ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΤΜΗΜΑ ΔΙΟΙΚΗΤΙΚΗΣ ΕΠΙΣΤΗΜΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ / ΤΕΛΙΚΗ ΑΝΑΦΟΡΑ ΠΡΑΚΤΙΚΗΣ ΑΣΚΗΣΗΣ

Κωνσταντίνα Σακέτου

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:
Παναγιώτης Λουρίδας

ΕΠΙΒΛΕΠΟΥΣΑ ΕΠΙΧΕΙΡΗΣΗΣ:
Φωτεινή Χαρπίδη

Ιούλιος 2022, Αθήνα

Μέρος Ι: Πτυχιακή Εργασία

Natural Language Processing in Software Testing: A Systematic Literature Review

Konstantina Saketou[1]

[1] Athens University of Economics and Business
www.aueb.gr

Abstract

The incorporation of Machine Learning and AI techniques to automate Software Testing processes has been a frequent phenomenon in recent years. Natural Language Processing (NLP) is the component of this family on which we ephazise in the present paper. Since the amount of knowledge in this area is constantly growing, there is a need to organize and provide a clear overview of the state-of-the-art technologies to both the scientific community and practitioners. This thesis paper is a Systematic Literature Review of the recent proposed NLP applications in Software Testing. We organize and classify the already existing knowledge included in a total of 44 scientific papers, based on the criterias of Contribution Type, Software Testing Stage, Software Testingg Type, Input Type, Output Type and also NLP Techniques used. Several significant findings of this review are: (1) the majority of the proposals refer to a new tool (2) many papers stick to popular and well-known NLP techniques like POS Tagging and TF-IDF to achieve their goal whereas few of them make use of more complex methods like LSTM RNNs. The present paper, aims to make current practices and upcoming needs accessible by simplifying the searching process and providing recent organized information that can be used as a starting point for further research.

Contents

Introduction

Software Testing is considered a very important stage of the Software Development Lifecycle. It aims to verify and review the quality of the whole system under development and to make sure that it functions according to all the requirements specified by the stakeholders. There are many different types of software testing and each one of them has different objectives regarding the aspect of the system that will be tested. Significant aspects include the system's functionality (Functional/Regression/Unit Testing), its overall performance (Performance/Stress Testing), its usability and also the approval from the end user and the degree to which he is satisfied with the system created (Acceptance Testing). Through the years, the mindset around software testing has developed. It is not being faced as a procedure that happens after the development stage but, on the contrary, it is now a necessary step towards the successful building of software. Testing happens during the whole development lifecycle, in parallel with everything else [1].

Because of the whole growth of the software testing mindset, it has now become a process that requires many resources in order to be successfully completed and it is one of the most expensive procedures [2]. For that reason, the need for integration of Automation Testing in software testing processes has been constantly increasing during the last years. Thanks to the use of automated test cases and automated testing in general, many benefits can be achieved such as reduction of test costs and effective reuse of code. According to a case study performed at a law-management software [3], the creation and use of automated GUI test cases improved significantly the whole testing process in many ways. For example, the test execution time reduced from 2 days to 1 hour and the customer satisfaction increased.

In addition to Software Test Automation, Machine Learning methods are being also used in order to optimize automation and make the whole testing process smarter and more effective. More specifically, Natural Language Processing (NLP) techniques have gained significant importance with applications that focus on different stages of the software testing process. A plethora of approaches have been proposed, with each one of them suggesting a different method regarding, for example, the requirement analysis phase, the test case implementation or execution. Some of these approaches target exclusively the NLP domain whereas others incorporate it into a wider method proposal for automation testing. The kind of the proposal may also vary from a simple algorithm to a whole new framework.

This paper is a Systematic Literature Review (SLR) study about the proposed applications of NLP methods that have been published in the last two decades and refer to the Software Testing domain. The already existing knowledge is being organized and presented here, filtered and structured according to the different criteria identified during the scientific literature research. Through this paper, a current and updated context is provided, regarding the fields of NLP and Software Testing that correspond to recent knowledge and applications which are accessible to anyone interested in the topics discussed.

Background and Related Work

This section sets the scene around the topic areas that will be discussed in this paper. Firstly, we describe basic information about the concept of NLP and we also provide some context related to Software Testing phases and processes. Then we briefly present studies we found that are similar to this one, compare them and consequently, identify the gap that this paper targets to fill.

1 Overview of Software Testing and NLP

Some kind of software is now present in every form of computer that exists these days, whereas no daily activity can be completed without using one. However, this leads to the expectations we form for that software to be extremely high since it has to perform at its best every time it is being used. Consequently, this has created the need to ensure the overall quality and functionality of that software. That is the role of Software Testing. According to Bourque et al [1], "*Software testing consists of the dynamic verification that a program provides expected behaviors on a finite set of test cases, suitably selected from the usually infinite execution domain*".

The testing process has four key issues [1]. The first one is that these days it is necessary to be performed using dynamic inputs and not static ones in order to ensure the adaptability of the program under test. Secondly, the number of test cases created should not be infinite. Even the simplest program can lead to countless possible test cases, which are obviously impossible to create and execute due to limitations in time and resources. The testing process gets even more chaotic when we refer to the testing of complex applications. Because of that, when it comes to testing it is important to create the suitable amount of tests which is adequate to assure the quality of the program under test. The third issue is that the selection of the testing techniques that will be used, should be performed very carefully, taking into consideration different aspects of the program to be tested, such as its functionality and other risks involved. Finally, the testing outcomes should be clear and easy for the test engineer to estimate if they are acceptable. In any other case, it means that the tests executed do not provide any substantial information for the condition and the quality of the software.

Another significant knowledge area that this paper focuses on is Natural Language Processing (NLP). According to Liddy E.D. [4], "*Natural Language Processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications*". NLP aims to process text with a human-like approach, by transforming textual information into a structure understandable by the computer. Nowadays, NLP is an intensely growing field and it keeps being constantly developed. It can be applied in a variety of ways, but Information Extraction (IE) is one which we will frequently come across in this paper.

Information Extraction aims to identify information structures and relationships between entities included in these structures from unstructured sources [5]. It is now a very well known

method in the NLP community with many applications in fields like machine learning, databases, web, document analysis and others. Sarawagi [5] refers to these applications by separating them into several categories with some of them being Enterprise Applications, Personal Information Management, Scientific Applications and Web Oriented Applications. Many of the ones explained in these categories are related to document processing (e.g. organization of files and projects), databases (e.g. data cleaning, management of citation databases) and management of user activity on the web (e.g. comparison shopping websites, placement of ads in webpages etc.). The wide usage of IE is, thus, very clear based on the different types of applications referred above and it is no surprise that many of the approaches studied in this paper utilize IE.

At the below section, we present the results of the research that was made in order to map the already existing Literature Reviews related to the subject we study in this paper.

2 Related Work

Several other papers have been created that are close to the current research area. Four such studies where found [6, 7, 8, 9] and are shortly presented in this section. Garousi et al. [6], conducted a Systematic Literature Mapping regarding NLP-assisted software testing methods suggested in a pool of 67 papers. This mapping study classifies and overviews a list of 38 identified tools. The goal of this paper was to provide practitioners with useful information about the reasons for which the use of such tools might benefit their current practice. However, it seems that on the spotlight of the study exist the different NLP tools proposed in the papers. The authors emphasize on these tools by further reviewing them, whereas other techniques or contribution types that may be included in the papers studied don't seem to take significant part of this Systematic Literature Review. Moreover, Ahsan et al [8], investigated 16 papers which contained a total of 6 NLP-based techniques and 18 tools with the same purpose. In addition to that, this study focuses on papers that contribute to the field of test case generation.

Trudova et al [7] created a Systematic Literature Review aiming to highlight the role of Artificial Intelligence in Automation Testing. They explore the influence of AI on the testing processes without focusing only on NLP techniques. The authors reviewed 34 papers where they identified 9 distinct software testing activities. The last study identified was the one conducted by Escalona et al [9], which focuses on NLP techniques regarding the automation of the test cases generation process. The analysis was performed between 24 approaches with a goal to determine the state of the art in the related area. The summary of the identified studies is displayed in Table 1.

We noticed that the wider knowledge area of Software Engineering is frequently studied as well, along with Natural Language Processing. We detected several papers related to different stages of that area and we state some of them in the following section. The first paper studies NLP during the Requirements Engineering (RE) process. RE contains activities related to maintaining, managing and creating the requirements in the design process. It is an essential step for the proper developing of every system and application in order to align with the corresponding specifications. Dalpiaz & Brinkkemper [10] propose a whole pipeline which combines several tools previously proposed in the literature in order to eliminate textual defects in user stories and, thus, to make the developing of the product architecture an easier process.

Another study touches the field of Release Planning which is tightly depended on the whole progress of the program development. Sharma & Kumar [11] focus on automatically managing user stories for Release Planning during Agile development. Specifically, they use NLP algorithms like RV Coefficient to categorize the user stories into different releases.

Table 2.1: SLRs identified about NLP and Software Testing

Title	Year	Reference	Papers studied	Focus Area	Empasis on
NLP-assisted software testing: A systematic mapping of the literature	2020	[6]	67	NLP	NLP Tools
Artificial Intelligence in Software Test Automation: A Systematic Literature Review	2020	[7]	34	Artificial Intelligence	Software Testing field
A Comprehensive Investigation of Natural Language Processing Techniques and Tools to Generate Automated Test Cases	2017	[8]	16	NLP	Test Case Generation
An overview on test generation from functional requirements	2011	[9]	24	NLP	Test Case Generation
This Paper	2022		44	NLP	Software Testing field

Having made a short reference to the already existing work, we notice that none of the above papers focuses on all three of NLP techniques, different application types (tool, framework, algorithm etc.) and different stages of software testing. In this paper, we emphasize on studies that refer to NLP techniques, possibly propose a variety of approach types and can improve different phases of the Software Testing stages (e.g. requirement analysis, implementation, execution). We aim to form a broader overview of the way that NLP and Software Testing fields now cooperate to optimize the quality of software.

Research Methodology

1 Categorization Criteria

This study aims to systematically classify the state-of-the-art in the field of Natural Language Processing combined with Software Testing processes. Through this procedure, this paper also targets to track possible trends and directions in current practices and techniques. In that way, possible future research opportunities are being also detected. To help with the categorization of the approaches and in order to give to this review a clear structure, we identify several categorization criteria based on which the present classification is performed. These criteria came up during the study of the different proposed applications.

1.1 Criterion 1: Contribution Type

This criterion refers to the kind of the approach that is being proposed. There is a number of possible types identified. Some of them include a new tool, a whole framework, a simple algorithm performing a small testing-related task (or not) or some of them may even contain a complex neural network. This paper embraces that kind of diversity since in that way, the different possible applications of NLP can be given prominence. We study the contribution type criterion since during the paper analysis we noticed the variety of existing approaches that are being proposed and developed by the scientific community. Thus, we would like to give an answer to the question regarding the number of these approach types, their complexity and, eventually, the way they achieve their target goal.

