



Test case prioritization based on human knowledge

Ícaro Prado Fernandes

(Federal University of São Paulo, São José dos Campos, São Paulo
 <https://orcid.org/0009-0005-7896-6643>, icaro.fernandes@unifesp.br)

Luiz Eduardo Galvão Martins

(Federal University of São Paulo, São José dos Campos, São Paulo
 <https://orcid.org/0000-0002-7266-5840>, legmartins@unifesp.br)

Abstract: Building quality software, that is, suitable for use and meeting user needs, is one of the biggest challenges in the software industry. Although it is possible to guarantee the proper functioning of software through testing activities, such activities are exhaustive in nature, as it is impossible to test all inputs of a minimally complex program. This work proposes a method to prioritize test cases based on human knowledge using a combination of factors evaluated in an assessment answered by 29 software industry professionals and 5 academics. The assessment confirmed that the proposed factors are relevant. Finally, a practical example that prioritizes test cases for a banking application was carried out and it was observed that the proposed method works properly.

Keywords: Test case prioritization, Software testing, Software quality

Categories: D.2.1, D.2.4, D.2.5

DOI: 10.3897/jucs.127870

1 Introduction

There are several challenges in the software industry, as they are increasingly present in people's life and in the most diverse industry segments. One of the great challenges faced by software engineering caused by the wide use of software in society is adapting to changes in requirements, costs, deadlines without compromising quality [Pressman and Maxim 2015, Sommerville 2011].

Producing software with quality is not a simple task, it is necessary to respect established deadlines and costs using the best practices, processes and tools, and finally, ensure that the software behaves properly and that is suitable for customer's use.

An activity that helps in evaluating and maintaining quality is software testing, according to [Myers et al. 2011] this is an activity that consists in executing a certain program with the purpose of revealing the presence of faults. According to [Chopra 2018] there is a terminology to cover incorrect program behavior from the time it was made until its revelation:

- **Error:** People make errors. When people make mistakes while coding, these mistakes are called errors or bugs.
- **Fault or Defect:** A missing DOI or incorrect statement in a program resulting from an error is a fault. So, a fault is the representation of an error.
- **Failure:** A failure occurs when a fault executes. The manifested inability of the program to perform a required function within specified limits is known as a failure.

The test case is an artifact responsible for describing and validating the behavior that the program must have within a specified limit. Its execution finds the failure, reveals the fault and allows the error to be corrected. That is why constant testing has a positive impact on the quality of the program, improving its behavior until it is suitable for use.

On the other hand, references indicate that the cost of testing activities is considered high in relation to the total cost of development process [Brooks 1995, Kaner et al. 1999, Myers et al. 2011, Pressman and Maxim 2015]. Therefore, it is essential to define and prioritize which tests should be carried out and the use of methods to perform test case prioritization (TCP) appears to optimize test execution [Rothermel et al. 1999].

There are several authors in the area of software testing, such as [Rothermel et al. 1999, Elbaum et al. 2000, Elbaum et al. 2001] who emphasize the importance of TCP, they explain that the TCP practice organizes the execution of test cases based on their importance, reduces the size of test suite and mitigates redundant test execution.

Studies have emerged highlighting the importance of considering people's opinions and knowledge during the testing phase, more specifically when carrying out TCP [Tonella et al. 2006, Malz and Göhner 2011, Srikanth et al. 2005, Krishnamoorthi and Mary 2009, Srikanth et al. 2016, Srivastva et al. 2008]. The methods to TCP proposed in these studies revealed that the use of human knowledge increases the effectiveness of test execution, because severe faults usually are revealed earlier.

There are several methods to prioritize test cases, however, it was observed in the literature that although some methods to TCP consider people's opinions, most of them use this factor by combining it with others, which makes the methods hybrid. These studies will be discussed in Section 2.

This paper proposes a method to TCP that takes advantage of the gap in the literature of methods purely based on human knowledge, once it considers only people's knowledge to perform TCP. The one proposed in this paper is built with the intention of performing TCP not only considering the knowledge of people who perform the tests, but also the customer and those who specify and code the software.

The rest of this paper is structured as follows: Section 2 is about background and related studies. Section 3 describes the research method. Section 4 presents the artifact design process. Finally, Section 5 presents results and discussions.

2 Background studies

The studies by [Singh et al. 2012, Catal and Mishra 2013, Khatibsyarbini et al. 2018] show that there are several methods to TCP in the literature. In order to simplify the analysis and understanding of these many methods, studies such as those carried out by [Yoo and Harman 2012, Catal and Mishra 2013] propose grouping TCP methods based on their characteristics, that is, the way they prioritize test cases. This grouping is usually done as follows:

- **Source code coverage:** methods for TCP based on code coverage prioritize test cases so that the source code is 100% covered by them. In other words, methods in this group analyze methods, functions, classes, objects and the code structure to define which tests should be executed.
- **Execution history:** In this group, methods usually use information from the test case execution history and the history of faults revealed by them to understand and determine how important its execution is.

