An Experiment with Focus on Security Through Large- Language Models Using Behavior-Driven Development

Shexmo Santos * , Tacyanne Pimentel , Marcus Silva , Luiz Santos , Fabio Rocha , Michel Soares

Posted Date: 22 September 2025

doi: 10.20944/preprints202509.1860.v1

Keywords: Automated tests; Behavior-Driven Development; Large Language Model (LLM); Security

 doi:10.20944/preprints202509.1860.v1

Article

An Experiment with Focus on Security Through Large-Language Models Using Behavior-Driven Development

Shexmo Richarlison Ribeiro dos Santos *, Tacyanne Bernadete Lima Pimentel, Marcus Vinicius Santana Silva, Luiz Felipe Cirqueira dos Santos, Fabio Gomes Rocha and Michel S. Soares

Federal University of Sergipe
* Correspondence: shexmor@gmail.com

Abstract

Requirements elicitation is a fundamental activity in a software development process, which means it needs to be carried out effectively, aiming the requirements to perform the behavior expected by the software. **Problem**: The need for assertiveness in requirements elicitation tasks can impact subsequent activities of software development. **Solution**: It is proposed an experiment focusing on the security non-functional requirement expressed in the ISO/IEC 25010:2023 Standard through Large-Language Models (LLMs) using Behavior-Driven Development (BDD). **Method**: Via an experiment, this study is qualitative and descriptive. It is possible to analyze, through statistical tests, the effectiveness of LLMs to elicit non-functional requirements. **Summarization of Results**: The study presented the effectiveness of using pre-defined prompts for test automation through machine learning. **Contributions and impact**: Through this experiment, it is possible to present an effective collaboration between human and artificial intelligence, generating time savings for automatic generation of tests.

Keywords: automated tests; behavior-driven development; large language model (LLM); security

1. Introduction

Behavior-Driven Development (BDD) introduces a flexible methodology that allows monitoring the functionalities to be developed by elaborating requirements through User Stories [1]. These requirements are validated through scenarios that outline the convenience criteria for the behavior of the functionalities in question.

Standard ISO/IEC 25010:2023 [2] describes the quality characteristics that can be employed to evaluate software quality, normally presented as non-functional requirements. By aligning non-functional requirements described in ISO/IEC 25010: 2023 with BDD examples, non-functional requirements can be documented and tested in a structured way. BDD ensures that software meets desired quality standards and provides a framework for evaluating the quality of software in use [3]. Regarding productivity, using Large-Language Models (LLMs) has become a way out of software development process steps when properly supervised [4].

ChatGPT and Gemini are LLMs which represent significant advancements in Artificial Intelligence (AI), particularly in natural language processing. Developed by OpenAI and Google DeepMind, respectively, these models reflect a shift from specialized systems to more adaptable and multimodal approaches. ChatGPT, based on the Generative Pre-trained Transformer architecture, excels at generating human-like responses and has been widely used in applications such as customer support, content creation, and educational assistance [5]. In contrast, Gemini stands out for its ability to integrate visual, auditory, and textual data, enabling more complex interactions and a deeper understanding of context. This evolution in AI technology signals a movement toward more general models capable of multifaceted and intricate interactions, reflecting current trends in pursuing broader and more versatile AI [6].

 doi:10.20944/preprints202509.1860.v1

The protocol used to direct the experiment described in this paper is formalized using part of the *Goal-Question-Metric* model [7] as follows: **To analyze** the application of BDD through LLMs, **with the purpose of** automating generation tests, **in relation to** non-functional requirements based on the security non-functional requirement of ISO/IEC 25010:2023, **from the point of view** of those involved in the software product, **in the context of** software quality". Table 1 presents the research questions proposed to achieve this purpose.

Table 1. Research questions.

Id	Question	Justification
Q1	What is the similarity of responses between ChatGPT and Gemini?	Present possible differences between these LLMs.
Q2	How can BDD be used to ensure quality related to the security non-functional requirement through an LLM?	Explain how this framework could contribute to eliciting non-functional requirements.