1.2 Criterion 2: Software Testing Stage

A significant criterion based on which we can classify the papers studied is the stage of Software Testing that they refer to. Does NLP have many applications in the Testing field? Maybe there are some approaches focusing on less popular testing phases? We include the Software Testing Stage criterion in this review, as an attempt to answer questions similar to the ones stated above. Every approach is possible to focus on a different testing phase. For example, some of them might refer to Requirement Analysis, Test Case Design and Development, Test Execution etc. Through this criterion, we are able to take a look into the broader spectrum of NLP applications in Software Testing. Moreover, this is a great chance to take a look into other testing phases apart from test case creation, which seems to be the main topic of interest when it comes to NLP and Testing.

1.3 Criterion 3: Software Testing Type

It is widely known that there is a significant number of different tests which are being executed to check a different aspect of an application. Does NLP have anything to do with all these types? Is it possible to contribute in any way on many of them? Maybe NLP is not always the answer? These kind of questions were generated during our analysis and they are also the motivation

behind the Testing Type criterion of this study. The applications studied in this paper each refer to some type of Software Testing. Some examples are Manual Testing, Automation Testing and Security Testing. In this section, the broader contribution of NLP in Software Testing is projected, while the reader can also identify possible usages of NLP on different aspects of testing.

1.4 Criterion 4: Input Type

There is also much interest about the different inputs of the proposed approaches. Reading through the studies, we notice a variety of data being processed to achieve the targeted result. Our questions regarding the possible contribution of NLP to different testing types lead us to further research about the information which practically builds the proposed approaches. What does this method need in order to perform this task? Should we provide it with complex data or its not that complicated after all? Given those questions, we want to map the level of complexity of the provided information on the papers' approaches. Several applications may accept as input a UML diagram, a graph/network or text written in a Domain Specific Language (DSL). These are some of the different types spotted during the analysis of the papers.

1.5 Criterion 5: Output Type

Another characteristic which is closely related with the one we just mentioned is the output type of the approaches. Since we study suggested applications from different stages of the Software Testing lifecycle, the outputs can vary. Depending on the stage they refer to, they can have different characteristics. Moreover, depending on the task they perform and its level of complexity, the output type will be different. The output is also a strong indicator of the overall goal of the approach. For example, if a tool generates test case code, then possibly its target is to automate the test case creation process to a certain degree. However, this can be determined by taking into consideration the input type as well. Apart from the most common output type we will come across here, which is the test case code, another interesting aspect in this section is that some approaches may even create something which may act as input to another program, increasing that way the level of flexibility in testing.

1.6 Criterion 6: NLP Techniques

In that case, we are interested in the type of NLP technique that is being used to perform the proposed approach. Which are those NLP methods frequently used to improve testing? What NLP process is usually followed on those approaches? What types of analyses are being performed? For example, some applications may use the very well known methods like the TF-IDF statistic, whereas others may use something less popular or even create a completely new NLP method. By categorizing the approaches according to this criterion, we get a better understanding of the latest NLP method trends that are being utilized and also, we identify new needs that have emerged in the NLP area by studying any new proposed NLP methods. This section contains significant information and is also one of the major focus knowledge areas of the present review.

2 Search and Selection of papers

The search process of the papers started by defining the corresponding search engines. The sources used were the ACM Digital Library[1], Scopus[2], IEEE Xplore[3] and Google Scholar. Then, we define the search terms and keywords. The ones used were *(nlp AND software testing), (nlp*

[1]https://dl.acm.org/

[2]https://www.scopus.com/standard/marketing.uri

[3]https://ieeexplore.ieee.org/Xplore/home.jsp

AND software testing) OR (nlp AND test automation) and *(nlp AND test automation)*. Due to the fact that the search results corresponded to thousands of papers, we excluded the papers that were written before the year 2000 to restrict this number. A big number of the papers that came up were not related at all to the current research area of interest. Also, we performed targeted search on several Conferences and Scientific Journals such as the International Conference on Software Engineering and the International Conference on Automated Software Engineering. The references of other previously published reviews were also another source utilized to track possible related studies.

After accessing the search results, the next step was to exclude a significant number of them and keep the ones that somehow correspond to the field we study. To achieve that, we read the titles and abstracts of those papers and we ended up with a total of 86 papers. Then, to further comprehend the contents of those 86 studies, we read and analyzed the full proposed approach of each one of them. During the reading, we excluded papers that generally refer to the field of Machine Learning without focusing on NLP. Also, papers that studied the broader area of Software Engineering and not specifically Testing were not of our interest. Many other papers studied the need for Automation in Software Testing making a short reference to the possible contribution of Machine Learning to that. These papers were also excluded during the searching process.

At that stage, all the papers of interest had been gathered and analyzed. Also, the ones that were not related to the goal of this study had been eliminated. After completing the above procedure, we get a final total consisting of 44 papers.

Results

After performing a thourough analysis on the gathered scientific papers, we came up with the categorization criteria presented in chapter 3. The contribution type criterion contains the categories of tool, framework, algorithm and technique. Starting from the tools, these are pieces of software containing programs that are used to support an application's or a system's functionality. Moreover, we consider as framework a total of techniques and tools combined in order to perform and/or facilitate a task. Regarding the algorithm, it is simply a sequence of instructions given to the computer with the aim of solving a specific problem. Finally, in the category of the techniques and approaches, we place the papers that propose a certain way of performing a task, usually using already existing tools and technologies. The reviewed papers have been mapped to those types and are presented in more detail in the following sections.

The next criterion is the Software Testing Stage. We identified papers which refered to a varitey of phases such as Requirement Analysis, Test Planning, Implementation and also Reporting. As we can see, we have found applications concerning a big part of the overall Testing lifecycle. As far as the Testing Type, we have made two kinds of categorizations here. The first one is Manual and Automated testing, since they are both two popular test types of major significance. We next categorized the approaches based on Integration, Acceptance and Crowdsourced testing. However, we also encountered many papers that do not set any limilation to the applicable testing type and, thus, provide more flexibility to that part. Another criterion we indentified is the input type of the proposed approach. Some of the most frequent types we found is that of NL test case descriptions, use case specification documents and also test case code. We further analyze these types at the corresponding section. As expected, we also refer to the output type as another criterion with the most common ones being the executable code, different tyes of models or diagrams (e.g. Use case/UML diagrams) and others. An interesting part of this criterion that we further analyze is that several approaches generate outputs which can then be provided as input to other tools or processes in general. Last but not least, we have the NLP Techniques criterion where we analyze in detail the identified NLP methods used in the papers. We study that section by separating it into two categories which are techniques refering to Natural Language Understanding and techniques refering to Natural Language Generation. The Natural Lnaguage Understanding component also contains a small section where we present different statistical NLP measures which are also used in the reviewed studies.

In this chapter, we present the results of our Systematic Literature Review and the study of the papers identified. We separately refer to each criterion individually by commenting and making a short reference to the results found. In some sections and always depending on the criterion, we further analyze any techniques or practices encountered.

1 Contribution Type

After reviewing all of the papers, we ended up with several contribution types proposed in their context. Our findings include tools, frameworks, algorithms and also techniques. In this section, we will refer to those categories that contain a big number of the papers we analyzed.

The first and most encountered contribution type is that of a tool. Almost 45,5% of the studied papers suggest some sort of tool for the ease of different software testing processes. Some of those tools are part of the overall approach, whereas others are the main focus of the study. Some of them target the test case execution process, others focus on test case generation or even other areas like defects prevention and requirement analysis facilitation. Pedemonte et al [12] present a way to convert manual functional tests into a state ready for automatic execution by using Machine Learning methods to process textual information. The tool contains semi-automatic functionality since in cases of textual information ambiguity, the user is prompted to interact with the tool to resolve the issue and provide the answer required to continue with the process. Another considerable approach is the CTRAS tool created by Hao et al [13] which aims at the summarization of duplicate test reports in oprder to make them easily understandable by developers. The tool idenitfies similarities in both textual information and screenshots and comprehensibly presents them into a single report.

Another significant contribution type we distinguished is that of a technique or approach. As we refered earlier, this type includes any proposal of approaches, processes, techniques and any other alternative way of performing a certain task. We grouped them all into this category since they all somehow refer to something very similar. Specifically, 25% of the studied papers suggest such an approach of that type. Rajaraman et al [14] propose in their study a way of idenitfying dependencies between tests. The authors apply Machine Learning techniques on log files in order to create a dependency graph which describes the connections and dependencies between the components of an application. This proposal widely facilitates the process of Integration Testing. We make a more detailed reference to that testing type at the corresponding section. Another paper that belongs to the technique category is that of Li et al [15]. Here they propose an automated way for extracting knowledge from historical bug reports and for also identifying relations between them. This functionality contributes to test troubleshooting activities and it can also enhance the reporting stage.

Several other papers focused on developing a Framework to achieve their goal. Some of them refer to the topic of test case generation while others emphasize on the earlier stage of requirement analysis and transformation. Lafi et al [16] created a framework which consists of different other processes in order to generate test cases. First of all, the use case descriptions are being processed in order to create a control flow graph and an NLP table of the system under test. Based on those, test paths are then generated which they will finally lead to the test case code. As far as the requirement analysis phase is concerned, Viggiato et al [17] created a framework in order to help testers improve the test cases they create. This framework accepts test cases in natural language as input and then provides recommendations and proposed changes. More specifically, it can provide recommendations for the improvement of the terminology of the test cases, it can identify potentially missing test steps from those test cases and lastly, it can prevent test case duplicates since it is able to identify similar test cases with the one which is currently being analyzed. All of the above three outputs contribute immensely to the proper and effective forming of test cases by making the testers' job much easier.

After the analysis of the identified contribution types of the papers studied, we notice that the majority of scientific contributions revovle around the proposal of a tool in order to perform

something new or to enhance a functionality. This is not something unexpected. Software Testing consists of many major processes, like test case generation, test planning, defect prevention and others, which shape all together the overall purpose of this knowledge area and are in need of improvements in order to move one step closer to complete automation of testing. Thus, many researchers and people interested in the field have figured out ways to enhance those methods by achieving great results.

2 Software Testing Stage

Another aspect of the papers we studied that we are interested in is the Stage of the Software Testing Lifecycle to which they apply. Some of the approaches we found refer to the Requirement Analysis phase, others to the Implementation and Development, Execution etc. The first one we will comment here is the Requirement Analysis. 25% of the studies come up with a proposal regarding that stage and they target to enhance the processing of the test requirements and specifications. This procedure happens before the construction of the test cases of any form and it is a critical stage which many times determines the overall outcome of the testing cycle. Thus, it is of major importance to effectivelly perform the Requirement Analysis for the optimal result.

