Extracting Modelling Information using Natural Language Processing

CRISTIAN PERLOG, University of Twente, The Netherlands

ABSTRACT

In the field of software engineering, testing has a high importance in ensuring the reliability and the quality of software. One approach to testing is Behavior-Driven Development, which uses natural language descriptions to define software behavior. This research focuses on automating the process of extracting essential elements from BDD scenarios using Natural Language Processing (NLP) techniques. This study is built on previous research that combines BDD and MBT to generate and execute test cases automatically from BDD scenarios. It evaluates NLP tools and techniques, and develops a tool used for extracting modelling information from BDD scenarios.

Additional Key Words and Phrases: Natural Language Processing, Behavior-Driven Development, Software Testing, Automation, Model-Based Testing

1 INTRODUCTION

In previous research [13], a language called IBDD, along with a hybrid grammar, was defined to support the transformation of BDD scenarios into formal models, which can then be used in the next phases of generating and executing test cases. However, the transformation from BDD to IBDD is done manually. This research builds upon that work by proposing a tool to automate this process, more specifically, for extracting the modelling information from BDD scenarios.

BDD:
Behavior-Driven Development (BDD) offers a framework for defining software behavior using domain-specific language in natural language sentences/scenarios. One of the main goals of BDD is to facilitate clear communication among stakeholders (both technical and business people) and ensure that the development process aligns with business goals [10]. It achieves this by expressing both the behavior and expected outcomes of the system. To translate BDD scenarios to executable test cases, essential information—such as variables, actions, and conditions—from the BDD scenarios needs to be extracted. However, this process is currently done manually [13], which takes considerable effort and is prone to errors due to different interpretations of the scenarios.

NLP:
To address this challenge, this research proposes a tool that utilizes Natural Language Processing (NLP) techniques to automate the extraction of modeling information from BDD scenarios. NLP is a field that focuses on providing a range of computational techniques for processing data encoded in natural language [3]. Given that BDD scenarios are written in natural language, NLP is well-suited for the interpretation of these scenarios. The main goals of the proposed tool in this research are to reduce the time and effort required for this phase of test preparation and to improve the consistency and accuracy of the resulting outputs. This automation could facilitate direct integration into Model-Based Testing (MBT) frameworks.

Thus, the **main research question** is:
How can Natural Language Processing (NLP) techniques be employed to automate the extraction of core elements—actions, variables, and conditions—from BDD scenarios?

Overview:
The paper is organized as follows: Section 2 provides a literature review of existing research on the application of NLP techniques in the context of BDD, as well as integration of BDD and MBT approaches in the context of automating the generation of test cases. Section 3 gives a brief summary of what precisely is expected the tool to be able to extract from BDD scenarios. Section 4 describes the main NLP tools considered for this research, as well as the rationale behind the final selection of NLTK. Section 5 details the methodologies applied in building the tool. In Section 6, a high-level architecture of the proposed tool is presented. Section 7 provides a guide for using the tool. Section 8 presents an example usage of the tool, and Section 9 discusses the evaluation of the tool's effectiveness. Finally, Section 10 concludes the paper and suggests directions for future work.

2 LITERATURE REVIEW

This section will include a review of existing research on the application of Natural Language Processing (NLP) techniques in the context of Behavior-Driven Development (BDD). Previous studies have explored the use of NLP to bridge the gap between human-readable BDD scenarios and formalized test specifications.

- For instance, Gupta et al. [7] explored generating multiple conceptual models from BDD scenarios using NLP. Their approach parsed BDD scenarios to extract actions, variables, and conditions, mapping these to UML diagrams such as use case, class, activity, and state machine diagrams. This automated model generation improved test specification, software quality, and communication among stakeholders by accurately representing relationships and dependencies in user stories.
- Similarly, Soeken et al. [11] introduced a novel approach to assisted BDD using NLP, which laid the groundwork for more advanced automation tools. Their research focused on automating the initial stages of Behavior Driven Development (BDD) by using natural language processing techniques to extract design information from acceptance tests. A methodology was proposed, where the user enters into a dialogue with the computer, which suggests code pieces extracted from natural language sentences. Specifically, the focus was on generating UML models, such as sequence diagrams and class diagrams. The approach was validated with a case study, and it was a significant step towards an automatization of BDD.
- Zameni et al. [13] proposed an intermediate language called IBDD to bridge the gap between informal BDD scenarios and formal Model-Based Testing (MBT) models. Their approach

