Analysis of Contributions at a Software Institute through the Introduction of a Pre-Trained Model for Requirements Classification

Flávia Oliveira*
Sidia Institute of Science and Technology
Manaus, Brazil
flavia.oliveira@sidia.com

Alay Nascimento
Sidia Institute of Science and Technology
Manaus, Brazil
alay.nascimento@sidia.com

Ana Paula Silva
Sidia Institute of Science and Technology
Manaus, Brazil
silva.ana@sidia.com

Leonardo Tiago
Sidia Institute of Science and Technology
Manaus, Brazil
leonardo.albuquerque@sidia.com

Lennon Chaves*
Sidia Institute of Science and Technology
Manaus, Brazil
lennon.chaves@sidia.com

ABSTRACT

Context: Ensuring that requirements are adequately covered in test cases represents a challenge in the software industry. Specifically, a Software Institute maintains a testing team that continuously analyzes the requirements to ensure their implementation in test cases. **Problem**: However, requirements analysis is a human-dependent process and faces a large volume of requirements received by the testing team. In addition, other activities compete with requirements analysis, requiring effort and allocation to ensure that a requirement is analyzed and incorporated into the test case. **Goal**: In order to automate the requirements analysis process, we developed a tool based on XLNet, a pre-trained model for classifying and determining if the requirement is part of the scope of the testing team. **Method**: To evaluate this tool, we conducted a study with a team of 4 members who analyze requirements, in which the participants conducted the requirements analysis manually and with the aid of the tool. The study consisted of two analyses: (1) quantitative, aimed at evaluating effectiveness (correctly classified requirements) and efficiency (requirements analysis time), and (2) qualitative, in which we developed a questionnaire to obtain feedback from the participants on the use of the tool. **Results**: In quantitative terms, the statistical tests indicated that there was no significant difference in terms of efficiency between manual and automated classification, with a p-value of 0.8824. Regarding effectiveness, a p-value of 0.0177 was obtained, however, the results showed that manual sorting is still more effective than tool-assisted sorting. Despite this, the qualitative results showed that 100% of the participants agreed that using the tool could improve their performance in the requirements analysis activity and identified positive points about its use, such as accuracy, speed of analysis, and a reduction in the effort dedicated to this activity. **Conclusions**: The results show that using the tool can bring benefits by automating the analysis and classification of requirements.

KEYWORDS

Requirements Classification, Pre-trained Models, Software Testing

*Both authors contributed equally in this research.

1 Introduction

Software testing is an essential step in the software development life cycle. It is during this step that the software is verified to meet the expected results according to the defined requirements, ensuring that it contains no defects [15, 30]. However, the effectiveness of testing depends directly on the correct identification and analysis of requirements, as they define the scope of what must be validated and, consequently, form the basis for constructing test cases [32].

Requirement analysis is a phase of testing in which software requirements are defined, analyzed, and documented [37]. This phase includes the following activities: defining the scope of the requirement, identifying and classifying the stakeholders, classifying the requirements, analyzing the requirements, and documenting the requirements [15]. One of the main challenges of this phase is the classification of requirements because, when performed correctly, it determines which testing team is responsible for validation. Failures in this activity can lead to problems such as validation by teams lacking knowledge of the requirement's scope, insufficient requirement coverage, and an increased risk of production failures due to incorrectly validated requirements [5].

Several studies have been conducted to address the challenges of the requirement classification activity using traditional Machine Learning algorithms, such as Naive Bayes [36] and Random Forest [35], as demonstrated in the studies by Fadhlurrohman et al. [16] and Surana et al. [42].

Among the most recent approaches, some authors have investigated the use of pre-trained language models for this task. For example, Kici et al. [26] explored models such as BERT [41], DistilBERT [39], ALBERT [27], and XLNet [48] to classify functional requirements, while Subahi [41] and Khan et al. [25] applied models such as BERT [41], ELECTRA [10], and RoBERTa [28] to the classification of non-functional requirements, including green software requirements. The results of these studies showed notable improvements in metrics such as accuracy and F1-score compared with traditional approaches, highlighting the potential of PLMs for automatic classification tasks. Despite these advances, few studies have focused on using Pre-trained Language Models (PLMs) to automate this activity in real-world case studies involving professionals directly engaged in requirements analysis.

In the context of a testing team responsible for performing functional tests at a software institute, requirements are received daily, and the team must analyze them to determine whether they fall within one of the team's scopes so that validation can be carried out during testing. Within the testing team, certain members, referred to as requirement analysts, are responsible for performing this activity, analyzing and classifying whether each requirement applies to the team. This requirement analysis process ensures that, if a requirement is part of the team's scope, the corresponding test cases and databases are updated, guaranteeing that it will be validated during the test execution stage. Analysts are responsible for updating the databases and requesting changes to the test cases. If a requirement is not analyzed, it will not be validated by the testing team, which may lead to project costs such as delivery delays or untested issues. Therefore, it is crucial that the process is performed both quickly and accurately.

However, given the large volume of requirements received by the team, manual execution demands significant effort and time, competing with other activities. To automate the analysis and classification of requirements, a tool was developed based on the fine-tuning of a pre-trained model, specifically, the XLNet model [48]. With this tool, requirements are automatically classified as applicable or not to the testing team, reducing the workload of team members, who can then focus only on relevant requirements, thereby optimizing time and resources.