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 [8].

Through this experiment, it will be possible to analyze the effectiveness of a prompt for requirements elicitation and its efficacy in test automation.

This study is divided as follows: **Theoretical concepts** point out the main points related to the application of ISO/IEC 25010:2023, BDD and LLM; **Related work** addresses studies related to this research; **Experiment** explains how the methodological steps of this study were guided by security non-functional requirement; **Results** present the data obtained; **Discussion** show the contributions of this study and how it advanced concerning related work; **Threats to validity** discuss the threats that were identified and mitigated; and, finally, **Conclusion** presents the contributions of this research to software development and information systems.

2. Theoretical Concepts

This Section discusses concepts inherent to ISO/IEC 25010: 2023, Behavior-Driven Development (BDD), and Large Language Model (LLM) in order to understand how the BDD framework can collaborate through LLMs aiming automated tests.

2.1. ISO/IEC 25010:2023

ISO/IEC 25010:2023 "defines a product quality model, which is applicable to ICT (Information and Communication Technology) products and software products" [2]. This emphasizes the creation of products that not only meet technical requirements but also connect with human needs. ISO/IEC 25010:2023 comprises eight quality attributes, or non- functional requirements. The focus of this research is on security, which is segmented into six subcharacteristics: confidentiality, integrity, non-repudiation, accountability, authenticity, and resistance.

Security assessment uses specific techniques and tools that allow for measuring compliance with the requirements and criteria established by ISO/IEC 25010:2023, improving the product development process, increasing user satisfaction, and gaining competitiveness in the software industry.

A process represents a set of activities to achieve a previously established goal. The capacity and maturity of a process are verified by the level of quality that a software achieves in terms of an

 doi:10.20944/preprints202509.1860.v1

expected result. Thus, it is clear that the quality of a system and its maturity must meet the needs of stakeholders [9].

Maturity models [10] are based on the evolution of processes and the existence of guidelines that guide and quantify both the implementation and improvement of these processes. Therefore, software security maturity refers to the ability of a product or system to protect information and data against vulnerabilities, in addition to managing the effects of an attack.

The continuous evolution of the software system makes it unreliable, presenting a greater propensity for errors, delays in delivery, and costs that often exceed forecasts. The software architecture serves as the fundamental basis, providing a high-level vision that has led to the development of several modeling techniques, design, and programming languages. With the increase in the use of software, its size and complexity have also increased, making it challenging to analyze and test in detail, directly reflecting on maintenance costs.

2.2. Behavior-Driven Development (BDD)

BDD emerged in 2003, conceived by Dan North [1] 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.

BDD uses the "3 amigos" technique [11], 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. User stories and their acceptance scenarios are created aimed 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 [12]. In this way, it is clear how BDD can collaborate to elicit non-functional requirements.

According to Bruschi et al. [13], 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 to the validity. Furthermore, with the time savings gained through reduced rework, one can focus on other activities inherent to the software.

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.

2.3. Large Language Model (LLM)

Large Language Models (LLMs), such as ChatGPT from OpenAI and Gemini from Google DeepMind, have impacted interaction with technology, allowing for more fluid and natural communication. ChatGPT, based on the transformer architecture [14], is a prominent example of this innovation, standing out for its ability to understand complex contexts and generate coherent and detailed responses.

Developed by the company OpenAI, ChatGPT uses the GPT-4 architecture and is trained on a wide range of texts, which makes it possible to address different topics in a natural and relevant way. Its ability to handle interactive dialogs and generate coherent text makes it a disruptive and valuable tool for various applications, including virtual assistants and content generation.