- **Human knowledge:** methods for TCP based on human knowledge prioritize test cases by considering information and opinion from people in charge of carrying out the tests and other people involved, such as software specifiers and coders.
- **Execution cost:** methods for TCP based on cost have a basic premise: they consider that test cases do not have the same execution cost. In other words, these methods assume that the test cases are different from each other, which is why they have their own characteristics. Therefore, the execution of each of them requires a specific effort.
- **Requirements:** In this group, methods to TCP always use functional and non-functional requirements as a source of information to analyze and define test prioritization.
- **Distribution:** Distribution-based TCP methods use the similar characteristics of test cases to group and then prioritize them. In other words, test cases are grouped based on their similarities.
- **Diagrams:** methods for TCP based on diagrams use information from Unified Modeling Language (UML) models to define test prioritization. These methods use visual representations or models to perform TCP, what is interesting considering that diagrams are widely used in the software industry to represent requirements, workflows and systems architecture.

A method belonging to a group does not prohibit it from using characteristics or factors from other groups to make TCP viable, as is the case with the methods explained below, which use human knowledge in prioritization, but combine it with other factors.

The study developed by [Srikanth et al. 2005] proposes a requirement-based method to TCP, but considering human knowledge. In this method, the customer provides the priority of the requirements and the developer provides an insight into how complex the implementation of this requirement is. To define the prioritization factor of a test case, these factors based on human knowledge are combined with two more factors, the requirement volatility, that is, the number of times it changes, and the fault proneness in requirement.

There are also methods that combine human knowledge with machine learning algorithms to enable TCP, a method of this type is the one proposed by [Tonella et al. 2006]. In this method, a test suite is defined and prioritized based on the human knowledge, this test suite is successively processed by a machine learning algorithm, this algorithm compares the test case with its peers, considering the complexity of the requirement covered by the test case and the history of failures that are covered by the test execution. In this method, human knowledge is also used when the machine learning algorithm cannot decide whether one test case is more important than another, in this scenario, human intervention occurs for decision making.

The method proposed by [Malz and Göhner 2011] performs TCP by combining human knowledge and fuzzy logic, this method defines the importance of the test case based on the tester's and developer's opinion about its execution complexity, failures history, history and quantity of changes, size, complexity and time to implement. These factors are subjected to a fuzzification process that will generate a test importance factor, which determines its execution priority.

Another method that uses human knowledge as one of the sources for TCP is the one proposed by [Khalid and Qamar 2019]. To keep the customer's preference, in this method,

the customer assigns a weight to business requirements, then clusters are created using K-means in order to group them. After that, for each test case present in the requirement, the cost and execution time are calculated using function points and complexity measures, in each cluster. Based on that, clusters are further classified into high, medium or low priority using K-Medoids algorithm.

The methods proposed in the studies by [Srikanth et al. 2005, Tonella et al. 2006, Malz and Göhner 2011, Khalid and Qamar 2019] confirm that using human knowledge contributes positively to the TCP process, however, all of these methods combine it with other factors. Aiming to fill a gap found in the literature, this paper proposes a method that uses only human knowledge to make TCP viable, therefore, it differs from the existing methods.

3 Research method

Based on the objective outlined for this paper, it was decided that the research method adopted would be Design Science Research (DSR), as shown in Figure 1. The DSR was chosen because it is suitable for researches that aims to build artifacts [Aken 2004]. The artifact proposed in this study is a method for TCP purely based on human knowledge.

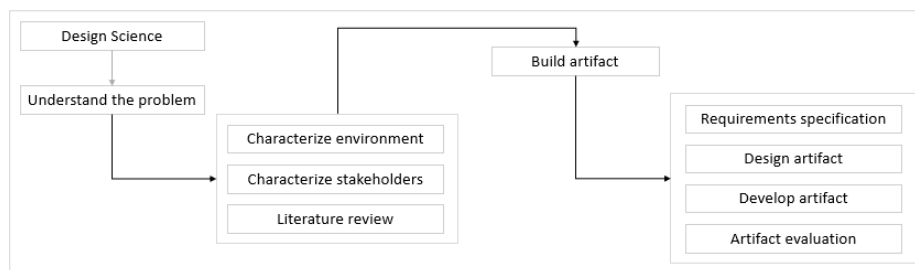


Figure 1: Design Science Research. Adapted from [Dresch et al. 2015]

Furthermore, [Simon 1996, Aken 2004] explain that applying the DSR increases the relevance of the research, since its results are prescriptive and assist in solutions to real-world problems. It also influences the rigor of the research, as it becomes reliable to the point of contributing as a knowledge base in specific areas.

3.1 Prioritization Factors

The algorithm underlying the proposed method prioritizes testing based on six factors: (1) importance of the process that is validated by the test case, (2) test case execution complexity, (3) priority for fixing failures revealed by the test case, (4) criticality of failures revealed by the test case, (5) customer perception, and (6) time required to execute the test case. We discuss below the six factors and why they were chosen for use in the proposed method.

- **Process importance (PI):** it is a measure to map the importance of the process or feature validated by the test case. A value is assigned to each test case ranging from

0 to 2, where 0 means that the process is critical and widely used, 1 means that the process is important and 2 means that the process is secondary or not used a lot. It is important to consider that in software, there are functionalities that can be classified as highly, little or rarely used, therefore, a test case that validates a highly used functionality should be executed before the others.

