The Impact of NLP on Software Testing

Sai Deepak Reddy Konreddy

Student at Woxsen University

deepakreddy.konreddy_2022@woxsen.edu.in

Abstract: *The number of applications being built and deployed everyday are increasing by leaps and bounds. To ensure the best user/client experience, the application needs to be free of bugs and other service issues. This marks the importance of testing phase in application development and deployment phase. Basically, testing is dissected into couple of parts being Manual Testing and Automation Testing. Manual testing, which is usually, an individual tester is given software guidance to execute. The tester would post the findings as "passed" or "failed" as per the guidance. But this kind of testing is very costly and time taking process. To eliminate these short comings, automation testing was introduced but it had very little scope and applications are limited. Now, that Artificial Intelligence has been foraying into many domains and has been showing significant impact over those domains. The core principles of Natural Language Processing that can be used in Software Testing are discussed in this paper. It also provides a glimpse at how Natural Language Processing and Software Testing will evolve in the future. Here we focus mainly on test case prioritization, predicting manual test case failure and generation of test cases from requirements utilizing NLP. The research indicates that NLP will improve software testing outcomes, and NLP-based testing will usher in a coming age of software testers work in the not-too-distant times.*

Keywords: Software Testing, Artificial Intelligence, NLP, Manual Testing, Automation Testing, Test Case Prioritization, Predicting Manual Test Case Failure and Generation of Test Cases from Requirements.

1. Introduction

Quality Assurance is a critical component which deals with checking the reliability of software applications. Testing, on the other hand, is a time-consuming process. Quality Assurance Analysts perform majority steps of software testing systematically in their traditional style. Every one of those steps is test-case development, where an individual tester derives a series of test cases from structured (systematic) specifications, which are also written in linguistic form. Test-case architecture is often a time-consuming task, so professionals are keen to benefit from some (slightly) automated method for extracting test suites from specifications. Many tech giants may save a lot of time and money by automating process of manually generating and documenting test cases through specifications. Besides that, if programme specifications shift, test cases must be updated, which requires additional time and effort.

Throughout the software development life cycle, software testing is a time-taking and expensive operation. Automation is a potential option for lowering research costs while improving the accuracy of the procedure. Test automation can be used to solve a variety of testing issues, including the creation of test data/cases, test prioritization, test case execution, and repair.
Most current test automation techniques depend on metrics derived from the applications under test's source code or specification models. In general, however, a considerable portion of processing is performed manually. These test cases are written in plain English. They lay out a scenario for the system under test to execute. Device acceptance testing phase, for instance, is a form of manual testing in which manual testers verify fully prepared functionality.

Again, from standpoint of automation, manual testing is particularly fascinating since it is among the most costly types of testing since it allows a person to evaluate the pass/fail outcomes on all and every run of test cases. This may be feasible for simple programs with a tiny proportion of manual test cases, however as the device increases in complexity, executing all manual test cases with any function update would become impossible to complete under the time constraints. When the production team practices a fast release practice, the time constraint becomes much more serious. Challenges that measure the probability of a test case execution loss are referred to as "test case failure estimation." When we have a tight schedule, we will use these forecasts to determine which component of current test cases to execute. Or, more importantly, in what order do we operate test cases in order to identify bugs sooner?

The majority of test prediction analysis studies focus on different standard programme indicators, such as LOC, difficulty, and turnover, at the application or configuration stage. Even so, in manual vulnerability scanning, these objects are usually not available or not linked to the manual test cases. As a result, we're looking at other test case failure prediction methods that don't depend on programme source code or component documentation in this research.

The traditional implementation outcome of a testing phase is among the better steps that does not include any software or configuration records. Many experiments, particularly in the sense of test automation, show that a test case's prior failure is a very strong predictor of its potential failure. In the context, it was found that historiography strategies outperformed text metrics methods for prioritising manual test cases.

The main question is just how quality assurance could help Software Testing by generating increasingly efficient and easy test cases in a timely manner even while fulfilling market standards and customer expectations. AI and its main elements, such as ML and Natural Language Processing (NLP), can help with software testing in a variety of ways.

Automated testing saves time and improves precision. The software development sector is fast focusing on the automated generation and execution of test cases. One of the most important factors in order for automate testing is to insure that the tests are accurate and that you will get the greatest returns. With ML, DL, and NLP models and technologies, companies can improve research results and produce smarter and more reliable test cases for programmes, as well as increase testing scope.

2. Literature Review

Quality Assurance is an essential segment of ensuring the accuracy of software applications. All of those steps are test-case creation, in which a specific tester derives a series of test cases from written conditions, which are also written in natural language. Test-case architecture is often a time-consuming task, so testers are keen to benefit from some automated method for extracting test suites from specifications.

