


# Test case prioritization methods: A systematic literature review

Icaro Prado Fernandes  [ Universidade Federal de São Paulo | [icaro.fernandes@unifesp.br](mailto:icaro.fernandes@unifesp.br) ]

Luiz Eduardo Galvão Martins  [ Universidade Federal de São Paulo | [legmartins@unifesp.br](mailto:legmartins@unifesp.br) ]

## Abstract

Software testing is a process that assists in achieving software quality. However, software testing used to be a repetitive activity that takes a long time to be performed. Several methods for test case prioritization have been created with the purpose to reduce the time of software testing activity without compromising its quality. The main idea of this review is to examine and classify test case prioritization methods based on the research questions created. In order to achieve the outlined objective, seventy-one primary studies that propose, analyze, or compare test case prioritization methods were used as a basis for the analysis. Sixty-eight of them were found by search string and four studies were manually included by the author. The main methods of each group were mapped, analyzed and described. Despite the existence of many methods for prioritizing test cases, none of them were designed especially to be applied in agile environments, therefore there are opportunities for improvement in this topic.

**Keywords:** *Test case prioritization, Software testing, Software quality, Methods for prioritizing test cases*

## 1. Introduction

The growing use of software across various markets and segments has added complexity to the development process, leading to significant advancements in processes, techniques, methods, and tools within software engineering.

According to Sommerville (2011), software engineering is an engineering discipline that focuses on all aspects of software production, including the stages of software specification, development, verification, validation, and evolution.

According to Pressman and Maxim (2019), the software development process comprises a series of activities, actions, and tasks involved in each stage of software production. The objective is to ensure that the software is delivered on time and meets the required quality standards to satisfy both investors and users. Delivering quality software is a significant challenge in software engineering, as quality can be subjective and is often defined as its suitability for use.

In addition to Sommerville (2011), Pressman and Maxim (2019), Fairley (1985), and Brooks (2021) converging on the importance and complexity of developing and delivering quality software, they also emphasize that software testing is a vital activity in the development process. While not a silver bullet for guaranteeing quality, Sommerville (2011) and Pressman and Maxim (2019) assert that proper testing can reveal incorrect, undesirable, or non-compliant software behaviors. It also shows the development team and clients that the software meets its requirements.

The term "properly" was used earlier because testing activities often tend to be repetitive and exhaustive, which inherently affects the balance of time, cost, and quality. To ensure efficiency in software testing activities, it is essential to address two critical questions: what should be tested and how should it be tested?

Rothermel et al. (1999) state that prioritizing test cases helps balance time, cost, and quality. This systematic literature review (SLR) will analyze existing test case prioritization (TCP) methods and group them based on the characteristics outlined by Catal and Mishra (2013) and Yoo

and Harman (2012).

The remainder of this paper is organized as follows: Section 2 presents concepts and a brief discussion about related work; Section 3 explains the research methodology adopted to conduct our SLR; Section 4 presents the results and analysis of this SLR; and Section 5 presents the conclusions and future work.

## 2. Background and Related Work

### 2.1 Concepts

The main concepts used in this SLR are defined below (Black, 2019):

**Software testing:** It is a process of evaluating a software application to ensure it meets specified requirements, functions as expected and is free from defects.

**Taxonomy:** A system that classifies and organizes entities into groups or categories based on shared characteristics, relationships, or hierarchies.

**Test case:** A detailed document or set of steps that outlines specific actions, conditions, inputs, and expected outcomes to verify whether a particular feature or functionality of a software application works as intended.

**Test case prioritization:** It is a process of organizing and ranking test cases in a software testing suite based on certain criteria, typically to achieve specific objectives like detecting critical bugs earlier, optimizing resource usage, or ensuring the most important functionalities are tested first.

### 2.2 Related Work

This section presents relevant previous studies that are related to TCP methods and served as a basis for the preparation of this SLR.

Yoo and Harman (2012) present an article with the purpose of providing a broad overview of the state of the art on test case selection, prioritization and reduction, and

also discusses open issues and possible opportunities for future research. Singh et al. (2012) did a search for existing TCP methods from 1997 to 2011 and selected 65 studies; of these studies, 49 were generators of 106 methods for TCP and 16 were based on comparative analysis. Catal and Mishra (2013) identified 120 papers related to methods for TCP in the period from 2001 to 2011 and reinforced the dominance of methods for TCP based on source code coverage.

In the review by Khatibsyarhini et al. (2018) 80 studies were identified and the distribution of the methods proposed in these studies was presented grouping them by characteristic: 18% coverage-based, 25% search-based, 10% fault-based, 9% requirement-based, 9% history-based, 7% risk-based, 5% Bayesian network based, 4% cost effective based, 4% multi-criteria, 4% model-based, 4% workflow and 4 others.

Rahmani et al. (2021) examines the evolution of TCP for regression testing, analyzing 48 studies published between 2017 and 2020. It presents four key reviews: (1) the advancement of approaches and techniques in regression TCP research, (2) the identification of variations in Software Under Test (SUT) utilized in TCP studies, (3) trends in metrics used to assess the effectiveness of TCP studies, and (4) the current state of requirements-based TCP.

Mohd-Shafie et al. (2022) carried out a systematic review to identify and analyze the state-of-the-art in Model-based test case generation (MB-TCG) and prioritization (MB-TCP), and approaches that integrate both. The research questions were formulated based on the need for this review, and relevant keywords were extracted to guide the literature search. To ensure reliability, prospective studies underwent a quality assessment, selecting only those that met the necessary standards. For full transparency, all research data from this review are available in a public repository. A total of 122 primary studies were finalized and included. One of the main findings is that the most common limitations in the existing approaches are the dependency on specifications, the need for manual interventions, and the scalability issue.

Hasnain et al. (2021) carried out a systematic review on Functional Requirement-Based Test Case Prioritization in Regression Testing. The researchers of this paper analyzed publications from 2009 to 2019 across seven major digital repositories, which are widely used for software engineering research. A thorough screening process was conducted, leading to the selection of 35 research papers to address the proposed research questions. The findings reveal that functional requirement-based TCP approaches have been extensively explored in primary studies. Additionally, fault size and the number of test cases are the most commonly discussed aspects of regression testing. This review also highlights that the iTrust system is frequently examined by researchers in primary studies.

Campos Junior et al. (2017) performed an SLR aiming to examine empirical research on TCP to synthesize reported effectiveness results and establish a foundation

for future studies. A systematic literature mapping was conducted to characterize TCP empirical studies, alongside a systematic literature review to analyze the effectiveness of reported TCP techniques. Among the selected studies from 1999 to 2016, a large number evaluated various TCP techniques. However, after applying quality assessment criteria, many were excluded due to potential methodological issues. The analyzed studies primarily focused on coverage-based TCP techniques. Additionally, findings suggest that factors such as faults, test case characteristics, and coverage granularity can significantly influence the effectiveness of TCP execution.

Ashima et al. (2020) carried out a comprehensive review to explore various prioritization techniques used by different researchers in recent years. Regression testing, a crucial method for verifying, testing, and upgrading software, plays a key role in this process. Test cases are scheduled and prioritized in a structured manner, enabling the detection of a maximum number of faults in the software, particularly technical issues. By identifying faults through test cases, the overall number of test cases is reduced, leading to lower execution costs. This approach ensures that high-priority test cases run first, effectively minimizing the cost, time, and effort involved in software testing.

As we can see from the related work discussed so far, the topic of test case prioritization is relevant, current, and has attracted great interest from the software engineering community. Although several reviews on TCP have been published in recent years, we believe that our SLR can add other contributions to the field, differentiating it from the others, which we mention below:

- Coverage period longer than most previous works, spanning from 1999 to 2022;
- Presentation and discussion of a software testing taxonomy based on the selected studies;
- A clear grouping of TCP methods and an in-depth analysis of each of them;
- A thorough search of TCP methods for agile software development teams;
- A more comprehensive SLR, not limited to specific domains, such as model-based, requirements-based or regression testing, as is the case with some SLRs already published on TCP.

### 3. Methodology

Defining a methodology is crucial for conducting any SLR, as it dictates the scope and quality of the selected studies. To analyze studies on "methods for TCP," a research methodology was structured to guide this SLR, as shown in Figure 1.

#### 3.1 Research questions and their motivations

The objectives of this SLR are:

- To identify the existence of studies that propose, analyze, or compare methods for TCP;

- To understand if any methods have been developed specifically to be applied in agile software development teams;
- To group the methods for TCP based on their characteristics and concisely describe them.

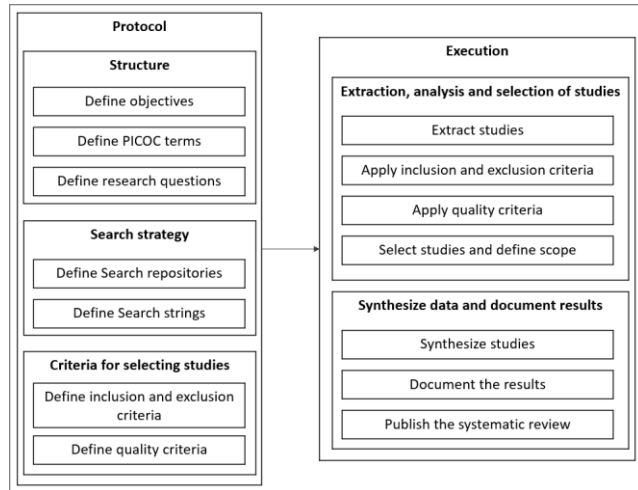


Figure 1. Research methodology.

