Classifying BDD Software Tests Using Machine Learning and NLP Techniques

David Ferreira Brandão[1], **Cleyton Mário de Oliveira Rodrigues** [1], **Wylliams Barbosa Santos**[1]

[1]Universidade de Pernambuco – (UPE)
50720-001 – RECIFE – PE – Brazil

{davfb.ferreira, cleyton.rodrigues}@gmail.com, wbs@upe.br

Abstract. *The increasing complexity and volume of Behavior-Driven Development (BDD) test scripts have made manual validation time-consuming, error-prone, and difficult to standardize. To address these challenges, this study proposes an automated approach for analyzing BDD scenarios through a hybrid solution that combines Natural Language Processing (NLP) and Machine Learning (ML) techniques. The solution consists of two components: an NLP-based validator that detects structural and linguistic inconsistencies in Gherkin steps, and a supervised ML classifier that assigns each step to one of three functional roles: Precondition, Action, or Expected Result independent of the original Gherkin keywords. The methodology includes the development and evaluation of a classification model trained on a labeled dataset of 1,500 synthetic BDD steps. Performance was validated using accuracy, precision, recall, and F1-score metrics, and further confirmed with real-world test data. The proposed system provides near-instant feedback per step, enabling efficient integration into real-time development workflows. This research demonstrates the feasibility of combining rule-based validation and machine learning classification to improve the quality, consistency, and maintainability of BDD test artifacts.*

1. Introduction

The software development life cycle (SDLC) is a structured process that guides software development from conception to implementation and maintenance. It consists of several stages, each with specific activities aimed at building quality software. These stages include Planning, Requirements Analysis, Design, Development (Implementation), Testing, Deployment, and Maintenance [Asha et al. 2023].

Among these stages, the testing phase plays a crucial role in verifying and validating the software, ensuring that it meets specified requirements and is free of defects. Testing is essential to the overall quality of the final product, and there are various methodologies to streamline this process. One such methodology is Behavior Driven Development (BDD), introduced by Dan North [North 2006a], which has become increasingly popular for its dual role as both a documentation method and a way to create automated test artifacts. BDD promotes collaboration between developers, testers and business stakeholders by providing a common language that bridges the gap between technical implementation and business requirements [OneDayTesting 2019]. Furthermore, the emphasis on clarity and continuous feedback in BDD aligns closely with the agile and refactoring principles highlighted by Fowler [Fowler 1999], and is increasingly relevant as AI-based approaches begin to intersect with software testing [Russell and Norvig 2020].

At the core of BDD is the Gherkin language syntax, originally developed by Aslak Hellesøy as part of the Cucumber project, which provides a structured format for defining system behavior in easily understandable scenarios [Hellesøy et al. 2017]. Gherkin enables stakeholders to write test cases in plain English (or other supported natural languages), ensuring that the intent of a feature is clearly communicated and accessible to all, including non-technical team members. The Gherkin syntax revolves around three main elements: ”Given”(preconditions), ”When”(actions), and ”Then”(expected results). These elements are the backbone of the test scenarios and provide a clear and consistent way to describe system behavior. [Chandorkar et al. 2022, Hellesøy et al. 2017]

Expressing requirements in Gherkin syntax using the BDD methodology ensures that every aspect of the software functionality is covered in a language that can be understood and verified by all team members. This shared understanding helps improve collaboration, reduces misunderstandings, and ensures that the development process is aligned with business goals. Furthermore, BDD allows for the direct development of automated tests from Gherkin scenarios, making the testing process more efficient and dynamic. This alignment of business and technical perspectives helps guarantee that the delivered software meets its intended functionality [Chandorkar et al. 2022].

However, writing tests without standardization can make their maintenance challenging and increase the likelihood of errors that may negatively affect the end user. BDD addresses these concerns by providing a structured framework that enhances the understanding and maintenance of tests. Nevertheless, the effectiveness of BDD depends heavily on adherence to its rules and best practices, which can be a challenge for less experienced users [OneDayTesting 2019].