TScIT 41, July 5, 2024, Enschede, The Netherlands
© 2024 University of Twente, Faculty of Electrical Engineering, Mathematics and Computer Science.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

translates IBDD scenarios into BDD Transition Systems (BDDTS), which are then converted into Symbolic Transition Systems (STS) for generating test cases. This method ensures that BDD scenarios can be automatically transformed into precise test cases, improving the quality of software testing.
- In [14] a method was proposed to streamline the conversion of BDD scenarios into test cases, thus reducing manual effort and laying a foundation for automated testing. This study introduced BDD Transition Systems (BDDTS) as formal models that map BDD scenarios into Symbolic Transition Systems (STS). The approach improves BDD further by enabling the automated generation and execution of comprehensive test cases, thereby bridging the gap between BDD scenarios and executable tests.

3 EXPECTATIONS AND REQUIREMENTS

To have a better understanding of the tool extraction requirements, a BDD scenario involving a real-world printer example (extracted from [14]) and its expected outcome is provided:

Given a controller job is in the scheduled jobs. When the printer starts printing the controller job And the printer completes printing the controller job. Then the controller job is in the printed jobs

In section 4 from [13], the grammar of the IBDD language is described in more detail. In practice, the following information is expected to be extracted from the aforementioned BDD scenario:

- "Local Variable": "controller job"
- "Context Variables": ["scheduled jobs", "printed jobs", "printer"]
- "Given": {"Boolean Functions": ["is in the (controller job, scheduled jobs)"]}
- "When": {"Printer": ["starts printing (controller job)", "completes printing (controller job)"]}
- "Then": {"Boolean Functions": ["is in the (controller job, printed jobs)"], "Actions": {}}

In short, the scenario consists of "Given", "When", and "Then" clauses. "Given", and "Then" steps describe the expectations from the system while the "When" step specifies the behavior of the system.

4 DESCRIPTION OF NLP TOOLS

In the process of automating the extraction of elements from BDD scenarios, two prominent NLP tools were considered: spaCy and NLTK. This section details the functionalities of each tool and explains the decision behind selecting NLTK for the tool development.

4.1 spaCy

spaCy [5] is a popular open-source NLP library designed for efficiency and ease of use. It provides pre-trained models for various languages and offers functionalities such as tokenization, part-of-speech (POS) tagging, named entity recognition (NER), and dependency parsing. Some of the key features of spaCy include:

- **Ease of Use**: spaCy's user-friendly API makes it accessible for both beginners and experienced developers.
- **Pre-trained Models**: The library includes pre-trained models that deliver high accuracy for common NLP tasks.
- **Speed**: spaCy is optimized for performance, making it suitable for processing large volumes of text quickly.

4.2 NLTK

The Natural Language Toolkit (NLTK) [2] is a comprehensive library for building NLP programs. It provides a wide range of tools for text processing, including tokenization, POS tagging, stemming, lemmatization, chunking, parsing, and semantic reasoning. The key advantages of NLTK include:

- **Flexibility and Customizability**: NLTK offers a broader range of functionalities and greater flexibility in customizing the NLP pipeline to suit specific tasks.
- **Educational Resources**: NLTK is well-documented [1] and widely used in academia (e.g.: [9] or [12]).
- **Wide Range of Algorithms**: The toolkit includes numerous algorithms for different NLP tasks, allowing for experimentation and optimization.

4.3 Comparison of spaCy and NLTK

The decision to select NLTK over spaCy for this research was based on the fact that it could give more control over the extraction process and the ability to implement custom algorithms for parsing BDD scenarios according to fit the requirements best. The following table provides a comparison of the key features of spaCy and NLTK (see Table 1).

Given the need for detailed analysis and manipulation of text for accurately identifying and extracting actions, variables, and conditions from BDD scenarios, NLTK's extensive capabilities, especially the support of various POS taggers, make it the preferred choice for this research.

5 METHODOLOGIES

Firstly, a thorough literature review of the application of NLP techniques in the context of BDD was conducted. It was identified that there is a gap in the testing preparation phase, specifically that the extraction of key elements from BDD scenarios is currently performed manually. Subsequently, research was carried out to understand the capabilities of NLP [3], as well as understanding BDD, how its scenarios are written and the main goal of this agile approach to software testing [10].