This experience report presents the results of applying a tool for analyzing and classifying requirements within a testing team in a real industrial setting. To evaluate the tool, we conducted a study in which it was used by four testing team members responsible for performing requirements analysis and classification, using a sample of ten requirements. The study compared manual and automated approaches, focusing on the time required to perform the classification (efficiency) and the accuracy rate of correct requirement classification (effectiveness). We also conducted a quantitative analysis supported by statistical testing, and to complement our findings, participants answered a seven-question survey, which was followed by a qualitative analysis of their responses.

The remainder of this paper is organized as follows: Section 2 presents the theoretical foundation and related work. Section 3 describes the requirements classification tool and Section 4 describes the Study Design. Section 5 presents the obtained results, which are discussed in the same section. Finally, Section 7 presents the conclusions of the study.

2 Theoretical Reference

2.1 Requirements Engineering and Traceability

In Requirements Engineering (RE), requirements are documents, usually written in natural language [2], that specify what a system should do or how it should behave [8]. They play a key role in facilitating communication among stakeholders, as successful RE depends on understanding the needs of users, customers, and other parties involved [8].

In the context of a software institute, where projects often involve innovation, applied research, and multiple stakeholders, by ensuring the clarity and traceability of requirements becomes an even more challenging task [3, 12]. This difficulty arises because, during the RE process, issues such as ambiguity [24], redundancy [20], or inconsistency [17] often emerge, which can directly compromise the accurate association of requirements with their respective test scopes [49].

This association is fundamental for defining an adequate test scope, as it allows the precise identification of which functionality, module, or category each requirement belongs to [18]. With this categorization, it becomes possible to direct tests to the appropriate areas of the system, ensuring that each requirement is covered by the most suitable test cases [40]. This promotes traceability between requirements and tests, improves functional coverage, and makes the verification process more efficient and aligned with the system's objectives [11].

Conversely, the absence of this association can lead to confusion between developers and testers, resulting in incomplete or misaligned verifications with the project's objectives [23]. This directly undermines traceability, functional coverage, and the final quality of the product [34].

To mitigate these problems, it is important to adopt approaches that automate the categorization and association of requirements with their respective test scopes. Emerging research highlights that the use of Pre-trained Language Models (PLMs) has proven promising for analyzing natural language texts, such as software requirements [13, 45]. Because they are trained on large volumes of data, these models are capable of understanding textual context, classifying relevant information, and extracting patterns with greater precision [14]. Their application in Requirements Engineering has contributed to improving traceability, increasing development efficiency, and enhancing the quality of delivered systems [13, 45].

2.2 Pre-Trained Language Models

In general, pre-trained models are algorithms that have already undergone an initial learning phase using large volumes of data [19, 47]. This phase enables the model to learn general linguistic representations, such as syntax, semantics, and contextual relationships between words [19, 47].

These models employ the transfer learning technique [22, 29], in which a model previously trained on a large dataset is later refined (through a process known as fine-tuning) using a smaller and more specific dataset targeted to particular tasks [21, 22]. This approach allows the reuse of previously acquired knowledge, resulting in more efficient training and improved performance [21, 29].

With the advancement of transformer-based architectures, which use attention mechanisms to capture contextual relationships between words, pre-trained models have become even more effective for transfer learning [44]. Models such as BERT [14], GPT [1], and T5 [33] have demonstrated significant results across various Natural Language Processing (NLP) tasks, highlighting the potential of this approach for text analysis and understanding [1, 14, 33].

The main tasks addressed by Pre-trained Language Models (PLMs) include text classification, question answering, named entity recognition (NER), text generation, automatic translation, summarization, and dialogue construction [9, 38]. These capabilities have been widely explored in diverse domains such as healthcare [46], law [6], and finance [7], and more recently, in Requirements Engineering (RE) [25, 26, 41]. In this last context, the application of PLMs extends beyond simple textual analysis—enabling the automation of critical tasks and improving the quality of artifacts produced throughout the software development life cycle [43].

2.3 Pre-Trained Models for Requirements Classification

Building on the consolidation of pre-trained models in the NLP field, several studies have demonstrated their effectiveness in identifying types of requirements and quality attributes in specification documents. For example, Khan et al. [25] proposed a transfer learning–based approach using XLNet [48] for the identification and classification of Non-functional Requirements (NFRs) according to quality attributes. The model outperformed others, such as BERT [14], ELECTRA [10], and DistilBERT [39], achieving an F1-score higher than 0.91. The proposed approach also showed significant gains in practical applications—for instance, a financial institution reported improvements in the stability and usability of its application after applying the model to analyze quality requirements.

Similarly, Kici et al. [26] conducted a comparative study of transformer based models, including BERT, RoBERTa [28], DistilBERT [39], ALBERT [27], and XLNet [48], for the task of classifying requirements in Software Requirements Specification (SRS) documents. Their work highlights the feasibility of using pre-trained models for textual classification in software engineering, even without domain-specific data.

Subahi [41], in turn, proposed a solution focused on sustainability in requirements engineering, using the BERT model to map non-functional requirements to sustainability dimensions (eco-technical and socioeconomic). The study reinforces the importance of fine-tuning with task-specific data to achieve strong performance in specialized domains and demonstrates how combining NLP and requirements engineering can support more sustainable design decisions from the early stages of the software life cycle.