- **Execution complexity (EC):** this measure is based on how difficult the test case is to execute considering the work for mapping and executing its dependencies and its preconditions, the test case extension in terms of steps and the number of details validated by it. A value is assigned to each test case ranging from 0 to 3, where 0 means that there are few dependencies and preconditions and the test execution is simple, 1 means that the test execution is simple but its dependencies and preconditions are not, 2 means that the test execution is not simple but its dependencies and preconditions are, and 3 means that both test execution and its dependencies and preconditions are not simple.
- **Possible fault priority (FP):** this measure allows the development team to reflect and identify which test cases could reveal faults that must be corrected urgently. Given that each test case describes a specific behavior to be tested, the priority for fixing a failure associated with a test case usually is different from the others. A value is assigned to each test case ranging from 0 to 2, where 0 means the failure must be fixed immediately, 1 means the failure must be fixed in the current sprint, and 2 means the failure can be fixed in a future sprint.
- **Possible fault criticality (FC):** this measure allows the development team to reflect and identify which test cases could reveal faults that interrupt the use of the application or some of its functionalities. A value is assigned to each test case ranging from 0 to 2, where 0 means the failure could break or interrupt the operation, 1 means the failure harms an important part of the operation, but can be overcome, and 2 means the failure harms a non-critical part of the operation, which is rarely used and can be overcome.
- **Customer perception (CP):** the purpose of this measure is to map the proximity that the test case has to the customer's use, because prioritizing tests that represent the customer's use can increase customer perception of value and satisfaction. A value is assigned to each test case ranging from 0 to 2 by the customer or their representative, where 0 means the test case represents an activity that will be frequently performed by the customer, 1 means the test case represents an activity that will be sporadically performed by the customer, and 2 means the test case represents an activity that will be rarely performed by the customer.
- **Execution time (ET):** is a subjective measure of how difficult the development team judges that executing the test case will be, considering its preconditions and postconditions, such as evidence of its execution. Here, a value in minutes is assigned to each test case.

3.2 Prioritization Process

The process to perform TCP using the proposed method is presented in Figure 2. There are five stakeholders in the process and an algorithm that prioritizes the test cases. The roles of each stakeholder are defined below:

The customer is responsible for providing their perception about each test case; therefore, the Customer perception (CP) factor is fulfilled by them.

The requirements analyst or product owner is responsible for providing insight into how important the process or flow validated by the test case is for the product, therefore, it is this person's responsibility to fulfill the Process importance (PI) factor.

The person in charge of executing the test is responsible for informing how complex its execution is and how long it will take to carry it out, that is, this person provides the factors Execution complexity (EC) and Execution time (ET).

The development team, which is made up of developers, testers and software engineers, is responsible for informing the other factors. They provide information about what is the priority (FP) and criticality (FC) of a possible fault in the test case. In cases where the requirements analyst is within the team, this person can also contribute indicating values for these factors.

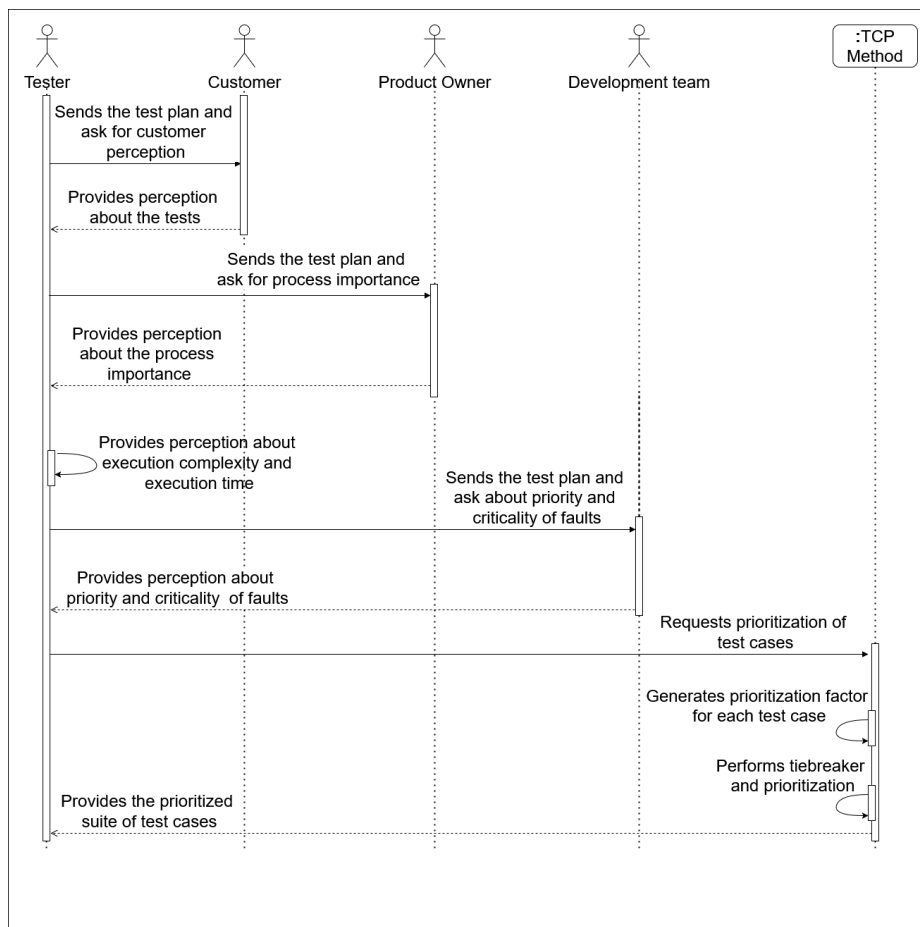


Figure 2: Prioritization Process

The way in which communication occurs between actors is not fixed, this is particular to each application domain. At this point, it is essential that the six prioritization factors used by the proposed method are filled in for each test case with values within the range allowed for each of them.

