Using Behavior-Driven Development (BDD) for Non-functional Requirements

Shexmo Santos , Tacyanne Pimentel , Fabio Gomes Rocha * , Michel Soares

Posted Date: 17 June 2024

doi: 10.20944/preprints202406.1061.v1

Keywords: Behavior-Driven Development; ISO/IEC/IEEE 25010; Non-functional requirements; Performance efficiency; Quality requirements

 doi:10.20944/preprints202406.1061.v1

Article

Using Behavior-Driven Development (BDD) for Non-Functional Requirements

Shexmo Santos †, Tacyanne Pimentel †, Fabio Rocha † and Michel Soares *

São Cristóvão, Sergipe, Brazil. Federal University of Sergipe; shexmor@gmail.com (S.S.); tacyblpimentel@gmail.com (T.P.); gomesrocha@gmail.com (F.R.)

* Correspondence: michel@dcomp.ufs.br

† These authors contributed equally to this work.

Abstract: Context: In Software Engineering, there must be clarity in communication among interested parts to elicit the requirements aimed at software development through frameworks to achieve the behaviors expected by the software. **Problem**: Lack of clarity in the requirements elicitation stage can impact subsequent stages of software development. **Solution**: We proposed a case study to achieve the proposed goal focusing on the performance efficiency characteristic expressed in the ISO/IEC/IEEE 25010 Standard using Behavior-Driven Development (BDD). **Method**: The case study was performed with professionals who use BDD to elicit the non-functional requirements of a company that develops software. **Summary of Results**: The results obtained were aspects related to the elicitation of non-functional requirements aimed at the performance efficiency characteristic of the ISO/IEC/IEEE 25010 Standard using BDD through the point of view of industry professionals. **Contributions and impact**: The article's main contribution is to demonstrate the effectiveness of using BDD to elicit non-functional requirements about the performance efficiency characteristic of the ISO/IEC/IEEE 25010 Standard.

Keywords: behavior-driven development; ISO/IEC/IEEE 25010; non-functional requirements; performance efficiency; quality requirements

1. Introduction

The software architect is the professional responsible for building the software, considering internal and external factors such as the architectural standard and clients, respectively [1]. For an architect to successfully develop software, he must know the aspects inherent to its creation, for example, the existing frameworks to achieve specific purposes to maintain quality software.

Framework is defined as a structure that serves as a basis for systematically building applications. There is also a need for frameworks used in software architecture to be able to meet such expectations external to agile methods, for example, framework Behavior-Driven Development (BDD), used in the life cycle of a system [2].

BDD introduces a flexible methodology that allows monitoring the characteristics to be developed by elaborating requirements through User Stories. These requirements are validated through scenarios that outline the convenience criteria for the behavior of the functionalities in question.

The BDD framework defines the software system's behavior using the "given, when, then" pattern, expressed in high-level, domain-specific natural language and executed through automated tests. BDD can document non-functional requirements (quality requirements) based on the performance efficiency characteristic presented in the ISO/IEC/IEEE 25010 [3] Standard.

The ISO/IEC/IEEE 25010 [4] Standard is part of the SQuaRE series of International Standards that guides the development process and proposes six divisions: Quality Management Division (ISO/IEC 2500n), Quality Model Division (ISO /IEC 2501n), Quality Measurement Division (ISO/IEC 2502n), Quality Requirements Division (ISO/IEC 2503n), Quality Assessment Division (ISO/IEC 2504n), SQuaRE Extension Division (ISO/IEC 25050 – ISO/IEC 25099).

 doi:10.20944/preprints202406.1061.v1

The ISO/IEC/IEEE 25010 Standard describes the quality characteristics that can be employed to evaluate software quality, such as non-functional requirements. By aligning the characteristics of the ISO/IEC/IEEE 25010 Standard with BDD examples, non-functional requirements can be documented and tested in a structured way. This technique ensures that software meets desired quality standards and provides a framework for evaluating the quality of software in use [5].

The goal of this study was to "**Analyze** the application of BDD, **with the purpose of** automating, **in relation to** non-functional requirements based on the characteristics of the ISO/IEC/IEEE 25010 Standard, **from the point of view** of those involved in the software product, **in the context of** the performance characteristic". The following research questions were proposed to achieve this purpose:

- How does BDD address difficulties in ensuring non-functional requirements?;
- How can BDD be used to ensure quality related to the performance efficiency characteristic of the ISO/IEC/IEEE 25010 Standard?;
- What are the benefits of using BDD to identify and automate non-functional tests?

In order to maintain software quality, there is a need for the structures used in its development to be designed in such a way as to contribute to the delivery of the final product, aiming to achieve what the user expects from the product and reduce rework arising from flaws in these structures [6]. Through this case study, we validate the proposed scenarios in a company that uses the BDD framework for cycles test, ensuring security in the quality of the product delivered according to testable scenarios for non-functional requirements in a real example.

This study splits as follows: **Related works**, where we address studies related to this research; **Theoretical concepts**, where we point out the main points related to the application of the ISO/IEC/IEEE 25010 Standard and BDD; **Methodology**, where we explain how we did the methodological steps of this study; **Case execution**, where we discuss the user stories used in this research; **Results**, where we present the data obtained; **Discussions**, where we explain the contributions of this study concerning related works; **Threats to validity**, where we discuss the threats that have been identified and mitigated; and, finally, **Conclusion**, where we present the contributions of this research to Software Architecture.