### 3.2 Research questions and their motivations

The objectives of this SLR are:

- To identify the existence of studies that propose, analyze, or compare methods for TCP;
- To understand if any methods have been developed specifically to be applied in agile software development teams;
- To group the methods for TCP based on their characteristics and concisely describe them.

TCP is important in agile development as it enhances testing efficiency, lowers costs, and speeds up the delivery of software. In the agile environment, where iterations are brief and time is restricted, exhaustive testing is not practical. Therefore, prioritizing tests enables the execution of the most critical scenarios first, ensuring that the most significant issues are identified promptly.

In order to achieve the stated goal, four research questions were investigated, as presented in Table 1. Each step of the research methodology will be described in detail below, in order to specify the protocol and actions carried out to develop this SLR.

### 3.2 PICOC

The protocol of this SLR follows the PICOC criteria that were suggested by Kitchenham and Charters (2007) to describe elements to construct a research question that can be researched, analyzed, and answered, these elements are presented and described as following.

*Population* is the element responsible for mapping the group of people interested in the review, in case of this study, the population is made up of software testing professionals and software engineers in general. *Intervention* is the element responsible for which technology, tool or procedure will be studied, here the objects under study are the test

case prioritization methods existing in the literature and how they work. *Comparison* is the element responsible for defining tools or methodologies for comparing the intervention. In this case, compare how the methods work and understand the positive and negative points of their application. *Outcome* is the element responsible for mapping and present results that add value to the population. In this case, identify the types of methods in the literature, group them and understand about key points of their application. *Context* in which the intervention and comparison take place is mapped. Basically, understand whether the methods have been tested in industry or academia.

Table 1. Research questions.

RQ1	What are the main methods for TCP in literature?
RQ2	How do the main methods for TCP in literature work?
RQ3	Has any method for TCP been specially developed to be applied by agile software development teams?
RQ4	What are the main advantages and disadvantages of applying the methods for TCP found in the literature?

### 3.3 Study selection process

The process of study selection was carried out in five steps, which are described below:

1. Selection of popular repositories on the topic of test case prioritization;
2. Searching through the repositories using search strings;
3. Initial filtering of the studies found by reading the title and abstract;
4. Detailed reading of all studies and application of the inclusion and exclusion criteria;
5. Quality assessment application.

RQ1 aims to assess the main methods found in the literature. RQ2 seeks to understand how the most relevant methods function. RQ3 explores the application or validation of these methods in agile software development. RQ4 investigates the strengths and weaknesses of using these methods.

### 3.4 Repository selection

An exploratory search on Google Scholar using the filter "Test Case Prioritization" was conducted to identify relevant articles and determine the most popular repositories. The following repositories (digital libraries) were selected: IEEE Xplore; ACM Digital Library; Science Direct (Elsevier only); and Springer.

### 3.5 Search string definition

To define the search strings, first a keyword analysis was done in order to cover the search questions, then synonyms were mapped, and finally the Boolean operators "AND"

and “OR” were used to create associations between the search terms. As illustrated in Figure 2, different search strings were used in each repository in order to find studies that propose, analyze, or compare methods for TCP.

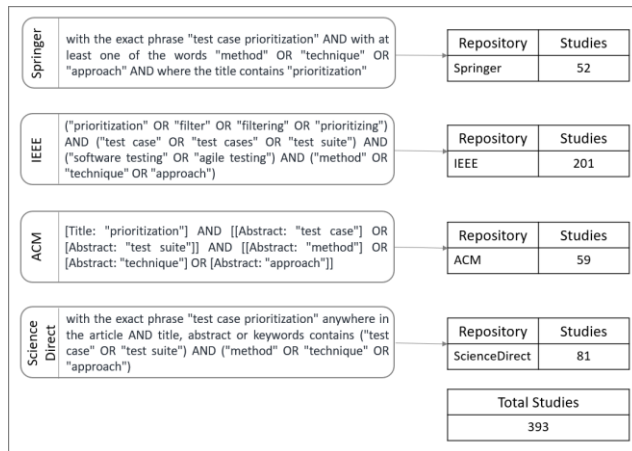


Figure 2. Studies found by search string.

The application of the search strings in the selected databases returned to a total of 393 studies, which went through a selection and filtering procedure for qualitative analysis.

### 3.6 Studies Selection

To refine the selection of studies for review, it was essential to identify the most pertinent ones within the population. Initially, the titles and abstracts of the 393 studies were examined to select those specifically related to the topic "methods for prioritizing test cases". Subsequently, these studies had to meet established inclusion and exclusion criteria.

An agreement test was applied to validate the selection criteria. Twenty articles, chosen at random, were analyzed separately by the two authors of this paper (the level of agreement was 87%). This process was important to enable the analysis and selection of studies, a detailed overview of the inclusion criteria is presented below:

- Articles from conferences and journals;
- Articles with a publication date from 1999 to 2022;
- Articles that propose methods for TCP;
- Articles that compare methods for TCP;
- Articles that map methods for TCP.

A detailed overview of the exclusion criteria is presented below, this process was important for removing duplicate studies and studies that were not directly associated with the topic:

- Articles less than 3 pages long;
- Duplicated articles;
- Articles not written in English;
- Articles that do not discuss methods for TCP;
- Articles that propose methods for TCP with characteristics that are not part of the established groups.

After applying the inclusion and exclusion criteria, the quality assessment questions were applied in order to check the studies capability to answer the research questions. The quality assessment was created with the following questions:

1. Are the article objectives clearly stated?
2. Are the research questions raised answered?
3. Are the variables used to construct the test case prioritization method specified?
4. Is the method for TCP work clearly described?
5. Is it possible to group the method for TCP based on their characteristics?
6. Does the proposed method for TCP compare with others?
7. Are the positive and negative points for applying the method presented?
8. Are agile methodologies mentioned in the article?
9. Is the application of the method for TCP in agile teams mentioned?

A score was developed to assess study quality: each “Yes” answer added 1 point, while each “No” answer added 0. Studies scoring 3 or below were excluded. Figure 3 shows the ratio of selected studies after reading, analysis, and filtering. The initial sample of 393 studies was reduced to 68, a decrease of about 82.70%.

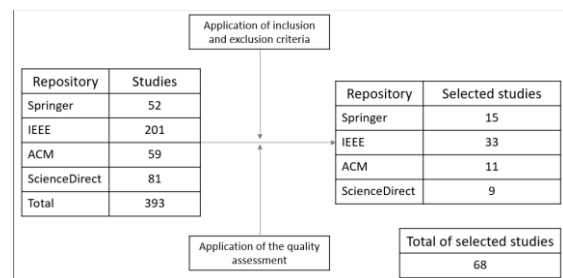


Figure 3. Study selection and reduction process.

Upon the completion of this selection and reduction stage, 68 studies were identified to manifest the capability to answer all of research questions.

### 3.7 Threats to Validity

A general issue related to publication bias is the trend of researchers to publish positive research outcomes rather than negative ones. This threat was considered low since the research questions in the SLR do not address the performance or efficiency of the approaches to TCP. The same applies to the risk of sponsorship bias. Information sources were not limited to specific publishers, journals, or conferences. Instead, the review covered a variety of scientific media, including major conferences and journals in the area sponsored by different publishers. To ensure reliable information, technical reports, works in progress, unpublished, or non-peer-reviewed publications were not included.

We obtained the initial studies using the search string from Section 3.5. We measured the quality of the search by

its recall, which is the ratio of retrieved relevant studies to all known relevant studies. Since we cannot know all relevant studies, we estimated recall using a pilot search on Google Scholar and refined the search string iteratively until it reached at least 85% recall for known studies. Despite this strategy, there remains a risk of missing relevant studies.

To ensure consistency in our data extraction form, we conducted a pilot test using 10% of the selected studies (7 out of 68). We simulated an analysis with this data to verify if it could answer our research questions. After refining the form, we created categories and parameterized attributes for a systematic extraction process. Although this strategy helped mitigate threats to consistency, risks such as missing important data and subjective judgments by researchers remain.

## 4. Results and Analysis

This section describes the results obtained with regard to the research questions raised. First, a summary of the studies used in this SLR is presented, then each research question is answered. In order to organize TCP methods, the works of Catal and Mishra (2013); and Yoo and Harman (2012) were carefully analyzed, based on what these works proposed, it was possible to arrive at a grouping of the methods for TCP based on their characteristics, this grouping is shown in Figure 4.

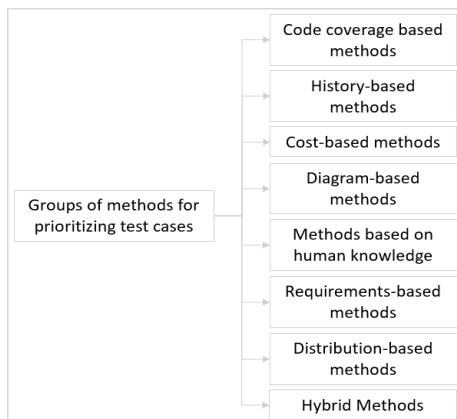


Figure 4. TCP methods groups.