The tester's interaction with the TCP method occurs through an Application Programming Interface (API), an endpoint was created to allow the tester to send the list of test cases to be prioritized by the method, after the tester has collected data from the other actors about the six prioritization factors for each test case.

3.3 Prioritization algorithm

Factor values are assigned by the actors for each test case, which makes them capable of prioritization by the proposed method. The prioritization factor for each test case is based on the distance it has to the test case with the highest possible priority. The further away the test case is from the one with the highest priority, the lower its execution priority will be.

For the comparison model between test cases work properly, it is necessary to know which test case has the highest possible priority, as in the proposed method the prioritization factors were mapped and a range of values for each of them was established, the test case with the highest possible priority is the one that receives zero for five of the six prioritization factors, this is demonstrated in Table 1.

Factor	Value
processImportance	0
executionComplexity	0
possibleFaultPriority	0
possibleFaultCriticality	0
customerPerception	0

Table 1: Test case with the highest possible priority

Five of the six factors will be used to calculate the prioritization factor, this is because the execution time factor is a tiebreaker criterion. When two test cases have the same prioritization factor, the one with the shortest execution time will be executed first.

The proposed test case comparison model uses dissimilarity measures to find out how different two test cases are on a scale that ranges from [0, infinity], the further away from zero the distance between them, the more different they are.

An effective and widely used measure of dissimilarity between objects in the most diverse markets and segments is the Euclidean distance [Gauch 1982, Alfakih 2018]. [Merchant and Castleman 2022] reinforce that Euclidean distance is by far the most commonly used distance measure. The calculation of the Euclidean distance is carried out using a mathematical formula, as shown in Figure 3.

The two objects in comparison must have the attributes (i,k) and (j,l) respectively. The results of the subtractions between (i,k) and (j,l) are squared and then added, finally, the square root is calculated on the result of the sum, the result obtained in this procedure is the Euclidean distance between the objects compared.

$$D_e = \sqrt{(i - k)^2 + (j - l)^2}$$

Figure 3: Adapted from [Merchant and Castleman 2022]

According to [Brower and Zar 1977], the smaller the value of the Euclidean distance between the compared objects, the closer they appear in terms of quantitative parameters, that is, the smaller the Euclidean distance between two objects, the closer they are.

Figure 4 presents the mathematical step by step to map how far a test case is from the one with the highest possible priority using Euclidean distance. The test case with the highest possible priority is represented by the letter h, while the test case in comparison is represented by the letter x.

$$\begin{aligned}
 a &= \text{Weight} * (h.\text{processImportance} - x.\text{processImportance})^2 \\
 b &= \text{Weight} * (h.\text{executionComplexity} - x.\text{executionComplexity})^2 \\
 c &= \text{Weight} * (h.\text{possibleFaultPriority} - x.\text{possibleFaultPriority})^2 \\
 d &= \text{Weight} * (h.\text{possibleFaultCriticality} - x.\text{possibleFaultCriticality})^2 \\
 e &= \text{Weight} * (h.\text{customerPerception} - x.\text{customerPerception})^2 \\
 f &= a + b + c + d + e \\
 \text{prioritizationFactor} &= \sqrt{f}
 \end{aligned}$$

Figure 4: Formula to calculate the prioritization factor

Note that a weight variable was added to the formula, the factors for prioritizing test cases were divided into two groups, primary and secondary. Primary factors have a weight represented by the value 1.0, secondary factors have a weight represented by the value 0.5. Of the five factors used in the formula, only the execution complexity is in the secondary group. Imagine that a test case is capable of revealing a critical fault in a critical functionality, or is extremely important in the customer's view, naturally, it will have a higher priority for execution, even though it is complex to execute.

3.4 Assessment of prioritization factors

The central element for developing a method to TCP is the definition of how it will be done, that is, the set of factors that will be used to say that the execution of one test case is more important than another.

Based on this, before codifying the method, an assessment about the adoption of the six prioritization factors was prepared and made available electronically via Google Forms, it was answered anonymously by 34 participants where 29 of them were software industry professionals and 5 of them were academics with knowledge in software testing.

The assessment occurred before the codifying of the method, because if any of the six prioritization factors did not make sense in the respondents opinion, it would be possible to adapt or remove the factor and propose the use of another.

The assessment's questions were not mandatory and were created based on the Likert scale. Created by Rensis Likert in 1932, it is a widely known measurement scale used around the world to measure preferences and perspectives [Júnior and Costa 2014], in addition, it is easy to use and analyze the degree of agreement.

Regarding the assessment, the first question maps the participant's profile, the options available for response were: "academic", "software industry professional" and "I do not wish to inform". To answer the remain questions, that is, from the second to the tenth question, the participant had available the answer options presented in Table 2.

Possible answers
A) Not important
B) It is a little important
C) Neutral
D) It is important
E) It is very important
F) I do not wish to inform

Table 2: Possible answers

The second question of the assessment is presented in Table 3. The purpose of this question is to understand how important, in the participant's opinion, it is to define a scope of tests and execute it in a prioritized manner.

Question 2
For you, how important is it to define a testing scope and execute the test cases within this scope in a prioritized manner?

Table 3: Second question

Table 4 presents the third question, the purpose of this question is to understand whether the participant agrees that the time to execute test cases is limited, so it is necessary to map and define a limited scope of them. This question is related to the prioritization factor: execution time (ET).