Sainani et al [18], for example, propose a method technique which idenitfies and classifies requirements from software engineering contracts. These documents contain all the necessary information about the product to be developed, and therefore tested, with the desired characteristics and specifications being a part of them. The developed approach analyzes the contract document using different NLP techniques, which we will discuss in a further section, and outputs the different requirements of the product as well as their type. The output might add important information regarding the product's architecture or the importance of each feature. Based on that, developers and testers get a clear overview of what is worth to be tested more and also the parts that need more attention during the development and testing. This method refers to very early steps on software testing. However, this does not mean that such a process is not significant for a successfull testing cycle. On the contrary, it sets a concrete base for all the other stages that follow.

Equal importance to the Requirement Analysis stage is given by Femmer et al [19]. They propose a tool called Smella, which identifies Requirement Smells inside Requirement documents. At this point, it is worth mentioning what Requirement Smells actually are. Code smells are parts of code that need to be somehow changed in order to enhance its structure, complexity or comprehension [20]. Consequently, Requirement Smells represent spots of Requirement Documents which need to be modified in a certain way in order for the product's specifications to be stated with the optimal way. For example, in a certain document it is possible to have the same requirement expressed in a different way. This might happen due to syntax mistakes or overall use of complex syntax in the document, which makes comprehension a harder task for the reader. This is the job of the Smella tool. By making good use of different NLP techniques and incorporating them into the tool, the authors managed to perform the processed described above.

The next stage we will refer to in this section is Software Test Planning which contains tasks like Effort Estimation and the overall planning of the roadmap of the testing process. Yang et al [21] propose a method called DivClass which aims to define the optimal prioritization of test reports inspection in Crowdsource Testing. This testing type is covered in a next section of this chapter, so we will not further explain it here. The proposed method initially performs numerous natural language processing tasks to transform the test report dataset it accepts as input. Then, the similarity of those test reports is calculated and finally, based on that, the inspection priority of each one is determined. This is a short description of the overall method. We perform detailed

reference to the different NLP techniques in Section 4.6. This approach can benefit the Software Testing process since it can reduce the time spent on the inspection of test reports which are not that critical or that they don't contain high importance information. On the contrary, substantial amount of time can be spent on inspecting test reports of higher priority. Another method which targets to make Software Testing faster is the one proposed by Tahvili et al [22]. Their approach predicts the execution time of manual test cases based on their textual specifications and also on historical data from previously executed test cases. Undoubtedly this approach has a lot to offer to the software testing community since it enhances the manual test case execution process which most of the times it ends up requiring the most amount of time to be completed. Taking into account that the tester now knows the approximate execution time of the test cases, it is easier for him/her to plan the overall test cycle and create the expected timetable.

The testing stage which gathers the majority of the attention in the scinetific literature is the implementation/development stage. That is when the test cases are created and the whole testing project comes to life. At this point, it is worth mentioning that almost 63% of the papers studied refer to that stage in some way. Because of that, we have a plethora of approach types each targeting a different issue. A very common task we frequently came across is the one of test case generation, usually from Requirement Documents expressed in Natural Language. Kamalakar et al [23], for example, created a tool called Kirby which generates test case code from textual Requirement Specifications. Code pieces of the System Under Test can be also used in order for the tool to better understand the structure and semantics. However, the framework proposed by Lafi et al [16] in order to perform the same task is different. This approach uses the Use Case Description of the Use Case Diagram expressed in UML. It is obvious that there is a variety of alternatives that achieve the same goals, which is the test case generation. Though this is not the only one we identified at the implementation stage. Viggiato et al [17] propose a framework which aims at improving manual test case descriptions. More specifically, this approach accepts the test cases expressed in natural language and generates recommendations for the improvement of those descriptions. The framework recommends test case terminology improvements, identifies possible missing steps from those test case and it is also capable of finding potential test cases that already exist in the test code and which are similar to the one provided as input. This is a very powerfull approach which makes the test case implementation a piece of cake. Developing test cases based on clear, complete and comprehensive descriptions eliminates the possibility of mistakes and makes the whole process much faster.

Some of the papers we studied aim to also enhance the stage of test case execution. Pedemonte et al [12] created a tool which converts manual test cases into automated ones. In their analysis, they emphasize on the importance of the automatic test case execution and that is why they pursuited the implementation of that task. Their approach has the ability of accepting user guidance if necessary, but 70% of the test cases on which they evaluated the tool converted automatically to automated test steps without user guidance. Consequently, we see that the execution stage is also considered a significant one for the sucessfull completion of the Software Testing process.

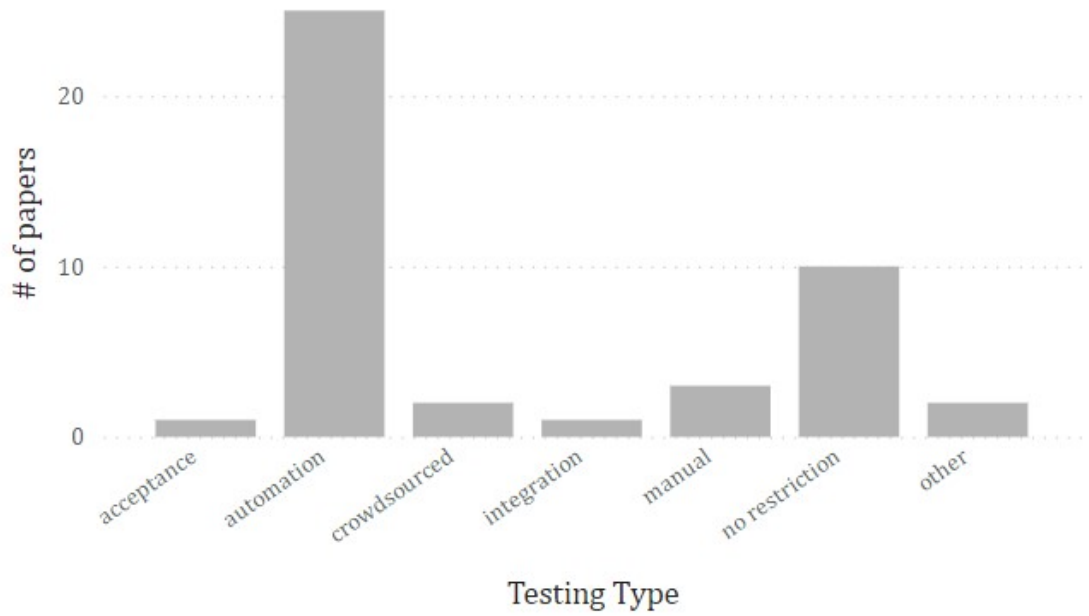
After analyzing the testing stages on which the studied papers refer to, we should make a short reference to a less popular issue of the testing world which is that of test report manipulation. Some of the studies we encountered, develop approaches that aim to transform test code and outcomes into a more human-friendly format. Hao et al [13] created a tool which not only identifies duplicate reports, but it also summarizes their content into a more comprehensive report. This process also happens on bug reports. On the other hand, Gonzalez et al [24] implemented a tool which performs a process opposite to what we have encountered so far in this review. The created tool converts JUnit Assertions into text written in English. Goal of this approach is to contribute to the maintenability, understandability and analysis of test code by reducing the

existence of complex and difficult to explain code.

3 Software Testing Type

In this section of our analysis, we focus on the test type that each approach refers to. The first kind of categorization we will perform is based on the two wide groups of manual and automated tests. Those two testing types are widely used in software testing with each one of them serving a different purpose. Manual Testing is a well-established technique where the tester interacts with the application in order to validate and ensure it functions as expected and according to requirements. He performs the steps described in the test cases by manually executing them. However, this is an extremely expensive process and requires a big amount of resources with time and money being some of them. This gap fills the use of Automated Testing, where important stages like test generation and execution are performed automatically. This process significantly decreases the amount of time needed to test the product and that is the reason why Automated Testing is widely used in Agile development where software releases are continuous and more frequent. Moreover, a big number of papers did not set any restrictions on the testing type to which they refer and they usually include improvement recommendations for the overall testing process. The results of the analysis based on Testing Type are also presented in Figure 1.

Figure 1: Classification of papers by Software Testing Type



We will firstly refer to the papers that contribute to the manual testing domain. A significant fact we notice from studying the approaches is that they refer to different phases of testing. With manual testing alone being a category of major importance, it is somehow reasonable for it to concern a big part of the software testing community. The Toucan4Test tool proposed by Zhang et al [25] facilitates the process of test cases forming and creation from requirements in natural language. More specifically, this approach automatically generates the test cases from specification documents in order for them to be manually executed from the testers. This entails that the above process outputs test cases in a comprehensive format which the executor can easily understand and that he can follow the test steps without any trouble. On the other hand, Tahvili et al [22] focus on manual test planning. Since we also referred to this approach in a previous section of this chapter, here we will shortly recall the context of the proposed method. The

authors proposed a technique which predicts the execution time of manual test cases. Such an ability is crucial for the optimized manual test execution since this is a very time consuming process. Thus, an effective scheduling of the test cases that are to be executed is more than important.

Although manual tests are a widely used testing method, automated testing has significantly grown the past few years and it continues to develop. This is also obvious from the fact that 25 out of all the papers studied in this review refer to this testing method. Consequently, these approaches correspond to various contribution types and testing stages. They refer to the automatic creation, forming and execution of test cases from different sources like requirement documents or use case diagrams. At this point, we should note that the majority of those techniques are applied during the implementation stage. One such example is the Litmus tool created by Dwarakanath et al [26] which generates test cases from requirement documents without imposing any restrictions to the input language format or syntax. The tool follows a five step process. After it verifies that a sentence of the document is testable, the processing and generation procedure of the test case begins. A similar tool was developed by Rane et al [27]. However, in that case the authors focused on optimizing the software testing process during Agile software development.

Apart from the above two big groups of manual and automated tests, we identified several other more specific testing types according to which we can further characterize some of the studied approaches. Those types are acceptance, integration and crowdsourced tests. The goal of acceptance tests is to determine whether the application meets functionality and usability criteria [28]. These tests are performed by the end users of the application. An example of such an approach is the one proposed by Wang et al [29] which targets the automatic generation of acceptance test cases and also, their automatic execution. Thus, the manual effort required is significantly reduced. Moving on to the Integration Testing type, it aims to verify that all the system modules and units communicate correctly to perform the functionality required [30]. This testing type is performed after Unit Testing which, on the other hand, tests each module of the application separately. The technique proposed by Rajaraman et al [14] targets Integration Testing. More specifically, goal of the approach is to identify interconnections and dependencies between different components of an application in order to optimize Integration Testing. The dependencies are determined based on the frequency of use and the importance of each component. Such characteristics exist in the application log files which are also the input of the proposed approach of the specific paper.