Figure 5 displays the number of methods for TCP found in a grouped form and the percentage of representation of each group. The characteristics used for grouping the methods were defined by Catal and Mishra (2013); and Yoo and Harman (2012). The number of methods in each group is an important measure, it allows a quantitative comparison between the groups and shows which group is more present in literature, however, it is a measure restricted to quantity and does not deep into the reasons why one group has a higher or lower number of methods than others.

In the taxonomy presented in Figure 6, software testing is defined as a process that has three major branches. The first one is about defining the scope of testing, and its purpose is to map what will be tested. After delimiting what will be tested, it is necessary to understand how to test; this is the goal of the second aspect. The third aspect is the prioritization of test cases, this aspect has the objective of defining the execution order of test cases and is the

central theme of this SLR.

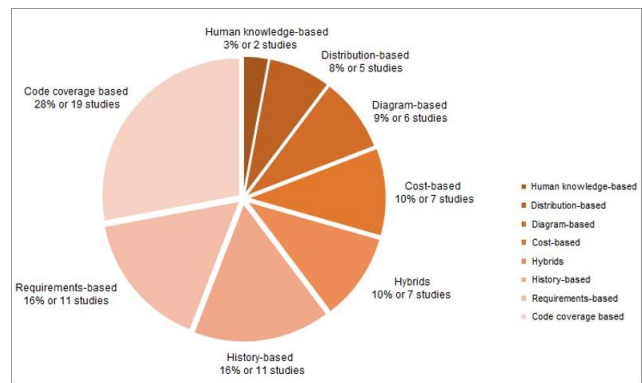


Figure 5. Proportion of methods per group.

In order to optimize prioritization activities, a vast amount of methods or approaches to prioritize test cases have been created. This wide range of methods is presented in the taxonomy in a grouped form, as it allows us to visualize different ways to prioritize test cases. The idea is that the decision of how to prioritize test cases starts from the choice of a certain group of methods, based on the characteristic that is most adherent to the application context.

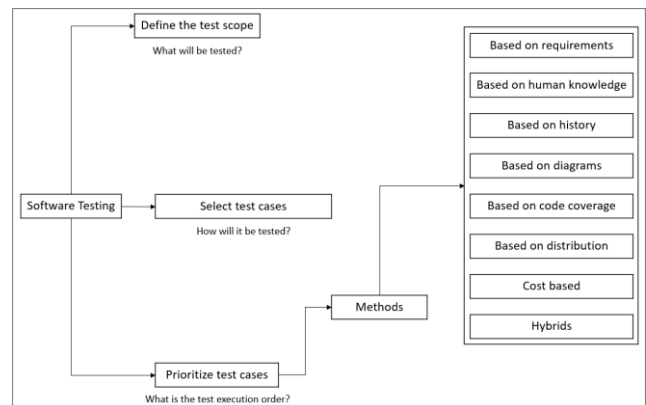


Figure 6. Software testing taxonomy.

Once you have determined a group of methods, you can individually analyze the methods in the chosen group and define which method will be the most effective according to its adherence to the application context. The software testing taxonomy presented by Figure 6 displays grouping the methods for TCP proposed by the analyzed studies, this grouping was carried out based on characteristics proposed by Catal and Mishra (2013); and Yoo and Harman (2012).

In order to answer **RQ1**, queries on the Google Scholar search platform were performed for all 68 selected studies, the purpose of this query was to map the number of citations of each study. Table 2 displays the number of times each study was cited in literature and its relevance (see Appendix A).

To define the relevance of the studies, the strategy was to relate the year of publication to the number of citations. Relevance is calculated based on dividing the number of the study citations by the number of years it has been published. In order to answer the research questions **RQ2 and RQ4**, in the next subsections (4.1 to 4.8) a discussion will be presented about the main methods for prioritizing



test cases and also their main advantages and disadvantages of application.

#### 4.1 Methods based on source code coverage

Methods for TCP based on code coverage prioritize the test cases so that the source code is 100% covered. This coverage can be done through various aspects of the code, the main ones found in the literature are shown in Figure 7.



Figure 7. Main aspects used in source code coverage-based methods.

The methods for TCP based on source code coverage found were Rothermel et al. (1999); Elbaum et al. (2000); Jones and Harrold (2003); Kwon et al. (2014); Zhang et al. (2013); Zhou (2010); Jiang et al. (2009); Huang et al. (2020); Solanki et al. (2015); Eghbali and Tahvildari (2016); Vedpal et al. (2014); Ganjkhani and Afsharchi (2019); Panigrahi and Mall (2014); Fang et al. (2014); Huang et al. (2014); Alazzam and Nahar (2018); Masri and El-Ghali (2009); Hao et al. (2014); Romano et al. (2018). The methods Chen et al. (2004); Zhu et al. (1997); Nie and Leung (2011) were manually included in the SLR because the search strings used did not find them automatically, these manually included studies were not considered in the accounting exposed in Figures 2 and 3.

Even though full coverage of the source code is a positive thing with regard to the ability to reveal faults, these methods are usually costly from the point of view of implementation and test execution time, reducing this cost is the main motivation of the most recent studies that propose methods of this type. The studies that propose methods based on source code coverage are 19 out of 68, that is, they represent about 28% of the sample used in this SLR, the first studies found on this group of methods are Rothermel et al. (1999); Elbaum et al. (2000); Jones and Harrold (2003).

In order to answer **RQ2**, the main methods for TCP from this group will be described as following. The study Rothermel et al. (1999) is the pioneer in the idea of developing methods for TCP to maximize source code coverage and serves as the basis for more recent studies. Rothermel et al. (1999) presents variations for TCP in order to provide different ways of covering the source code, these are branch-total, branch-additional, statement-total, statement-additional, total-fault and additional-fault.

The branch-total method prioritizes test cases based on the number of decision structures or branches they cover, the more branches a test case covers, the higher its execution priority. Similarly, the statement-total method prioritizes test cases based on the amount of source code statements they cover.

The branch-additional and statement-additional methods prioritize test cases based on the decision structures and branches (branch-additional) or instructions

in the code (statement-additional) that are not yet covered, the idea is to make sure that at least one test case covers each decision structure, branch or instruction. The total-fault and additional-fault methods prioritize test cases according to their probability of revealing failures based on their execution history, in additional-fault a source code analysis is performed when selecting a test case that assigns a lower priority to test cases similar to the one selected.

Elbaum et al. (2000) extends the scope of Rothermel et al. (1999) by including the function-total and function-additional methods, both of which take functions in the source code for TCP, in the function-total method tests are prioritized according to the total functions they cover; in the function-additional method tests are prioritized according to the total functions that have not yet been covered.

Jones and Harrold (2003) proposes methods for reducing and prioritizing the test suite, essential test cases are initially selected, then is assigned a contribution to the test case and later there is the removal of test cases with lower contribution, according to code coverage criteria. The prioritization of the defined test suite uses the concept of the additional method proposed in Rothermel et al. (1999) as a basis, re-calculating the contribution of test cases after selecting each test case according to criteria of code coverage in instructions and entities.

Zhang et al. (2013) also uses the study Rothermel et al. (1999) as a basis for developing a TCP method by combining the general concepts of the total and additional methods, which respectively prioritize test cases based on the total number of code elements covered by test cases and number of code elements not yet covered by test cases.

Another approach to prioritize test cases based on source code coverage is the prioritization from retrieving information from the code itself. In order to cover the least tested code, Kwon et al. (2014) proposes the retrieval of Term Frequency and Inverted Document Frequency information.

Term Frequency is the periodicity in which an element (line of code, method or class) of the source code is executed by a test case, and Inverted Document Frequency is the opposite of Document Frequency, which is the number of test cases that cover a given element. The logic is that if an element has a high Inverted Document Frequency, this element is not covered by many test cases, so the test case that covers this element will have its priority increased. In this approach, elements of code that are executed by fewer test cases have a higher probability of having faults, so the test cases that cover these elements should be prioritized; the prioritization of the test cases is done by using a similarity index, calculated from a linear regression model taking into account Term Frequency and Inverted Document Frequency.

Still on source code coverage, there are also methods that make use of similarity analysis between the test cases and the source code to determine a suite of tests with the highest possible coverage of the code, these methods usually use code elements such as snippets, functions, branches and

methods. Zhou (2010); Jiang et al. (2009); Huang et al. (2014) are examples of methods with this feature, they are based on the Adaptive Random Testing (ART) method created by Chen et al. (2004), ART uses randomness to define and prioritize test cases applying statistical concepts of uniform distribution to extend fault detection.

The most recent study found on prioritization methods based on source code coverage is Huang et al. (2020), the method proposed in this study is called code combinations coverage based prioritization (CCCP) and selects the test case that covers the largest number of source code elements, then performs a calculation to evaluate the code combinations coverage (CCC) of the other test cases, the prioritization of the other test cases occurs according to the largest CCC calculated. The method proposed in Huang et al. (2020) is based on a combination of two existing methods in the literature, Zhu et al. (1997); Nie and Leung (2011).