2. Related Works

Haoues *et al.* [7] conducted a study to show which type of architecture could follow the quality characteristics expressed in the ISO/IEC/IEEE 25010 Standard. The authors identified that it is necessary to think about quality throughout the entire software development life cycle to deliver the final product and maintain the quality characteristics of the ISO/IEC/IEEE 25010 Standard. Therefore, the architecture used must aim to meet the non-functional requirements demanded to maintain the quality of what was requested as a final product; thus, when carrying out a case study using BDD to elicit non-functional requirements, it will be possible to identify difficulties about guaranteeing such requirements.

Estdale and Georgiadou [8] explored the scope and interpretation of the quality models of the ISO/IEC/IEEE 25010 Standard, presenting aspects such as non-functional requirements inherent to software development. Non-functional requirements directly improve when implemented in products through measurable characteristics that meet a fixed set of specifications, aiming to satisfy usage and consumption expectations. In this way, this case study will identify how BDD will seek to guarantee quality related to the performance efficiency characteristic of the ISO/IEC/IEEE 25010 Standard.

According to Jarzkebowicz and Weichbroth [9], there is a greater need for attention when dealing with the elicitation of non-functional requirements. Therefore, the authors conducted a systematic review followed by interviews to identify how non-functional requirements were elicited and documented. The authors discovered through the results of the studies that non-functional requirements are an essential part of the software development process; however, they are carried out differently, with each company eliciting the requirements differently. Therefore, understanding the

 doi:10.20944/preprints202406.1061.v1

elicitation of non-functional requirements is essential, as it can impact the entire execution of software development. Therefore, through the use of BDD to elicit non-functional requirements, it will be possible to identify how the difficulties encountered by the authors could be mitigated.

Olsson, Sentilles, and Papatheocharous [10] also performed a Systematic Literature Review that analyzed empirical research for non-functional requirements. They observed that the ISO/IEC/IEEE 25010 Standard was mentioned in the articles analyzed by the authors, which had the function of directing the quality of the software during its development. However, researchers need to understand more when reporting possible solutions regarding non-functional requirements, which affects practitioners' acceptance. Therefore, validating research through methods such as case studies can effectively identify whether what they publish is in line with the daily challenges in the industry. Thus, this case study can contribute to the use of BDD in identifying and automating non-functional tests to maintain quality and achieve the behavior expected by the software.

3. Theoretical Concepts

In this Section, we discuss concepts inherent to the ISO/IEC/IEEE 25010 Standard and Behavior-Driven Development (BDD) in order to understand how the BDD framework can collaborate with the elicitation of non-functional requirements.

3.1. ISO/IEC/IEEE 25010 Standard

The ISO/IEC/IEEE 25010 Standard addresses software quality. By relying on this Standard, the software can meet the needs of stakeholders, such as performance efficiency [11].

The purpose of the ISO/IEC/IEEE 25010 Standard is to guide the evaluation of the quality of softwares and systems. The Standard is divided into eight quality attributes, or characteristics, as shown in Figure 1.

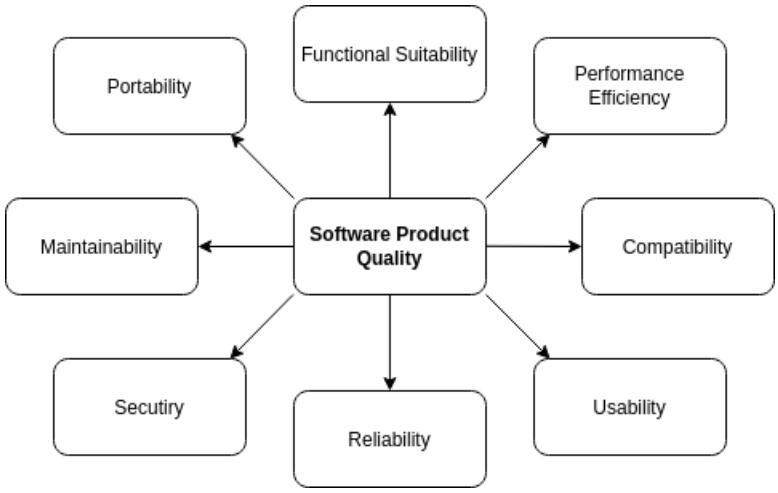

Figure 1. Quality Characteristics

The focus characteristic of this study was performance efficiency, which, in turn, is divided into three sub-characteristics: time behavior, resource utilization, and capacity.

This assessment uses specific techniques and tools that allow for measuring compliance with the requirements and criteria established by the Standard, improving the product development process, increasing user satisfaction, and gaining more competitiveness in the software industry.

A process represents a set of activities to achieve a previously stipulated goal. The capacity and maturity of a process are verified by the level of quality of what a software achieves related to an expected result. The software development process focuses on delivering a software implementation [12]. Thus, it is clear that the quality of a system and its maturity must meet the needs of stakeholders [8].

 doi:10.20944/preprints202406.1061.v1

3.2. Behavior-Driven Development (BDD)

BDD emerged in 2003, conceived by Dan North [13] as a framework of notable flexibility. North outlined the need for a framework focused on solving communication and team integration issues. Test-Driven Development (TDD) does not address these gaps while maintaining a focus on test automation, a factor that contributes significantly to product quality. Thus, while TDD is primarily dedicated to technical aspects, BDD seeks to improve communication and collaboration between the technical team and stakeholders. In this context, BDD is a framework mainly oriented towards observing system behavior.