Question 3
Consider the statement below and choose an alternative that matches your opinion: "The time for testing execution is limited, therefore it is important to define a finite scope of test cases that is capable of revealing the greatest number of faults and ensuring that the system is suitable for use, that is, that it behaves in the correct way."

Table 4: Third question

Table 5 presents the fourth question, the purpose of this question is to understand whether in the participant's opinion, executing test cases in a prioritized manner is more efficient than executing them randomly.

Question 4
Consider the statement below and choose an alternative that matches your opinion: "Executing test cases in a prioritized manner is more efficient than executing them randomly."

Table 5: Fourth question

Table 6 presents the fifth question, this question is related to the prioritization factor: importance of the process validated by the test case for the product (PI). From the response, it is possible to understand the participant's degree of agreement with the use of this prioritization factor.

Question 5
Consider the statement below and choose an alternative that matches your opinion: "In a system that has several functionalities, some functionalities are more important than others."

Table 6: Fifth question

Table 7 presents the sixth question, this question is related to the following prioritization factor: test case execution complexity (EC). From the response, it is possible to understand the participant's degree of agreement with the use of this prioritization factor.

Question 6
Consider the statement below and choose an alternative that matches your opinion: "The test case is an artifact that has a set of actions that will be executed to validate whether a certain functionality behaves properly. Factors such as: generation of the test input and output, dependencies, preconditions and the amount of detail that must to be validated make one test case more complex to execute than another. Therefore, the greater the complexity to execute the test case, the longer the time required to execute it."

Table 7: Sixth question

Table 8 presents the seventh question, this question is related to the following prioritization factor: priority of fixing a fault revealed by the test case (FP). From the response,

it is possible to understand the participant's degree of agreement with the use of this prioritization factor.

Question 7
Consider the statement below and choose an alternative that matches your opinion: "If a test case is capable of revealing a fault that needs to be fixed immediately, it is important to ensure that this test case will be executed."

Table 8: Seventh question

Table 9 presents the eighth question, this question is related to the following prioritization factor: criticality of the existence of a possible fault revealed by the test case (FC). From the response, it is possible to understand the participant's degree of agreement with the use of this prioritization factor.

Question 8
Consider the statement below and choose an alternative that matches your opinion: "If a test case is capable of revealing a fault that interrupts the functioning of a critical part of the system, it is important to ensure that this test case will be executed."

Table 9: Eighth question

Table 10 presents the ninth question, this question is related to the following prioritization factor: customer perception (CP). From the response, it is possible to understand the participant's degree of agreement with the use of this prioritization factor.

Question 9
Consider the statement below and choose an alternative that matches your opinion: "It is important that the test cases reflect the customer's use of the product, therefore, it is essential to understand how the customer uses the product to carry out tests that are close to reality."

Table 10: Ninth question

Finally, Table 11 presents the tenth and final question. This question seeks to understand the degree of agreement among participants about the importance of using human knowledge to prioritize test cases.

Question 10
Consider the statement below and choose an alternative that matches your opinion: “It is important to consider the knowledge that the people who are part of the team have to build and prioritize the scope of test cases that will be executed.”

Table 11: Tenth question

After applying the assessment, it was possible to collect the opinion of 29 software industry professionals and 5 academics who provided support and evidence on the use of the factors proposed by the method to make TCP viable and that human knowledge is important in this process. The quantitative result of the assessment will be presented in the results section.

4 Artifact design

To build the artifact, which is a method for prioritizing test cases based on human knowledge using the six proposed prioritization factors, requirements were extracted and prioritized from the diagram presented in Figure 2 in Subsection 3.2, which explained the prioritization process, these requirements are presented in Table 12.

Priority	Requirement
1	Receiving the test case prioritization request
2	Calculate the prioritization factor using Euclidean distance
3	Prioritize test cases based on the calculated prioritization factor
4	Tiebreaker of test cases with the same prioritization factor based on their execution time

Table 12: Requirements to be developed

The first requirement to be implemented is the endpoint that will receive from the tester a list of test cases that will be prioritized. Next, an algorithm is implemented to calculate the prioritization factor based on the Euclidean distance between each test case received in the request and the test case with the highest possible priority.

After calculating the prioritization factor, the test cases are ordered based on it, in this way, they are prioritized from the test case closest to the one with the highest possible priority to the one further away from it.

The last requirement to be implemented is the tiebreaker based on execution time, that is, if two test cases have the same prioritization factor, the one with the shorter execution time will be prioritized over the other.

According to [Sommerville 2011], the architectural design of a system is a creative process, where development team designs a systemic structure to meet the specified requirements. To design the artifact proposed in this study the guidelines of Clean Architecture were adopted, a software architecture standard created by Robert Cecil

Martin in 2012. Normally, Clean Architecture is visually illustrated as shown in Figure 5, where it is possible to visualize the layers and the communication between them.

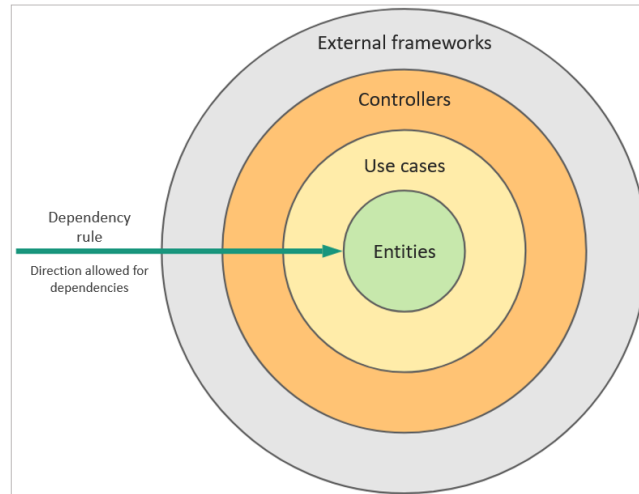


Figure 5: Clean architecture

At the center of the model are the entities, these are known and used by all layers. [Martin 2012] describes entities as the application's business objects that encapsulate general, high-level business rules.