In order to answer RQ4, the main advantages and disadvantages of the analyzed methods will be described. The disadvantage of methods using ART Zhou (2010); Jiang et al. (2009); Huang et al. (2014) is the complexity and time required to select and prioritize test cases, despite this, the experiments performed in Zhou (2010); Jiang et al. (2009); Huang et al. (2014) demonstrate that these methods are more efficient at revealing faults compared to simply random prioritization. The disadvantage of the reduction method proposed by Jones and Harrold (2003) is that by removing a test case from the suite permanently, the ability to reveal faults in the suite may be affected. The advantages of methods for TCP based on source code coverage are a reduction in the cost of manual test execution and also an increase in the fault detection rate.

## 4.2 Methods based on human knowledge (RQ1 and RQ2)

Human knowledge-based methods for TCP are the least representative in literature analyzed in this SLR, in general its proposal is to prioritize the test cases taking into consideration information and opinions of the people performing the tests.

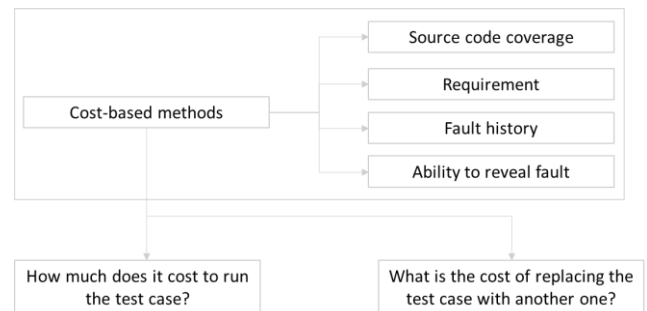
In order to answer RQ2, the main methods for TCP from this group will be described as following. In the method proposed by Tonella et al. (2006), a suite of tests is defined and prioritized by the tester, the method successively refines the prioritization by comparing the priority of a test case with its peers using a sorting and ranking function.

The method proposed by Malz and Göhner (2011) combines the knowledge of the test team with fuzzy logic to prioritize the test cases. The data source used to prioritize the test cases are the test teams, the developers, and the agents, which are the test cases and modules of the software. This method considers the historical information of execution and failure of test cases and the knowledge of people to assign each test case a value that corresponds to its execution priority. This test case priority determination is performed through fuzzy logic, because according to the authors it offers the possibility to reduce human interaction and also the imprecision and bias of human evaluation.

In order to answer RQ4, the main advantages and disadvantages of the analyzed methods will be described. The positive point of the method proposed by Tonella et al. (2006) is its wide applicability, as little information is required, this method can be applied in several contexts. In contrast, the scalability of the proposed method is compromised as the test suite increases, since the data input is performed by a human tester. The method proposed by Malz and Göhner (2011) solves the deficiency of the method proposed by Tonella et al. (2006) in the aspect of scalability, however in the aspect of complexity in development and execution, the method proposed by Tonella et al. (2006) is more advantageous. One gap found in the analyzed works was the participation of developers and also customers in the prioritization of test cases, this could increase the efficiency and quality of the testing activity, since different perspectives tend to improve the coverage and ordering of the test suite.

## 4.3 Cost-based methods

Cost-based TCP methods have a basic premise: they consider that test cases do not have the same execution cost. In this SLR, by analyzing the methods proposed by Elbaum et al. (2001); Alspaugh et al. (2007); Parashar et al. (2012); Wang et al. (2014); Lin et al. (2014); Bryce et al. (2011); Wu et al. (2014) it is concluded that cost-based TCP methods are complementary to other kinds of methods. This statement is proven based on the combination of the aspects used along with the cost of executing the test cases to prioritize them. Figure 8 shows the main aspects used by these methods in cost-based prioritization.



**Figure 8.** Main aspects used as a source of information for cost-based methods.

In order to answer RQ2, the main methods for TCP from this group will be described as following. For example, Elbaum et al. (2001) uses historical data to help prioritize test cases. In this method, the cost of the test case is set based on its execution history, even if the runtime history does not accurately reflect the future execution cost of the test case, one must accept this assumption and be satisfied with this historical measurement that even if not exact is accurate enough. Alspaugh et al. (2007) uses information from source code coverage to help prioritize test cases. In this method, test cases are prioritized by combining their code coverage, their execution cost, and the benefit of executing them

over similar or nearby test cases. There are also methods that take into consideration the difference in severity of faults revealed by test cases Elbaum et al. (2001); Alspaugh et al. (2007); Parashar et al. (2012). The concern in revealing faults in these methods goes beyond the quantitative analysis of faults and also adopts qualitative criteria, considering the severity and cost of their existence.

Another relevant method to prioritize test cases based on cost is the one proposed by Wang et al. (2014), this method seeks to optimize the prioritization of test cases aiming to minimize the cost and maximize the test suite. This method considers the cost of test case execution in terms of time and resource allocation for execution; the code coverage that can be achieved with the defined test suite; the test suite's ability to detect flaws and, finally, the prioritization dimension, which relates the amount of test cases that can be prioritized before all available resources are allocated.

Lin et al. (2014) explains that most methods focus on decreasing the size of the test suite, however they do not usually take into consideration the cost of test case execution, in the author's view this is an opportunity for improvement in methods, since the differences in execution costs between test cases are significant for defining and prioritizing the test cases that will be executed. The method proposed by Lin et al. (2014) focuses on reducing the test suite by taking into consideration the historical cost of running the test cases, the requirements covered by the test cases, and also the possibility that each test case can be replaced by others during test suite reduction.

In order to answer **RQ4**, the main advantages and disadvantages of the analyzed methods will be described. The main advantage found in cost-based TCP methods is their ability to take into consideration the cost of executing the test cases; furthermore, assuming that the faults revealed by the test cases have a different degree of severity, that is, software impact is an important factor in prioritization.

In contrast, cost-based methods usually have a higher complexity when compared to coverage-based methods. This is because cost-based methods measure, analyze and classify a larger amount of variables to assist in prioritization, some examples of this are the methods Alspaugh et al. (2007); Wang et al. (2014); Wu et al. (2014) that in addition to analyzing code coverage, also take into consideration the cost of executing and replacing test cases and the cost of faults in test cases.

#### 4.4 Diagram-based methods

Diagram-based TCP methods use information from UML (Unified Modeling Language) models to perform TCP, the methods of this type found in the literature are Korel et al. (2005, 2008); Mahali and Mohapatra (2018); Sapna and Mohanty (2009); Wang et al. (2015a); Panda et al. (2019). Figure 9 shows the main aspects used by these methods in diagram-based prioritization.



**Figure 9.** Main aspects used as a source of information for diagram-based methods.

In order to answer **RQ2**, the main methods for prioritizing test cases from this group will be described as following. In the method proposed by Korel et al. (2005), the test cases are allocated to a high or low priority test suite according to their relevance for coverage and validation of changes made in a model called EFSM (Test Prioritization Using System Models), the EFSM is used to capture different aspects of the system's behavior; the prioritization of the test cases contained in the high and low priority suites occurs randomly.

This model, called EFSM, serves as the basis for test map-ping and is diagrammed using graphs, where states are nodes and transitions are edges. To extract, map and derive test cases from the EFSM model, the following UML diagrams are used: sequence diagram and activity diagram.

The method proposed by Korel et al. (2008) was created in order to improve the method proposed by Korel et al. (2005) by including three heuristics based on model dependency analysis.

1. The first heuristic considers that each code change is called a transition in the EFSM model, the test case that performs a higher number of transitions should have a higher priority than test cases that perform a lower number of transitions, because it has a higher probability of revealing faults;
2. The second heuristic considers that test cases from the lowest priority suite can also reveal faults, so the elaborate algorithm tries to give them a better chance of being selected;
3. The third heuristic considers that each transition should have an equal chance of being executed, this heuristic tries to balance the number of executions of the transitions, so that the test cases are prioritized according to the transition that has been validated a smaller number of times.

The method proposed by Mahali and Mohapatra (2018) uses the sequence and activity diagrams modeled from the requirements to generate an ASG (Activity Sequence Graph), through the independent paths contained in the ASG the test cases are mapped, each different path in the ASG is a test case. Simultaneously, the ASG information is stored in a database, which is queried to define the priority of the test cases.

In order to answer **RQ4**, below the main advantages and disadvantages of the analyzed methods will be described. The advantage presented by the authors in Korel et al. (2005, 2008) over diagram-based methods is that they are less complex than coverage-based methods;

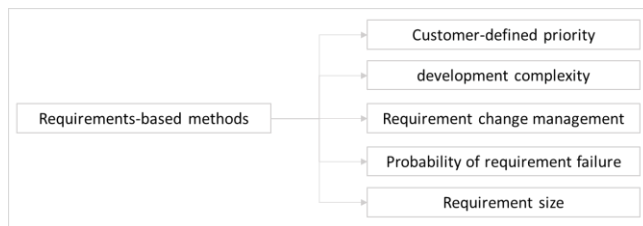


the disadvantage presented in Korel et al. (2005, 2008) is that not all information extracted from EFSM models is useful, there is information that does not extend fault detection.

One point to be taken into consideration for the methods in this group is that the requirements specification does not always include a sequence or activity diagram. These diagrams usually address architectural aspects and are not always mandatory artifacts in the software development process, so it is necessary to keep in mind that when you choose to use methods for prioritizing test cases from this group, the diagrams should be treated as prerequisites for TCP to occur.