In order to promote collaboration and communication, BDD introduced gherkin, a remarkable way of expressing specifications that culminate in user stories and acceptance scenarios in natural language. This approach enables integration with testing tools, allowing the adoption of BDD to monitor system requirements and evaluate its behavior, ensuring that it is validated as ready at the end of each iteration. BDD has been widely recognized by the academic community and industry, demonstrating significant benefits, such as more effective communication, reduced release delivery time, and greater accuracy in requirements elicitation [14].

BDD uses the "3 amigos" technique, where a meeting is held between the product owner, tester, and developer in order to objectively and concisely specify the expected behavior by the system when eliciting the requirements, in this case, user stories and their acceptance scenarios, in order to outline the behavior expected by the software. As such stories need to be written directly and transparently to achieve the proposed objective, BDD needs to make its scenarios testable [15]. In this way, it is clear how BDD can collaborate to elicit non-functional requirements.

According to Bruschi *et al*. [2] improved communication through BDD results in better quality collaboration between those involved in the process, as the requirements are standardized by the "3 amigos" in order to also help with the documentation and automation of tests, necessary factors the validity. Furthermore, with the time savings gained through reduced rework, one can focus on other activities inherent to the software. Therefore, after outlining the requirement to be implemented, the next step is to write it through BDD following the gherkin language.

From this point on, the features begin to be written, implemented, and validated, seeking to maintain the behavior expected by the requirements elicited in each release in order to deliver what is being expected by the system. In addition to helping with communication and collaboration between the parts involved [16], as it is written in English, the BDD also helps with the dynamic documentation of the system [14,15,17]. Dynamic documentation, or living documentation, refers to the agility required for updates to process documentation. With these aspects, it is expected that the use of BDD will save time and money, as it reduces the need for rework due to poor quality releases, has better quality validation [5,18], in addition to a better understanding of the code [3].

According to Heck and Zaidman [19], the quality of requirements is an essential aspect of the quality of the final product. There is a need for requirements to be elicited in a clear, objective, and unambiguous manner to deliver what is expected by the customer. Aiming to analyze how a company's non-functional requirements were elicited, we conducted the following case study.

4. Methodology

Evidence-Based Software Engineering is a methodology whose purpose is to deliver systems with greater reliability [20] through, for example, case studies. The methodology used in this research on the application of BDD for non-functional requirements and automation, based on the performance efficiency characteristic of the ISO/IEC/IEEE 25010 Standard, was conducted through a case study in collaboration with a company that develops software.

Case studies have been increasingly used in Software Engineering [21] because they use empirical data collection that seeks to understand a specific population. Thus, Software Engineering is based on observing actions aimed at software and their respective responses [22], as, in this way, it is possible to use the observed aspects to validate how the system should behave. One of the research purposes in

 doi:10.20944/preprints202406.1061.v1

Software Engineering focuses on how software engineers and other professionals in the field carry out development, operation, and maintenance in different situations [23]. Therefore, according to Perry, Sim, and Easterbrook [24], a case study is a scientific method carried out in a planned manner, from the elaboration of questions to the presentation of results.

This research is exploratory and descriptive [23,25], given the inherent need to elicit non-functional requirements through agile frameworks, in order to integrate quality validation. Some guidelines pointed out by Melegati and Wang [21] and Peterson [26] were followed, which involve a clear definition of research questions and goals, appropriate selection of relevant cases for research, collection, and analysis of data from various sources such as interviews, observations, and documentation. There is also a need to ensure the validity and reliability of the data through triangulation and member checking.

This research analyzed the case of a team that adopted BDD for non-functional requirements and test automation based on the performance efficiency characteristic of the ISO/IEC/IEEE 25010 Standard. Experienced teams are understood to have more than one year of experience in adopting BDD; however, they still need to apply it to non-functional requirements. The case study focused on analyzing requirements elicitation in one of the products developed by a Brazilian software manufacturer. The tested product is for customer management and product pricing. This Section describes the steps followed during the research, highlighting the team members' participation in the process.

Below, for better understanding, a BPMN diagram of the steps of this research is presented in Figure 2.

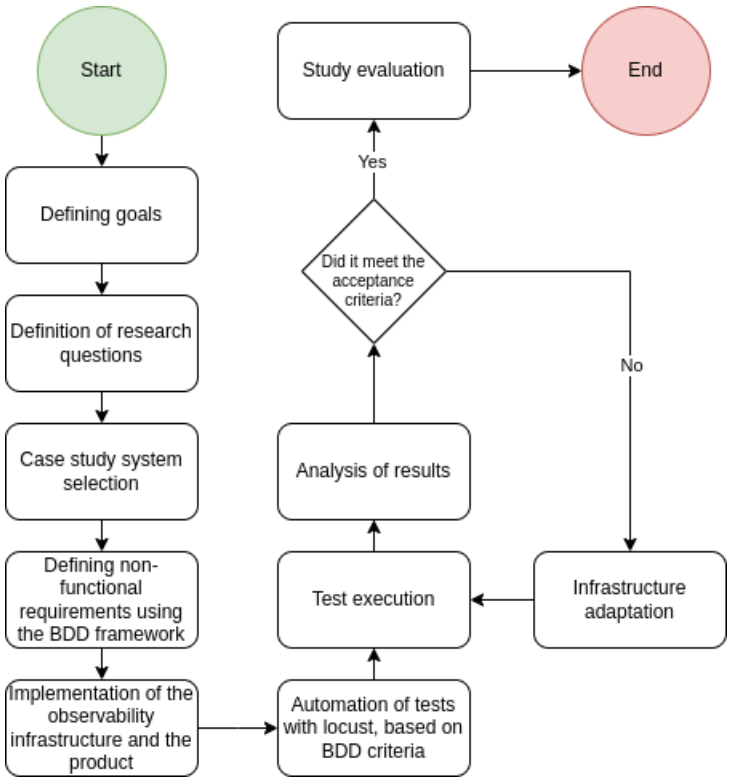

Figure 2. Step by Step of the research

When the research topic was defined, the method that achieved the relationship proposed as a general goal between the ISO/IEC/IEEE 25010 Standard and the BDD was chosen, followed by the elaboration of research questions, carrying out the case study, list of threats validity and how we mitigate them and, finally, conclusion.