During the literature review, no studies were found that directly addressed the problem proposed in this document. However, analogous models exist that classify rules in different types of textual documents. One of these models, ”RClassify”developed an architecture to categorize rules in software requirements documents [Asha et al. 2023]. To the best of our knowledge, no existing approach integrates rule-based best practice validation and functional step classification in a unified solution for BDD scenario analysis. This work proposes the development of a new model architecture, inspirated by RClassify, and specifically tailored for use in BDD testing. To address the problem, some Machine Learning (ML) and Natural Language Processing (NLP) techniques will be applied.

Problems related to the extraction and classification of rules in natural language are widely discussed in the literature. Techniques such as NLP and ML are frequently used to address these challenges [Asha et al. 2023, Anish et al. 2022, Lafi and abdelQader 2023, Rajbhoj et al. 2023]. Given these challenges, this paper explores the following research question: ”Can the combination of NLP and ML-based approaches enhance the identification of structural and semantic patterns in BDD test scenarios, thereby improving their quality and maintainability?”

2. Objective

The primary objective of this work is to design and implement a hybrid classification system that leverages Natural Language Processing (NLP) and Machine Learning (ML) techniques to automatically analyze and classify BDD (Behavior-Driven Development) test steps written in Gherkin. This system performs both semantic validation, ensuring

adherence to best practices and structural conventions, and functional categorization of test steps into preconditions, actions, and expected results, providing a comprehensive solution for improving the quality and maintainability of BDD artifacts.

3. State of the Art

The adoption of advanced techniques in software development has led to the increasing use of methodologies such as BDD, which aims to bridge the gap between the technical development process and business needs. In BDD, software requirements are described collaboratively between developers, testers, and stakeholders using a common, easily understandable language [Farooq et al. 2023]. To facilitate this communication, the BDD technique is often used, as it allows the expected system behavior to be described in structured and standardized scenarios. This structure improves the understanding of requirements and facilitates the creation of automated tests, aligning development with the desired behavior [OneDayTesting 2019].

ML is a subfield of Artificial Intelligence (AI) that enables systems to learn patterns from data and make decisions without being explicitly programmed for each case. When combined with NLP, which focuses on the interaction between computers and human language, it becomes possible to automate the analysis and understanding of complex texts, such as BDD scenarios [Farooq et al. 2023, Taneja and Vashishtha 2022].

The use of NLP techniques for business rule extraction has been widely explored by various authors across different industries. Among the works analyzed in the literature review, notable examples include the use of NLP for data integration between different systems in the electric power industry [Xiao et al. 2023], the creation of a classifier and spell checker for low-resource languages in NLP [Mati et al. 2021], and the extraction of metadata from unstructured scientific documents in PDF format by combining NLP with Computer Vision (CV) [Boukhers and Bouabdallah 2022].

In the field of business rule classification, the literature shows a growing application of ML and Deep Learning techniques to classify these rules in requirement documents. Some examples include the application of deep learning to classify business rules from software requirements specifications [Anish et al. 2022], the use of ML to classify business rules with the aim of improving the software requirements elicitation process [Lafi and abdelQader 2023], and even the development of DocToModel, an ML model that enables the automated extraction of structured information from natural language texts [Rajbhoj et al. 2023].

As demonstrated, there are several approaches and techniques for the extraction and classification of rules in text documents. These approaches include the use of ML, Deep Learning, and NLP, and in some cases, a combination of these techniques [Ye et al. 2021], to achieve greater effectiveness in meeting the proposed objectives.

While these approaches demonstrate the effectiveness of NLP and ML techniques for textual analysis in software documentation, they focus primarily on classification or rule extraction. Few works explore the combined use of these techniques in the context of BDD testing, especially with an emphasis on enforcing best practices and improving scenario quality.

However, ensuring semantic correctness and stylistic adherence remains a manual

and error-prone task. Best practices—such as writing steps in third person, avoiding imperative tone, ensuring logical sequencing, and eliminating semantic duplication—are frequently violated, especially by less experienced users.