The approaches proposed in these studies achieved performance metrics such as accuracy and F1-score above 80%, reinforcing the potential of pre-trained language models to support the automatic classification of requirements, whether by type or quality attribute. Collectively, these works demonstrate that, with appropriate adjustments, such models are capable of effectively processing natural language texts and generating consistent classifications.

In contrast to these approaches, the present work aims to evaluate the practical application of a pre-trained model in the task of classifying requirements using real world data. The objective is to investigate the feasibility of this method within an organizational context and to identify its potential limitations.

3 A Tool for Automated Requirement Classification

3.1 Contextualization

The testing team in which this research was conducted is organized by work front. For example, one group is responsible for analyzing requirements that arrive to be validated in tests, another group implements test cases, another group manages defects, and so forth. Thus, the testing team has several groups specializing in specific tasks. Our article describes only the participation of the group that analyzed the requirements. The requirements refer to carrier features implemented in software projects embedded in smartphones or tablets. Thus, the requirements are constantly updated according to the client's needs.

Regarding the nature of the requirements, the testing team constantly deals with changing demands. We identified two main types of complexity: (1) low complexity, associated with similar and routine requirements (e.g., recurring app version updates), and (2) high complexity, associated with disruptive and unprecedented requirements (e.g., the inclusion of an AI-based functionality). It is also important to highlight that, within the context of the team and the projects assessed by the testing team, once a high-complexity requirement becomes familiar to the team, it becomes known and is therefore considered low complexity. The occurrence of high-complexity requirements is quite rare in the team, as they only arise when there is a major change in the business rules of the project.

3.2 Description and Features

To achieve the goal of automating the analysis and classification of requirements, a tool was developed based on a pre-trained and fine-tuned version of the XLNet model [48]. This tool receives the title and description of a requirement as input, processes the data, and outputs its classification. The tool is available online[1] and operates internally within the Software Institute. However, due to a non-disclosure agreement, we cannot provide the source code or implementation details.

XLNet [48] is a pre-trained language model based on the transformer architecture. It was selected after conducting a viability study that compared different PLMs, including BERT [14] and ELECTRA [10]. In our context, XLNet demonstrated superior performance, achieving 93.16% AUC (Macro), while BERT reached 91.28% and ELECTRA 90.17% in the task of classifying requirements within the testing team evaluated in this study. The details about this results are described in our previous work [31].

3.3 Structure of Inputs and Outputs

The requirements received by the testing team contain various fields of information; however, for input to the tool, only the following text-based fields are used:

- **Title**: A brief summary of the requirement.
- **Description**: A detailed explanation of the requirement, including all information that needs to be validated during test execution.

[1]Due to non-disclosure agreement, the tool is not available for external users. It works only into the institute.

The tool then produces an output in text format that represents the classification assigned to the submitted requirement. In this tool, two classification categories are available:

- **Applicable**: The requirement is applicable to the scope of the testing team. The requirement falls within the testing scope; therefore, the testing team must analyze it to ensure that it is properly covered by the test cases.
- **Not Applicable**: The requirement is not applicable to the scope of the testing team. In this case, another technical team is responsible for analyzing the requirement and ensuring its coverage in the corresponding test cases.

3.4 Tool Usage

Figure 1 illustrates the steps involved in the interaction between a user and the tool to perform the automatic classification of requirements. The step-by-step process of this interaction is as follows:

(1) The user enters the title and description of the requirement into the tool's text input field.
(2) The tool sends the text to the XLNet model [48] for inference, where the model determines whether the requirement is applicable to the testing team's scope.
(3) The model performs the inference.
(4) The tool displays the model's output to the user.

In the next section, we discuss the details of this study's design, based on the tool described above.

4 Study Design

In this section, we describe how the study was conducted, presenting its planning, instruments, and execution process.

4.1 Planning

4.1.1 Goal. We defined the goal of this study using the goal definition template of the Goal Question Metric (GQM) [4] paradigm as follows:

"Analyze **the process of using an automated tool to classify requirements compared to the manual process** With the purpose of **evaluating** With respect to **efficiency and effectiveness** From the **researchers** point of view In the context of **practitioners from a testing team in a Software Institute**".

4.1.2 Research Questions. To achieve our goal, we aim to answer the following research questions:

RQ1: How effective and efficient is the use of a tool for automatically classifying requirements?

RQ2: What are the perceptions of testing team members regarding the use of a tool that automatically classifies requirements?

4.1.3 Context. The context of this study involves professionals from a testing team working at a Software Institute. The team receives requirements on a daily basis and must analyze and classify them in two categories (applicable and not applicable). The team is composed exclusively of software testing professionals, with no additional roles such as software developers, scrum masters, or designers. Their main activities include analyzing new incoming requirements, creating or updating test cases to ensure coverage, executing test cases, and reporting defects.

4.1.4 Variables. For this study, we have defined the following independent and dependent variables:

- **Independent Variables**: The method of requirement classification, which in this study was conducted in two ways: manually and automatically.
- **Dependent Variables**:
  - **Efficiency** Indicator, computed by the ratio between the number of classified requirements and the time spent on classification. The unit of analysis for this variable is requirements per minute.
    Efficiency is computed according to the equation below:
    $$\text{Efficiency} = \frac{\text{Total of Requirements Classified}}{\text{Total of Time Spent for Classification}} \tag{1}$$
  - **Effectiveness** Indicator, computed by the ratio between the total number of correctly classified requirements and the total number of requirements used for classification. In this way, effectiveness is computed as follows:
    $$\text{Effectiveness} = \frac{\text{Total of Requirements Classified Correctly}}{\text{Total of Requirements Classified}} \tag{2}$$

4.1.5 Hypotheses. For this study, we have the following hypotheses:

- **Null Hypothesis (H_{01}):** There is no difference in the classification of requirements manually and using a tool regarding the **efficiency** indicator.
- **Alternative Hypothesis (H_{A1}):** There is a difference in the classification of requirements manually and using a tool regarding the **efficiency** indicator.
- **Null Hypothesis (H_{02}):** There is no difference in the classification of requirements manually and using a tool regarding the **effectiveness** indicator.
- **Alternative Hypothesis (H_{A2}):** There is a difference in the classification of requirements manually and using a tool regarding the **effectiveness** indicator.