 doi:10.20944/preprints202406.1061.v1

4.1. Case Study

The protocol used to direct this case study was formalized using part of the *Goal-Question-Metric* [27] model defined below: "**Analyze** the application of BDD, **with the purpose of** automating, **in relation to** non-functional requirements based on the characteristics of the ISO/IEC/IEEE 25010 Standard, **from the point of view** of those involved in the software product, **in the context of** the characteristic of performance". In order to achieve the proposed goal, we created the research questions expressed in Table 1.

Table 1. Research questions

Research questions		
Identification	Question	Rationale
RQ1	How does BDD address difficulties in ensuring non-functional requirements?	This question aimed to understand how BDD can mitigate situations arising from the elicitation of non-functional requirements.
RQ2	How can BDD be used to ensure quality related to the performance efficiency characteristic of the ISO/IEC/IEEE 25010 Standard?	This question aimed to explain how this framework could contribute to eliciting non-functional requirements.
RQ3	What are the benefits of using BDD in identifying and automating non-functional tests?	This question aimed to bring positive points to adopting BDD in the testing cycle.

We applied this case study to a system with mixed CRM and people management characteristics, so we chose it due to its relevance to the company. A multidisciplinary team was involved during the case study with the following team members:

- A software architect responsible for defining the architecture and non-functional requirements;
- Two developers in charge of implementing the acceptance scenarios;
- A Team Lead, responsible for leading the process and helping with communication between team members;
- A test analyst tasked with automating performance tests using Locust in Python;
- A DevOps, responsible for configuring the infrastructure and deploying the system.

The software architect created user stories and acceptance scenarios that described non-functional performance requirements based on the characteristics of the ISO/IEC/IEEE 25010 Standard, following the gherkin language. These user stories and acceptance scenarios were presented to the team and relevant stakeholders to gain their approval, ensuring a common understanding of the non-functional requirements.

The tests were automated after the user stories were written, and the acceptance and approval criteria were set by those involved. The test analyst was responsible for automating performance tests using the Locust tool in Python, ensuring that acceptance scenarios were translated into executable tests.

The tests were carried out, and the metrics available in the acceptance criteria were monitored using analysis tools. During the execution of performance tests, the multidisciplinary team, including the software architect, test analyst, and DevOps, tracked the results and metrics generated by monitoring and analyzing the results using Grafana and Jaeger tools.

 doi:10.20944/preprints202406.1061.v1

4.2. Interview

Before executing the automation tests, one of the authors interviewed the team lead who used BDD in this case study. In a brief conversation, the team lead informed us of how the requirements were approved by interested parts to achieve the behavior expected by the software. Furthermore, he mentioned the importance of integration between the team, the process, and the quality of the software, aiming to obtain releases that meet what was requested, stating the importance of using BDD from the elicitation of the requirement to the execution of the test cases.

In order to guarantee the performance efficiency characteristic expressed in the ISO/IEC/IEEE 25010 Standard, as well as its sub-characteristics, based on the elicitation of requirements, user stories were created with their respective acceptance criteria reported in the following Section.

5. Case Execution

The execution of the case study is based on the functionalities and scenarios defined following the ISO/IEC/IEEE 25010 standard using BDD. The case study intended to evaluate the system's performance efficiency concerning three sub-characteristics: time behavior, resource utilization, and capacity. The user stories and acceptance criteria, following the gherkin language, are described below so that each Subsection corresponds to one of the sub-characteristics addressed in this study regarding the ISO/IEC/IEEE 25010 Standard.

5.1. User Story: System Efficiency (Time Behavior)

Description: As a system's administrator, the team seeks to ensure the system can handle 100 concurrent users for 7 days, ensuring efficiency is consistent during periods of high demand.

Scenario: Time behavior test

Given that the system is working correctly and can accommodate 100 simultaneous users, as shown in Figure 3.

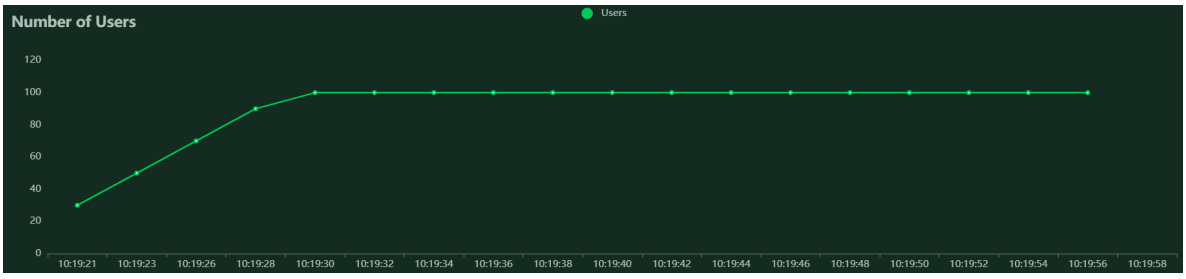

Figure 3. Number of users

When 100 simultaneous users access the system for 7 consecutive days, each user creates an opportunity every 5 minutes, as shown in Figure 4.