3. Related Work

Rajhoj et al. [15] carried out a case study using ChatGPT to analyze the requirements elicitation stage in the software life cycle to save time and focus on other parts of the process. The authors concluded that using Generative AI is an area of research to be explored. Using the experiment described in this paper to automate related tests may contribute to this study by demonstrating whether it is effective in using ChatGPT to automate tests.

 doi:10.20944/preprints202509.1860.v1

Santos et al. [16] carried out a case study to analyze the effectiveness of using BDD in eliciting non-functional requirements regarding the performance efficiency. The authors concluded that BDD effectively elicits non-functional requirements in a way that achieves the study's objective. With this new experiment, it may be possible to analyze the elicitation of non-functional requirements guided by ISO/IEC 25010:2023 through the LLM used, generating if positive, time savings in software development.

Olsson, Sentilles, and Papatheocharous [17] performed a Systematic Literature Review that analyzed empirical research for non-functional requirements. They observed that ISO/IEC 25010:2023 was mentioned in the articles analyzed by the authors, which had the function of directing the quality of software during its development. However, researchers need to understand more when reporting possible solutions regarding non-functional requirements, which affects practitioners' acceptance. The experiment described in this paper can contribute to the use of BDD in identifying and automating non-functional tests using an LLM to maintain quality and save time.

Kasauli et al. [18] conducted a case study with seven companies to analyze requirements engineering in developing large-scale agile systems. The authors concluded that there is a need for a solid approach to requirements engineering. Using the experiment described in this paper, it will be possible to observe whether using a LLM effectively generate automated tests.

According to Jarzkebowicz and Weichbroth [19], 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, by each company.

4. Experiment

Evidence-Based Software Engineering is a methodology who- se purpose is to deliver systems with greater reliability [20]. The methodology used in this research on the application of BDD for non-functional requirements and automation, based on the security non-functional requirements of ISO/IEC 25010:2023 using generative AI, is conducted through a descriptive, qualitative experiment.

Guidelines pointed out by Melegati and Wang [21] and Peterson [22] 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 observations, and documentation. The methodological steps of this study are presented in Figure 1.

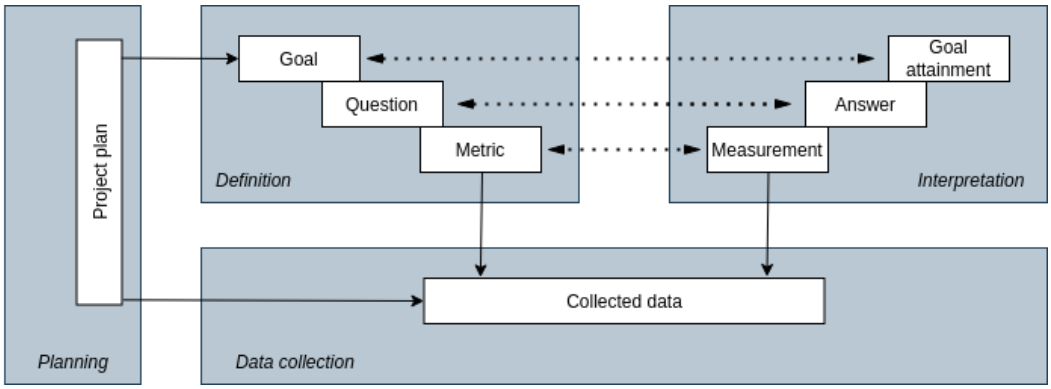

Figure 1. The 4 phases of the GQM method [adapted [23]].

With the goal, questions, and metrics inherent to this study outlined, this paper could be analyzed through data obtained in the experiment effectively. This research analyzed an experiment of adopting BDD for non-functional requirements and test automation based on the security non-functional requirements of ISO/IEC 25010:2023 using generative AI.

 doi:10.20944/preprints202509.1860.v1

4.1. Experiment Execution

The model for executing this experiment follows Rajbhoj et al. [15], as shown in Figure 2.