This gap is addressed in our work through the development of a two-part model. The first component is a rule-based NLP validator that detects violations of BDD best practices and reports them back to the user without modifying or classifying the test steps. The second component is a machine learning classifier trained to categorize each step as a precondition, action, or expected result. This classification is independent of the original keyword and is designed to detect potentially misplaced steps.

To the best of our knowledge, this is the first work that combines best-practice validation with structural classification in BDD scenarios, offering a dual-layer analysis that improves both the quality and correctness of test artifacts and can be integrated into real-world development pipelines.

4. Methodology

This work proposes an approach that combines Natural Language Processing (NLP) and Machine Learning (ML) techniques to improve the quality and correctness of test steps written in Gherkin. The methodology is structured around two independent but complementary modules: a rule-based validator for static and semantic analysis, and a supervised classifier for step categorization.

4.1. Architecture Overview

The proposed solution is composed of two main components. The first component, the **NLP Rule Validator**, performs static and semantic analysis of the test steps to verify their adherence to Gherkin best practices. While it does not classify the steps, it provides valuable feedback to the user regarding potential issues in the structure and style of the steps. This feedback ensures that the test steps follow the established conventions, improving their readability and maintainability.

The second component, the **ML Classifier**, is responsible for classifying each test step into one of three categories: Precondition, Action, or Expected Result. This classification is independent of the original Gherkin keywords (Given, When, Then), enabling the identification of potential misplacements in the test logic. By doing so, the classifier ensures that each step is categorized according to its function, regardless of the keyword used.

The final implementation integrates the outputs of both modules. The NLP Rule Validator enhances the quality of the input test steps, ensuring they follow best practices, while the ML Classifier provides functional categorization, which supports better test maintenance and validation throughout the software development process.

4.2. NLP Rule Validator Module

The NLP validator is based on a set of heuristic rules synthesized from well-established Gherkin and BDD literature, including third-person descriptive form, logical sequence of steps, avoiding redundancy, and ensuring semantic clarity [Smart 2014, Wynne and Hellesoy 2017, North 2006b]. These rules are applied to each scenario to assess its semantic and structural integrity. If a rule is violated, the tool generates a warning to the user. The rules implemented include the following:

- **Third-person validation:** Ensures steps are written in a neutral, descriptive form, avoiding imperative verbs.
- **Conjunction redundancy:** Detects multiple conjunctions like ”And”or ”But”within the same step.
- **Step sequence check:** Verifies the logical order of steps within a scenario, such as ”Given”before ”When”, and ”When”before ”Then”.
- **Excessive use of same step:** Detects multiple lines being repeated with the same step or conjunction.
- **Semantic similarity:** Computes similarity scores between steps in the same feature file to flag redundant or overly similar phrases.

The NLP module serves as a static validator and does not interact with the ML training pipeline.

4.3. ML Classification Module

The ML component classifies each Gherkin step into its corresponding category: Precondition, Action, or Expected Result. Each step is preprocessed and vectorized using a combination of TF-IDF and Word2Vec techniques and then passed to a trained classifier model. This functional classification is independent of the original Gherkin keywords (Given, When, Then), allowing the system to identify misplacements and inconsistencies in step labeling.

While simpler approaches based on regular expressions could validate syntactic patterns, the semantic nature of step classification requires a learning-based approach. Understanding the contextual role of a test step—particularly when natural language phrasing varies significantly—demands models capable of generalizing patterns from labeled data. The following supervised classifiers were evaluated:

