Development

Rashmi Popli is an Associate Professor at J.C. Bose University of Science and Technology, YMCA, Faridabad, India. She has 21 years of rich experience in Teaching, and 4 research scholars are pursuing PhD under her guidance and supervision. Her areas of specialisation include Machine Learning, Software Engineering, Software Testing, Network Security and automation of software. She has published more than 50 research papers in various International Journals and conferences. She is a lifetime member of ISTE and CSI. She is also holding the position of Faculty In-Charge, Industrial Collaboration and Consultancy and Coordinator IQAC.

Naresh Chauhan is working as a Professor in J.C. Bose University of Science and Technology, YMCA, India. He has received his Ph.D. (Computer Engineering) from MD University, Rohtak (Haryana) in 2008, M.Tech. (Information Technology) from GGS Indraprastha University, Delhi in 2004 and B.Tech. (Computer Engg.) from NIT Kurukshetra, in the year 1992. He has about 30 years of experience in teaching and the industries. He served Bharat Electronics Ltd. and Motorola India Ltd. His research interests include Internet technologies, Software Engineering,Software Testing and Real time systems. He has published two books on Software Testing and Operating Systems published by Oxford University Press, India.