The third testing type we will talk about in this section is Crowdsourced Testing. Crowdsourced Testing has been growing in popularity during the last few years with many platforms having already been developed to serve that purpose, such as CrowdSprint[1] and TestBirds[2]. In Crowdsourced Testing, test tasks are available online for anyone who is willing to perform them and test various software products. When the test task is completed, a test report is submitted describing the software's behavior [31]. A paper that focuses on this testing type is one we have previously encountered written by Yang et al [21]. The authors propose a method called DivClass which prioritizes the test reports coming from Crowdsource testers. Also, similar reports are identified and combined into a signle report for more effective management of time during test report inspection.

Having expanded on the Software Testing Type characteristic of this review, we notice an overall tendency of the scientific community mainly towards Automation Testing approaches and methods to further evolve this practice, which is currently gaining a lot of attention. We also came up to a number of papers which emphasized on a more specific testing type without making

[1]https://crowdsprint.com/crowdsourced-testing/

[2]https://www.testbirds.com/services/quality-assurance/

distinctions regarding the above categories of Automated and Manual tests. Such test types are Integration, Acceptance and Crowdsourced Testing. However, the approaches proposed there can be utilized during both automated and manual testing.

4 Input Type

The input type of the studied approaches is another significant criteria based on which we will characterize them. During our analysis, we notice that the nature of the input that each method accepts varies from being some sort of test case code, requirement and use case documents. However, the majority of the inputs represent information written in natural language.

We will start by referring to the approaches that need test case code in order to perform the functionality required. The tool proposed by Gonzalez et al [24] accepts as input the test code of the system under test and in particular the part of it which contains the assertions performed to check different parts of the testing process. After the JUnit assertions have been provided to the tool, then a summary of the code is generated. More specifically, the assertion code is translated into natural language statements in English. That way, the test case code obtains a comprehensive format and the maintenance of the test cases becomes way easier and faster. Another approach which has test case code as one of its accepted inputs is the one proposed by Kamalakar et al [23] which we have previously referenced again in this chapter. The goal of the approach is the generation of test cases from the textual descriptions of the software's functionality. However, code pieces of the software under test can be also provided as input to the approach in order for the tool to better understand the behavior and expected functions of the application that is being tested.

A very frequent input type we came across during our analysis is that of test case descriptions written in natural language. The method proposed by Kirinuki et al [32] aims to facilitate the maintenance of the automated test scripts where the identification of web elements on web pages is necessary. In a web application elements like buttons, text input fields etc. tend to change as the application develops. That means that the corresponding test scripts need to constantly adapt to the new changes. However, such a process is very time consuming and not effective. Thus, the authors move towards script-free testing by proposing a technique where testing is performed through executable test cases in natural language with no need to create test scripts. Arruda et al [33] on the other hand, developed a tool for test script generation which accepts as input the textual test case descriptions. This tool generates reusable code and is also able to adapt and integrate with different test generation frameworks for even better results.

The use cases and user stories of the system under test is another common input type referenced into the papers studied. That information can be provided through different forms like diagrams or textual documents. However, we noticed that textual descriptions outweigh the rest of the accepted input formats. The method proposed by Nogueira et al [34] is one of the many which aim at the automatic test case generation. However, its accepted input type is the application's use cases written in a Controlled Natural Language (CNL) proposed by the authors. As stated in the paper, CNL is a subset of English that can be processed and translated into a formal language with possibly a different format [34]. On the other hand, the approach of Mulla et al [35] incorporates the user stories into software testing by specifically focusing on Agile practices. This approach exposes the importance of user story artifacts in the whole Software Testing Lifecycle by offering, at the same time, reusability of code, increased test case generation speed and reliability.

After the analysis of the different input types of the suggested approaches, we cannot question their variety and diversity. However, there is no doubt that most of them share a common

characteristic. The input information is usually expressed in natural language in order for the corresponding method or tool to perform the task required. Since the papers we study combine Natural Language Processing Techniques with Software Testing, it is more than reasonable for the proposed approaches to make good use of textual information of all sorts. This is another way through which the whole contribution of the NLP knowledge area is evident.

5 Output Type

We are now moving on to the output type characteristic based on which we will continue our analysis. The different output types distinguished seem to serve multiple purposes. Several approaches generate results ready to be used and to enhance the testing process, whereas others create information which can then act as input into another tool, framework or process. Test reports is another frequently encountered type which sometimes is also seen to incorporate bug reporting information.

The first output type we will refer to is any form of generated code. The corresponding papers of this output type implement automatic test case generation from different input information and as a result, they generate executable test case code. The approach of Mueller et al [36] does exactly that based on the textual requirements of the application's behavior. This is achieved by firstly transforming abstract textual information into a more normalized format (Textual Normal Form – TNF) and then, the authors make good use of the UML diagram representation where the initial information is transferred. The UML diagram is translated into classification trees which finally, generate the output code. At this point, we should note that the generated result uses the SystemVerilog language to represent the corresponding outcome. According to Bergeron et al [37], "*SystemVerilog is the first truly industry-standard language to cover design, assertions, transaction-level modeling and coverage-driven constrained random verification.*", meaning that SystemVerilog provides a huge flexibility and support when it comes to working in areas like high-level data types, object oriented programming, assertions and others. That way, integration does not act as a barrier when it comes to moving and exchanging knowledge and information related to different areas each.

Our analysis now continues with the papers which generate information that can be then provided as input to other processes and frameworks. The method proposed by Li et al [38] identifies similar test steps by using clustering techniques. The generated output is the created clusters which contain the common test steps. Those clusters can be then taken into advantage in order to create or even refactor API test methods. Either by acting as input to another tool or by simply being a useful source of information to the Test Engineer, this approach achieves the development of simpler and clean test code.

Different forms of models and diagrams is another frequent output type we came across during our analysis. Some of the corresponding approaches refer to the implementation stage, but the number of methods that target Test and Requirement Analysis is equally noticeable. Harmain et al [39] developed a tool called CM-Builder which aims at facilitating the process of test analysis. This tool accepts textual information regarding the corresponding software's requirements and then formally presents that information into a UML class diagram, consisting of the necessary object classes of the system. This class diagram can be then provided to any Test Analyst for further improvement. Thanks to the above process, the overall analysis of the system to be tested is being both enhanced and accelerated. Similar is the goal of the MaramaAIC tool developed by Kamalrudin et al [40]. However, the authors focus much more on the textual requirements improvement, thus after the input specification documents have been provided to the tool, the generated output is either a model or diagram containing the identified recommendations. Those

recommendations can be proposals to provide better consistency, completeness and correctness to the requirements.

Having concluded our analysis on the output types of the studied approaches, we see that there are two major types which stand out. Those are the executable test code and any form of model or diagram describing some type of information. Approaches that generate test code usually aim at the direct implementation and execution of the test cases and have taken up the mission to accelerate the overall testing process. On the other hand, papers that propose small and more detailed tools or methods and target the improvement of one specific stage or procedure, provide small but very useful improvements to small steps of the testing process. Nonetheless, both of the above directions, effectively and in their own way, enhance the testing journey. A more detailed overview about the studied papers and their matching to the above criteria is shown in Table 2 below.

Table 2. List of the studies reviewed by Contribution Type and Software Testing Stage

Title	Ref.	Contribution Type	Software Testing Stage	Input	Output
A systematic approach to automatically derive test cases from use cases specified in restricted natural languages	[25]	Tool	Requirement Analysis	NL Test Case Descriptions	Test Case Specifications in RTCM format
DASE: Document-Assisted Symbolic Execution for Improving Automated Software Testing	[41]	Tool	Requirement Analysis	Code Comments and Manual Pages	Detected bugs, Code Coverage results
Extracting and Classifying Requirements from Software Engineering Contracts	[18]	Framework	Requirement Analysis	Software Engineering Contracts	Architectural Requirements
Determining Software Inter-Dependency Patterns for Integration Testing by applying Machine learning on Logs and Telemetry data	[14]	Approach/ Technique	Requirement Analysis	Log Files	Identified Dependences

Processing Natural Language Requirements	[42]	Tools	Requirement Analysis	Requirement Document	a Web-based environment aiding in NL requirements gathering
Rapid Quality Assurance with Requirements Smells	[19]	Tool	Requirement Analysis	Requirement Document	Identified Requirement Smells
CM-Builder: An Automated NL-based CASE Tool	[39]	Tool	Requirement Analysis	Requirement Document	UML Diagram in a semantic network
MaramaAIC: tool support for consistency management and validation of requirements	[40]	Tool	Requirement Analysis	Textual Requirement Specifications	Corrected use case model/diagram
Detecting System Use Cases and Validations from Documents	[43]	Approach/ Technique	Requirement Analysis	Requirement Document	Detected use cases and rules
Score-Based Automatic Detection and Resolution of Syntactic Ambiguity in Natural Language Requirements	[44]	Tool	Requirement Analysis	Requirement Documents	Possible word interpretations and proposal for ambiguity resolving
Crowdsourced Test Report Prioritization Based on Text Classification	[21]	Approach/ Technique	Test Planning	Test Report Dataset	Test Reports in recommended order
Towards Execution Time Prediction for Manual Test Cases from Test Specification	[22]	Approach/ Technique	Test Planning	NL Test Case Descriptions	Predicted Execution Time
Automatically Generating Tests from Natural Language Descriptions of Software Behavior	[23]	Tool	Implementation	NL Test Scenario Descriptions, Code pieces of developed software	Test Case Code

Towards Transforming User Requirements to Test Cases Using MDE and NLP	[45]	Framework	Implementation	User Requirements container (URCon)	Container with annotated user requirements (AnnURCon)
Litmus: Generation of Test Cases from Functional Requirements in Natural Language	[26]	Tool	Implementation	Requirements Document	Test Case Code
Automation and consistency analysis of test cases written in natural language: An industrial context	[33]	Tool	Implementation	NL Test Case Descriptions	Test Case Code
NAT2TESTSCR: Test case generation from natural language requirements based on SCR specifications	[46]	Framework	Implementation	Requirements written in SysReq CNL	Software Cost Reduction Specifications Document
Towards Automatic Functional Test Execution	[12]	Tool	Test Execution	NL Test Case Descriptions	Adjusted Automated Test and Report
Abstract Flow Learning for Web Application Test Generation	[47]	Tool	Implementation	NL Test Case Descriptions	ML-based trainable test flow generation system
A Natural Language Programming Approach for Requirements-based Security Testing	[48]	Framework	Implementation	NL Test Case Descriptions	Executable Security Test Cases
Automatic Generation of Acceptance Test Cases from Use Case Specifications: an NLP-based Approach	[29]	Tool	Implementation	User Story, Test Scenario Descriptions	Test Case Code