- **BERT (fine-tuned transformer):** A transformer-based NLP model capable of learning deep contextual relationships between words. Fine-tuned for the classification task using labeled Gherkin steps.
- **CatBoost (CB):** A gradient-boosting algorithm optimized for categorical data, offering strong performance with minimal preprocessing and resistance to overfitting.
- **Long Short-Term Memory (LSTM):** A type of recurrent neural network (RNN) that captures sequential dependencies in text, well-suited for natural language tasks.
- **Multilayer Perceptron (MLP):** A feedforward neural network with one or more hidden layers, trained through backpropagation for multi-class classification.
- **Multinomial Naive Bayes (GNB):** A probabilistic classifier based on Bayes' theorem, effective for high-dimensional text classification with strong independence assumptions.
- **Random Forest (RF):** An ensemble method that constructs multiple decision trees and combines their outputs to improve classification accuracy and reduce overfitting.
- **XGBoost (XGB):** An efficient and scalable gradient-boosting algorithm that iteratively improves model accuracy by correcting previous errors.

All models were trained and evaluated using a labeled dataset of BDD test steps, manually annotated with their correct classifications. Model performance was assessed through cross-validation and metrics including accuracy, precision, recall, and F1-score.

4.4. Feedback from NLP and ML Modules

In the final implementation, both modules are executed sequentially. The user receives two complementary forms of feedback: validation warnings generated by the NLP module, which flag violations of structural or semantic rules, and classification results produced by the ML model, which indicate the functional role of each step—Precondition, Action, or Expected Result—regardless of the original keyword used. This dual feedback mechanism enables the identification of both writing inconsistencies and logical misplacements, supporting automated review and improving the clarity, correctness, and maintainability of BDD scenarios.

4.5. Type of Research

This study is classified as applied research, as it addresses a practical challenge in software testing: improving the quality of BDD scenarios written in Gherkin through automated analysis.

Its objectives are descriptive, exploring how NLP and ML techniques can be combined to perform both qualitative validation and functional classification of test steps. The NLP module applies rule-based checks to identify violations of best practices, while the ML module categorizes steps into Precondition, Action, or Expected Result.

Methodologically, the approach is mixed: the NLP component provides non-predictive, qualitative feedback, and the ML component is quantitatively assessed using accuracy, precision, recall, and F1-score. This combination ensures a comprehensive analysis of both structure and semantics in BDD tests.

4.6. Tools and Techniques

This study combines NLP and ML techniques to enhance the quality of BDD test scenarios written in Gherkin. Implemented in Python, it leverages open-source libraries for text processing, vectorization, and model development.

SpaCy was used for tokenization and semantic analysis, offering efficient tools for validating linguistic structure. Rule validation relied on custom logic to check best practices, such as third-person narrative and correct sequencing of ”Given,When,”and ”Then”steps.

Text vectorization used TF-IDF and Word2Vec. TF-IDF evaluates term relevance as follows:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

where

$$\text{TF}(t, d) = \frac{\text{Count of term } t \text{ in document } d}{\text{Total number of terms in } d}, \quad \text{IDF}(t) = \log\left(\frac{N}{\text{DF}(t)}\right)$$

Word2Vec generates dense vectors that capture semantic similarity between words. The combination improves both syntactic and contextual understanding for classification.

Models were developed using Scikit-learn (for Random Forest, Naive Bayes, XGBoost) and TensorFlow/Keras (for LSTM and BERT). This setup enabled a hybrid system for validating structure and classifying functional roles in BDD scenarios.

4.7. Model Architecture

The architecture proposed in this work consists of two independent yet complementary modules: an NLP-based rule validator and a supervised ML classifier. These modules operate in parallel, each addressing a distinct aspect of BDD test step analysis.

The **NLP validator** performs static and semantic analysis on each step using heuristic rules derived from established Gherkin and BDD best practices. It provides qualitative feedback to the user by identifying violations such as the use of imperative verbs instead of third-person descriptions, excessive conjunctions, or incorrect step sequencing. This component is rule-driven and does not interact with the machine learning pipeline.

The **ML classifier** processes preprocessed and vectorized Gherkin steps and assigns each one to a functional category: Precondition, Action, or Expected Result. This classification is based solely on the content of the step, independent of its original Gherkin keyword (Given, When, or Then), allowing the model to detect semantically misplaced steps.