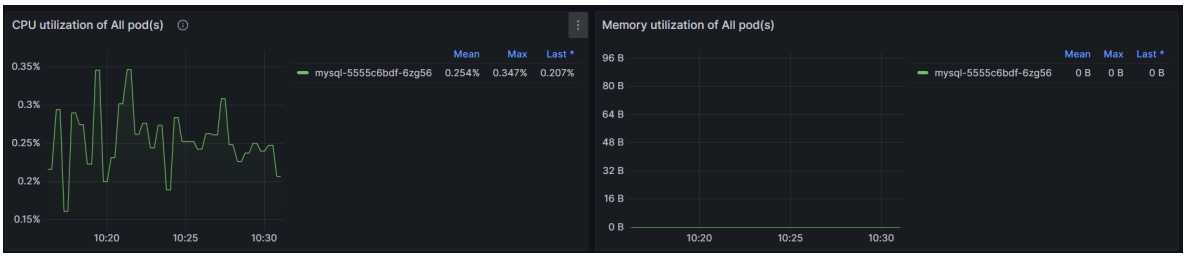

Figure 4. Memory usage

 doi:10.20944/preprints202406.1061.v1

Then the system must maintain an average response time of less than 5 seconds, not present critical errors during the test, and the use of server resources must remain below 80%, as shown in Figure 5.

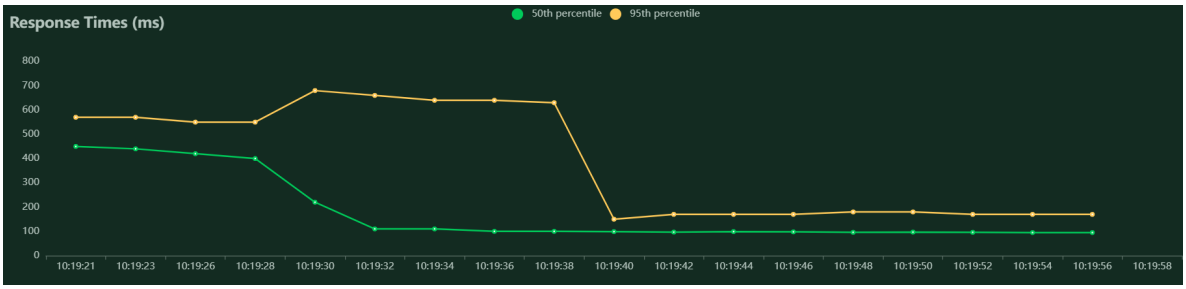

Figure 5. Time to response

5.2. User Story: Resource Utilization

Description: As a system's administrator, the team seeks to ensure that the system can handle 1000 concurrent users for 30 minutes, ensuring that server resource utilization remains below 85% during load testing.

Scenario: Resource utilization test

Given that the system is working correctly and has capacity for 1000 simultaneous users.
When 1000 simultaneous users access the system for 30 minutes.
Then server resource utilization should not exceed 85%, and the system should not experience critical errors during testing.

5.3. User Story: Capacity

Description: As a system's administrator, the team seeks to ensure that the system can handle 2,700 concurrent users for 30 minutes, ensuring that system capacity is sufficient to meet demand during peak usage.

Scenario: System capacity test

Given that the system is working correctly and has a capacity for 2,700 simultaneous users.
When 2,700 simultaneous users access the system for 30 minutes.
Then server resource utilization should not exceed 90%, and the system should not experience critical errors during testing.

6. Results

Below are the answers to the research questions presented in Table 1.

6.1. QP1 - How Does BDD Address Difficulties in Ensuring Non-Functional Requirements?

The BDD framework intended for test automation, such as Cucumber and JBehave, was initially designed to address the system's functional units, which resulted in significant challenges in the integration between user stories and their corresponding scenarios and subsequent automation. However, as evidenced in this investigation, it is possible to considerably mitigate the difficulties associated with the practical application of BDD in contexts involving non-functional requirements by adopting complementary tools.

In the present study, we opted for the combined use of Pytest-BDD in conjunction with Locust. While the former makes it possible to apply BDD in Python environments, the latter facilitates

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 June 2024 doi:10.20944/preprints202406.1061.v1

performance testing. As a result, an integration enables the validation of the acceptance criteria established by the BDD, focusing on evaluating the system's performance. This approach demonstrated that adopting complementary tools, such as Locust, allows the framework adopted for BDD to validate the acceptance criteria efficiently, providing more excellent reliability to the results obtained.

Adopting BDD combined with complementary tools emerges as a complete and adequate procedure to satisfy the system's non-functional requirements, allowing efficient compliance monitoring with the proposed scenarios.

6.2. QP2 - How Can BDD Be Used to Ensure Quality Related to the Performance Efficiency Characteristic of the ISO/IEC/IEEE 25010 Standard?

Solis and Wang [28] identified essential characteristics of BDD, highlighting scenario models and automated acceptance tests with mapping rules as preponderant elements. Such characteristics play a fundamental role in ensuring software quality. In this study, we adopted a set of tools based on user stories and acceptance criteria, enabling test automation and monitoring of user perspectives. To this end, we used Locust to evaluate system performance and Jaeger to analyze communication between services, as illustrated in Figure 6.