Automatic Generation of Test Cases for Agile using Natural Language Processing	[27]	Tool	Implementation	NL Test Case Description	Test Case Code
Cluster-Based Test Scheduling Strategies Using Semantic Relationships between Test Specifications	[49]	Framework	Implementation	Test Specifications	Test Case Clusters ready for scheduling
Clustering Test Steps in Natural Language toward Automating Test Automation	[38]	Framework	Implementation	NL Test Case Descriptions	Test Case Clusters
Generation of Executable Testbenches from Natural Language Requirement Specifications for Embedded Real-Time Systems	[50]	Framework	Implementation	Textual Requirement Specifications	SystemVerilog Code
Automated Test Cases Generation From Requirements Specification	[16]	Framework	Implementation	Use Case Description model	Test Paths, NLP table
Constructing Test Cases Using Natural Language Processing	[51]	Tool	Implementation	Functional Requirement Document	Test Case Table
NLP-assisted Web Element Identification Toward Script-free Testing	[32]	Framework	Implementation	NL Test Case Descriptions	Test Procedure in NL
NLP-Based Requirements Formalization for Automatic Test Case Generation	[36]	Approach/ Technique	Implementation	Functional Requirements	Requirements Model

Syntactic Rules of Extracting Test Cases from Software Requirements	[52]	Approach/ Technique	Implementation	Functional Requirements	Syntactic Rules for Test Case Generation
Test Case Reuse based on ESIM Model	[53]	Approach/ Technique	Implementation	New Test Case	Test Cases Similarity
A Linguistic Analysis Engine for Natural Language Use Case Description and Its Application to Dependability Analysis in Industrial Use Cases	[54]	Approach/ Technique	Implementation	Use Case Descriptions	Use Case Description Model
Using Natural Language Processing Techniques to Improve Manual Test Case Descriptions	[17]	Framework	Implementation	NL Test Case Descriptions	Test Case Improvement Recommendations
A Natural Language Processing (NLP) Framework for Embedded Systems to Automatically Extract Verification Aspects from Textual Design Requirements	[55]	Framework	Implementation	Textual Design Requirements	Verification Assertions
Automatically Generating Precise Oracles from Structured Natural Language Specifications	[56]	Tool	Implementation	Test Case Specifications Document	Test Case Code
On Learning Meaningful Assert Statements for Unit Test Cases	[57]	Approach/ Technique	Implementation	Test and Focal method context	Assert statements

Building Combinatorial Test Input Model from Use Case Artefacts	[58]	Approach/ Technique	Implementation	Use Case Specifications	Test Case Model
Automatic generation of test cases and test purposes from natural language	[34]	Tool	Implementation & Test Execution	CNL Use Cases	Test Case Code
A Fine-Grained Approach for Automated Conversion of Junit Assertions to English	[24]	Tool	Reporting	Test Case Code Assertions	English text
CTRAS: Crowdsourced Test Report Aggregation and Summarization	[13]	Tool	Reporting	Test Report	Textual information and/or screenshots of duplicate tests
Towards the identification of bug entities and relations in bug reports	[15]	Approach/ Technique	Reporting	Bug Reports	Relations between bug entities
Mining Historical Test Logs to Predict Bugs and Localize Faults in the Test Logs	[59]	Tool	Test Planning	Test Logs	Identified Faults
The Potent Combo of Software Testing and NLP	[35]	Algorithm	Requirement Analysis	User Stories, Test Scenario Description	Test Case Code
Assisted Behavior Driven Development Using Natural Language Processing	[60]	Tool	Implementation	Test Case Code, NL Test Case Descriptions	UML class/sequence diagram

6 NLP Techniques

The field of Natural Language Processing (NLP) consists of two different components each referring to a specific task. The first one is Natural Language Understanding (NLU) and the second one is Natural Language Generation (NLG) [4, 61]. NLU involves the study and processing of text in multiple levels and aspects (e.g. syntactical, morphological etc.), in order to map the given textual information into useful representations.
NLU contains several steps which actually represent different analysis types [4]. The first kind is

Lexical Analysis which has to do with the understanding of a word's structure. The content of this analysis represents the Lexicon of the language which is being analyzed and contains a total of words and phrases of this language. Next we have Syntactic Analysis, during which sentences are parsed, based also on grammar rules, in order to analyze the relationship between words. Semantic Analysis is another NLP step which identifies all the possible meanings of a sentence based on the interaction of each individual meaning of the words comprising it. For example, the sentence "I will send a friend to my letter" will be rejected during Semantic Analysis. The next type is Discourse Analysis. This kind of analysis focuses on the meaning of a text which is conveyed by the interaction of the meanings of the different sentences it contains. Lastly, we have Pragmatic Analysis where sentences are re-interpreted on their real meaning like, for example, the identification of the subject on sentence pronouns like "him", "them" etc.
As far as NLG is concerned, it refers to the creation of meaningful texts of different formats based on an input resource. The process consists of three phases which are identifying the goal of the task, planning on how this goal can be achieved taking into consideration the available resources and finally, implementing the plan created into text [61]. The majority of the papers studied in this review perform NLU. However, several papers performing NLG were also identified.

6.1 Natural Language Understanding – NLU

We will firstly analyze the techniques used which belong to the NLU component. Since lexical analysis aims to understand the meaning of each word individually, Part-Of-Speech (POS) tagging is one of those techniques who widely contribute to this process. POS tagging is used in the majority of the reviewed papers because it performs a very important task for the text processing procedure. Goal of this technique is to assign to each word the part of speech tag to which it belongs. POS tagging distinguishes whether a word is a noun, a verb, an adverb etc. In our case, it is frequently used in user requirements analysis in order to transform them into use cases which finally helps generate the required test cases [45, 29, 27, 56, 58, 35]. Moreover, we also encountered POS tagging being used in proposed methods that focus on bug reporting and ambiguity detection in user requirements. Osama et al [44], for example, use it to assign three different types of ambiguities (semantic, syntactic and syntax) to the information identified.
However, several authors moved one step further from POS tagging. Specifically, we identified two papers that adopted a slightly altered approach. Pedemonte et al [12] utilize BIO tagging [62]. BIO tagging facilliates the process of creating or identifying chunks of words or sentences in a text. In contrast to POS, BIO tagging does not assign only one label to a token, but it inserts a prefix to each token's tag which declares its position at a chunk. There are three possible prefixes which are B – a tag is at the beggining of a chunk, I – a tag is inside a chunk and O – a tag does not exist in a chunk. The authors of this paper use this tagging method during the processing of functional test steps to finally transform them into automated tests. BIO tagging specifically helps them identify the test steps more accurately into the text, considering that each test step can represent a single chunk of information.
The second paper we found that does not use the widely known POS tagging method is the one by Carvalho et al [46]. The authors here implemented a framework for the generation of automated tests by incorporating into the process the Software Cost Reduction method, which detects and corrects errors during the requirement analysis phase. In this approach, a customized POS tagger was created in order for all the needs of the study to be satisfied. This was necessary, because the input of the approach is text written in a Controlled Natural Language (CNL) created by the authors called SysReq. The creation of a CNL also requires a lexicon of that language which will contain a number of its words and phrases. Thus, SysReq lexicon was also created. In general, it is very possible that one word has more than one meanings depending on its way of use inside a sentence. Since we now have a new total of textual representations, POS tagger may fail to identify and deal with this problem as expected. Consequently, a parser customized and adapted to SysReq-CNL was developed. This parser searches all possible classifications of a word into

SysReq lexicon by also taking into consideration any following words that are possibly necessary in order to extract the suitable meaning each time. For example, if we want to analyze the phrase "according to", the new parser will attempt to classify both "according" and also "according to" with the correct classification to be finally obtained from "according to" [46].

We are now moving on to the Syntactic level of analysis in NLP which mainly contains parsing techniques used to identify the relationships between words in a text. During the parsing process, we noticed a frequent use of the Stanford CoreNLP Parser[3]. This is an open source software product developed by the Stanford Natural Language Processing (NLP) Group available for everyone to use. The parser generates a phrase structure tree (PST) from input sentences in different languages. A PST describes the relationships and dependencies between the words of a sentence as well as its structure. This approach is used on several papers we reviewed [60, 27, 44]. Apart from the Stanford Parser, PST were also generally used in several other proposed approaches in order to achieve the same processing target as discussed above [39, 35]. The Natural Language Took Kit (NLTK)[4] is another open source project spotted being used during our analysis. NLTK contains many libraries and tools for performing NLP tasks – including parsing – using the Python programming language. The approach of Tahvili et al [22] predicts the execution time of manual test cases and uses NLTK parsing to identify the activities contained in a manual test. Moreover, the proposal of Preeti et al [58] includes the step of parsing UML diagrams containing use case specifications into their analysis.
On the other hand, Arruda et al [33] make use of Parsing Expression Grammar (PEG) [63] as a way to deal with encountered ambiguities on the text during parsing. The authors created their own CNL syntax and rules which then incorporated into the PEG parser. PEG parsing is slightly different than the above parsing methods discussed since, in that case, ambiguity problems are solved based on the grammar rules and priority choices set during construction. Textual ambiguities also concern Sinha et al [54] who didn't develop a new CNL to their approach. Instead, they don't restrict the textual input of their method by using shallow parsing and a Finite State Transducer (FST) to perform it. Shallow parsing allows us to be able to get part of the information included in the resulting parse tree. However, with POS tagging we can only keep the last layer of the tree which contains just the POS tags. As far as FSTs are concerned, Finite State methods have proven to be a very efficient and much faster way to perform different NLP tasks like lexical and syntactic analysis [64], especially in context-free approaches.

The next step in NLP is Semantic analysis, where the goal is to extract the actual or dictionary meaning of a word. For that task, we noticed the use of WordNet in several papers[60, 27, 33]. WordNet is a lexical database and contains semantic information for words in different languages available for everyone who works on NLP tasks. Sinha et al [54] introduce semantic information to their approach through the Dictionary Concepts Annotator they created. This component uses an extensible domain dictionary to assign verbs to predefined actions/classes. For example, the verb "CHANGE" will be assigned to the following classes: "UPDATE" – 78%, "OUTPUT" – 15% and "INPUT" – 6% [54].

A worth mentioning method we identified being used in order to optimize both syntactic and semantic analysis tasks is the Word2Vec technique [38, 18, 53]. Word2Vec [65], frequently used for word embedding processing, is a total of algorithms and architectures which transform the meaning of words into vector representations. That kind of representation, helps NLP tasks efficiently identify groups of similar words. The approach of Li et al [38] aims at clustering similar test steps in order for them to be assigned to the same test method and, in that way, empowering code reuse. Word2Vec significantly contributes to this paper by vectorizing the description of

[3]https://stanfordnlp.github.io/CoreNLP/parser-standalone.html
[4]http://www.nltk.org/

test sentences. The output vectors are then made use in word embeddings for the creation of the final test step clusters. However, Tahvili et al [49] use a variation of Word2Vec called Doc2Vec. Doc2Vec [66] is ideal for vectorizing whole textual documents, in contrast to Word2Vec which vectorizes individual words. The proposed approach accepts as input test specification documents and clusters them in order to finally proceed to scheduling. Such a task could not be performed by Word2Vec since here we are not talking about the vectorization of small test steps but, instead, we vectorize a total of textual documents.