## 4.5 Requirements-based methods

Requirements-based TCP methods use information from requirements to prioritize test cases, the methods of this type found in the literature are Srikanth et al. (2005); Arafeen and Do (2013); Krishnamoorthi and Sahaaya Arul Mary (2009); Hettiarachchi et al. (2016); Srikanth et al. (2016b); Srivastva et al. (2008); Panda and Mohapatra (2021); Hettiarachchi et al. (2014); Ma et al. (2016); Kavitha et al. (2010); Liu et al. (2014). Figure 10 shows the main aspects used by these methods in requirements-based prioritization.



**Figure 10.** Main aspects used as a source of information for requirements-based methods.

In order to answer RQ2, the main methods for TCP from this group will be described as following. The study Srikanth et al. (2005) is the pioneer in the idea of developing methods to prioritize test cases aiming at using requirements as the main source of information. The method proposed in this study for prioritizing test cases based on requirements takes into account four factors: priority of the requirements according to the customer, complexity to develop the requirement according to the developer, volatility of the requirement and failure propensity of the requirement.

Srikanth et al. (2005) developed the PORT (Prioritization Of Requirements for Testing) tool to support the proposed method, the idea of PORT is to receive the values assigned to the four factors for a given requirement and then calculate a prioritization factor for it. The priority of the test cases that cover a given requirement is the result of the product between the prioritization factor of the requirement associated to the test case and the coverage of the requirement provided by the test case.

Srikanth et al. (2016b) analyzed the method proposed by Srikanth et al. (2005) and came to the conclusion that the most relevant factors for prioritizing test cases are: the priority assigned by the customer and the propensity of the

requirement to fail. Srikanth et al. (2016b) thoroughly investigated these two factors and applied TCP based on them, the result was that using the two factors in combination amplifies the effectiveness of TCP, therefore, the authors propose the evolution of PORT to PORT 2.0 using only these two factors for TCP.

Arafeen and Do (2013) performs the clustering of requirements using K-means, from the clustered requirements, a test case traceability matrix is used to extract and group the test cases associated with each cluster of requirements. With the test case cluster formed, prioritization of the test cases in each cluster occurs through a complexity metric that is calculated by considering a combination of three factors:

1. Number of lines of code: total lines of the class, not counting comments and blank lines;
2. Nested block depth: the number of nested rows in a method or function;
3. McCabe Cyclomatic Complexity: Number of linearly independent paths through the decision structures contained in the method.

Prioritization of test case clusters occurs based on customer requirements, delivery roadmap, developer implementation commitment, and source code change information.

The prioritization of test cases proposed by the method developed by Krishnamoorthi and Sahaaya Arul Mary (2009) is based on six factors: customer priority, changes in requirements, implementation complexity, integrity, traceability, and failure impact. In this method, the prioritization of test cases occurs according to the steps described below.

1. Get the total number of requirements and the total number of test cases planned for the project to be tested;
2. Assign value to the six factors for all requirements, this assignment is done by people involved in development;
3. Assign a weight to each requirement, this weight is calculated based on the average between the six factors;
4. Analyze and map the test cases that cover each requirement;
5. Assign a weight to each test case associated with the requirement based on the fraction of the requirement covered by the test case;
6. Sort the test cases in descending order of their weights. The test cases with the highest weight will be executed before the others.

There are also methods that use fuzzy inference to improve the result of test cases prioritization, for example the method proposed by Hettiarachchi et al. (2016) uses the modification status, complexity, security and size of the requirements as risk indicators, these risk factors of each requirement are estimated by a system that uses fuzzy logic. In general, the premise adopted in this method is that if a requirement is critical and cannot contain failures,

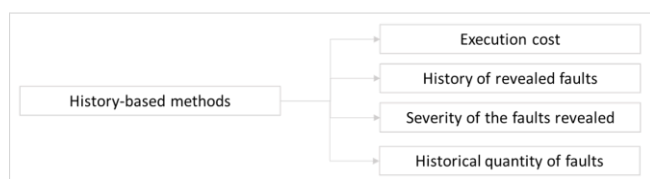
the test cases linked to it have a high priority.

In order to answer **RQ4**, the main advantages and disadvantages of the analyzed methods will be described. The main advantages found in the group of requirements-based methods are: considering the priority assigned by the customer and considering the opinion of people from the development team when prioritizing the test cases. Another advantage to consider is that the requirement is a mandatory artifact within the software development process, that is, it is a source of information with no risk of non-existence. This advantage is mainly taken into consideration when making a comparison between diagram-based and requirements-based methods for prioritizing test cases.

On the other hand, the efficiency of this group of methods could be compromised when applied in environments where requirements are highly changeable, it is necessary to consider that the requirement needs to be well written, that is, be clear, achievable, necessary and testable. Meeting these aspects and ensuring the understanding and alignment of all the actors that participate in the software development process is not a simple task.

#### 4.6 History-based methods

Methods for history-based TCP use information from the execution history of the test cases and the fault history revealed by them, the methods of this type found in the literature are Kim and Porter (2003); Park et al. (2008); Fazlalizadeh et al. (2009); Khalilian et al. (2012); Huang et al. (2012); Marijanet et al. (2013); Wang et al. (2015b); Wang and Zeng (2016); Srikanth et al. (2016a); Kim et al. (2017); Magalhães et al. (2021). Figure 11 displays the main aspects used by these methods in history-based prioritization.



**Figure 11.** Main aspects used as a source of information for history-based methods.

In order to answer **RQ2**, the main methods for TCP from this group will be described. The study Kim and Porter (2003) pioneered the idea of developing methods to prioritize test cases using history as the main source of information and uses a combination of test case execution history and a function coverage analysis in the source code to prioritize the test cases, which occurs in two steps. First, a weight is assigned to each role according to the number of test cases that cover it; the less coverage a role has, the higher the weight of test cases associated with it. The second step is to set the historical value of the test case based on its execution history. Test cases with a higher weight, that is, a lower historical execution frequency, are given a higher priority.

Park et al. (2008) proposes a method that focuses on using information about test case cost and fault severity history, a historical value for the test case is calculated based on these factors and will serve as the basis for prioritization.

The method developed by Fazlalizadeh et al. (2009) uses the method developed by Kim and Porter (2003) as a basis, which is improved by including the fault detection capability that the test case has as a factor for prioritizing the test cases, which is now performed based on a combination of the execution history and the number of faults revealed by the test case.

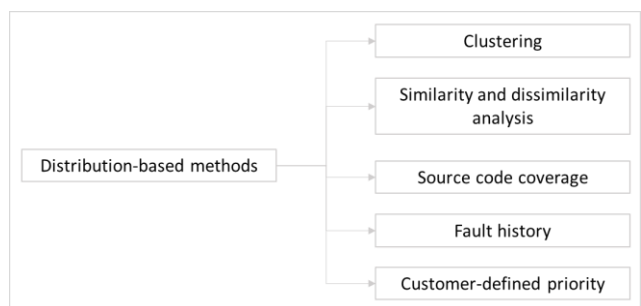
In order to answer **RQ4**, the main advantages and disadvantages of the analyzed methods will be described. The advantage of the method proposed by Wang and Zeng (2016) is that it consults the customer and developers to prioritize the test cases, the disadvantage found in Wang and Zeng (2016) is that the method does not consider that there may be redundant faults in the fault division for each requirement.

Since cost, type and severity of failure may have different priority depending on the context, the method proposed by Park et al. (2008) could be increased to consider this point and allow the parameterization of different levels of cost, type and severity of failure in the calculus of the historical value.

Analyzing methods from this group, one advantage found is that considering the test case execution history for future prioritizations is a way to increase the failure detection rate in regressive testing, since the test cases that usually reveal a higher historical number of failures are the ones that usually validate the most complex parts of the application.

#### 4.7 Distribution-based methods

Distribution-based TCP methods use the similar aspects of test cases to group and then prioritize them. The methods of this type found in the literature are Leon and Podgurski (2003); Carlson et al. (2011); Khalid and Qamar (2019); Biswas et al. (2022); Gokilavani and Bharathi (2021), Figure 12 displays the main aspects used by these methods in distribution-based prioritization.



**Figure 12.** Main aspects used as a source of information for distribution-based methods.

In order to answer **RQ2**, the main methods for TCP from this group will be described as following. The study performed by Leon and Podgurski (2003) is a pioneer in the idea of developing methods to prioritize test cases in

order to group and distribute them based on their characteristics, its operation goes through a dissimilarity function that generates a real number to represent the degree of dissimilarity among the test cases, which are selected by group and prioritized according to the order in which the groups are created.

Another very relevant method in this group is the one pro-posed by Carlson et al. (2011), in which the grouping of the test cases is done by similarity in terms of code coverage and their prioritization can be done in several ways, as described below.

1. Code coverage: prioritize the test cases in each cluster according to how many methods they cover;
2. Code complexity: prioritize the test cases based on the average ratio between the number of lines of code in the class and the dependency between methods;
3. Failure history: division of the number of failures detected by the total number of failures, coming from the test execution history;
4. Combination of fault history and code complexity: arithmetic mean between code complexity and the ratio of detected faults.