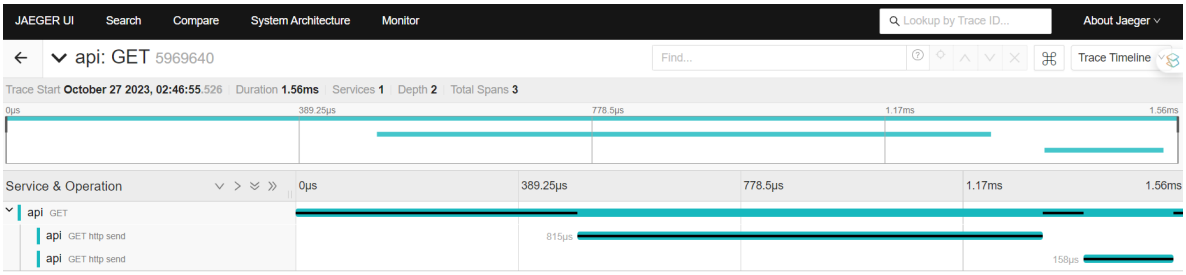

Figure 6. Communication using Jaeger

Figure 6 presents Jaeger. This system shows communication between services, demonstrating performance and how each microservice communicates, allowing us to understand if there is an internal problem between services. This approach facilitates the work that links the need for performance that BDD pointed out and the possibility of verifying the behavior, not just of the response, but of the components linked to the response.

Figure 7 shows Grafana analyzing the containers of running services, allowing us to understand memory and processor consumption during tests, thus allowing us to cross-reference information from the point of view of the user with Locust, the service with Jaeger and the server with Grafana, having a 360 view of the system's performance.

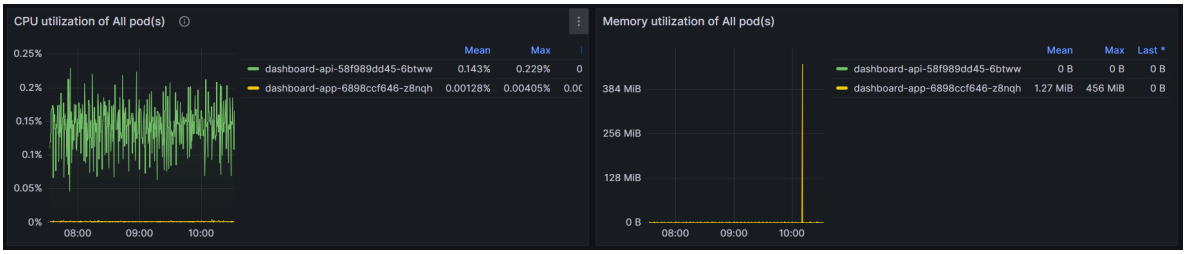

Figure 7. Evaluation of infrastructure components

The assessment of infrastructure components, monitored by Grafana, provided a comprehensive and end-to-end view of the system's performance.

 doi:10.20944/preprints202406.1061.v1

6.3. QP3 - What Are the Benefits of Using BDD in Identifying and Automating Non-Functional Tests?

The application of BDD to ensure the quality of a system's characteristics offers a series of advantages, particularly highlighting the substantial improvement in communication between the architect and the product development team. Furthermore, its intrinsic integration with automation processes provides greater transparency and ease in validating the system, ensuring that it effectively fulfills the characteristics outlined during the design process.

As evidenced in Section 4.1, BDD becomes more transparent and accurate in requirements elicitation when using the gherkin language. In this way, we achieved the behaviors expected by the software, achieving success in automating non-functional tests. Furthermore, as demonstrated in Figure 2, the importance of acceptance criteria meeting the tests to guarantee the quality initially proposed by eliciting the requirement becomes evident.

7. Discussions

The execution of these test scenarios made it possible to evaluate the system's performance concerning the sub-characteristics inherent to the performance efficiency characteristic of the ISO/IEC/IEEE 25010 Standard and ensure that the defined non-functional requirements were met. Thus, scripts were created for testing in Python with Locust. When running, the configurations for each user and execution scenario are made on the screen, as shown in Figure 8.

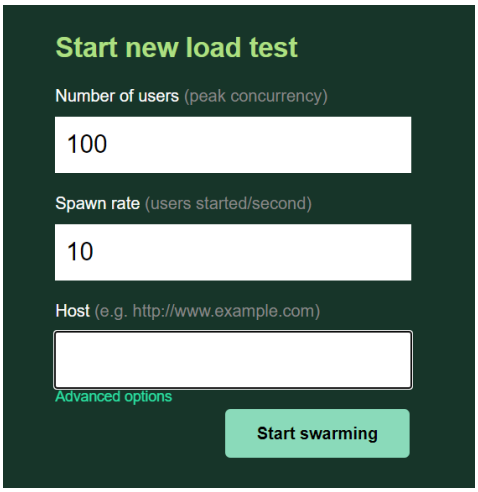

Figure 8. Configuring Locust

As mentioned by Haoues *et al.* [7], to maintain software quality following ISO/IEC/IEEE 25010 Standard, there is a need to focus on the quality aspect throughout the entire cycle of software life. In this way, BDD managed to present positive results from eliciting requirements, carried out objectively among team members, to the automation of tests to meet the proposed acceptance criteria.

According to Estdale and Georgiadou [8], non-functional requirements present improvements regarding the functioning of the software. As they are considered more complex, it is expected to have difficulty in eliciting non-functional requirements. Through the gherkin language used by BDD, there was a better understanding of what to expect from the software's behavior, so that user stories were described with a clear purpose.

According to Jarzkebowicz and Weichbroth [9], it is necessary to understand how non-functional requirements are elicited. Through this study, we perceived the necessary steps for the elicitation of non-functional requirements, as shown in Figure 2, in order to realize the need to validate the requirement through user stories and their respective acceptance criteria to identify whether what was expected by the software was achieved or not.