We will finally refer to a process identified towards the end of the NLP steps described above which is the Anaphora Resolution process. Some of the papers studied [39, 36, 54] include this task as part of their approach. During Anaphora Resolution, pronouns are identified in the sentence and are replaced with the noun phrases they refer to. That way, the meaning of sentences becomes more clear and it is easier to spot dependences and semantic relationships between them. Gröpler et al [36] apply Pronoun Resolution using the algorithm proposed by Lappin et al [67] in order to identify third person pronouns in formal requirement documents. Moreover, Sinha et al [54] have dedicated a whole component of their approach to Anaphora Resolution where they use a specialized version of [68] to achieve the expected results.

Statistical Measures for NLP

Apart from the different techniques used in the papers, many of them also utilize different statistical metrics to measure frequency and similarity of entities like words and sentences. In this section, we present the most popular metrics among the ones we identified.

The first measure in our analysis is TF-IDF which stands for Term Frequency - Inverse Document Frequency. TF-IDF measures the importance of a word, sentence or lemma in a document compared to its importance in the corresponding collection of documents or corpus [5]. Li et al [38] use it for this exact purpose along with Amar et al [59]. However, apart from the classic TF-IDF approach, [59] uses a variation of it called line-IDF. In this paper, the goal of the authors is to prevent bugs in code through historical test logs processing. Thus, they needed to also measure the importance of a log line, since in this approach, rare log lines should be strong fault indicators. Consequently, line-IDF represents the importance of a log line in a test log and is defined as:

$$line - IDF_{l,d} = \log \frac{N}{N_l}$$

, where N is the number of logs for a test and N_l is the total number of logs that contain the line l. TF-IDF is also used by Kirinuki et al [32] during web element processing. More specifically, they use the metric to weigh words among elements in order to possibly identify and assign unique words to different web elements.

Cosine similarity is another measure we noticed being frequently used in the proposed approaches [23, 32, 53, 59]. After the vectorization of the information we want to compare (e.g. words, phrases, lines of text), the authors use it as a way to identify any resemblance between the elements. It is defined as [69]:

$$cosine\ similarity = \cos\theta = \frac{\overrightarrow{W_1} \cdot \overrightarrow{W_2}}{|W_1||W_2|}$$

, where W_1 and W_2 represent the two vectorized words we are comparing.
The proposed approaches of [53, 59] make use of the above measure in order to compare words in a text or document which usually contains test case descriptions. Kamalakar et al [23] incorporate cosine similarity in their total of calculations they perform. They use a hybrid method in order to

calculate similarity depending on each case their approach comes across, with cosine similarity being used in cases of sentence matching. Kirinuki et al [32] use a weighted mean of two cosine similarities in order to compare a target vectorized string to a web element defining it like so:

$$\frac{\alpha \times cos\_sim(v_{target} + v_{text}) + cos\_sim(v_{target} + v_{attr})}{\alpha + 1}, \alpha \geq 1$$

, where v_{target} is the vectorized string and v_{text} and v_{attr} represent information of the web element's structure (e.g. class, area-hidden etc.).

Moving to another statistical metric, Relaxed Word-Mover's Distance (RWMD) is used in the approach of Li et al [38] in order to calculate the similarity of test steps. RWMD is a variation of Word-Mover's Distance (WMD) proposed in [70] which calculates the Euclidean distance between word embeddings. Small distance equals to dissimilarity, whereas big distance means that the two words are more likely to be similar. RWMD can be more efficiently computed comparing to WMD, while still providing the authors with the information requested. The authors define the RWMD distance of two sentences x and x' as:

$$RWMD(x, x') = \max(\, l_1(x, x'), l_2(x, x')\,), where$$

$$l_1(x, x') = \sum_{i,j=1}^{n} t_{i,j}|v_i - v_j|_2 \quad , s.t.\ t_{i,j} = \begin{cases} x_i, & j = argmin_j|v_i - v_j|_2 \\ 0, & otherwise \end{cases}$$

$$l_2(x, x') = \sum_{i,j=1}^{n} t'_{i,j}|v_i - v_j|_2 \quad , s.t.\ t'_{i,j} = \begin{cases} x'_j, & j = argmin_j|v_i - v_j|_2 \\ 0, & otherwise \end{cases}$$

At the above formulas, we consider as x_i the normalized frequency of the word w_i in the text, whereas the vector v_i is the word embedding of w_i.

6.2 Natural Language Generation – NLG

Having completed our analysis on the NLU techniques used in the studied papers, in this section we will refer to the NLG techniques identified. Specifically, we found two papers which make use of text generation methods and, thus, have natural language outputs. Gonzalez et al [24] created a code summarization tool which specifically translates JUnit assertions to English text. That way, the maintainability, understandability and overall analysis of software and tests is being highly improved. To achieve that, the authors used the SimpleNLG engine. SimpleNLG was introduced by Gatt et al [71] in order to support large-scale data-to-text NLG systems that perform summarization of numerical and symbolic data. It is a Java library which performs lexical, syntactic and morphological analysis on given input and it, also, provides a variety of lexical (e.g. AdverbPosition, VerbType etc.) and phrasal (e.g. Tense, InterrogationType etc.) features.

The second paper performing NLG is the one by Watson et al [57] who proposed ATLAS (AuTomatic Learning of Assert Statements). ATLAS was created to predict a meaningful assert statement which can be used to asses the correctness of a focal method given also a test method. The ATLAS workflow consists of several steps. First of all, we have the extraction of test methods from different Java projects available on Github. Those methods proceed, then, to data mining tasks gathering the ones containing the "@Test" annotation as well as all the other declared methods of the projects. Then, during the data filtering stage, the methods found are separated to test and focal ones and, after the context of each focal method has been identified, ATLAS generates pairs of those test and focal methods. The relevant assert statements are generated as well. In order for ATLAS to learn how to successfully generate those assert statements, it uses

a Recurrent Neural Network (RNN) encoder-decoder model to automate this process. RNNs are being frequently used in NLG tasks since it is a great way to transfer information and they can also effectively identify patterns in data.

In our case, the Encoder is a single layer bi-directional RNN which consists of two distinct LSTM RNNs and takes as input the tokenized test method and the context of the focal method as a stream of tokens. Here, the authors made use of bi-directionality in the network in order for the encoder to take into consideration both the tokens that come before and the tokens that come after as context for the token under analysis. This practice offers more flexibility to the overall task.
As far as the Decoder is concerned, it is a double layer LSTM RNN which takes as input a context vector with fixed length and transforms it into a sequence of output tokens. It uses a copy mechanism trained on the raw source code of the focal and test methods which facilitates the process of determining the source where the output tokens will be copied from.

To conclude, we see that NLG tasks can end up being pretty complex depending on the target task and the expected quality of the output. However, such tasks seem to be very popular among the NLP community since these days they offer a wide spectrum of possible applications in many fields and even everyday tasks. Once again, a more detailed mapping of the studied approaches is presented in Table 3 below.

Table 3. List of the studies reviewed by NLP Techniques

Title	Reference	NLP Techniques	NLP Tools
DASE: Document-Assisted Symbolic Execution for Improving Automated Software Testing	[41]		Stanford CoreNLP Typed dependency
Extracting and Classifying Requirements from Software Engineering Contracts	[18]		Word2Vec
Determining Software Inter-Dependency Patterns for Integration Testing by applying Machine learning on Logs and Telemetry data	[14]	Lemmatization, Tokenization, Parsing	
Processing Natural Language Requirements	[42]	Word Tagging, Fuzzy Matching	
Rapid Quality Assurance with Requirements Smells	[19]	POS Tagging	
CM-Builder: An Automated NL-based CASE Tool	[39]	Anaphora Resolution	Stanford CoreNLP Parser

MaramaAIC: tool support for consistency management and validation of requirements	[40]	POS Tagging	
Detecting System Use Cases and Validations from Documents	[43]	POS Tagging, Tokenization	
Score-Based Automatic Detection and Resolution of Syntactic Ambiguity in Natural Language Requirements	[44]	POS Tagging	Stanford CoreNLP Parser
Crowdsourced Test Report Prioritization Based on Text Classification	[21]	Bag-of-Words Model, Synonymy Replacement	Language Technology Platform
Towards Execution Time Prediction for Manual Test Cases from Test Specification	[22]		Python NLTK
Automatically Generating Tests from Natural Language Descriptions of Software Behavior	[23]	Cosine Similarity	
Towards Transforming User Requirements to Test Cases Using MDE and NLP	[45]	POS Tagging	
Litmus: Generation of Test Cases from Functional Requirements in Natural Language	[26]		LinkGrammar Parser
Automation and consistency analysis of test cases written in natural language: An industrial context	[33]		Parsing Expression Grammar (PEG), WordNet
NAT2TESTSCR: Test case generation from natural language requirements based on SCR specifications	[46]	Customized POS Tagger (using Software Cost Reduction method)	
Towards Automatic Functional Test Execution	[12]	BIO Tagging	
Abstract Flow Learning for Web Application Test Generation	[47]	LSTM RNNs	

A Natural Language Programming Approach for Requirements-based Security Testing	[48]	Tokenization	
Automatic Generation of Acceptance Test Cases from Use Case Specifications: an NLP-based Approach	[29]	POS Tagging	
Automatic Generation of Test Cases for Agile using Natural Language Processing	[27]	POS Tagging	Stanford CoreNLP Parser, WordNet
Cluster-Based Test Scheduling Strategies Using Semantic Relationships between Test Specifications	[49]		Doc2Vec
Clustering Test Steps in Natural Language toward Automating Test Automation	[38]	TF-IDF, Relaxed Word-Mover's Distance	Word2Vec
Generation of Executable Testbenches from Natural Language Requirement Specifications for Embedded Real-Time Systems	[50]	Lemmatization	
Automated Test Cases Generation From Requirements Specification	[16]	Control Flow Graph, NLP Table	
Constructing Test Cases Using Natural Language Processing	[51]	Tokenization, Document Parsing	
NLP-assisted Web Element Identification Toward Script-free Testing	[32]	TF-IDF, Cosine Similarity	
NLP-Based Requirements Formalization for Automatic Test Case Generation	[36]	Anaphora Resolution	Algorithm of [67]
Syntactic Rules of Extracting Test Cases from Software Requirements	[52]	POS Tagging, Cosine Similarity	Penn TreeBank