In the next layer are the use cases, in this layer the functionalities and business rules are implemented according to the specified requirements, this layer could also be called business logic.

The controller layer contains the classes responsible for enabling communication between the outermost layer of the architecture and the business rules. [Valente 2020] explains that in applications that use API to enable communication with the API consumer, the controller classes are responsible for implementing the API endpoints. These classes must receive requests and forward them to the corresponding use cases. And also, do the opposite: receive the results returned by the use cases, convert them into JSON format and provide them to the API consumer.

In the external frameworks layer are the classes that enable tooling for persistence in databases and construction of user interfaces. For [Martin 2012], these structures and tools are kept in the outermost layer because this is where they can cause the least impact in the event of a change or update.

This layered isolation protects the architecture core, where the business rules and entities are. As explained previously, the tester's interaction with the TCP method occurs through an API, the API contract that will be used to enable TCP by the proposed method, the request and response structure, is presented by Figure 6.

```
Request design
{
  "externalId": "string",
  "testCases": [
    {
      "testCaseId": "string",
      "processImportance": "int",
      "executionComplexity": "int",
      "possibleFaultPriority": "int",
      "possibleFaultCriticity": "int",
      "approximateExecutionTime": "int",
      "customerPerception": "int"
    }
  ]
}

Response design
{
  "externalId": "string",
  "testCases": [
    {
      "testCaseId": "string",
      "processImportance": "int",
      "executionComplexity": "int",
      "possibleFaultPriority": "int",
      "possibleFaultCriticity": "int",
      "approximateExecutionTime": "int",
      "customerPerception": "int",
      "prioritizationFactor": "double"
    }
  ]
}
```

Figure 6: API contract

Note that in the request, the tester sends an identifier and a list of test cases, each test case in the list has its identifier and the six prioritization factors that the method uses. Then, the prioritization algorithm will calculate the prioritization factor for each test case and find out how far it is from the test case with the highest possible priority.

After that, the method will order the test cases by how close they are to the case with the highest possible priority, in other words, the closer the test case's prioritization factor is to zero, the higher its execution priority will be.

The next step is the treatment of test cases that have the same prioritization factor, the tiebreaker occurs based on the execution time factor, the test case that has the shortest execution time will gain priority compared to the one that has the same prioritization factor but takes longer to execute.

Finally, the API returns to the tester the test cases sent by him, ordered by the prioritization factor calculated by the method. To be transparent with the tester, the API response contains the prioritization factor calculated by the proposed method for each test case.

In order to exemplify how the proposed method works and demonstrate that its operation through the proposed API works properly, a practical example was made using real software, this will be presented in the results section.

5 Results and conclusions

This section is structured as follow: first, the assessment results are presented and discussed. Afterwards, a practical example of using the method through the created API is presented and discussed.

5.1 Assessment results

The application of the assessment had the collaboration of 34 participants, of which 29 were software industry professionals and 5 academics, all participants answered all questions, that is, there were no abstentions.

Table 13 aims to directly relate the questions present in the questionnaire with its object of evaluation, presenting the participants' degree of agreement on the aspect covered by it. The agreement presented in the table is the perceptual of answers obtained with total and partial agreement for each question.

Question	Assessment objective	Agreement
Q02	Understand the importance of prioritizing test cases	97%
Q03	Degree of agreement with the use of the factor: execution time (ET)	91,2%
Q04	Running tests in a prioritized manner is more efficient than running them randomly	94,1%
Q05	Degree of agreement with the use of the factor: importance of the process validated by the test case for the product (PI)	97,1%
Q06	Degree of agreement with the use of the factor: test case execution complexity (EC)	94,1%
Q07	Degree of agreement with the use of the factor: priority of fixing a possible fault revealed by the test case (FP)	100%
Q08	Degree of agreement with the use of the factor: possible fault criticality (FC)	97,1%
Q09	Degree of agreement with the use of the factor: customer perception (CP)	100%
Q10	Degree of agreement among participants about the importance of using human knowledge to prioritize test cases	94,1%

Table 13: Assessment result

The application of the assessment was of great importance for the logical evaluation of the constructed method, aspects that support the construction of the artifact were evaluated, for example: the questions (Q03, Q05, Q06, Q07, Q08 and Q09) are related to the proposed prioritization factors; the relevance of the topic test case prioritization is addressed by the questions (Q02, Q04) and the use of human knowledge as a basis for prioritizing tests is addressed by the question (Q10).

The result of the assessment showed, based on the degree of agreement obtained, that:

- It makes sense to use the six proposed factors for test case prioritization;
- Test case prioritization is important and run test cases in a prioritized manner is, in general, more effective than running them randomly;
- It is important to consider human knowledge to prioritize test cases.

5.2 Practical example

A practical example of using the proposed method was carried out, which consisted of a functional validation based on functionalities contained in a real system, more specifically a Brazilian mobile banking system.