Regarding the Systematic Literature Review with a focus on non-functional requirements carried out by Olsson, Sentilles, and Papatheocharous [10], the authors observed the need to carry out studies

 doi:10.20944/preprints202406.1061.v1

that validate what has been researched in academia in order to deliver answers to assertions consistent with the reality of the industry. In this way, this research contributed to understanding the elicitation of non-functional requirements with BDD in an current situation.

As North [13] mentioned, BDD was created to mitigate problems arising from Test-Driven Development (TDD), so BDD is focused on the behavior of releases, while TDD focuses directly on testing. Thus, the author also states that BDD has the benefits of improving communication and readability, aspects perceived in this research in the elicitation of non-functional requirements so that there was the participation of the parts involved in the development of releases, such as was described in Section 4.

8. Threats to Validity

Threats to validity are mitigated during the research to carry out a study with more excellent reliability [23]. The descriptions made by Zhou *et al.* were used [29] regarding types of validity.

8.1. Construction Validity

Understanding the ISO/IEC/IEEE 25010 Standard and the BDD was necessary for the case study of this research to be carried out. Based on what we presented in Section 3, it became possible to identify the importance of analyzing whether the elicitation of non-functional requirements through BDD followed the performance efficiency characteristic set out in the ISO/IEC/IEEE 25010 Standard.

8.2. Internal Validity

We carried out this research with the participation of four collaborators: two carried out the collection of information in pairs regarding the ISO/IEC/IEEE 25010 Standard and BDD, another carried out the case study, and, finally, the latter carried out the refinement of information throughout the article writing process.

8.3. External Validity

Aiming to maintain quality in results, we considered the members who would be part of the team that would elicit the non-functional requirements set out in Section 4. In this way, it was possible to understand each team member's role in eliciting and implementing a requirement.

8.4. Conclusion Validity

This research is replicable using the methodological steps described in Section 4 and theoretical research regarding the ISO/IEC/IEEE 25010 Standard and BDD.

9. Conclusion

In this study, the BDD framework was adopted for the elicitation of non-functional requirements and the ISO/IEC/IEEE 25010 Standard, precisely the performance efficiency characteristic with its three sub-characteristics, namely, time behavior, resource utilization, and capacity. Each subcharacteristics was associated with a user story, accompanied by one or more acceptance scenarios, as exemplified in Section 5. This approach made it possible to guide user testing, establishing links between efficiency characteristics and how requirements are being elicited through the BDD gherkin language.

Furthermore, as it is written in gherkin language, BDD contributed to eliciting non-functional requirements about BDD's objectivity for writing focused on the system's behavior. In this way, the use of BDD helped those involved in the requirements elicitation process to mitigate possible ambiguities in writing user stories and their respective acceptance scenarios.

The use of BDD as a framework in the software life cycle leads to improved communication on the part of the members involved in the process to improve the software development stages, from requirements elicitation to documentation and maintenance. This research brought a holistic approach

 doi:10.20944/preprints202406.1061.v1

to the system's quality requirements by relating BDD to the ISO/IEC/IEEE 25010 Standard. Through this study, BDD showed its effectiveness following the performance efficiency characteristic expressed in the ISO/IEC Standard /IEEE 25010 as per Section 6.

The obstacles encountered in this study were mitigated according to Section 8 to guarantee the results' quality. The goal of this research was achieved, and the research questions expressed in Table 1 were answered according to Section 6, in addition to the process used for requirements elicitation and test automation being detailed, aiming to clarify the step-by-step process used, so that other researchers can replicate it.

As BDD is a framework used from requirements elicitation to software implementation and maintenance, communication between stakeholders and the development team is necessary to contribute to the quality of the software delivered. Thus, when using BDD in software development, related people present its importance so that BDD can achieve its goal proposed by Dan North: a framework that assists in communication and delivers releases according to expected behavior. Therefore, this study involves people, processes, and techniques to maintain the system's quality entirely.

As suggestions for future work, the methodology used in this research could be replicated in a new study with teams that use BDD using other quality characteristics of the ISO/IEC/IEEE 25010 Standard. A survey could also be carried out with a focus on the quality characteristics of the ISO/IEC/IEEE 25010 Standard with professionals who use BDD in order to identify whether software development has been based on the quality characteristics expressed in the ISO/IEC/IEEE 25010 Standard. Finally, an experiment could be carried out that compared the use of BDD to elicit non-functional requirements with and without the ISO/IEC/IEEE 25010 Standard as a reference to observe the impact caused by the quality expressed in the Standard.

Author Contributions: Conceptualization, F. R. and M. S.; methodology, F. R.; software, S. S. and F. R.; validation, F. R. and M. S.; formal analysis, F.R.; investigation, S. S. and T. P.; resources, S. S., T. P. and F. R.; data curation, F. R.; writing—original draft preparation, S. S. and T. P.; writing—review and editing, S. S., T. P., F. R. and M. S.; visualization, S. S., T. P., F. R. and M. S.; supervision, M. S.; project administration, M. S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Acknowledgments: We would like to acknowledge the CAPES/BRAZIL scholarship.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kruchten, P. What do software architects really do? *Journal of Systems and Software. Elsevier* **2008**, *V.81*, 2413–2416.
2. Bruschi, S.; Xiao, L.; Kavatkar, M.; others. Behavior Driven-Development (BDD): a case study in healthtech. Pacific NW Software Quality Conference, 2019.
3. Guerra-Garcia, C.; Nikiforova, A.; Jiménez, S.; Perez-Gonzalez, H.G.; Ramírez-Torres, M.T.; Ontañon-García, L. ISO/IEC 25012 - Based methodology for managing data quality requirements in the development of information systems: Towards Data Quality by Design . *Data and Knowledge Engineering* **2023**, *145*, 102152–102152. doi:10.1016/j.datak.2023.102152.
4. Valdés-Souto, F.; Núñez-Varela, A.S.; Pérez-González, H.G. Evaluating the software quality non-functional requirement through a fuzzy logic-based model based on the ISO/IEC 25000 (SQuaRE) standard. *2019 7th International Conference in Software Engineering Research and Innovation CONISOFT* **2019**, pp. 16–25.
5. Binamungu, L.P.; Embury, S.M.; Konstantinou, N. Maintaining Behaviour Driven-Development specifications: Challenges and opportunities. 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering SANER. IEEE, 2018, pp. 175–184.
6. Karagöz, G.; Sözer, H. Reproducing failures based on semiformal failure scenario descriptions. *Software Quality Journal* **2017**, *25*, 111–129.