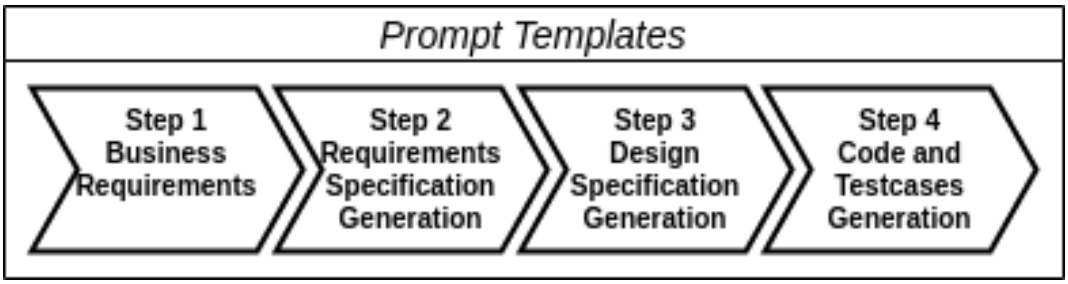

Figure 2. Software Development Life Cycle Process using Generative AI [adapted [15]].

The experiment analyzed in this paper focused on the effectiveness of AI-generated writing through prompts created to maintain quality in software development while saving time in requirements elicitation and test automation. Execution of the experiment is based on functionalities and scenarios defined following ISO/IEC 25010:2023 using BDD and generative AI.

The presented paper analyzed the security non-functional requirement and its respective six sub-characteristics, namely: confidentiality, integrity, non-repudiation, accountability, authenticity, and resistance.

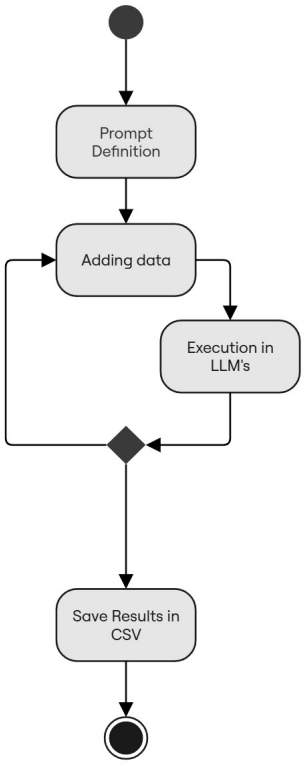

Figure 3. Prompt Automation Process.

A separate prompt is defined for each of the subcharacteristics that are the subject of this study. The information added to each non-functional requirement is allocated according to the need for each subcharacteristic to be used in the LLMs. Results brought by the LLMs are refined when necessary so that the prompt could present an acceptable result.

 doi:10.20944/preprints202509.1860.v1

4.2. Evaluated Metric

Similarity is the metric evaluated in this study. It is related to the similarity of results obtained between ChatGPT and Gemini, aiming to identify if these LLMs present similar or significantly different results. Through these statistics, it will be possible to present assertive results with proven information.

5. Results

Answers to RQs in Table 1 are presented as follows.

5.1. Q1—What Is the Similarity of Responses Between ChatGPT and Gemini?

To measure the similarity of responses between ChatGPT and Gemini, it is used the Kruskal-Wallis test. This non-parametric test fits for comparing independent distributions. Our hypothesis are shown as follows.

- Metric: Similarity coefficient.
- Alternative Hypothesis (H1): LLMs do not generate similarity to each other.
- Null Hypothesis (HN1): LLMs generate similarity to each other.

Confidentiality
Statistic: 6.8181
p-value: 0.0090
Results indicated a Kruskal-Wallis statistic of 6.8181 and a p-value of 0.0090. Tests showed evidences concerning differences between the LLMs. The extremely low p-value (less than 0.05) suggests that there is a statistically significant difference in the similarity of responses between ChatGPT and Gemini.