This type of application was chosen because it is observed that their use, according to [Febraban 2023], is constantly growing and is already the main form of interaction between banks and their customers in Brazil.

The last major innovation widely available in banking applications was the Brazilian Instant Payment (PIX), a payment method created by the Central Bank of Brazil in 2020. According to [Bacen 2024] using PIX, resources are transferred between accounts in a few seconds, at any time or day, the transfer is carried out using a key, a unique code that represents the account.

Before PIX, resource transfers were made via Available Electronic Transfer (TED), to transfer the money it was necessary to provide information about the financial institution and the receiver. A limitation of this model is the transfer window once it only occurs in business hours and business days.

The test suite to evaluate how the method will work was raised based on the functionalities of mobile banking applications widely used for transferring resources, according to [Febraban 2023] and is presented in Table 14.

Test Case	Title
CT01	Perform PIX by QR Code key
CT02	Perform PIX by phone number key
CT03	Perform PIX by CPF key
CT04	Perform PIX by CNPJ key
CT05	Perform PIX by email key
CT06	Schedule a PIX
CT07	Perform external transfer (TED)
CT08	Perform internal transfer (TED)
CT09	Make payment of bill
CT10	Check account balance
CT11	Consult daily statement
CT12	Consult monthly statement

Table 14: Suite of test cases

As this is a practical example to confirm the adequate functioning of the method, the values for the six prioritization factors of each test case were obtained based on the volume of operations performed in accordance with [Febraban 2023] and are presented

in Table 15. In real situations in the software industry, prioritization factors will be informed by the actors involved in the prioritization process.

Test Case	PI	EC	FP	FC	CP	ET
CT01	0	1	0	0	0	20
CT02	0	0	0	1	1	15
CT03	0	0	0	0	0	12
CT04	0	0	0	0	0	15
CT05	0	0	0	1	1	15
CT06	0	2	0	1	1	25
CT07	0	0	1	1	0	20
CT08	1	0	1	1	1	15
CT09	1	1	1	1	1	12
CT10	0	0	1	0	0	5
CT11	0	0	2	1	1	8
CT12	2	1	2	1	2	30

Table 15: Filling the prioritization factors for each test case in the suite

To make the viewing simpler, Table 15 abbreviates the name of the prioritization factors, where the Process Importance is PI; Execution Complexity is EC; Possible Fault Priority is FP; Possible Fault Criticality is FC; Customer Perception is CP and Execution time in minutes is ET.

A request was sent to the API that performs the prioritization of test cases by passing the values from Table 15. The API response was an ordered list of test cases based on their prioritization factor, the prioritization result is displayed in Table 16.

Test Case	Title	Calculated prioritization factor
CT03	Perform PIX by CPF key	0.0
CT04	Perform PIX by CNPJ key	0.0
CT01	Perform PIX by QR Code key	0.7071067811865476
CT10	Check account balance	1.0
CT02	Perform PIX by phone number key	1.4142135623730951
CT05	Perform PIX by email key	1.4142135623730951
CT07	Perform external transfer (TED)	1.4142135623730951
CT08	Perform internal transfer (TED)	2.0
CT06	Schedule a PIX	2.0
CT09	Make payment of bill	2.1213203435596424
CT11	Consult daily statement	2.449489742783178
CT12	Consult monthly statement	3.6742346141747673

Table 16: Prioritization result

The result of the practical example demonstrated that the method works and that the result of the prioritization requested by it was satisfactory, since the test cases directly related to the PIX functionality received high execution priority, together with the account balance check, an important function from the perspective of checking transactions carried out by PIX and other features of the application.

In addition, this practical example also proved the tiebreaker based on the execution time factor works properly, even though may not be sufficient in situations where the prioritization factor of the test cases are the same and their execution time are also the same, in these situations it would be important to think about and adopt some additional tiebreaker factor.

5.3 Conclusions

We have investigated the relevance of the information gathered from the user to create a method for test case prioritization based on human knowledge. The proposed method integrates the tester's knowledge with the one of the customer, the requirements analyst and the rest of the team based on a combination of six prioritization factors.

From a theoretical point of view, the method is appealing because unlike existing methods in the literature that combine people's perception with other factors, the method of this work is purely based on human knowledge and, because of this, it is widely applicable, in several testing contexts.

It was confirmed by 29 software industry professionals and 5 academics that using the six proposed factors to carry out prioritization makes sense.

Although still preliminary, the practical example results show that the proposed method works properly and that future work can be carried out with the aim of improving the tiebreaker mechanism, as the proposed one may not be enough in situations where there are many test cases to be prioritized with the same values for the prioritization factors.

Future work can be carried out by conducting a complete investigation of the applicability and cost-benefit of the proposed method, when compared to others, this involves application of the method in real-world scenarios, with the involvement of all actors: testers, developers, requirements analysts and customers.

References

- [Aken 2004] Aken, J. E. V. Management research based on the paradigm of the design sciences: the quest for field-tested and grounded technological rules. *Journal of management studies*, v. 41, n. 2, p. 219-246, 2004.
- [Alfakih 2018] Alfakih, A. Y. *Euclidean distance matrices and their applications in rigidity theory*. Springer International Publishing, 2018.
- [Bacen 2024] Bacen. Brazil Central Bank. Available at: <https://www.bcb.gov.br/estabilidade/financeira/pix>. Access in: 31 jan. 2024.
- [Brooks 1995] Brooks, F. P.; *The Mythical Man-Month (anniversary ed.)*. Addison-Wesley, 1995.
- [Brower and Zar 1977] Brower, J. E.; Zar, J. H. *Field & laboratory methods for general ecology*. 2.ed. Dubuque: Wm. C. Brown Publishers, 1977. 226p.
- [Catal and Mishra 2013] Catal, C.; Mishra, D. Test case prioritization: a systematic mapping study. *Software Quality Journal*, v. 21, n. 3, p. 445-478, 2013.