Test Case Reuse based on ESIM Model	[53]	Cosine Similarity	Word2Vec
A Linguistic Analysis Engine for Natural Language Use Case Description and Its Application to Dependability Analysis in Industrial Use Cases	[54]	Shallow Parsing, Anaphora Resolution	Finite State Transducers, Dictionary Concepts Annotator, Specialized version of [68]
Using Natural Language Processing Techniques to Improve Manual Test Case Descriptions	[17]	Tokenization, n-grams, BERT-based language models	
A Natural Language Processing (NLP) Framework for Embedded Systems to Automatically Extract Verification Aspects from Textual Design Requirements	[55]	POS Tagging, Sentence Splitting	
Automatically Generating Precise Oracles from Structured Natural Language Specifications	[56]	POS Tagging	
On Learning Meaningful Assert Statements for Unit Test Cases	[57]	Data Filtering, Context Understanding, LSTM RNN Decoder/Encoder model	
Building Combinatorial Test Input Model from Use Case Artefacts	[58]	POS Tagging	Python NLTK
A systematic approach to automatically derive test cases from use cases specified in restricted natural languages	[25]	NL Restriction Rules	
Automatic generation of test cases and test purposes from natural language	[34]	Document Parsing	
A Fine-Grained Approach for Automated Conversion of Junit Assertions to English	[24]		SimpleNLG

CTRAS: Crowdsourced Test Report Aggregation and Summarization	[13]	Tokenization, Synonym Identification, Jaccard Distance, PageRank	
Towards the identification of bug entities and relations in bug reports	[15]	POS Tagging, Word Embeddings	Stanford CoreNLP
Mining Historical Test Logs to Predict Bugs and Localize Faults in the Test Logs	[59]	TF-IDF, line-IDF, Cosine Similarity	
The Potent Combo of Software Testing and NLP	[35]	POS Tagging	Stanford CoreNLP Parser
Assisted Behavior Driven Development Using Natural Language Processing	[60]		Stanford CoreNLP Parser, WordNet

Discussion

This section provides an overview of the mapping that took place on the main part of this review based on the criteria discussed in Chapter 3. We discuss trends noticed in each criterion regarding the usage of some techniques and any frequent patterns identified at the studied proposed approaches. This, finally, leads us to several conclusions about the current trends of NLP applications in the Software Testing field.

Starting from the Contribution Type characteristic, we notice a big tendency of the scientific community in developing frameworks and tools. This probably happens because these type of methods align with today's needs, since many times they combine complex functionality with applications to trending business scenarios. Frameworks and tools are widely used by corporations and smaller businesses in order to function fast and effectively. Consequently, the scientific community adapts to today's practices and aims at improving and facilitating the current practice by developing methods which easily adapt to the business world.

Our analysis, then, proceeded to the Software Testing Stage criterion with the Implementation stage being on top of the identified types. This stage is both a very important and also a very time consuming part of the Testing lifecycle, and sometimes it may act as a barrier in the current Agile Development methodology followed by many organizations. A very big number of the papers we studied in this review have created approaches to automate the test case creation process, with that requiring very little human effort to be completed. That way, test case development becomes almost fully automated and the duration of the testing process decreases significantly. This again proves that the academic research keeps up with current trending practices in order to contribute new and useful knowledge.

As far as the Testing Type criterion is concerned, the vast majority of the approaches focuses on Automated Testing. As we have discussed in earlier sections of this review, the test automation field has been rapidly growing the last few years along with the overall tendency for automation existing in numerous parts of the corporate world and also everyday life. Taking into consideration the need for adaptation to new changes, the authors of many of the papers we refered here have brought automation processes one step closer to business practices. As we noticed, this is usually performed through the development of methods which automate tasks like test case creation and enhance different requirement analysis procedures.

Regarding the input type characteristic, we frequently encountered approaches that operate given documents containing natural language requirements and specifications about the tests to be developed. These documents act as a base for the whole testing process since they describe necessary prerequisites of the system under test. They also contain the expected functionality of the product based on which it is later going to be tested for. Consequently, regardless of the target of the proposed approach, requirement documents are an important source of information for software testing, which is sometimes difficult to make use of because of the fact that they

contain natural language text that is not a very friendly data format for the computer to process. However, thanks to NLP, that kind of data types are being now more frequently and easily utilized.

Moving on to the output type, we find that test case code is one of the main data types generated by the proposed approaches. This finding aligns with the fact that many of the papers focus on the implementation stage, as we refered earlier. Thus, the test case generation process creates executable test code for the system under test.

The last criterion based on which we analyzed the papers is the NLP Techniques used in the approaches. We separated the NLP techniques in NLU and NLG methods and discussed them individually. What we eventually ended up on, is that the majority of the papers perform NLU tasks and use popular methods to achieve that. A widely and well-known technique used during Lexical Analysis is tagging and, more specifically, POS Tagging. POS Tagging identifies the part of speech to which the given words belong. However, apart from the classic use of POS Tagging, we identified several cases to which different customizations have been applied to the method. Those customizations are related to the choice of different tags than the classic parts of speech [44], whereas another paper uses BIO Tagging to further facilitate the process [12]. Other authors even proceeded to create their own customized POS Tagger [46].
Moreover, the use of NLP tools and libraries is also pretty often. Two distinct ones we noted are the Stanford CoreNLP library and the Python NLTK. Both of them are mainly used to support NLP parsing tasks, even though they provide a variety of functionalities. Word2Vec is another popular tool which turned out to be pretty popular among the papers reviewed. This tool facilitates the process of transforming the semantic representation of words into vectors, a task necessary in order to utilize word embeddings. To conclude, several statistical measures were also used by the authors with some of them being widely known to the NLP community. The metrics applied in most of the papers are TF-IDF and cosine similarity. Both of them provide an indication of the semantic similarity between two elements using their vector representations.
Overall, we notice a tendency to continue using long-established popular NLP methods, and customizing them if necessary to achieve the requested goal. However, this doesn't mean that attempts to incorporate more unusual practices have not been made. LSTM RNNs and Relaxed Word-Mover's Distance are two examples of methods which we saw being used in a low frequency among the approaches. We believe, though, that the desired complexity of the technique and the quality of the result play an important role in the type of method which will be used during development. During this review, we discovered that it is not necessary to use complex methods in order to achieve substantial results and contribute to this knowledge area. Significant approaches have been developed that highly improve testing processes by making use of simple yet effective NLP methods.

Conclusion and Future Work

This thesis paper classified the state-of-the-art and the practices of NLP used in the Software Testing process. We reviewed 44 scientific papers written after the year 2000 that focus on this knowledge area and contribute to different testing tasks. This Systematic Literature Review organizes the already existing knowledge of the scientific literature and it is a well structured source of information for both researchers and practitioners who operate in the software testing field.

Also, the incorporation of a popular Machine Learning process — NLP — in this knowledge area increases the level of alignment of this review with current practice and trends. By creating this study, we aim to make current practices and upcoming needs accessible to anyone interested. We simplified the searching process by providing recent organized information which can be used as a starting point for further research.

Given that, the present review motivates the business community to utilize outcomes and results of this paper in order to express or verify current needs and challenges faced in technical processes during practice. The scientific community is encouraged to use this study and the above expressed needs and challenges as a source of information in order to contribute to the solution of those problems.

Bibliography

[1] Pierre Bourque, Robert Dupuis, Alain Abran, James W Moore, and Leonard Tripp. *The guide to the Software Engineering Body Of Knowledge v3.0*. IEEE, 2014.

[2] Vahid Garousi and Frank Elberzhager. Test Automation: not just for test execution. *IEEE Software*, 34(2):90–96, 2017.

[3] Vahid Garousi and Erdem Yildirim. Introducing automated GUI testing and observing its benefits: an industrial case study in the context of law-practice management software. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 138–145. IEEE, 2018.

[4] Elizabeth D Liddy. Natural Language Processing. *In Encyclopedia of Library and Information Science, 2nd Ed. NY. Marcel Decker, Inc.*, 2001.

[5] Sunita Sarawagi. Information Extraction. *Found. Trends Databases*, 1(3):261–377, mar 2008.

[6] Vahid Garousi, Sara Bauer, and Michael Felderer. NLP-assisted software testing: A systematic mapping of the literature. *Information and Software Technology*, 126:106321, 2020.

[7] Dhaya Sindhu Battina. Artificial Intelligence in Software Test Automation: A Systematic Literature Review. *International Journal of Emerging Technologies and Innovative Research (www. jetir. orgl UGC and issn Approved), ISSN*, pages 2349–5162, 2019.

[8] Imran Ahsan, Wasi Haider Butt, Mudassar Adeel Ahmed, and Muhammad Waseem Anwar. A comprehensive investigation of natural language processing techniques and tools to generate automated test cases. In *Proceedings of the Second International Conference on Internet of things, Data and Cloud Computing*, pages 1–10, 2017.

[9] María José Escalona, Javier J Gutierrez, Manuel Mejías, Gustavo Aragón, Isabel Ramos, Jesús Torres, and Francisco José Domínguez. An overview on test generation from functional requirements. *Journal of Systems and Software*, 84(8):1379–1393, 2011.

[10] Fabiano Dalpiaz and Sjaak Brinkkemper. Agile requirements engineering: From user stories to software architectures. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*, pages 504–505, 2021.

[11] Sarika Sharma and Deepak Kumar. Agile release planning using natural language processing algorithm. In *2019 Amity International Conference on Artificial Intelligence (AICAI)*, pages 934–938, 2019.

[12] Pablo Pedemonte, Jalal Mahmud, and Tessa Lau. Towards automatic functional test execution. In *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces*, pages 227–236, 2012.

[13] Rui Hao, Yang Feng, James A. Jones, Yuying Li, and Zhenyu Chen. CTRAS: Crowdsourced Test Report Aggregation and Summarization. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 900–911, 2019.

[14] R Rajaraman, P K Kapur, and Deepak Kumar. Determining Software Inter-Dependency Patterns for Integration Testing by applying Machine learning on Logs and Telemetry data. In *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pages 1080–1084, 2020.

[15] Bin Li, Ying Wei, Xiaobing Sun, Lili Bo, Dingshan Chen, and Chuanqi Tao. Towards the identification of bug entities and relations in bug reports. *Automated Software Engineering*, 29(1):1–31, 2022.

[16] Mohammed Lafi, Thamer Alrawashed, and Ahmad Munir Hammad. Automated Test Cases Generation From Requirements Specification. In *2021 International Conference on Information Technology (ICIT)*, pages 852–857, 2021.

[17] Markos Viggiato, Dale Paas, Chris Buzon, and Cor-Paul Bezemer. Using Natural Language Processing techniques to improve manual test case descriptions. In *International Conference on Software Engineering-Software Engineering in Practice (ICSE-SEIP)*, 2022.

[18] Abhishek Sainani, Preethu Rose, Vivek Joshi, and Smita Ghaisas. Extracting and Classifying Requirements from Software Engineering Contracts. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 147–157, 08 2020.

[19] Henning Femmer, Daniel Méndez Fernández, Stefan Wagner, and Sebastian Eder. Rapid Quality Assurance with Requirements Smells. *Journal of Systems and Software*, 123:190–213, 2017.

[20] Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[21] Yuxuan Yang and Xin Chen. Crowdsourced Test Report Prioritization Based on Text Classification. *IEEE Access*, pages 1–1, 2021.

[22] Sahar Tahvili, Mehrdad Saadatmand, Markus Bohlin, Wasif Afzal, and Sharvathul Hasan Ameerjan. Towards Execution Time Prediction for Manual Test Cases from Test Specification. In *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 421–425, 2017.