Khalid and Qamar (2019) develop a method for distribution-based TCP that takes the customer's priorities into consideration. In this method, the customer gives a weight or importance to the business requirement, so the customer's needs are given higher priority. Based on the importance reported by the customer, three clusters are created using K-Means and classified as high, medium and low priority, each test case has its cost and execution time calculated, based on these calculated factors the test cases are distributed into sub clusters, classified as high, medium or low priority using the K-Medoids algorithm and finally prioritized based on customer priority, cost and execution time.

One of the recent relevant works in this group of methods is the one proposed by Gokilavani and Bharathi (2021), in which the selection of test cases is performed using the PCA (principal component analysis) algorithm, the attributes used for clustering and prioritization are defined using dimensionality reduction and clustering is performed using K-means, and finally the test cases in each group are prioritized using a classification algorithm.

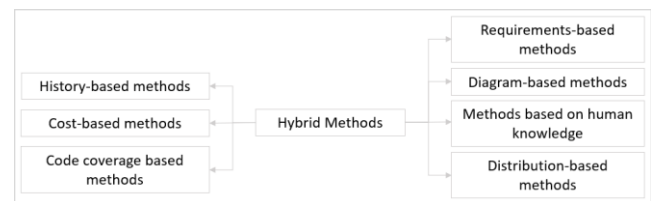
In order to answer **RQ4**, the main advantages and disadvantages of the analyzed methods will be described. An advantage of the method proposed by Khalid and Qamar (2019) is the presence of the customer in the prioritization, once the customer informs the importance that a certain requirement has, it is possible to prioritize and direct the efforts in quality assurance of what has a higher value for the customer.

In the method proposed by Carlson et al. (2011), the advantage presented is the combined use of source code coverage aspects with distribution aspects, according to the authors the combination was a determining factor for effectiveness in fault detection. The methods Khalid and Qamar (2019); Gokilavani and Bharathi (2021) have the limitation of specifying the amount of clusters in advance in

K-Means, however by using this algorithm the ability to find similar or related test cases is extended, which contributes to the reduction of the test suite and prioritization of test cases.

## 4.8 Hybrid Methods

Hybrid TCP methods are those that combine elements from two or more groups of methods to prioritize test cases without the predominance of one group over another, as demonstrated by the Figure 13. Methods of this type found in the literature are Yadav and Dutta (2016); Dobuneh et al. (2014); Su et al. (2020); Siddik and Sakib (2014); Jahan et al. (2020); Thomas et al. (2014); Yadav and Dutta (2020).



**Figure 13.** Hybrid methods combine features from two or more method groups.

In order to answer **RQ2**, the main methods for TCP from this group will be described as following. In the method proposed by Yadav and Dutta (2016) a fuzzy logic is developed and used for prioritizing the test cases based on the calculation and combination of execution time, failure detection rate and requirement coverage. Dobuneh et al. (2014) proposes a method that prioritizes the test cases based on the highest number of HTTP requestson the web pages, the length of the HTTP request and the dependencies between HTTP requests, this data used in the prioritization of the test cases is obtained from the application log files.

Siddik and Sakib (2014) prioritizes test cases based on a combination of:

- Requirements specification: search for the similarity of requirements and create clusters to group them using K-means;
- Source code: calculation of the priority of source code based on three pieces of information: the total number of lines of code executed, the depth of the aligned block of code in a class or method, and the level of association and dependency between two methods or classes;
- UML diagrams: extracting information from UML diagrams and converting it to XML so that the main activities contained in the diagrams are mapped and prioritized.

By using an algorithm called topic modeling, the method proposed by Thomas et al. (2014) performs the extraction of linguistic data from test cases to map in a vector the topics covered by each test case. Subsequently, a distance analysis is performed between the test cases in the vector to map the similarity between them, since it is assumed that similar test cases are functionally equal and tend to reveal the same faults. The prioritization of test

cases in Thomas et al. (2014) occurs with the purpose of selecting test cases in a prioritized way, maximizing the average distance between them and thus obtaining a greater coverage and comprehensiveness of functionality.

In order to answer **RQ4**, the main advantages and disadvantages of the analyzed methods will be described. In Yadav and Dutta (2016) we see that using fuzzy logic to prioritize test cases is somewhat safe and effective, however, it has added complexity. The method proposed by Dobuneh et al. (2014) is exclusively applied to web systems and has a dependency on good log files, which besides having the necessary information for prioritization need to be in a format that favors the retrieval and reuse of information. When analyzing the methods of this type, in general, a common positive point was found, the hybrid combination of the prioritization criteria increases the percentage of faults found in the applications.

#### 4.9 Prioritizing tests in agile environments (RQ3)

In order to answer **RQ3**, the 68 studies proposing methods for prioritizing test cases contained in this SLR were analyzed. The analysis was performed on two fronts, the first being the textual search in the study for the terms “agile”, “agile methods”, “agile software testing”, “scrum” and “kanban”. The result obtained was that the searched terms were not mentioned in any of the 68 studies.

The second front of analysis was a careful reading of the sections elaborating the prioritization method proposed by the study and its application in a given context; the purpose here was to search for a method developed or applied in an environment where agile methods are present. The result obtained was that none of the 68 studies reviewed proposed or applied a method for prioritizing test cases in an environment that uses agile methods.

## 5. Conclusion

This study aimed to identify, analyze, or compare methods for TCP. To achieve this, four research questions were raised and answered using a designed research strategy shown in Figure 1. Studies were selected, evaluated, and data extracted to address these questions. Results are presented in graphs, figures, and tables, with answers provided for each method group proposed by Catal and Mishra (2013) and Yoo and Harman (2012).

The main findings resulted from this SLR are the following:

- There are a vast number of methods for TCP in literature, which makes the subject relevant to science and industry;
- None of the methods for prioritizing test cases have been developed specifically for agile software development teams;
- Not all methods for prioritizing test cases that have been studied have advantages or disadvantages over methods from the same or a different group of

features;

- The studies comparing a prioritized execution of test cases with a non-prioritized or randomly prioritized execution concluded that prioritizing test cases extends fault detection and optimizes the time it takes to execute test cases manually or automatically;
- It is possible to create hybrid methods for TCP using features from more than one group, however their efficiency in fault detection needs to be compared with methods from each group in isolation, this may be a point to be discussed in future research;
- Cost-based TCP methods also use elements from more than one group of methods, however they have one predominance in prioritization: the cost of running or replacing the test case; this predominance makes these cost-based methods not classified as hybrids.

New research can be conducted on the topic of test case prioritization, in order to analyze the application of the methods in environments that use agile methodologies, this analysis can be conducted as follows:

- Analysis of adherence and effectiveness of the criteria or parameters used by the method to prioritize test cases;
- Ability of the methods to reveal faults in agile environments, where deliveries are iterative and incremental;
- Compare the impact caused by using and not using a method in prioritizing test cases, considering quantity and criticality of the flaws found together with the time required to reveal and correct them;
- Analysis of the impact of using the method on the team's delivery capacity. The purpose of this point would be to understand if the use of a method to prioritize the test cases causes any impact on the team's delivery flow.

Future work should aim to qualitatively understand why certain feature groups have more or fewer methods than others. This SLR and existing studies focus mainly on quantitative aspects, leaving a qualitative gap that needs exploring.

## References

- Alazzam, I. and Nahar, K. M. O. (2018). Combined source code approach for test case prioritization. In Proceedings of the 2018 International Conference on Information Science and System - ICISS '18, New York, New York, USA. ACM Press.
- Alspaugh, S., Walcott, K. R., Belanich, M., Kapfhammer, G. M., and Soffa, M. L. (2007). Efficient time-aware prioritization with knapsack solvers. In Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE)