The final implementation integrates outputs from both modules into a unified reporting workflow, as illustrated in Figure 1. While the NLP module flags stylistic and structural issues, the ML classifier identifies logical misalignments. Together, they provide comprehensive feedback that enhances both the quality and functional coherence of BDD scenarios.

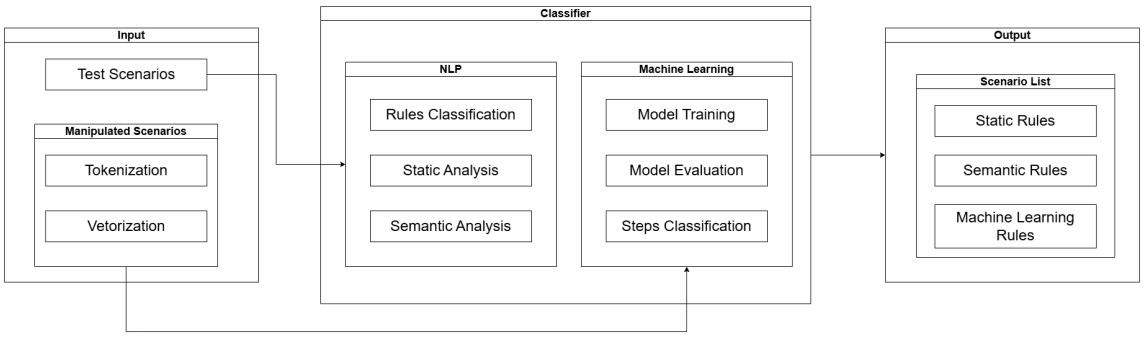

Figura 1. Proposed classification model architecture

4.8. Database

Due to confidentiality constraints, the use of real project data for training was not authorized by the sponsoring organization. To address this, a synthetic dataset was developed by the author, following guidelines provided by the company's innovation team. The dataset was designed to emulate the structure and linguistic characteristics of real BDD scenarios while omitting any proprietary information.

The resulting dataset consists of 1,500 manually labeled Gherkin steps, equally distributed across the three target categories: 500 labeled as *Precondition* (originally `Given`), 500 as *Action* (`When`), and 500 as *Expected Result* (`Then`). Care was taken

to maintain class balance and ensure that the syntactic and semantic patterns reflected real-world usage.

Although real project data was not used during training, access was granted to a real-world dataset from a global laptop development company for validation purposes. This dataset was used to assess the model's generalization capability. The performance metrics obtained on the real dataset were consistent with those from the synthetic set, suggesting that the synthetic data accurately captures the structure and semantics of BDD test steps.

4.9. Evaluation Metrics

The performance of the proposed machine learning models was quantitatively evaluated using standard classification metrics: Accuracy, Precision, Recall, F1-score, and Standard Deviation. These metrics are widely adopted for assessing multi-class classification tasks and provide comprehensive insights into the model's predictive capabilities.

Accuracy measures the proportion of correct predictions over the total number of predictions and is calculated as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP, TN, FP, and FN represent the counts of true positives, true negatives, false positives, and false negatives, respectively.

Precision evaluates the proportion of correctly predicted positive instances among all instances predicted as positive:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall indicates the proportion of actual positive instances that were correctly identified:

$$\text{Recall} = \frac{TP}{TP + FN}$$

F1-score is the harmonic mean of Precision and Recall, offering a balance between the two:

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

To provide a fair assessment across the three target classes (Precondition, Action, Expected Result), macro-averaging was used for Precision, Recall, and F1-score. This ensures that each class contributes equally to the final evaluation, regardless of sample size.

Standard Deviation was also computed to quantify the variability in model performance across different runs or folds, offering insights into model stability and robustness.

5. Results

The classification models were evaluated using the test dataset, and their performance was assessed through Accuracy, Precision, Recall and F1-score. The goal was to identify which model achieved the best results in the task of categorizing Gherkin steps.