Next, a small testing phase of the NLP tools NLTK [2] and spaCy [5] was conducted, after which a final decision to choose NLTK was made. Initially, a dataset of BDD scenarios was collected from books [4, 10], research papers [13, 14] and some of them were generated by ChatGPT [8]. These scenarios were used in testing the extraction of modeling information.

After developing the initial functionality of extracting the modelling information, the focus shifted to improving the functionality to handle a bigger variety of BDD scenarios. Finally, an evaluation was conducted to determine the tool's effectiveness.

5.1 Dataset Collection and Annotation

The dataset for the research consists of BDD scenarios collected from various sources [4, 8, 10]. This section describes the dataset collection, annotation, and the preprocessing of scenarios.

Table 1. Comparison of spaCy and NLTK features

Feature	spaCy	NLTK
Ease of Use	User-friendly API, accessible for both beginners and experienced developers	More complex API, requires deeper understanding of NLP concepts
Pre-trained Models	Includes high-accuracy pre-trained models for various languages	Fewer pre-trained models, but supports integration with external models
Speed	Optimized for performance, suitable for large text volumes	Slower performance due to extensive functionalities and flexibility
Tokenization	Efficient, accurate tokenization	Highly customizable tokenization
POS Tagging	High accuracy with pre-trained models	Highly customizable, supports various taggers
Named Entity Recognition (NER)	High accuracy with pre-trained models	Customizable, but requires more setup
Dependency Parsing	Robust and efficient dependency parsing	Customizable parsing with support for different algorithms
Stemming and Lemmatization	Basic support	Comprehensive support with customizable algorithms
Chunking and Parsing	Limited chunking capabilities	Advanced chunking and parsing with support for custom grammars
Semantic Reasoning	Limited support	Extensive support for semantic analysis and reasoning

5.1.1 Data Collection. The data collection process involved identifying relevant sources of BDD scenarios, as well as collecting a big variety of scenarios. The focus was on books that provided diverse and comprehensive examples of BDD scenarios. Additionally, some scenarios were generated using AI tools and then adapted to meet specific requirements. Specifically, making sure that the 'Given' part of the scenario specifies the required system state, that is a precondition for the BDD scenario. The latter should describe the action (or sequence of actions) performed by either the system or the environment. Finally, the 'Then' step describes the action(s) the system performs after 'When' and/or the final state of the system [14].

5.1.2 Data Annotation. To facilitate the training and evaluation of the NLP tool, the collected scenarios were annotated with the expected modeling elements: actions, variables, and conditions. This manual annotation process involved:

- **Identifying Key Elements**: Annotating each scenario with location variables, context variables, boolean functions, actors and the actors' actions.

5.2 Scenario Preprocessing

Preprocessing the scenarios is essential to standardize the format and remove noise. The preprocessing steps include:

- **Text Normalization**: Converting text to lowercase, removing punctuation, and standardizing whitespace.
- **Tokenization**: Splitting the text into tokens (words and punctuation marks).
- **POS Tagging**: Assigning part-of-speech tags to each token using NLTK's POS tagger.
- **Chunking**: Grouping tokens into phrases based on their POS tags to identify potential actions, variables, and boolean functions.

5.3 Tool Development

The development of the tool involved the following phases:

5.3.1 Initial Functionality. The initial phase focused on creating the basic functionality for extracting necessary data from the BDD scenarios. This included developing methods for tokenization, POS tagging, noun phrase extraction, boolean function extraction, and section extraction (Given, When, Then).

5.3.2 Functionality Improvement. After the initial functionality was implemented, the focus shifted to improving the extraction accuracy and performance of the tool. This involved refining the algorithms and NLP techniques to handle complex scenarios more effectively.

5.4 Evaluation

An evaluation phase (Section 7) was conducted to assess the tool's effectiveness. This involved testing the tool on a variety of BDD scenarios to measure its accuracy in extracting local variables, context variables, actors (and their actions) and boolean functions.

6 HIGH-LEVEL ARCHITECTURE OF THE TOOL

The tool is designed to extract structured data from Behavior-Driven Development (BDD) scenarios using Natural Language Processing (NLP) techniques provided by NLTK and incorporating its possibilities to fit the requirements. Here's a high-level overview of its architecture, showing the components:

(1) **Input Module**:
   - Responsible for receiving BDD scenario text input from the user.
   - Provides an interface for continuous input until the user decides to stop.