Integrity
Statistic: 19.1731
p-value: 0.0017
Results indicated a Kruskal-Wallis statistic of 19.1731 and a p-value of 0.0017. Tests showed evidences concerning differences between the LLMs. The extremely low p-value (less than 0.05) suggests that there is a statistically significant difference in the similarity of responses between ChatGPT and Gemini.

Non-repudiation
Statistic: 9.81
p-value: 0.0074
Results indicated a Kruskal-Wallis statistic of 9.81 and a p-value of 0.0074. Tests showed evidences concerning differences between the LLMs. The extremely low p-value (less than 0.05) suggests that there is a statistically significant difference in the similarity of responses between ChatGPT and Gemini.

Accountability
Statistic: 5.33
p-value: 0.0209
Results indicated a Kruskal-Wallis statistic of 5.33 and a p-value of 0.0209. Tests showed evidences concerning differences between the LLMs. The extremely low p-value (less than 0.05) suggests that there is a statistically significant difference in the similarity of responses between ChatGPT and Gemini.

Authenticity
Statistic: 5.33
p-value: 0.0209

 doi:10.20944/preprints202509.1860.v1

Results indicated a Kruskal-Wallis statistic of 5.33 and a p-value of 0.0209. Tests showed evidences concerning differences between the LLMs. The extremely low p-value (less than 0.05) suggests that there is a statistically significant difference in the similarity of responses between ChatGPT and Gemini.

Resistance
Statistic: 5.33
p-value: 0.021
Results indicated a Kruskal-Wallis statistic of 5.33 and a p-value of 0.021. Tests showed evidences concerning differences between the LLMs. The extremely low p-value (less than 0.05) suggests that there is a statistically significant difference in the similarity of responses between ChatGPT and Gemini.
Tests have shown evidence that there are differences between the groups. H1 supported.

5.2. Q2—How Can BDD Be Used to Ensure Quality Related to the Security Non-Functional Requirements Through an LLM?

BDD could elicit the non-functional requirements concisely and objectively through the quality of the prompt writing. Through the gherkin pattern, using BDD infers short and clear sentences in a way that allows an assertive result.

LLMs like ChatGPT and Gemini offer significant advantages, such as:

- **Requirements creation and validation**: ChatGPT and Gemini help one clearly and precisely develop non-functional requirements, such as security.
- **Requirements gathering**: During this stage, ChatGPT and Gemini can transform complex information placed through the prompt into precise specifications to generate test scenarios aligned with ISO/IEC 25010: 2023 Standard through BDD.
- **Documentation automation**: ChatGPT and Gemini expand the ability to generate documentation and facilitate the generation of technical documentation and test scenarios.
- **Productivity and Consistency**: Automating documentation can save time and reduce manual effort and consistent text generation helps maintain the quality of requirements.

By writing prompts with satisfactory results from a technical point of view, it is possible to identify the possibility of automating tests of non-functional requirements used in this study through BDD, ensuring assertiveness in the behavior expected by the requirement.

6. Discussion

Experiment described in this paper has made it possible to relate the automated test code generation process with humans and machine learning. Through prompts outlined with clear objectives, AIs can assist in delivering acceptance criteria with quality. The prompt model created for this experiment is presented as follows.

Listing 1. Instruction on the security non-functional requirement.

You are the requirements analyst responsible for defining non-functional requirements that align with the characteristics and sub-characteristics of Standard 25010 related to information security. As a requirements analyst, I want you to define the non-functional requirements, linked to <sub-feature>, for <application detail> with the following constraints <constraints> and expect the following result: user story model and acceptance criteria of gherkin type in <programming language> with a table of non-functional requirements to be met.

Rajhoh et al. [15] explained the need to advance studies on AIs in the requirements elicitation stage within the software development process. In this experiment, it is possible to present acceptable results through the prompts. In addition, the study carried out by Kasauli et al. [18] presented the need for a solid approach to requirements elicitation, while the paper presented by Jarzkebowicz and Weichbroth [19] presented results on the elicitation of non-functional requirements, carried out

 doi:10.20944/preprints202509.1860.v1

differently in the different companies studied, so that the prompts created in this experiment can be given as an alternative for both problems.