5.1. Model Performance Comparison

Table 1 presents the performance of each model on the hold-out test set using macro-averaged evaluation metrics. Among the tested classifiers, XGBoost achieved the best balance of accuracy and F1-score, followed closely by MLP. The LSTM and CatBoost models, on the other hand, demonstrated significantly lower performance.

Tabela 1. Performance Metrics of Classification Models (Macro-Averaged)

Model	Accuracy	Precision	Recall	F1 Score
MLP	0.705	0.706	0.709	0.700
LSTM	0.322	0.107	0.333	0.162
RF	0.500	0.341	0.509	0.407
GNB	0.527	0.590	0.531	0.522
XGB	**0.727**	**0.750**	**0.726**	**0.727**
CB	0.334	0.111	0.333	0.167
BERT	0.548	0.742	0.557	0.484

5.2. Performance Analysis of Classification Models

The performance results summarized in Table 1 reveal meaningful trends in the behavior of the evaluated models. Surprisingly, despite the widespread expectation of superior performance from neural architectures like LSTM and BERT, traditional machine learning models such as XGBoost and MLP delivered the highest overall results in this specific classification task.

XGBoost stood out as the most effective model, achieving the highest scores in all four metrics: **Accuracy (0.727)**, **Precision (0.750)**, **Recall (0.726)**, and **F1 Score (0.727)**. MLP also performed well, presenting consistent values around 0.70, demonstrating that the feature representation using TF-IDF combined with Word2Vec provided a strong foundation for simpler neural architectures.

Conversely, models like LSTM and CatBoost showed underwhelming performance. Their low macro-averaged scores may be attributed to the incompatibility between their architecture assumptions and the structure of the transformed features or the small sample size used for training.

Although BERT achieved a relatively high precision score (0.742), its recall and F1 score were significantly lower (0.557 and 0.484, respectively). This indicates a tendency to favor conservative predictions—fewer positive classifications made with higher certainty—resulting in a performance that, while notable in specific contexts, was overall inferior to the more balanced results obtained by MLP and XGBoost.

5.3. Statistical Significance Analysis

To determine whether the observed performance differences between the models are statistically significant, the **Friedman test** was applied to the F1 scores of each model.

Friedman Test Results:

- **Friedman Statistic:** 21.6857
- **p-value:** 0.0006

The result indicates a statistically significant difference in the F1 performance of at least some of the models ($p < 0.05$). This suggests that the models do not perform equivalently, and further pairwise comparisons are warranted to identify which models are significantly different from each other.

5.4. Student's T-Test

To further investigate the differences between the top-performing models, **MLP** and **XGBoost**, a pairwise **Student's T-Test** was conducted on their F1 scores. These models were selected due to their consistent and superior performance across all evaluation metrics, with **XGBoost** showing the highest overall F1 score and **MLP** following closely behind.

Tabela 2. Pairwise T-Test Result Between MLP and XGBoost (F1 Scores)

Model Comparison	t-statistic	p-value
MLP × XGBoost	3.2154	0.0157

The p-value of 0.0157 indicates a statistically significant difference between the F1 scores of MLP and XGBoost. This result confirms that **XGBoost** outperforms **MLP** in terms of average F1 score, reinforcing the effectiveness of gradient-boosted trees for this classification task when combined with hybrid vectorization techniques.

6. Conclusion

This study introduced an approach for evaluating and classifying Behavior-Driven Development (BDD) test scenarios written in Gherkin, leveraging both Natural Language Processing (NLP) and Machine Learning (ML) techniques. The proposed solution addresses two critical challenges in BDD-based test automation: ensuring compliance with syntactic and semantic best practices, and accurately classifying test steps based on their functional roles.

The architecture integrates a rule-based NLP validator, which performs static analysis of the test steps to enforce structural and stylistic conventions, and a supervised ML classifier that categorizes steps as Preconditions, Actions, or Expected Results, irrespective of the initial Gherkin keywords. This dual-layered approach provides both qualitative feedback on test quality and quantitative evaluation of classification performance, ensuring a comprehensive solution for test scenario analysis.