4.1.6 Participant Selection. The participants selected for the study were software testers responsible for analyzing and classifying requirements within the testing team. In total, four participants took part in the study. The sampling method adopted was non-probabilistic and based on convenience, as only members of the testing team who perform requirement analysis and classification were included. Furthermore, all participants had more than one year of experience with the activity of requirements classification.

4.2 Instrumentation

For this study, the following instruments were developed:

- **Informed Consent Form (ICF)**: To ensure the ethical integrity of the research, participants were provided with a consent form through which they agreed to take part in the study. In addition, the Software Institute has an ethics committee that ensures all research follows established ethical guidelines. Thus, before any study is conducted, internal approval is required to confirm that no ethical standards are violated.
- **Presentation**: A slide presentation designed to explain to participants how the study would be conducted and its objectives.

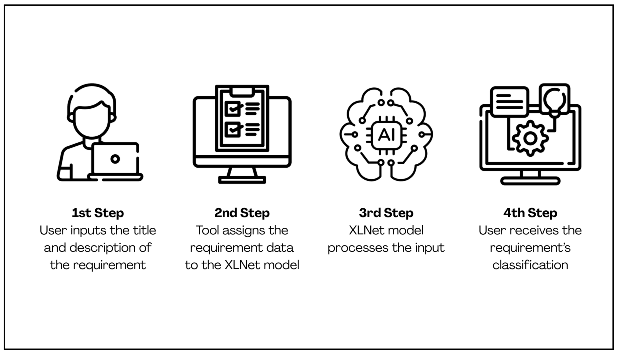
1st Step
User inputs the title and description of the requirement
2nd Step
Tool assigns the requirement data to the XLNet model
AI
3rd Step
XLNet model processes the input
4th Step
User receives the requirement's classification

Figure 1: Process of the user interacting with the system

- **Spreadsheet with requirements**: A spreadsheet containing ten requirements to be analyzed both manually and using the tool. The spreadsheet included fields for participants to record the start and end times of each analysis, as well as a field to classify each requirement as "Applicable" or "Not Applicable".
- **Post-experiment form**: A form used for the final evaluation of participants' intention to use the tool. It consisted of five open-ended questions and two statements rated on a five-point Likert scale to assess participants' level of agreement.
- **Notebook**: Each participant used a notebook with Wi-Fi access to connect to the internet and interact with the requirement classification tool.

4.3 Execution

4.3.1 Pilot Study. First, a pilot study was conducted to evaluate the quality of the research instruments. The pilot study was carried out with a tester responsible for requirements analysis during test case development. Since this tester possessed the relevant expertise, the participant was considered representative for assessing the study's instruments.

4.3.2 Study Implementation. After the pilot study, a meeting room was scheduled for all participants. All invited participants attended the session, bringing their own notebooks used for their regular work activities. Following this preparation, the study was conducted as described in Table 1. In total, the study lasted 85 minutes.

4.3.3 Analysis and Interpretation. To support this study, we conducted two types of analysis: (1) quantitative and (2) qualitative. The quantitative analysis involved measuring efficiency and effectiveness indicators. First, we tested the normality of the data using the Shapiro–Wilk test, followed by a statistical test. We then calculated the p-value and analyzed the results based on the proposed null and alternative hypotheses. Subsequently, we carried out a qualitative analysis based on participant feedback, consolidating the main points raised by the participants into the study's research findings.

5 Results

5.1 Quantitative Analysis

To address **RQ1**, we conducted a quantitative analysis. First, we collected all the responses entered in the requirements spreadsheet and then calculated the efficiency and effectiveness metrics.

Table 2 presents a summary of the distribution of efficiency and effectiveness for each participant. Figure 2 shows the effectiveness boxplot comparing manual and automated classifications, while Figure 3 displays the efficiency boxplot, likewise comparing manual and automated classifications.

Regarding the **effectiveness** indicator, the boxplot in Figure 2 shows that the manual classification process is more effective than the automated classification process, which is only represented by a "line" in the boxplot, indicating the values are concentrated in only one value. This result indicates that, for the requirements selected in this study, the tool was not able to correctly classify the requirements for the testing team. To better understand these results and obtain statistical validation, we conducted a statistical test. First, we applied the Shapiro–Wilk test, which indicated that the data did not follow a normal distribution. Consequently, we proceeded with a non-parametric test for independent samples. The Mann–Whitney test, performed with a significance level of 0.05, yielded a p-value of 0.0177, confirming a statistically significant difference in effectiveness between manual and automated classification using the tool. Thus, we conclude that manual classification was significantly more effective than automated classification and that, statistically, the tool did not contribute to improving the effectiveness of the classification process.