- [Chopra 2018] Chopra, R. Software testing: a self-teaching introduction. Mercury Learning and Information, 2018.
- [Dresch et al. 2015] Dresch, A.; Lacerda, D. P.; Júnior, J. A. V. A. Design Science Research: método de pesquisa para avanço da ciência e tecnologia. Bookman Editora, 2015.
- [Elbaum et al. 2000] Elbaum, S.; Malishevsky, A. G.; Rothermel, G. Prioritizing test cases for regression testing. In: Proceedings of the 2000 ACM SIGSOFT international symposium on Software testing and analysis. 2000. p. 102-112.
- [Elbaum et al. 2001] Elbaum, S.; Malishevsky, A.; Rothermel, G. Incorporating varying test costs and fault severities into test case prioritization. In: Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001. IEEE, 2001. p. 329-338.
- [Febraban 2023] Febraban. Pesquisa FEBRABAN de Tecnologia Bancária 2023. Available at: <https://portal.febraban.org.br/pagina/3106/48/pt-br/pesquisa>. Access in: 31 jan. 2024.
- [Gauch 1982] Gauch, H. G. Multivariate analysis in community ecology. Cambridge: Cambridge University Press, 1982. 298p.
- [Júnior and Costa 2014] Júnior, S. D. S.; Costa, F. J. Mensuração e escalas de verificação: uma análise comparativa das escalas de Likert e Phrase Completion. PMKT—Revista Brasileira de Pesquisas de Marketing, Opinião e Mídia, v. 15, n. 1-16, p. 61, 2014.
- [Kaner et al. 1999] Kaner, C.; Falk, J.; Nguyen, H. Q. Testing computer software. John Wiley & Sons, 1999.
- [Khalid and Qamar 2019] Khalid, Z.; Qamar, U. Weight and cluster based test case prioritization technique. In: 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON). IEEE, 2019. p. 1013-1022.
- [Khatibsyarbini et al. 2018] Khatibsyarbini, M.; Isa, M. A.; Jawawi, D. N.; Tumeng, R. Test case prioritization methods in regression testing: A systematic literature review. Information and Software Technology, v. 93, p. 74-93, 2018.
- [Krishnamoorthi and Mary 2009] Krishnamoorthi, R.; Mary, S. S. A. Factor oriented requirement coverage based system test case prioritization of new and regression test cases. Information and Software Technology, v. 51, n. 4, p. 799-808, 2009.
- [Malz and Göhner 2011] Malz, C.; Göhner, P. Agent-based test case prioritization. In: 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops. IEEE, 2011. p. 149-152.
- [Martin 2012] Martin, C. R. The Clean Code Blog - The Clean Architecture. Available at: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>. Access in: 14 out. 2023.
- [Merchant and Castleman 2022] Merchant, F.; Castleman, K. Microscope image processing. Academic press, 2022.
- [Myers et al. 2011] Myers, G. J.; Sandler, C.; Badgett, T. The art of software testing. John & Sons, 2011.
- [Pressman and Maxim 2015] Pressman, R. S.; Maxim, B. R. Engenharia de software: uma abordagem profissional. 8. ed. McGraw-Hill, 2015.
- [Rothermel et al. 1999] Rothermel, G.; Untch, R. H.; Chu, C.; Harrold, M. J. 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). IEEE, 1999. p. 179-188.
- [Simon 1996] Simon, H. A. The sciences of the artificial. MIT press, 1996.
- [Singh et al. 2012] Singh, Y.; Kaur, A.; Suri, B.; Singhal, S. Systematic literature review on regression test prioritization techniques. Informatica, v. 36, n. 4, 2012.

- [Sommerville 2011] Sommerville, I. Engenharia de software. 9. ed. Pearson, 2011.
- [Srikanth et al. 2016] Srikanth, H.; Hettiarachchi, C.; Do, H. Requirements based test prioritization using risk factors: An industrial study. *Information and Software Technology*, v. 69, p. 71-83, 2016.
- [Srikanth et al. 2005] Srikanth, H.; Williams, L.; Osborne, J. System test case prioritization of new and regression test cases. In: 2005 International Symposium on Empirical Software Engineering, 2005. IEEE, 2005. p. 10 pp.
- [Srivastva et al. 2008] Srivastva, P. R.; Kumar, K.; Raghurama, G. Test case prioritization based on requirements and risk factors. *ACM SIGSOFT Software Engineering Notes*, v. 33, n. 4, p. 1-5, 2008.
- [Tonella et al. 2006] Tonella, P.; Avesani, P.; Susi, A. Using the case-based ranking methodology for test case prioritization. In: 2006 22nd IEEE international conference on software maintenance. IEEE, 2006. p. 123-133.
- [Valente 2020] Valente, M. T. O. Engenharia de software moderna. Independente, 2020. 395p.
- [Yoo and Harman 2012] Yoo, S.; Harman, M. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, v. 22, n. 2, p. 67-120, 2012.