[23] Sunil Kamalakar, Stephen H Edwards, and Tung M Dao. Automatically Generating Tests from Natural Language Descriptions of Software Behavior. In *8th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE-2013)*, pages 238–245, 2013.

[24] Danielle Gonzalez, Suzanne Prentice, and Mehdi Mirakhorli. A Fine-Grained Approach for Automated Conversion of JUnit Assertions to English. In *Proceedings of the 4th ACM SIGSOFT International Workshop on NLP for Software Engineering*, NL4SE 2018, page 14–17, New York, NY, USA, 2018. Association for Computing Machinery.

[25] Man Zhang, Tao Yue, Shaukat Ali, Huihui Zhang, and Ji Wu. A systematic approach to automatically derive test cases from use cases specified in restricted natural languages. In *International Conference on System Analysis and Modeling*, pages 142–157. Springer, 2014.

[26] Anurag Dwarakanath and Shubhashis Sengupta. Litmus: Generation of Test Cases from Functional Requirements in Natural Language. In *17th International conference on Applications of Natural Language Processing and Information Systems*, pages 58–69, 06 2012.

[27] Prerana Pradeepkumar Rane. *Automatic generation of test cases for Agile using Natural Language Processing*. PhD thesis, Virginia Tech, 2017.

[28] Glenford J Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing*. John Wiley & Sons, 2011.

[29] Chunhui Wang, Fabrizio Pastore, Arda Goknil, and Lionel Briand. Automatic generation of acceptance test cases from use case specifications: an NLP-based approach. *IEEE Transactions on Software Engineering*, 2020.

[30] Hareton KN Leung and Lee White. A study of integration testing and software regression at the integration level. In *Proceedings. Conference on Software Maintenance 1990*, pages 290–301. IEEE, 1990.

[31] Qiang Cui, Junjie Wang, Guowei Yang, Miao Xie, Qing Wang, and Mingshu Li. Who should be selected to perform a task in crowdsourced testing? In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 75–84. IEEE, 2017.

[32] Hiroyuki Kirinuki, Shinsuke Matsumoto, Yoshiki Higo, and Shinji Kusumoto. NLP-assisted Web Element Identification Toward Script-free Testing. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 639–643, 2021.

[33] Filipe Arruda, Flávia Barros, and Augusto Sampaio. Automation and consistency analysis of test cases written in natural language: An industrial context. *Science of Computer Programming*, 189:102377, 2020.

[34] Sidney Nogueira, Hugo LS Araujo, Renata Araujo, Juliano Iyoda, and Augusto Sampaio. Automatic generation of test cases and test purposes from natural language. In *Brazilian Symposium on Formal Methods*, pages 145–161. Springer, 2015.

[35] Nilofar Mulla and Naveenkumar Jayakumar. The potent combo of software testing and nlp. In *ICDSMLA 2019*, pages 1623–1632. Springer, 2020.

[36] Robin Gröpler, Viju Sudhi, Emilio José, Calleja García, and Andre Bergmann. NLP-Based Requirements Formalization for Automatic Test Case Generation. In *29th International Workshop on Concurrency, Specification and Programming (CS&P'21)At: Berlin, Germany*, 09 2021.

[37] Janick Bergeron, Eduard Cerny, Alan Hunter, and Andrew Nightingale. *Verification methodology manual for SystemVerilog*, volume 1. Springer, 2006.

[38] Linyi Li, Zhenwen Li, Weijie Zhang, Jun Zhou, Pengcheng Wang, Jing Wu, Guanghua He, Xia Zeng, Yuetang Deng, and Tao Xie. Clustering Test Steps in Natural Language toward Automating Test Automation. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, page 1285–1295, New York, NY, USA, 2020. Association for Computing Machinery.

[39] Harmain Mohamed Harmain and R Gaizauskas. CM-Builder: an automated NL-based CASE tool. In *Proceedings ASE 2000. Fifteenth IEEE International Conference on Automated Software Engineering*, pages 45–53. IEEE, 2000.

[40] Massila Kamalrudin, John Hosking, and John Grundy. Maramaaic: tool support for consistency management and validation of requirements. *Automated software engineering*, 24(1):1–45, 2017.

[41] Edmund Wong, Lei Zhang, Song Wang, Taiyue Liu, and Lin Tan. Dase: Document-assisted symbolic execution for improving Automated Software Testing. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 620–631. IEEE, 2015.

[42] Vincenzo Ambriola and Vincenzo Gervasi. Processing Natural Language Requirements. In *Proceedings 12th IEEE International Conference Automated Software Engineering*, pages 36–45. IEEE, 1997.

[43] Smita Ghaisas, Manish Motwani, and Preethu Rose Anish. Detecting system use cases and validations from documents. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 568–573, 2013.

[44] Mohamed Osama, Aya Zaki-Ismail, Mohamed Abdelrazek, John Grundy, and Amani Ibrahim. Score-Based Automatic Detection and Resolution of Syntactic Ambiguity in Natural Language Requirements. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 651–661, 2020.

[45] Sai Chaithra Allala, Juan P Sotomayor, Dionny Santiago, Tariq M King, and Peter J Clarke. Towards transforming user requirements to test cases using MDE and NLP. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 350–355. IEEE, 2019.

[46] Gustavo Carvalho, Diogo Falcao, Flávia Barros, Augusto Sampaio, Alexandre Mota, Leonardo Motta, and Mark Blackburn. NAT2TESTSCR: Test case generation from natural language requirements based on SCR specifications. *Science of Computer Programming*, 95:275–297, 2014.

[47] Dionny Santiago, Peter J. Clarke, Patrick Alt, and Tariq M. King. Abstract Flow Learning for Web Application Test Generation. In *Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*, page 49–55, New York, NY, USA, 2018. Association for Computing Machinery.

[48] Xuan Phu Mai, Fabrizio Pastore, Arda Göknil, and Lionel Briand. A natural language programming approach for requirements-based security testing. In *29th IEEE International Symposium on Software Reliability Engineering (ISSRE 2018)*. IEEE, 2018.

[49] Sahar Tahvili, Leo Hatvani, Michael Felderer, Wasif Afzal, Mehrdad Saadatmand, and Markus Bohlin. Cluster-Based Test Scheduling Strategies Using Semantic Relationships between Test Specifications. In *Proceedings of the 5th International Workshop on Requirements Engineering and Testing*, RET '18, page 1–4, New York, NY, USA, 2018. Association for Computing Machinery.

[50] Wolfgang Mueller, Alexander Bol, Alexander Krupp, and Ola Lundkvist. Generation of executable testbenches from Natural Language requirement specifications for embedded real-time systems. In *Distributed, Parallel and Biologically Inspired Systems*, pages 78–89. Springer, 2010.

[51] Ahlam Ansari, Mirza Baig Shagufta, Ansari Sadaf Fatima, and Shaikh Tehreem. Constructing Test cases using Natural Language Processing. In *2017 Third International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB)*, pages 95–99, 2017.

[52] Satoshi Masuda, Tohru Matsuodani, and Kazuhiko Tsuda. Syntactic rules of extracting test cases from software requirements. In *Proceedings of the 2016 8th International Conference on Information Management and Engineering*, pages 12–17, 2016.

[53] Xiaomeng Chen, Jinbo Wang, Shan Zhou, Panpan Xue, and Jiao Jia. Test Case Reuse based on ESIM Model. In *2021 8th International Conference on Dependable Systems and Their Applications (DSA)*, pages 700–705. IEEE, 2021.

[54] Avik Sinha, Amit Paradkar, Palani Kumanan, and Branimir Boguraev. A linguistic analysis engine for natural language use case description and its application to dependability analysis in industrial use cases. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pages 327–336. IEEE, 2009.

[55] Muhammad Waseem Anwar, Imran Ahsan, Farooque Azam, Wasi Haider Butt, and Muhammad Rashid. A Natural Language Processing (NLP) framework for embedded systems to automatically extract verification aspects from textual design requirements. In *Proceedings of the 2020 12th International Conference on Computer and Automation Engineering*, pages 7–12, 2020.

[56] Manish Motwani and Yuriy Brun. Automatically Generating Precise Oracles from Structured Natural Language Specifications. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 188–199, 2019.

[57] Cody Watson, Michele Tufano, Kevin Moran, Gabriele Bavota, and Denys Poshyvanyk. On Learning Meaningful Assert Statements for Unit Test Cases. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 1398–1409, 2020.

[58] S Preeti, B Milind, Medhini S Narayan, and Krishnan Rangarajan. Building combinatorial test input model from use case artefacts. In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 220–228. IEEE, 2017.

[59] Anunay Amar and Peter C. Rigby. Mining Historical Test Logs to Predict Bugs and Localize Faults in the Test Logs. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 140–151, 2019.

[60] Mathias Soeken, Robert Wille, and Rolf Drechsler. Assisted Behavior Driven Development using Natural Language Processing. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 269–287. Springer, 2012.

[61] Diksha Khurana, Aditya Koli, Kiran Khatter, and Sukhdev Singh. Natural Llanguage Processing: State of the art, current trends and challenges. *arXiv preprint arXiv:1708.05148*, 2017.

[62] Lance A Ramshaw and Mitchell P Marcus. Text chunking using transformation-based learning. In *Natural language processing using very large corpora*, pages 157–176. Springer, 1999.

[63] Bryan Ford. Parsing expression grammars: a recognition-based syntactic foundation. In *Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 111–122, 2004.

[64] Jerry R Hobbs, Douglas Appelt, John Bear, David Israel, Megumi Kameyama, Mark Stickel, and Mabry Tyson. Extracting Information from Natural-Language Text. *Finite-state language processing*, page 383, 1997.

[65] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[66] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[67] Shalom Lappin and Herbert J. Leass. An Algorithm for Pronominal Anaphora Resolution. *Comput. Linguist.*, 20(4):535–561, dec 1994.

[68] Christopher Kennedy and Branimir Boguraev. Anaphora for everyone: Pronominal anaphora resolution without a parser. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*, 1996.

[69] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[70] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From Word Embeddings To Document Distances. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 957–966, Lille, France, 07–09 Jul 2015. PMLR.

[71] Albert Gatt and Ehud Reiter. SimpleNLG: A realisation engine for practical applications. In *Proceedings of the 12th European Workshop on Natural Language Generation (ENLG 2009)*, pages 90–93, 2009.

[72] Dr Srinivas Perala and Ajay Roy. A review on Test Automation for test cases generation using NLP techniques. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, 12(6):1488–1491, 2021.

[73] John Smart. *BDD in Action: Behavior-driven development for the whole software lifecycle*. Simon and Schuster, 2014.

[74] Selenium Contributors. Page Object Models, January 2022.

[75] KR1442 Chowdhary. Natural Language Processing. *Fundamentals of artificial intelligence*, pages 603–649, 2020.