Regarding **efficiency**, Figure 3 shows that the manual classification process presents a visual distribution similar to that of the automated classification. To confirm this observation, we conducted a statistical analysis. First, we verified whether the efficiency data followed a normal distribution using the Shapiro–Wilk test, which indicated that it did not. Consequently, we applied the non-parametric Mann–Whitney test with a significance level of 0.05. The test yielded a p-value of 0.8824, indicating no statistically significant difference in efficiency between the manual and automated classification processes. Therefore, we conclude that the use of the tool does not affect the speed of the classification process, that is, the number of requirements analyzed per minute remains the same regardless of whether the tool is used.

Table 1: Table with Activities Performed in the Study

Activities Performed	Objective	Duration
1. Completion of the ICF	All participants read the ICF and agreed to participate in the study.	5 min
2. Presentation	The facilitator, whose role was played by one of the researchers explained on a slide the next steps of how the study would be conducted.	10 min
3. Manual Execution	In this step, the participants manually analyzed all 10 requirements that were present in the requirements spreadsheet, recording the start and end time of the analysis, and also informing the classification of the requirement: "Applicable" or "Not Applicable".	25 min
4. Automated Execution	Similar to the previous step, the participants had to record the start and end time of the analysis, but this process now involved the support of the classification tool.	25 min
5. Post-Experiment Form	After the experiments, the participant was invited to fill out the post-experiment form in order to provide feedback about their usage experience.	20 min

Table 2: Efficiency and Effectiveness by Participant

	Efficiency - Requirements per Minute		Effectiveness	
Participant	**Manual**	**Automated**	**Manual**	**Automated**
P1	5.00	3.33	0.80	0.70
P2	1.67	1.67	1.00	0.70
P3	2.00	1.43	1.00	0.70
P4	1.67	2.50	1.00	0.70

Considering the results obtained for the efficiency and effectiveness indicators, and following the statistical analyses, we cannot reject the H_{01} hypothesis and, therefore, cannot draw conclusions regarding the efficiency indicator. However, we can reject the H_{02} hypothesis and accept the H_{A2} hypothesis concerning the effectiveness indicator. Although we rejected the null hypothesis for effectiveness, the results indicate that automated requirement classification is less effective than manual classification, as the effectiveness metric decreases when using the automated process. Furthermore, there is no significant difference in efficiency between the two classification methods, as confirmed by the statistical test results for this indicator.

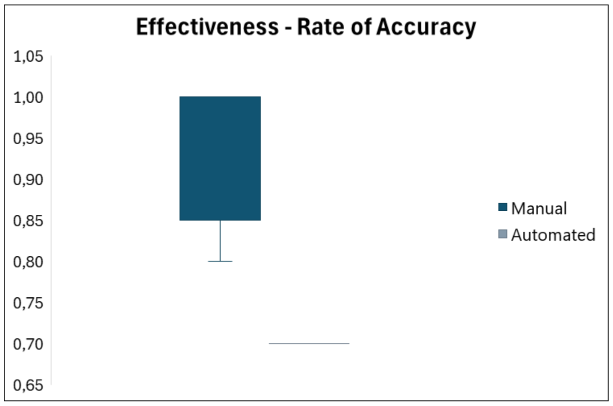

Figure 2: Boxplot of Effectiveness using Manual and Automated Classification

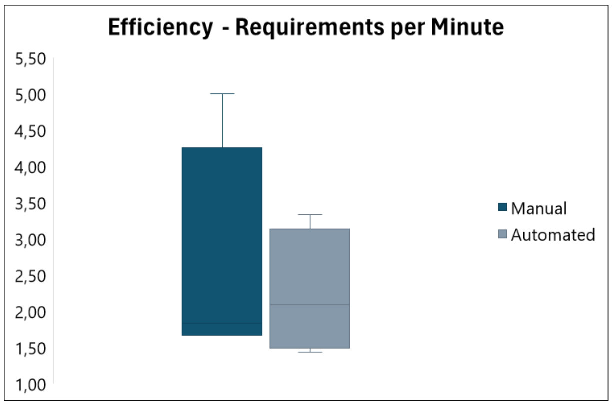

Figure 3: Boxplot of Efficiency using Manual and Automated Classification

5.2 Qualitative Analysis

After the experiment, participants completed a feedback form on the use of the tool for analyzing and classifying requirements. This qualitative feedback was then used to answer **RQ2**.

5.2.1 Experience of Use: The feedback comparing the use of the tool with the manual process was positive. Participants described the experience as good and noted that using the tool was pleasant, simple, and particularly useful when dealing with complex requirements. As stated by P2: "*It was a good experience. The result of the analysis by the tool was almost 100% correct...*" and complemented by P4: "*Pleasant... The use of the tool helps to simplify the process of identifying requirements, being specially useful in cases of complex requirements*".