Experimental results revealed that traditional ML models, particularly XGBoost and MLP, significantly outperformed more complex neural models, such as LSTM

and BERT. XGBoost achieved the highest performance across all macro-averaged metrics—accuracy, precision, recall, and F1-score—indicating that hybrid vectorization techniques (TF-IDF and Word2Vec), when combined with robust ensemble learners, are highly effective in this domain.

The statistical significance of these results was confirmed through the Friedman test, which revealed meaningful differences in model performance. Further pairwise T-tests demonstrated a statistically significant difference between MLP and XGBoost, with XGBoost outperforming MLP in terms of F1-score. These findings underscore the importance of effective feature engineering and the advantage of ensemble models in structured textual tasks like BDD step classification.

In summary, this approach offers a preliminary step toward automating BDD test analysis by providing a structured framework aimed at improving the correctness and maintainability of test scenarios. Future work will focus on extending the NLP validator to include auto-correction capabilities and exploring the application of transfer learning to improve the generalization of the ML classifier across diverse datasets.

Referências

Anish, P. R., Lawhatre, P., Chatterjee, R., Joshi, V., and Ghaisas, S. (2022). Automated labeling and classification of business rules from software requirement specifications. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, pages 53–54.

Asha, R., Padmalata, N., Ajim, P., Piyush, K., and Vinay, K. (2023). Rclassify: Combining nlp and ml to classify rules. In *2023 IEEE 31st International Requirements Engineering Conference (RE)*.

Boukhers, Z. and Bouabdallah, A. (2022). Vision and natural language for metadata extraction from scientific pdf documents: A multimodal approach. *2022 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*.

Chandorkar, A., Patkar, N., Sorbo, A. D., and Nierstrasz, O. (2022). An exploratory study on the usage of gherkin features in open-source projects. *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*.

Farooq, M. S., Omer, U., Ramzan, A., Rasheed, M. A., and Atal, Z. (2023). Behavior driven development: A systematic literature review. *IEEE Access*.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional.

Hellesøy, A., Wynne, M., and Tooke, S. (2017). *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. Pragmatic Bookshelf.

Lafi, M. and abdelQader, A. (2023). Automated business rules classification using machine learning to enhance software requirements elicitation. *2023 International Conference on Information Technology (ICIT)*.

Mati, D. N., Hamiti, M., Selimi, B., and Ajdari, J. (2021). Building spell-check dictionary for lowresource language by comparing word usage. *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*.

North, D. (2006a). Introducing bdd. *Better Software*, 8(2):29–34.

North, D. (2006b). Introducing bdd. *Better Software*.

OneDayTesting (2019). Gherkin: Concepts and benefits. Accessed on: September 4, 2024.

Rajbhoj, A., Nistala, P., Kulkarni, V., Soni, S., and Pathan, A. (2023). Doctomodel: Automated authoring of models from diverse requirements specification documents. *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*.

Russell, S. and Norvig, P. (2020). *Artificial Intelligence: A Modern Approach*. Pearson, 4th edition.

Smart, J. F. (2014). *BDD in Action: Behavior-driven development for the whole software lifecycle*. Manning Publications.

Taneja, K. and Vashishtha, J. (2022). Comparison of transfer learning and traditional machine learning approach for text classification. *2022 9th International Conference on Computing for Sustainable Global Development*.

Wynne, M. and Hellesoy, A. (2017). *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. Pragmatic Bookshelf.

Xiao, J., Du, W., Xu, Z., and Qian, Y. (2023). Cross-system data integration based on rule-based nlp and node2vec. *2023 8th International Conference on Data Science in Cyberspace (DSC)*.

Ye, Y., Xie, X., Jin, H., and Wang, D. (2021). A hybrid model combined with svm and cnn for community content classification. In *2021 IEEE 23rd International Conference on High Performance Computing & Communications; 7th International Conference on Data Science & Systems; 19th International Conference on Smart City; 7th International Conference on Dependability in Sensor, Cloud & Big Data Systems & Application*.