(2) **Processing Module**:

- **Tokenization**: Splits the input text into sentences and words.
- **POS Tagging**: Tags each word with its part of speech.
- **Noun Phrase Extraction**: Identifies and extracts noun phrases from the sentences.
- **Location Variable Extraction**: Identifies the primary location or entity involved in the scenario.
- **Context Variable Extraction**: Extracts context variables that provide additional information about the scenario.
- **Boolean Function Extraction**: Extracts logical conditions and boolean functions from the sentences.
- **Actor and Action Extraction**: Identifies actors and their respective actions within the scenarios.
- **Section Extraction**: Separately processes the 'Given', 'When', and 'Then' sections of the BDD scenarios to extract relevant data.

(3) **Data Aggregation Module**:
- Aggregates and structures the extracted data into a (Python) dictionary format [6].
- Combines and cleans data from different sections to form a clear output.

(4) **Output Module**:
- Outputs the structured data to the console or saves it to a JSON file.
- Supports generating unique filenames for each scenario if saving to files.

7 HOW TO USE THE TOOL

7.1 Preparing The Environment

- Ensure having Python installed.
- Ensure having the required NLTK resources installed:

```
pip install nltk
python -m nltk.downloader punkt
    stopwords maxent_ne_chunker words
    averaged_perceptron_tagger
```

7.2 Saving the Script

- Save the provided Python script to a file, e.g., `bdd_extractor.py`.

7.3 Run the Script

- Open a terminal or command prompt and navigate to the directory containing `bdd_extractor.py`.
- Run the script with an optional output file argument in order to save the output:

```
python bdd_extractor.py -o
    extracted_data.json
```

7.4 Interactive Input

- The script will prompt entering BDD scenarios one by one.
- Type a BDD scenario and press Enter. For example:
  `Given a controller job is in the scheduled jobs and the controller job is a production job. When the printer starts printing the controller job and completes printing the controller job. Then the controller job is in the printed jobs.`
- Continue entering scenarios. Type `stop` to end the input loop.

7.5 Output

- If specified an output file (using the command [-o filename.json]), the tool will save each scenario's structured data to a new JSON file with a unique name.
- If no output file is specified, the tool will print the structured data to the console.

8 EXAMPLE USAGE

8.1 Running the Tool

```
python bdd_extractor.py -o scenarios.json
```

This command runs the program and creates a json file, where the extracted data is going to be saved.

8.2 Entering Scenarios

Enter the BDD scenario (or type 'stop' to quit): `Given a controller job is in the scheduled jobs and the controller job is a production job. When the printer starts printing the controller job and completes printing the controller job. Then the controller job is in the printed jobs.`
Enter the BDD scenario (or type 'stop' to quit): `Given the printer controller is running. When the operator restarts the printer controller. Then the printed jobs clean up is executed.`
Enter the BDD scenario (or type 'stop' to quit): `stop`

8.3 Output

Two JSON files, `scenarios_1.json` and `scenarios_2.json`, will be created, each containing the structured data for the respective scenarios entered.

Below are the results of the scenario: `Given a controller job is in the scheduled jobs. When the printer starts printing the controller job And the printer completes printing the controller job. Then the controller job is in the printed jobs.`

- "Local Variable": "controller job"
- "Context Variables": ["scheduled jobs", "printed jobs"]
- "Given": {"Boolean Functions": ["is in the (controller job, scheduled jobs)"]}
- "When": {"printer": ["starts printing (the controller job)", "completes printing (the controller job)"]}
- "Then": {"Boolean Functions": ["is in the (controller job, printed jobs)"], "Actions": {}}

8.4 Script Overview

Here's a brief overview of the key functions in the script:

- `extract_noun_phrases(words_pos)`: Extracts noun phrases from a POS-tagged sentence. It looks for sequences of nouns and adjectives and collects them into phrases.
- `extract_boolean_functions(sentence)`: This function extracts boolean functions from a sentence by identifying auxiliary verbs and forming function phrases. It processes the sentence to determine the start and end of functions, handling parameters and ensuring that only meaningful functions are extracted. More precisely, once an auxiliary verb is found, it accumulates subsequent verbs, prepositions, particles, and determiners to form a meaningful boolean expression. The parameters are set using the extracted noun phrases from the sentence.
- `extract_given(data)`: This function processes the 'Given' section of the BDD scenario. It tokenizes the input data into sentences, extracts noun phrases, and identifies boolean functions. It also compiles a list of system variables (location and/or context variables) and parameters relevant to the 'Given' context. Both, the system variables and parameters are the noun phrases present in the 'Given' sentence/step.
- `extract_when(data)`: This function processes the 'When' section of the BDD scenario. The actors are extracted by checking if the word is a noun or a pronoun and if no action has been started yet. If so, this would be the current actor. Next, once an actor is identified, the function looks for verbs to find the actions. It compiles a dictionary of actors and their actions, including any parameters involved.
- `extract_then(data)`: This function processes the 'Then' section of the BDD scenario. It extracts boolean functions and actions, similar to the 'Given' and 'When' sections. It also compiles system variables and parameters, ensuring a comprehensive representation of the expected outcomes. This is done similarly, using the noun phrases.
- `extract_bdd_scenario(data)`: This function combines the processing of 'Given', 'When', and 'Then' sections. It merges noun phrases and location/context variables from 'Given' and 'Then', determines the location variable, and compiles a complete structured representation of the BDD scenario. The location variable is extracted by combining and counting the noun phrases present in both the 'Given' and 'Then' steps. If there are multiple such variables, the most frequent one is selected. In the case multiple noun phrases have the same frequency, the first noun phrase occurring in the 'Given' step is selected.
- `main()`: This function handles user input and output operations. It prompts the user to enter BDD scenarios, processes them using the extraction functions, and saves the results to JSON files. It manages the overall flow of the script and ensures that scenarios are correctly processed and stored.

9 EVALUATION OF EFFECTIVENESS

This section presents the evaluation results of the BDD scenario extraction tool, focusing on its effectiveness. The tool was tested on 10 various BDD scenarios (see Appendix A) to measure its accuracy in extracting local variables, context variables, and boolean functions. Most of the tested BDD scenarios are extracted from [14], being the most relevant to this research, while the other five [8, 10] are different from the real-world printer examples, hence making it a better testing set.

9.1 Scenarios and Results

After having tested the effectiveness of the tool based on the scenarios, below are shown the achieved results.

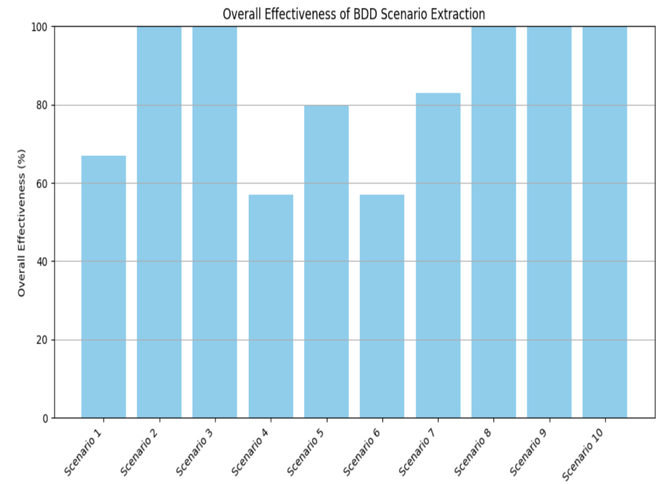

Fig. 1. Overall effectiveness of 10 different BDD scenarios

The overall effectiveness for each scenario is calculated using the following formula:

$$\text{Effectiveness of a BDD scenario} = \left(\frac{\text{Number of Correct Elements}}{\text{Total Number of Elements}}\right) \times 100\%$$

Here, an element is every item from the modelling information. For example, context variables could be a list of multiple variables, in which case, every variable would count as an element.

$$\text{Tool's Effectiveness} = \frac{\sum \text{Overall Effectiveness of Each Scenario}}{\text{Number of Scenarios}}$$

The overall effectiveness of the tool, calculated across the 10 scenarios, is 84.4%.

9.2 Detailed Analysis

The tool was evaluated based on its ability to accurately extract local variables, context variables, and the required information from each step of the BDD scenarios ('Given', 'When' and 'Then'). For an element to be considered correct, all related information in the annotated data (expected results) must correspond exactly to the actual results provided by the tool. Specifically, the local variable must match exactly, all data in the context variables list must be identical, and for the boolean functions, both the function name and its parameters must be correct. Lastly, for the 'Actions', all

components—actor(s), action name(s), and their parameters—must match precisely.