5.2.2 Contributions of the Tool: Regarding the potential contributions of the tool to the team, participants stated that it could assist in the analysis process by performing a preliminary classification of requirements. They also suggested that the tool could generate a statement about the requirement's impact on the team, which would then be reviewed by the members responsible for the activity. After this review, the team could update their database with the new requirement and subsequently submit the corresponding

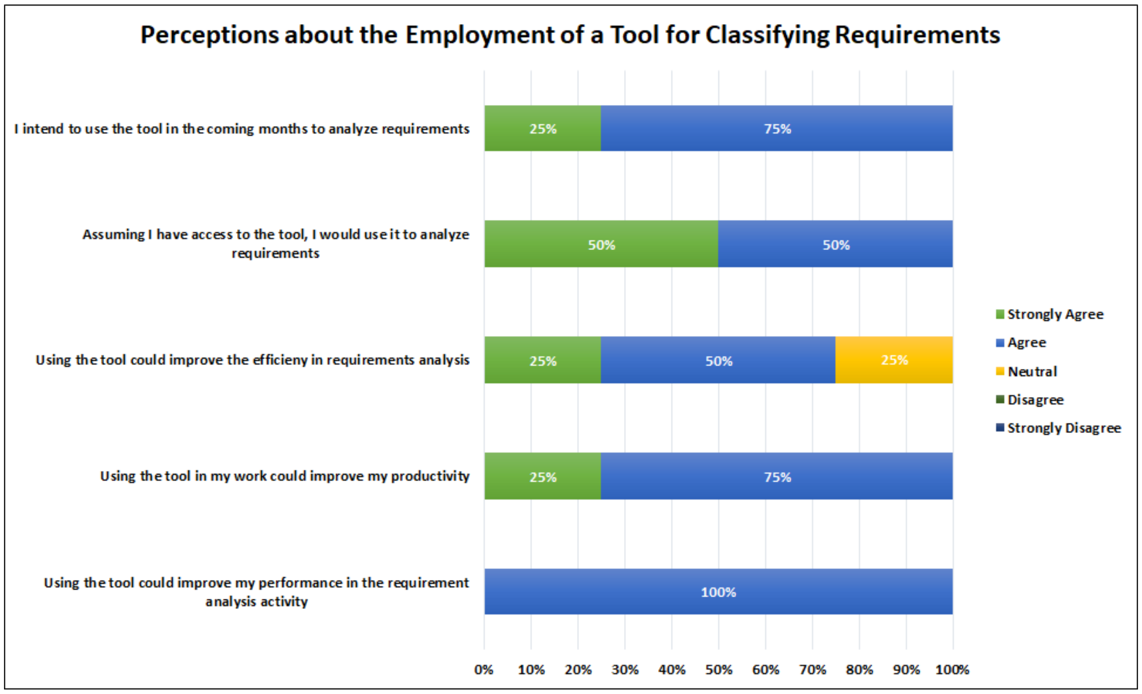

Figure 4: Participants' Perceptions on the Use of the Tool for Requirements Classification

changes to the test cases, as stated by P3: "*The tool can help providing a verdict in advance for each requirement, so the members of the requirement analysis team can follow up with the remaining tasks, such as updating the database, communicating the team and etc.*". Participants perceived another contribution that can come from a future version of the tool, which is to indicate the test cases that are impacted by the requirement, as mentioned by P2: "*I believe that it can help by performing the analysis in advance, then classifying and, in the future, by indicating the impacted test cases*".

5.2.3 Positive Aspects: Participants identified the following positive aspects of the tool during the classification process: precision, speed, coherent analysis, reduced effort for requirements not yet applied, and the ability to provide faster analyses for complex requirements that would otherwise require more time to analyze manually. As mentioned by P2: "*I found the tool to be precise and very quick...*" e complemented by P4: "*The analysis of complex requirements can become easier, thus allowing for quicker answers*".

5.2.4 Intention of Use: To verify the participants' intention to use the tool, Figure 4 shows the participants' consolidated answers, which were scored by the participants in a 5-point Likert scale. As can be seen in Figure 4, 100% of participants agree that the tool can improve the performance of requirement analysis. When questioned about the possible increase in productivity, 75% of participants agreed and 25% strongly agreed that the tool can increase the productivity of requirement analysis. When questioned if the use of the tool can improve the effectiveness of requirement analysis, 50% of participants agreed with the statement, 25% strongly agreed and 25% were neutral. Regarding the use of the tool for requirement analysis, 50% agree and 50% strongly agree that they would use the tool. In regards to the sentence "*I intend to use the tool in the following months to analyse requirements*", 75% agree and 25% strongly agree.

6 Threats and Limitations of The Study

It is important to highlight a few threats and limitations of this study:

- The study was conducted within a software institute that operates under confidentiality policies and a specific development environment; therefore, the findings cannot be generalized to other testing teams.
- The study involved only four participants, all of whom were members of the requirements analysis team; thus, they represent a small but relevant sample for the study.
- The requirements analyzed were specific to the institute in which the study was conducted.
- A total of ten requirements were considered in the analysis; consequently, the measured execution time may vary for other sets of requirements.

7 Conclusions

This paper reports on the application of a tool based on a pre-trained model (XLNet) [48] to support the requirement classification process within a testing team at a Software Institute. To this end, we conducted a study with four participants responsible for identifying if the requirement is within the team's scope. The experiment was structured so that participants first performed manual classification of the requirements and then used the automated requirements classification tool. Through this study, we collected data on both manual and automated classification times, as well as the corresponding classification results. To complement the analysis, participants also provided feedback through a post-experiment questionnaire.

We analyzed the collected data using both quantitative and qualitative approaches. In the quantitative analysis, we evaluated efficiency and effectiveness indicators and found that: (1) with respect to efficiency, there was no statistically significant difference between manual and automated classification using the tool, leading us to retain the null hypothesis H_{01}; and (2) with respect to effectiveness, we obtained a p-value of 0.0177, allowing us to reject the null hypothesis H_{02}, although the results indicated that manual classification was more effective than automated classification. Therefore, in response to **RQ1**, we conclude that the tool achieves efficiency comparable to the manual classification process but lower effectiveness.