- 2007 - WEASEL Tech '07, New York, New York, USA. ACM Press.
- Arafeen, M. J. and Do, H. (2013). Test case prioritization using requirements-based clustering. In 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation.
- Ashima, G. Shaheamlung and K. Rote. (2020). A comprehensive review for test case prioritization in Software Engineering, 2020 International Conference on Intelligent Engineering and Management (ICIEM), London, UK, 2020, pp. 331–336, doi: 10.1109/ICIEM48762.2020.9160217.
- Biswas, S., Rathi, R., Dutta, A., Mitra, P., and Mall, R. (2022). A regression test case prioritization technique targeting 'hard to detect' faults. *International Journal of System Assurance Engineering and Management*, pages 1066–1081.
- Black, R. (2019). *Foundations of software testing: ISTQB certification (4th ed.)*. Cengage Learning.
- Brooks, Jr, F. (2021). *Mythical man-month, anniversary edition*, the. Addison Wesley, Boston, MA, 2 edition.
- Bryce, R. C., Sampath, S., Pedersen, J. B., and Manchester, S. (2011). Test suite prioritization by cost-based combinatorial interaction coverage. *Int. J. Syst. Assur. Eng. Manag.*, 2(2):126–134.
- Campos Junior, H. S., Araújo, M. A. P., David, J. M. N., Braga R., Campos, F. and Ströele, V. (2017). Test case prioritization: a systematic review and mapping of the literature. In *Proceedings of the XXXI Brazilian Symposium on Software Engineering (SBES '17)*. Association for Computing Machinery, New York, NY, USA, 34–43. <https://doi.org/10.1145/3131151.3131170>
- Carlson, R., Do, H., and Denton, A. (2011). A clustering approach to improving test case prioritization: An industrial case study. In 2011 27th IEEE International Conference on Software Maintenance (ICSM). IEEE.
- Catal, C. and Mishra, D. (2013). Test case prioritization: a systematic mapping study. *Softw. Qual. J.*, 21(3):445–478.
- Chen, T. Y., Leung, H., and Mak, I. K. (2004). Adaptive random testing. In *Advances in Computer Science - ASIAN 2004. Higher-Level Decision Making*, Lecture notes in computer science, pages 320–329. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Dobunch, M. R. N., Jawawi, D. N., Ghazali, M., and Malakooti, M. V. (2014). Development test case prioritization technique in regression testing based on hybrid criteria. In 8th. Malaysian Software Engineering Conference (MySEC), pages 301–305.
- Eghbali, S. and Tahvildari, L. (2016). Test case prioritization using lexicographical ordering. *IEEE Trans. Softw. Eng.*, 42(12):1178–1195.
- Elbaum, S., Malishevsky, A., and Rothermel, G. (2001). Incorporating varying test costs and fault severities into test case prioritization. In *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, pages 329–338.
- Elbaum, S., Malishevsky, A. G., and Rothermel, G. (2000). Prioritizing test cases for regression testing. *Softw. Eng. Notes*, 25(5):102–112.
- Fairley, R. E. (1985). *Software engineering concepts*. In McGraw-Hill series in software engineering and technology.
- Fang, C., Chen, Z., Wu, K., and Zhao, Z. (2014). Similarity-based test case prioritization using ordered sequences of program entities. *Softw. Qual. J.*, 22(2):335–361.
- Fazlalizadeh, Y., Khalilian, A., Abdollahi Azgomi, M., and Parsa, S. (2009). Incorporating historical test case performance data and resource constraints into test case prioritization. In *Tests and Proofs, Lecture notes in computer science*, pages 43–57. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Ganjkhani, E. and Afsharchi, M. (2019). An effective test case prioritization by combination of strategies. *SN Appl. Sci.*, 1(9).
- Gokilavani, N. and Bharathi, B. (2021). Test case prioritization to examine software for fault detection using PCA extraction and k-means clustering with ranking. *Soft Comput.*, 25(7):5163–5172.
- Hao, D., Zhang, L., Zhang, L., Rothermel, G., and Mei, H. (2014). A unified test case prioritization approach. *ACM Trans. Softw. Eng. Methodol.*, 24(2):1–31.
- Hasnain, M., Pasha, M.F., Ghani, I. et al. *Functional Requirement-Based Test Case Prioritization in Regression Testing: A Systematic Literature Review*. *SN COMPUT. SCI.* 2, 421 (2021). <https://doi.org/10.1007/s42979-021-00821-3>
- Hettiarachchi, C., Do, H., and Choi, B. (2014). Effective regression testing using requirements and risks. In 2014, the Eighth International Conference on Software Security and Reliability.
- Hettiarachchi, C., Do, H., and Choi, B. (2016). Risk-based test case prioritization using a fuzzy expert system. *Inf. Softw. Technol.*, 69:1–15.
- Huang, R., Chen, J., Li, Z., Wang, R., and Lu, Y. (2014). Adaptive random prioritization for interaction test suits. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, New York, NY, USA. ACM.
- Huang, R., Zhang, Q., Towey, D., Sun, W., and Chen, J. (2020). Regression test case prioritization by code combinations coverage.
- Huang, Y.-C., Peng, K.-L., and Huang, C.-Y. (2012). A history-based cost-cognizant test case prioritization technique in regression testing. *J. Syst. Softw.*, 85(3):626–637.
- Jahan, H., Feng, Z., and Mahmud, S. M. H. (2020). Risk-based test case prioritization by correlating system methods and their associated risks. *Arab. J. Sci. Eng.*, 45(8):6125–6138.
- Jiang, B., Zhang, Z., Chan, W. K., and Tse, T. H. (2009).

- Adaptive random test case prioritization. In 2009 IEEE/ACM International Conference on Automated Software Engineering, pages 233–244.
- Jones, J. A. and Harrold, M. J. (2003). Test-suite reduction and prioritization for modified condition/decision coverage. *IEEE Trans. Softw. Eng.*, 29(3):195–209.
- Kavitha, R., Kavitha, V. R., and Suresh Kumar, N. (2010). Requirement based test case prioritization. In 2010 IEEE International Conference on Communication Control and Computing Technologies.
- Khalid, Z. and Qamar, U. (2019). Weight and cluster based test case prioritization technique. In 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON).
- Khalilian, A., Abdollahi Azgomi, M., and Fazlalizadeh, Y. (2012). An improved method for test case prioritization by incorporating historical test case data. *Sci. Comput. Program.*, 78(1):93–116.
- Khatibsyarhini, M., Isa, M. A., Jawawi, D. N. A., and Tumeng, R. (2018). Test case prioritization approaches in regression testing: A systematic literature review. *Inf. Softw. Technol.*, 93:74–93.
- Kim, J., Jeong, H., and Lee, E. (2017). Failure history data-based test case prioritization for effective regression test. In *Proceedings of the Symposium on Applied Computing*, New York, NY, USA.
- Kim, J.-M. and Porter, A. (2003). A history-based test prioritization technique for regression testing in resource constrained environments. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*.
- Kitchenham, B. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering.
- Korel, B., Koutsogiannakis, G., and Tahat, L. H. (2008). Application of system models in regression test suite prioritization. In 2008 IEEE International Conference on Software Maintenance.
- Korel, B., Tahat, L., and Harman, M. (2005). Test prioritization using system models. In 21st IEEE International Conference on Software Maintenance (ICSM'05), pages 559–568.
- Krishnamoorthi, R. and Sahaaya Arul Mary, S. A. (2009). Factor oriented requirement coverage based system test case prioritization of new and regression test cases. *Inf. Softw. Technol.*, 51(4):799–808.
- Kwon, J.-H., Ko, I.-Y., Rothermel, G., and Staats, M. (2014). Test case prioritization based on information retrieval concepts. In 2014 21st IEEE Asia-Pacific Software Engineering Conference.
- Leon, D. and Podgurski, A. (2003). A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases. In 14th IEEE International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.
- Lin, C.-T., Tang, K.-W., and Kapfhammer, G. M. (2014). Test suite reduction methods that decrease regression testing costs by identifying irreplaceable tests. *Inf. Softw. Technol.*, 56(10):1322–1344.
- Liu, W., Wu, X., Zhang, W., and Xu, Y. (2014). The re-search of the test case prioritization algorithm for black box testing. In 2014 IEEE 5th International Conference on Software Engineering and Service Science.
- Ma, T., Zeng, H., and Wang, X. (2016). Test case prioritization based on requirement correlations. In 2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD).
- Magalhães, C., Mota, A., and Momente, L. (2021). UI test case prioritization on an industrial setting: A search for the best criteria. *Softw. Qual. J.*, 29(2):381–403.
- Mahali, P. and Mohapatra, D. P. (2018). Model based test case prioritization using UML behavioural diagrams and association rule mining. *Int. J. Syst. Assur. Eng. Manag.*, 9(5):1063–1079.
- Malz, C. and Göhner, P. (2011). Agent-Based test case prioritization. In 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops.
- Marijan, D., Gotlieb, A., and Sen, S. (2013). Test case prioritization for continuous regression testing: An industrial case study. In 2013 IEEE International Conference on Software Maintenance.
- Masri, W. and El-Ghali, M. (2009). Test case filtering and prioritization based on coverage of combinations of program elements. In *Proceedings of the Seventh International Workshop on Dynamic Analysis*, New York, NY, USA.
- Mohd-Shafie, M.L., Kadir, W.M.N.W., Lichter, H. et al. Model-based test case generation and prioritization: a systematic literature review. *Softw Syst Model* 21, 717–753 (2022). <https://doi.org/10.1007/s10270-021-00924-8>
- Nie, C. and Leung, H. (2011). A survey of combinatorial testing. *ACM Comput. Surv.*, 43(2):1–29.
- Panda, N., Acharya, A. A., and Mohapatra, D. P. (2019). Test scenario prioritization for object-oriented systems using UML diagram. *Int. J. Syst. Assur. Eng. Manag.*, 10(3):316–325.
- Panda, N. and Mohapatra, D. P. (2021). Test scenario prioritization from user requirements for web-based software. *Int. J. Syst. Assur. Eng. Manag.*, 12(3):361–376.
- Panigrahi, C. R. and Mall, R. (2014). A heuristic-based regression test case prioritization approach for object-oriented programs. *Innov. Syst. Softw. Eng.*, 10(3):155–163.
- Parashar, P., Kalia, A., and Bhatia, R. (2012). Pair-wise time-aware test case prioritization for regression testing. In *Information Systems, Technology and Management, Communications in computer and information science*, pages 176–186. Springer Berlin Heidelberg, Berlin,