- **Local Variable**: Correct in 90% of scenarios (9/10)
- **Context Variables**: Correct in 70% of scenarios (7/10)
- **Boolean Functions (Given)**: Correct in 90% of scenarios (9/10)
- **Actions (When)**: Correct in 80% of scenarios (8/10)
- **Boolean Functions (Then)**: Correct in 57.14% of scenarios (4/7), the remaining three scenarios did not have boolean functions (in the 'Then' step)
- **Actions (Then)**: Correct in 66.6% of scenarios (2/3), the other scenarios did not include any actions

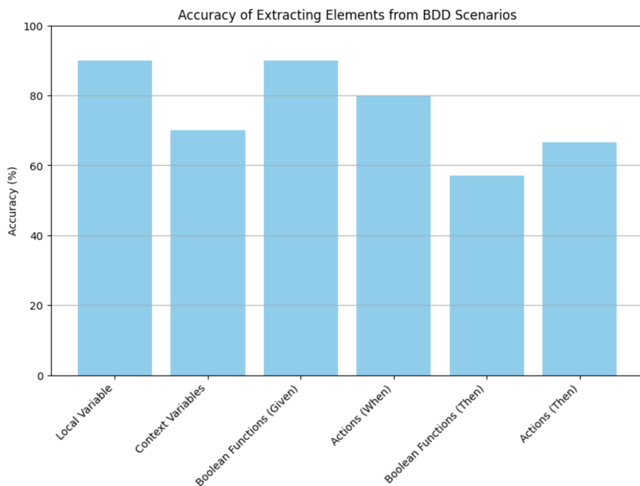

Fig. 2. Results of each step of extracting the key elements

The results above show that the tool performed well in extracting the 'Local Variable', as well as the boolean functions in the 'Given' step, with areas for improvement in extracting actions, context variables, as well as boolean functions that are more complex, such as those that have multiple noun phrases, or consisting of special cases (discussed in next subsection).

9.3 Special Cases in BDD Scenarios

It is important to note that BDD scenarios can contain special cases that may affect the extraction process:

- **String Values**: Scenarios may include specific string values that need to be identified and extracted correctly. For example:
  `Given I am not yet playing When I start a new game Then I should see "Welcome to Codebreaker!" And I should see "Enter guess:"`

  As of the current state of the tool, these cases would need either to be written in a different format, such that it won't contain string values (this example would be: ... Then I should see the welcoming message And I should see the guessing message), or be handled differently as specific cases as an additional feature of the tool.
- **Numbers and Symbols**: Scenarios may contain numbers or special symbols that need to be handled appropriately. For example:
  `"Given the secret code is 1234. When I guess 1234. Then the mark should be ++++`

  In a similar way, this case would require a different approach in either the way the scenario is written, or, alternatively, be handled separately.
- **Ambiguous Words**: Scenarios may contain words that can function as both verbs and nouns, making it challenging to extract the required data correctly. For example:
  `Then the controller job is in the waiting jobs.`
  `Given a user has finished shopping.`

  Words like "waiting" and "shopping" can be both nouns and verbs, depending on the context. These cases require advanced parsing techniques to accurately determine their roles within the scenarios. Therefore, the context variables might be extracted incorrectly, because they rely on the function used in extracting noun phrases. This means that having ambiguous words in BDD scenarios is currently very likely to lower the effectiveness of extracting context variables, as depicted in Figure 2.

9.4 Discussion

The evaluation results indicate that the tool performs well in extracting the boolean functions in the 'Given' step, with a 90% accuracy rate. Similarly, the extraction of local variables is highly accurate, achieving a 90% correctness rate across scenarios. However, the extraction of context variables and the 'Then' parts of the scenarios demonstrate areas for improvement, with accuracies of 70% and 57.14% respectively. The extraction of actions within the 'Then' step also shows an average score of 66.6% for the scenarios that included actions. Handling the "Then" parts of the scenarios still requires further adjustments.

Another observation from the Figure 1 is that the tool does not perform as well in more complex scenarios. More specifically, scenarios 1, 6, and 7, they all have boolean functions in the 'Then' step, which have parameters formed out of more than one noun phrase, which makes it more challenging to correctly extract them. Scenarios 4 and 5 both make use of the ambiguous word 'waiting', which is incorrectly labeled by the POS tagging as a verb, while it being an adjective describing the noun 'jobs'.

Special cases, such as ambiguous words, string values and symbols, present additional challenges that need to be addressed. Future work will focus on improving the tool's ability to extract and handle these elements with a better accuracy.

9.5 Summary

The preliminary evaluation of the BDD scenario extraction tool demonstrates its potential effectiveness in automating the extraction of modelling information from BDD scenarios. While the tool shows

promising results, particularly in extracting the boolean functions, further refinement is needed to improve its accuracy in other areas.