The qualitative analysis, addressing **RQ2**, revealed that participants viewed the tool positively, as a means to support the classification of complex requirements, facilitate early classification, increase agility, and reduce effort in cases where the requirement is not applicable. Moreover, participants expressed that the tool could enhance the performance and productivity of the requirements classification activity and indicated their intention to use it in future requirements analysis tasks.

For future work, in addition to verifying whether a requirement is applicable to the testing team, we plan to investigate the automatic identification of test cases that should be updated based on each requirement, aiming to establish automated traceability between requirements and test cases. Further studies are also needed to explore the lessons learned from applying this new approach.

ACKNOWLEDGMENTS

This paper is a result of the Research, Development & Innovation Project (ASTRO) performed at Sidia Institute of Science and Technology sponsored by Samsung Eletrônica da Amazônia Ltda., using resources under terms of Federal Law No. 8.387/1991, by having its disclosure and publicity in accordance with art. 39 of Decree No. 10.521/2020. The authors extend their gratitude to the Software Engineering Laboratory for its contribution to the infrastructure.

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Waad Alhoshan, Alessio Ferrari, and Liping Zhao. 2023. Zero-shot learning for requirements classification: An exploratory study. *Information and Software Technology* 159 (2023), 107202.

[3] RICARDO CORREIA BARROS. 2018. *A Importância da Gestão de Requisitos para Projetos de Desenvolvimento de Software.* Ph. D. Dissertation. BS Thesis, Campus Sao Paulo (IFSP), Instituto Federal de Educaçao, Ciência e ....

[4] Victor R Basili. 1994. Goal, question, metric paradigm. *Encyclopedia of software engineering* 1 (1994), 528–532.

[5] Elizabeth Bjarnason, Per Runeson, Markus Borg, Michael Unterkalmsteiner, Emelie Engström, Björn Regnell, Giedre Sabaliauskaite, Annabella Loconsole, Tony Gorschek, and Robert Feldt. 2014. Challenges and practices in aligning requirements with verification and validation: a case study of six companies. *Empirical software engineering* 19 (2014), 1809–1855.

[6] Ilias Chalkidis, Manos Fergadiotis, Prodromos Malakasiotis, Nikolaos Aletras, and Ion Androutsopoulos. 2020. LEGAL-BERT: The muppets straight out of law school. *arXiv preprint arXiv:2010.02559* (2020).

[7] Zhiyu Zoey Chen, Jing Ma, Xinlu Zhang, Nan Hao, An Yan, Armineh Nourbakhsh, Xianjun Yang, Julian McAuley, Linda Petzold, and William Yang Wang. 2024. A survey on large language models for critical societal domains: Finance, healthcare, and law. *arXiv preprint arXiv:2405.01769* (2024).

[8] Betty H.C. Cheng and Joanne M. Atlee. 2007. Research Directions in Requirements Engineering. In *Future of Software Engineering (FOSE '07)*. 285–303. doi:10.1109/FOSE.2007.17

[9] Ha Na Cho, Tae Joon Jun, Young-Hak Kim, Heejun Kang, Imjin Ahn, Hansle Gwon, Yunha Kim, Jiahn Seo, Heejung Choi, Minkyoung Kim, et al. 2024. Task-Specific Transformer-Based Language Models in Health Care: Scoping Review. *JMIR Medical Informatics* 12 (2024), e49724.

[10] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555* (2020).

[11] Aline Gomes Cordeiro and André Luís Policani Freitas. 2011. Priorização de requisitos e avaliação da qualidade de software segundo a percepção dos usuários. *Ciência da Informação* 40, 2 (2011).

[12] Gabriela Oliveira da Trindade and Márcia Lucena. 2016. Rastreabilidade de Requisitos em Metodologias Ágeis: um Estudo Exploratório. In *Simpósio Brasileiro de Sistemas de Informação (SBSI)*. SBC, 478–485.

[13] Eliane Maria De Bortoli Fávero and Dalcimar Casanova. 2021. BERT_SE: A Pre-trained Language Representation Model for Software Engineering. *arXiv e-prints* (2021), arXiv–2112.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186.

[15] Asmaa H Elsaid, Rashed K Salem, and Hatem M Abdul-kader. 2015. Automatic framework for requirement analysis phase. In *2015 Tenth International Conference on Computer Engineering & Systems (ICCES)*. IEEE, 197–203.

[16] Daffa Hilmy Fadhlurrohman, Mira Kania Sabariah, Muhammad Johan Alibasa, and Jati Hiliamsyah Husen. 2023. Naive Bayes Classification Model for Precondition-Postcondition in Software Requirements. In *2023 International Conference on Data Science and Its Applications (ICoDSA)*. IEEE, 123–128.

[17] Irit Hadar, Anna Zamansky, and Daniel M Berry. 2019. The inconsistency between theory and practice in managing inconsistency in requirements engineering. *Empirical Software Engineering* 24, 6 (2019), 3972–4005.

[18] Ines Hajri, Arda Goknil, Fabrizio Pastore, and Lionel C Briand. 2020. Automating system test case classification and prioritization for use case-driven testing in product lines. *Empirical Software Engineering* 25 (2020), 3711–3769.

[19] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. 2021. Pre-trained models: Past, present and future. *AI Open* 2 (2021), 225–250.