- Heidelberg.
- Park, H., Ryu, H., and Baik, J. (2008). Historical value-based approach for cost-cognizant test case prioritization to improve the effectiveness of regression testing. In 2008, the Second International Conference on Secure System Integration and Reliability Improvement.
- Pressman, R. and Maxim, B. (2019). Loose leaf for software engineering: A practitioner's approach. McGraw-Hill Education, 9<sup>th</sup> edition.
- Rahmani, A., Ahmad, S., Jalil, I. E. A. and Herawan, A. P. (2021). A Systematic Literature Review on Regression Test Case Prioritization. *International Journal of Advanced Computer Science and Applications(IJACSA)*, 12(9), 2021. <http://dx.doi.org/10.14569/IJACSA.2021.0120929>
- Romano, S., Scanniello, G., Antoniol, G., and Marchetto, A. (2018). Spiritus: A simple information retrieval regression test selection approach. *information and software technology*, v. 99. pages 62–80.
- Rothermel, G., Untch, R. H., Chu, C., and Harrold, M. J. (1999). Test case prioritization: an empirical study. In *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99)*. 'Software Maintenance for Business Change' (Cat. No.99CB36360).
- Sapna, P. G. and Mohanty, H. (2009). Prioritization of scenarios based on uml activity diagrams. In 2009 First International Conference on Computational Intelligence, Communication Systems and Networks, pages 271–276.
- Siddik, M. S. and Sakib, K. (2014). RDCC: An effective test case prioritization framework using software requirements, design and source code collaboration. In 2014 17th International Conference on Computer and Information Technology (ICCIT).
- Singh, Y., Kaur, A., Suri, B., and Singhal, S. (2012). Systematic literature review on regression test prioritization techniques. *Informatica (Ljubl.)*, 36(4).
- Solanki, K., Singh, Y., and Dalal, S. (2015). Test case prioritization: An approach based on modified ant colony optimization (m-ACO). In 2015 International Conference on Computer, Communication and Control (IC4).
- Sommerville, I. (2011). *Software Engineering*. Pearson, Upper Saddle River, NJ, 9<sup>th</sup> edition.
- Srikanth, H., Cashman, M., and Cohen, M. B. (2016a). Test case prioritization of build acceptance tests for an enterprise cloud application: An industrial case study. *J. Syst. Softw.*, 119:122–135.
- Srikanth, H., Hettiarachchi, C., and Do, H. (2016b). Requirements based test prioritization using risk factors: An industrial study. *Inf. Softw. Technol.*, 69:71–83.
- Srikanth, H., Williams, L., and Osborne, J. (2005). System test case prioritization of new and regression test cases. In 2005 International Symposium on Empirical Software Engineering, 2005.
- Srivastva, P. R., Kumar, K., and Raghurama, G. (2008). Test case prioritization based on requirements and risk factors. *Softw. Eng. Notes*, 33(4):1–5.
- Su, W., Li, Z., Wang, Z., and Yang, D. (2020). A meta-heuristic test case prioritization method based on hybrid model. In 2020 International Conference on Computer Engineering and Application (ICCEA).
- Thomas, S. W., Hemmati, H., Hassan, A. E., and Blostein, D. (2014). Static test case prioritization using topic models. *Empir. Softw. Eng.*, 19(1):182–212.
- Tonella, P., Avesani, P., and Susi, A. (2006). Using the case-based ranking methodology for test case prioritization. In 2006 22nd IEEE International Conference on Software Maintenance.
- Vedpal, Chauhan, N., and Kumar, H. (2014). A hierarchical test case prioritization technique for object oriented software. In 2014 International Conference on Contemporary Computing and Informatics (IC3I).
- Wang, S., Buchmann, D., Ali, S., Gotlieb, A., Pradhan, D., and Liaaen, M. (2014). Multi-objective test prioritization in software product line testing: an industrial case study. *Proceedings of the 18th International Software Product Line Conference*, 1:32–41.
- Wang, X., Jiang, X., and Shi, H. (2015a). Prioritization of test scenarios using hybrid genetic algorithm based on UML activity diagram. In 2015 6th IEEE International Conference on Software Engineering and Service Science (IC-SESS).
- Wang, X. and Zeng, H. (2016). History-based dynamic test case prioritization for requirement properties in regression testing. In 2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED), pages 41–47.
- Wang, Y., Zhao, X., and Ding, X. (2015b). An effective test case prioritization method based on fault severity. In 2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS).
- Wu, H., Nie, C., and Kuo, F.-C. (2014). Test suite prioritization by switching cost. In 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops.
- Yadav, D. K. and Dutta, S. (2016). Test case prioritization technique based on early fault detection using fuzzy logic. In 2016 3<sup>rd</sup> International Conference on Computing for Sustainable Global Development (INDIACom), pages 1033–1036.
- Yadav, D. K. and Dutta, S. (2020). Regression test case selection and prioritization for object oriented software. *Microsyst. Technol.*, 26(5):1463–1477.
- Yoo, S. and Harman, M. (2012). Regression testing minimization, selection and prioritization: a survey. *Softw. Test. Verif. Reliab.*, 22(2):67–120.
- Zhang, L., Hao, D., Zhang, L., Rothermel, G., and Mei, H. (2013). Bridging the gap between the total and additional test-case prioritization strategies. In 2013 35th International Conference on Software Engineering (ICSE).
- Zhou, Z. Q. (2010). Using coverage information to guide test case selection in adaptive random testing. In 2010 IEEE

34th Annual Computer Software and Applications Conference Workshops. 29(4):366–427.

Zhu, H., Hall, P. A. V., and May, J. H. R. (1997). Software unit test coverage and adequacy. *ACM Comput. Surv.*,



## Appendix A

This appendix contains Table 2.

**Table 2.** Selected Studies

Article	Year	Citations	Relevance
Alazzam and Nahar (2018)	2018	6	1
Alspaugh et al. (2007)	2007	57	3,35
Arafeen and Do (2013)	2013	195	17,73
Biswas et al. (2022)	2022	1	0,5
Bryce et al. (2011)	2011	69	5,31
Carlson et al. (2011)	2011	159	12,23
Dobunch et al. (2014)	2014	12	1,2
Eghbali and Tahvildari (2016)	2016	65	8,13
Elbaum et al. (2001)	2001	520	22,61
Elbaum et al. (2000)	2000	588	24,5
Fang et al. (2014)	2014	105	10,5
Fazlalizadeh et al. (2009)	2009	33	2,2
Ganjkhani and Afsharchi (2019)	2019	1	0,2
Gokilavani and Bharathi (2021)	2021	21	7
Hao et al. (2014)	2014	132	13,2
Hettiarachchi et al. (2014)	2014	37	3,7
Hettiarachchi et al. (2016)	2016	101	12,63
Huang et al. (2014)	2014	19	1,9
Huang et al. (2020)	2020	31	7,75
Huang et al. (2012)	2012	153	12,75
Jahan et al. (2020)	2020	25	6,25
Jiang et al. (2009)	2009	312	20,8
Jones and Harrold (2003)	2003	582	27,71
Kavitha et al. (2010)	2010	42	3
Khalid and Qamar (2019)	2019	7	1,4
Khalilian et al. (2012)	2012	73	6,08
Kim et al. (2017)	2017	16	2,29
Kim and Porter (2003)	2002	610	27,73
Korel et al. (2005)	2005	158	8,32
Korel et al. (2008)	2008	110	6,88
Krishnamoorthi and Sahaaya Arul Mary (2009)	2009	130	8,67
Kwon et al. (2014)	2014	24	2,4
Leon and Podgurski (2003)	2003	244	11,62
Lin et al. (2014)	2014	50	5
Liu et al. (2014)	2014	8	0,8
Ma et al. (2016)	2016	23	2,88
Magalhães et al. (2021)	2021	4	1,33
Mahali and Mohapatra (2018)	2018	18	3
Malz and Göhner (2011)	2011	19	1,46
Marijan et al. (2013)	2013	198	18
Masri and El-Ghali (2009)	2009	18	1,2
Panda et al. (2019)	2019	6	1,2
Panda and Mohapatra (2021)	2021	4	1,33
Panigrahi and Mall (2014)	2014	34	3,4
Parashar et al. (2012)	2012	5	0,42
Park et al. (2008)	2008	110	6,88
Romano et al. (2018)	2018	27	4,5
Rothermel et al. (1999)	1999	891	35,64
Sapna and Mohanty (2009)	2009	38	2,53
Siddik and Sakib (2014)	2014	18	1,8
Solanki et al. (2015)	2015	15	1,67
Srikanth et al. (2005)	2005	277	14,58
Srikanth et al. (2016a)	2016	45	5,63

Continuation of Table 2			
Article	Year	Citations	Relevance
Srikanth et al. (2016b)	2016	69	8,63
Srivastva et al. (2008)	2008	42	2,63
Su et al. (2020)	2020	9	2,25
Thomas et al. (2014)	2014	189	18,9
Tonella et al. (2006)	2006	128	7,11
Vedpal et al. (2014)	2014	11	1,1
Wang et al. (2014)	2014	97	9,7
Wang and Zeng (2016)	2016	47	5,88
Wang et al. (2015a)	2015	16	1,78
Wang et al. (2015b)	2015	20	2,22
Wu et al. (2014)	2014	18	1,8
Yadav and Dutta (2016)	2016	24	3
Yadav and Dutta (2020)	2020	19	4,75
Zhang et al. (2013)	2013	208	18,91
Zhou (2010)	2010	85	6,07
End of Table			