10 CONCLUSION AND FUTURE WORK

The research presented in this paper proposes a method and tool used for the purpose of automating the extraction of modelling information from Behavior-Driven Development (BDD) scenarios using Natural Language Processing (NLP) techniques. The primary goal of the tool is to reduce manual effort and improve the consistency and accuracy of test preparation, resulting in a more efficient cycle of automating test case generation using BDD along MBT.

The tool uses NLTK to process and extract actions, variables, and conditions from BDD scenarios, showing promising results in its preliminary evaluation. The tool successfully identifies and extracts key elements with a high degree of accuracy, particularly the local variables, and the boolean functions that do not consist of many noun phrases. However, the tool's performance in handling the context variables and actions, as well as special cases involving ambiguous words, string values and symbols, still has room for improvement.

Answer to the main research question:
How can Natural Language Processing (NLP) techniques be employed to automate the extraction of core elements—actions, variables, and conditions—from BDD scenarios?

The tool achieves this by making use of POS tagging and noun phrase extraction to identify key elements. Boolean functions are extracted by identification of the auxiliary verbs and logical phrases. Next, the steps 'Given', 'When', 'Then' are handled separately to capture the structure of BDD scenarios accordingly. Then, the extracted data is aggregated into a structure suitable to be further used in the testing phase.

Future Improvements:
The main area of future work for this research would be improving the effectiveness of the tool in the process of extracting modelling information from the BDD scenarios. That would require making a better algorithm for recognizing and extracting noun phrases to handle complex BDD scenarios better. Also, there is still needed to handle special cases from BDD scenarios, such as those containing string values, symbols and/or digits. The conditional logic could also be improved to handle the extraction of boolean functions and its parameters more accurately. The graphical use interface (GUI) may also benefit from having the functionality of batch processing, allowing the user to insert multiple BDD scenarios at once and/or be able to upload files with scenarios.

Addressing these areas, the tool would become more robust, accurate and user-friendly. Therefore its utility in the cycle of software testing would increase further.

ACKNOWLEDGMENTS

The author of the thesis would like to thank Petra van den Bos and Tannaz Zameni for the assistance and supervision during the period allocated for this research.

During the preparation of this work the author used ChatGPT [8] in order to generate BDD scenarios, which were used in the development/testing of the tool, as well as to improve the readability of the work. After using this tool/service, the author reviewed and edited the content as needed and takes full responsibility for the content of the work.

REFERENCES

[1] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python.* O'Reilly Media, Inc.
[2] Steven Bird, Edward Loper, and Ewan Klein. 2023. Natural Language Toolkit. https://www.nltk.org/
[3] K.R. Chowdhary. 2020. Natural Language Processing. In *Fundamentals of Artificial Intelligence.* Springer, New Delhi, 397–428. https://doi.org/10.1007/978-81-322-3972-7_19
[4] Eric Evans. 2004. *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Addison-Wesley Professional.
[5] Explosion. 2024. spaCy 101: Everything you need to know. https://spacy.io/usage/spacy-101
[6] Python Software Foundation. 2024. Dictionaries. https://docs.python.org/3/tutorial/datastructures.html#dictionaries.
[7] Abhimanyu Gupta, Geert Poels, and Palash Bera. 2023. Generating multiple conceptual models from behavior-driven development scenarios. *DATA KNOWLEDGE ENGINEERING* 145, Article 102141 (2023), 29 pages. http://doi.org/10.1016/j.datak.2023.102141
[8] OpenAI. 2024. ChatGPT. https://chat.openai.com/
[9] Krasen Samardzhiev, Andrew Gargett, and Danushka Bollegala. 2018. Learning Neural Word Salience Scores. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*, Malvina Nissim, Jonathan Berant, and Alessandro Lenci (Eds.). Association for Computational Linguistics, New Orleans, Louisiana, 33–42. https://doi.org/10.18653/v1/S18-2004
[10] J. F. Smart and J. Molak. 2023. *BDD in Action: Behavior-driven development for the whole software lifecycle.* Simon and Schuster.
[11] Mathias Soeken, Robert Wille, and Rolf Drechsler. 2012. Assisted Behavior Driven Development Using Natural Language Processing. In *Objects, Models, Components, Patterns*, Carlo A. Furia and Sebastian Nanz (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 269–287. https://doi.org/10.1007/978-3-642-30561-0_19
[12] Jiawei Yao. 2019. Automated Sentiment Analysis of Text Data with NLTK. *Journal of Physics: Conference Series* 1187, 5 (apr 2019), 052020. https://doi.org/10.1088/1742-6596/1187/5/052020
[13] Tannaz Zameni, Petra van den Bos, Arend Rensink, and Jan Tretmans. 2024. An Intermediate Language to Integrate Behavior-Driven Development Scenarios and Model-Based Testing. https://api.semanticscholar.org/CorpusID:268377309
[14] Tannaz Zameni, Petra van Den Bos, Jan Tretmans, Johan Foederer, and Arend Rensink. 2023. From BDD Scenarios to Test Case Generation. In *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW).* 36–44. https://doi.org/10.1109/ICSTW58534.2023.00019