 doi:10.20944/preprints202406.1061.v1

7. Haoues, M.; Sellami, A.; Ben-Abdallah, H.; Cheikhi, L. A guideline for software architecture selection based on ISO 25010 quality related characteristics. *International Journal of System Assurance Engineering and Management* **2017**, *8*, 886–909.
8. Estdale, J.; Georgiadou, E., Applying the ISO/IEC 25010 Quality Models to Software Product: 25th European Conference, EuroSPI 2018, Bilbao, Spain, September 5-7, 2018, Proceedings; 2018; pp. 492–503. doi:10.1007/978-3-319-97925-0_42.
9. Jarzębowicz, A.; Weichbroth, P. A qualitative study on non-functional requirements in agile software development. *IEEE Access* **2021**, *V. 9*, 40458–40475.
10. Olsson, T.; Sentilles, S.; Papatheocharous, E. A Systematic Literature Review of empirical research on quality requirements. *Requirements Engineering. Springer* **2022**, *V.27*, 249–271.
11. International Organization for Standardization. ISO/IEC/IEEE 25010. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010/, 2011.
12. Miguel, J.P.; Mauricio, D.; Rodríguez, G. A review of software quality models for the evaluation of software products. *arXiv preprint arXiv:1412.2977. Cornell University* **2014**.
13. North, D. Introducing BDD. https://dannorth.net/introducing-bdd/, 2006.
14. Pereira, L.; Sharp, H.; de Souza, C.; Oliveira, G.; Marczak, S.; Bastos, R. Behavior-Driven Development benefits and challenges: reports from an industrial study. Proceedings of the 19th International Conference on Agile Software Development: Companion, 2018, pp. 1–4.
15. Silva, T.R.; Fitzgerald, B. Empirical findings on BDD story parsing to support consistency assurance between requirements and artifacts. In *Evaluation and Assessment in Software Engineering*; 2021; pp. 266–271.
16. Couto, T.; dos Santos Marczak, S.; Callegari, D.A.; Móra, M.; Rocha, F. On the Characterization of Behavior-Driven Development Adoption Benefits: A Multiple Case Study of Novice Software Teams. *Anais do XXI Simpósio Brasileiro de Qualidade de Software, 2022, Brasil.* **2022**.
17. Nascimento, N.; Santos, A.R.; Sales, A.; Chanin, R. Behavior-Driven Development: an expert panel to evaluate benefits and challenges. Proceedings of the XXXIV Brazilian Symposium on Software Engineering, 2020, pp. 41–46.
18. Moe, M.M. Comparative Study of Test-Driven Development TDD, Behavior-Driven Development BDD and Acceptance Test–Driven Development ATDD. *International Journal of Trend in Scientific Research and Development* **2019**, *3*, 231–234.
19. Heck, P.; Zaidman, A. A Systematic Literature Review on quality criteria for agile requirements specifications. *Software Quality Journal* **2018**, *26*, 127–160.
20. Kitchenham, B.A.; Dyba, T.; Jorgensen, M. Evidence-based software engineering. Proceedings. 26th International Conference on Software Engineering. IEEE, 2004, pp. 273–281.
21. Melegati, J.; Wang, X. Case survey studies in software engineering research. Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2020, pp. 1–12.
22. Wohlin, C.; Höst, M.; Henningsson, K. Empirical research methods in software engineering. *Empirical methods and studies in software engineering: Experiences from ESERNET. Springer* **2003**, pp. 7–23.
23. Runeson, P.; Höst, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering. Springer* **2009**, *V.14*, 131–164.
24. Perry, D.E.; Sim, S.E.; Easterbrook, S. Case Studies for Software Engineers. Proceedings of the 28th international conference on software engineering, 2006, pp. 1045–1046.
25. Yin, R.K. Case study methods. American Psychological Association, 2012.
26. Petersen, K. Guidelines for Case Survey Research in Software Engineering. *Contemporary empirical methods in software engineering* **2020**, pp. 63–92.
27. Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M.C.; Regnell, B.; Wesslén, A. *Experimentation in software engineering*; Springer Science & Business Media, 2012.
28. Solis, C.; Wang, X. A study of the characteristics of behaviour driven development. 2011 37th EUROMICRO conference on software engineering and advanced applications. IEEE, 2011, pp. 383–387.
29. Zhou, X.; Jin, Y.; Zhang, H.; Li, S.; Huang, X. A map of threats to validity of Systematic Literature Reviews in Software Engineering. 2016 23rd Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2016, pp. 153–160.

 doi:10.20944/preprints202406.1061.v1