Santos et al. [16] carried out a case study on BDD's effectiveness in eliciting non-functional requirements. The experiment presented in this paper showed that adopting BDD's effectiveness extends to using LLMs when it comes to non-functional requirements, automating the process, and generating time savings. Furthermore, this study contributes to Olsson, Sentilles, and Papatheocharous [17] since this study is an empirical study carried out in a real situation.

When creating BDD, Dan North sought to mitigate the testing-related problems arising from Test-Driven Development (TDD) [1]. Using the gherkin language, BDD is adopted systematically and concisely, thus saving time for carrying out parts of the software development process, such as requirements elicitation. The use of BDD, together with LLMs, brings an approach that companies can practice to assist teams in routine activities and gain more time to carry out other activities related to software development.

7. Threats to Validity

Threats to validity are mitigated during the research to carry out a study with more excellent reliability [24]. The descriptions made by Zhou et al. were used [25] regarding types of validity.

7.1. Construction Validity

Understanding ISO/IEC 25010:2023, the BDD framework, and how LLM work were necessary for the experiment of this research to be carried out. Considering Section 2, it became possible to identify the importance of analyzing whether the elicitation of non-functional requirements through an LLM and BDD followed the security non-functional requirement set out in the ISO/IEC 25010:2023 Standard.

7.2. Internal Validity

Five researchers are involved in this study: three carried out the collection of informations regarding the ISO/IEC 25010:2023 Standard, BDD and LLM, two carried out the experiment, and, finally, the latter carried out the refinement of information throughout the article writing process.

7.3. External Validity

Statistical test approved the validity of this study, so that it is possible, through the methodological steps expressed in Section 4, to replicate this experiment. Through the statistical test carried out, the study could be made reliable.

7.4. Conclusion Validity

The researchers had no objections regarding the methodological steps used and the results obtained. This study presents the verified and validated results.

8. Conclusion

Experiment presented in this paper related the use of the BDD framework with LLMs to elicit non-functional security requirements and their sub-characteristics arising from ISO/IEC 25010:2023. It is possible to verify the integrity of the information through the Kruskal-Wallis statistical test to present solid results.

The prompt made it possible to facilitate management of the requirements elicitation process by having the prompt refined with the objective of delivering the answer needed for a functionality; for example, it integrates the process with stakeholders and AI to systematize the process in a methodological way.

BDD is used throughout the software development, from requirements elicitation to documentation. Leveraging LLMs for test automation helps stakeholders save time on tasks that can be outsourced

 doi:10.20944/preprints202509.1860.v1

to AI. It is essential to mention that using LLMs can help professionals perform their work with quality, so AI is a facilitator in the process, not the main actor.

Through this study, it is possible to advance the understanding of using LLMs to elicit non-functional requirements. The integrative cycle between the human, the system, and the process is beneficial regarding quality of the requirements elicited, helping automate part of the software development process.

As suggestions for future work, this article could be expanded and also used other evaluation metrics and other LLMs to validate in greater depth than in this experiment the use of different LLMs in eliciting non-functional requirements using BDD. A case study could be carried out in a company that used BDD and LLMs for software development to validate the results regarding pre-defined requirements elicitation prompts.

Acknowledgments: We would like to thank CAPES/BRAZIL for the financial support (Shexmo's master scholarship).