A APPENDIX

The appendix includes the 10 BDD scenarios used for testing the effectiveness of the proposed tool, and for two of them the annotated data (expected results) along with the extracted information by the tool will be provided.

A.1 Scenario 1

Given a controller job is in the scheduled jobs.
When the printer starts printing the controller job and the printer completes printing the controller job.
Then there is a printed output and the printed output is a hard copy of the controller job.

Expected Results:

```
{
  "Local Variable": "controller job",
  "Context Variables": [
   "scheduled jobs", "printer", "printed output", "hard copy"
```

```
  ],
  "Given": {
    "Boolean Functions": [
      "is in the (controller job, scheduled jobs)"
    ]
  },
  "When": {
    "printer": [
      "starts printing (the controller job)",
      "completes printing (the controller job)"
    ]
  },
  "Then": {
    "Boolean Functions": [
      "is a (printed output)",
    "is a (printed output, hard copy of the controller job)"
    ]
  }
}
```

Actual Results:

```
{
  "Local Variable": "controller job",
  "Context Variables": [
   "printer", "hard copy", "printed output", "scheduled jobs"
  ],
  "Given": {
    "Boolean Functions": [
      "is in the (controller job, scheduled jobs)"
    ]
  },
  "When": {
    "printer": [
      "starts printing (the controller job)",
      "completes printing (the controller job)"
    ]
  },
  "Then": {
    "Boolean Functions": [
      "is a (printed output)",
     "is a (printed output, hard copy, controller job)"
    ],
    "Actions": {}
  }
}
```

A.2 Scenario 2

Given a controller job is printing.
When the operator pauses the printing of the controller.
Then the controller job is paused.

Expected Results:

```
{
  "Local Variable": "controller job",
  "Context Variables": ["controller", "operator",
  "printing"],
  "Given": {
    "Boolean Functions": [
      "is printing (controller job)"
    ]
  },
  "When": {
    "operator": [
      "pauses (the printing of the controller)"
    ]
  },
  "Then": {
    "Boolean Functions": [
      "is paused (controller job)"
    ],
    "Actions": {}
  }
}
```

Actual Results:

```
{
  "Local Variable": "controller job",
  "Context Variables": ["controller", "operator",
  "printing"],
  "Given": {
    "Boolean Functions": [
      "is printing (controller job)"
    ]
  },
  "When": {
    "Actions": {
      "operator": [
        "pauses (the printing of the controller)"
      ]
    }
  },
  "Then": {
    "Boolean Functions": [
      "is paused (controller job)"
    ],
    "Actions": {}
  }
}
```

A.3 Scenario 3

Given a controller job is paused.
When the operator resumes printing the controller job.
Then the controller job is printing.

A.4 Scenario 4

Given a controller job is paused.
When the operator moves the controller job to the waiting jobs before the printer completes printing.
Then the controller job is in the waiting jobs and the controller job is not in the printed jobs.

A.5 Scenario 5

Given a controller job is in the waiting jobs.
When the operator moves the controller job to the scheduled jobs.
Then the controller job is in the scheduled jobs.

A.6 Scenario 6

Given a user has forgotten their password.
When the user requests a password reset.
Then the user receives an email with instructions to reset their password.

A.7 Scenario 7

Given a user has finished shopping.
When the user continues to checkout.
Then there is a printed output and the printed output is a hard copy of the controller.

A.8 Scenario 8

Given a user has completed their purchase.
When the user submits their order.
Then the user receives a confirmation email.

A.9 Scenario 9

Given the customer is unauthenticated.
When they choose to collect their order.
Then they should be asked to supply contact details.

A.10 Scenario 10

Given the kettle has water.
When I turn the kettle on.
Then the water should boil.