[20] Elisabeth Henkel, Nico Hauff, Lena Funk, Vincent Langenfeld, and Andreas Podelski. 2024. Scalable Redundancy Detection for Real-Time Requirements. In *2024 IEEE 32nd International Requirements Engineering Conference (RE)*. IEEE, 193–204.

[21] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *International conference on machine learning*. PMLR, 2790–2799.

[22] Shenson Joseph and Herat Joshi. 2024. ULMFiT: Universal Language Model Fine-Tuning for Text Classification. *International Journal of Advanced Medical Sciences and Technology (IJAMST)* 4 (2024).

[23] Marcos Kalinowski, Michael Felderer, Tayana Conte, Rodrigo Spínola, Rafael Prikladnicki, Dietmar Winkler, Daniel Méndez Fernández, and Stefan Wagner. 2016. Preventing incomplete/hidden requirements: reflections on survey data from Austria and Brazil. In *Software Quality. The Future of Systems-and Software Development: 8th International Conference, SWQD 2016, Vienna, Austria, January 18-21, 2016, Proceedings 8*. Springer, 63–78.

[24] Erik Kamsties. 2005. Understanding ambiguity in requirements engineering. *Engineering and Managing Software Requirements* (2005), 245–266.

[25] Muhammad Amin Khan, Muhammad Sohail Khan, Inayat Khan, Shafiq Ahmad, and Shamsul Huda. 2023. Non functional requirements identification and classification using transfer learning model. *IEEE Access* 11 (2023), 74997–75005.

[26] Derya Kici, Aysun Bozanta, Mucahit Cevik, Devang Parikh, and Ayşe Başar. 2021. Text classification on software requirements specifications using transformer models. In *Proceedings of the 31st Annual International Conference on Computer Science and Software Engineering*. 163–172.

[27] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).

[28] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[29] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. 2023. Recent

advances in natural language processing via large pre-trained language models: A survey. *Comput. Surveys* 56, 2 (2023), 1–40.

[30] Glenford J Myers, Corey Sandler, and Tom Badgett. 2011. *The art of software testing.* John Wiley & Sons.

[31] Alay Nascimento, Flávia Oliveira, Leonardo Tiago, and Lennon Chaves. 2025. Who Should Test the Requirement? A Comparative Study on Requirements Classification for Assigning Test Teams using the Pre-Trained Models. In *Anais do XXXIX Simpósio Brasileiro de Engenharia de Software* (Recife/PE). SBC, Porto Alegre, RS, Brasil, 671–677. doi:10.5753/sbes.2025.11009

[32] Roger S Pressman. 2005. *Software engineering: a practitioner's approach.* Palgrave macmillan.

[33] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research* 21, 140 (2020), 1–67.

[34] Patrick Rempel and Parick Mäder. 2016. Preventing defects: The impact of requirements traceability completeness on software quality. *IEEE Transactions on Software Engineering* 43, 8 (2016), 777–797.

[35] Steven J Rigatti. 2017. Random forest. *Journal of Insurance Medicine* 47, 1 (2017), 31–39.

[36] Irina Rish et al. 2001. An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, Vol. 3. Seattle, USA, 41–46.

[37] Luiz Gustavo Mendes Rodrigues. 2006. *Um modelo de avaliação dos requisitos no processo de desenvolvimento de software.* Ph. D. Dissertation. Tese de Doutorado, Universidade Estadual de Campinas, Instituto de Computação.

[38] Sebastian Ruder, Matthew E Peters, Swabha Swayamdipta, and Thomas Wolf. 2019. Transfer learning in natural language processing. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Tutorials.* 15–18.

[39] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).

[40] Miriam Sayão and Julio Cesar Sampaio do Prado Leite. 2006. Rastreabilidade de requisitos. *RITA* 13, 1 (2006), 57–86.

[41] Ahmad F Subahi. 2023. Bert-based approach for greening software requirements engineering through non-functional requirements. *IEEE Access* 11 (2023), 103001–103013.

[42] Chetan Surana Rajender Kumar Surana, Dipesh B Gupta, Sahana P Shankar, et al. 2019. Intelligent chatbot for requirements elicitation and classification. In *2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT).* IEEE, 866–870.

[43] Rosalia Tufano, Luca Pascarella, and Gabriele Bavota. 2023. Automating code-related tasks through transformers: The impact of pre-training. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE).* IEEE, 2425–2437.

[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[45] Julian von der Mosel, Alexander Trautsch, and Steffen Herbold. 2023. On the Validity of Pre-Trained Transformers for Natural Language Processing in the Software Engineering Domain. *IEEE Transactions on Software Engineering* 49, 4 (2023), 1487–1507. doi:10.1109/TSE.2022.3178469

[46] Dandan Wang and Shiqing Zhang. 2024. Large language models in medical and healthcare fields: applications, advances, and challenges. *Artificial Intelligence Review* 57, 11 (2024), 299.

[47] Haifeng Wang, Jiwei Li, Hua Wu, Eduard Hovy, and Yu Sun. 2023. Pre-trained language models and their applications. *Engineering* 25 (2023), 51–65.

[48] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems* 32 (2019).

[49] Zhenzhen Yang, Rubing Huang, Chenhui Cui, Nan Niu, and Dave Towey. 2025. Requirements-Based Test Generation: A Comprehensive Survey. *arXiv preprint arXiv:2505.02015* (2025).