References

1. North, D. Introducing BDD. https://dannorth.net/introducing-bdd/, 2006.
2. 7, T.C.I.J.S. ISO/IEC 25010:2023. https://www.iso.org/standard/78176.html, 2023.
3. Binamungu, L.P.; Embury, S.M.; Konstantinou, N. Maintaining Behaviour Driven-Development specifications: Challenges and opportunities. In Proceedings of the 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering SANER. IEEE, 2018, pp. 175–184.
4. Sauvola, J.; Tarkoma, S.; Klemettinen, M.; Riekki, J.; Doermann, D. Future of software development with generative AI. *Automated Software Engineering* **2024**, *31*, 26.
5. Brown, T.B. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* **2020**.
6. DeepMind, G. Gemini: Advancing Language Understanding and Generation with Deep Neural Networks. *Journal of Artificial Intelligence Research* **2024**.
7. Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M.C.; Regnell, B.; Wesslén, A.; et al. *Experimentation in software engineering*; Vol. 236, Springer, 2012.
8. Karagöz, G.; Sözer, H. Reproducing failures based on semiformal failure scenario descriptions. *Software Quality Journal* **2017**, *25*, 111–129.
9. 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. https://doi.org/10.1007/978-3-319-97925-0_42.
10. April, A.; Abran, A., Maturity Models in Software Engineering. In *Software Maintenance Management: Evaluation and Continuous Improvement*; 2008; pp. 41–67. https://doi.org/10.1002/9780470258033.ch2.
11. Cucumber. Who Does What? https://cucumber.io/docs/bdd/who-does-what/, 2019.
12. 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.
13. Bruschi, S.; Xiao, L.; Kavatkar, M.; et al. Behavior Driven-Development (BDD): a case study in healthtech. In Proceedings of the Pacific NW Software Quality Conference, 2019.
14. Radford, A.; Wu, J.; Amodei, D.; Sutskever, I. Language Models are Few-Shot Learners. *arXiv preprint arXiv:2005.14165* **2019**. Acesso em: 10 ago. 2024.
15. Rajbhoj, A.; Somase, A.; Kulkarni, P.; Kulkarni, V. Accelerating Software Development Using Generative AI: ChatGPT Case Study. In Proceedings of the Proceedings of the 17th Innovations in Software Engineering Conference, 2024, pp. 1–11.
16. Santos, S.; Pimentel, T.; Rocha, F.G.; Soares, M.S. Using Behavior-Driven Development (BDD) for Non-Functional Requirements. *Software* **2024**, *3*, 271–283. https://doi.org/10.3390/software3030014.
17. Olsson, T.; Sentilles, S.; Papatheocharous, E. A Systematic Literature Review of empirical research on quality requirements. *Requirements Engineering. Springer* **2022**, *V.27*, 249–271.
18. Kasauli, R.; Knauss, E.; Horkoff, J.; Liebel, G.; de Oliveira Neto, F.G. Requirements engineering challenges and practices in large-scale agile system development. *Journal of Systems and Software* **2021**, *172*, 110851. https://doi.org/https://doi.org/10.1016/j.jss.2020.110851.
19. Jarzębowicz, A.; Weichbroth, P. A qualitative study on non-functional requirements in agile software development. *IEEE Access* **2021**, *V. 9*, 40458–40475.

 doi:10.20944/preprints202509.1860.v1

20. Kitchenham, B.A.; Dyba, T.; Jorgensen, M. Evidence-based software engineering. In Proceedings of the Proceedings. 26th International Conference on Software Engineering. IEEE, 2004, pp. 273–281.
21. Melegati, J.; Wang, X. Case survey studies in software engineering research. In Proceedings of the Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2020, pp. 1–12.
22. Petersen, K. Guidelines for Case Survey Research in Software Engineering. *Contemporary empirical methods in software engineering* **2020**, pp. 63–92.
23. Van Solingen, R.; Berghout, E.W. *The Goal/Question/Metric Method: a practical guide for quality improvement of software development*; McGraw-Hill, 1999.
24. Runeson, P.; Höst, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering. Springer* **2009**, *V.14*, 131–164.
25. Zhou, X.; Jin, Y.; Zhang, H.; Li, S.; Huang, X. A map of threats to validity of Systematic Literature Reviews in Software Engineering. In Proceedings of the 2016 23rd Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2016, pp. 153–160.