Many methods focused on NLP have been suggested in the research to decrease the manual effort of translating NL specifications to test cases. An input collection of specifications written in natural language is needed for this method. A collection of test cases is then immediately derived from the procedural criteria after a sequence of Natural Language Processing steps. Natural Language Processing methods were used in other software testing practices, such as the test oracle issue, in addition to the test-case design process. For example, the technique described created test oracles for evaluating three “extraordinary” behaviours from Javadoc comments, potentially saving individual testers time when deciding test oracles for that reason.

Many Natural Language Processing-based approaches and tools have been introduced in the last few decades to increase the reliability of software testing. Here, we use the term "Natural Language Processing-assisted software testing" to refer to all-Natural Language Processing-based strategies and resources that can help with any software testing operation, such as test-case modelling and test oracle operations.

Considering the enormous amount of expertise in the field of Natural Language Processing-assisted quality assurance, a professional or (fresh) investigator would find it difficult to study and gain a view of this field. As previously said, analysts are keen to receive assistance from any (partly) automated solution that will enable them and save resources when removing tests from specifications. Realizing that we can adjust a known methodology to forecast and optimise quality assurance in their very own sense can theoretically aid businesses and research technicians develop software testing performance. Besides that, the plurality of tech professionals, according to the writers' knowledge and the opinions of other researchers and academics, do not (aggressively) read research related articles. The system has the due features: Identifying testing, choosing to complete the major important testing goals, trying to devise a more cost-effective process to finish the existing specific goals, and achieving a substantial reduction as the evaluation expenditure is scaled down are all aspects of the system. As a result, we've seen personally how important research articles like this one are in providing a rundown of the whole field and serving as a "map" to the pool of information in the field, such that a professional can get a quick overview of what's there before having to search back and learn every article in the area.

Alike analytical linguistics and AI have an influential macro called NLP. NLP is described as a "set of analytical tools for detecting and rendering naturally produced texts in order to achieve human-like speech recognition". Speech comprehension, natural-language comprehension, and genetic production are common Natural Language Processing challenges.

While Nature Language constructs are directive from a lexical standpoint, the difficulty of interpretation totally makes language comprehension a difficult concept to grasp. According to one analysis, the phrase "List the prices of the company generated in ninety seventy-three with the sale of its products generated in ninety seventy-four." had four hundred fifty-five semantic-syntactic compiles. This exemplifies the difficulties of processing and analysis: while textual variant spellings are an innate ability in humans, it is hard to express to a computer all of the tiny complexities that comprise Natural Language. As a result, several posts of Natural Language Processing have arisen to examine various facets of Natural Language Processing from various perspectives. In the field of information engineering, Information Extraction is also the most commonly used Natural Language Processing solution.

The format of the material, which can range through text data to HTML, XML, and sub–Natural

Language, plays a role in Information Extraction; the methods that used handle the text should be defined; and the level of optimization in the gathering, tagging, and extracting phase must be taken into account.

Many methods focused on Natural Language Processing have been suggested throughout the research to minimize the energy work of translating specifications published in Natural Language to test cases. Automated testing for both the testcase design process is possible with such methods. An output collection of specifications written in NL is needed for this method. A collection of test cases is then automatically derived from the textual specifications after a sequence of Natural Language Processing steps. Of note, the accuracy of such a process is not flawless, necessitating the use of a professional tester for peer examination.

The form of Natural Language specifications used as feedback by a timely update is a critical consideration for Natural Language -assisted software testing. Few studies suggested a test case failure forecasting model for manual testing as a non-code/specification-based classifier for test collection, prioritization, and elimination. The findings revealed that combining a basic historiography function with a linear regression analysis would reliably predict the performance of test cases. Furthermore, the NLP-based technique will increase the precision of forecasts made by the base classifier. The research suggested a method that uses NLP to automatically generate test cases from practical requirements. The suggested program's aim was to minimize software testers' commitment and time spent testing the product.

Although some methods are based programme specifications to be presented in confined (governed) Natural Language, others accept specifications that enable for even more flexibility in the way sustainable development are published, as we will discuss in this research study. Managed Natural Language are subgroups of Natural Language generated by limiting the syntax and terminology in order to limit or remove uncertainty and ambiguity in the Natural Language Processing-based methodology for removing test cases from specifications.

The methods which use function variables as contributions to forecast improves the precision of standard historiography methods, which only describe each test case via an ID (no feature is mined). The studies suggested a test case failure forecasting model for manual testing as a non-code/specification-based classifier for test collection, prioritization, and elimination. The findings revealed that combining a basic historiography function with a linear regression analysis would reliably predict the performance of test cases. Furthermore, the NLP-based technique will increase the precision of forecasts made by the base classifier. The researchers suggested a method that uses natural language processing to automatically generate test cases from practical requirements. The suggested program's aim was to minimize software testers' commitment and time spent testing the product.

POS is a basic Natural Language Processing technique that we use to explore our proposal functionality. Aggregation Part of Speech is a technique for extracting terms from test cases and then analyzing them. And use a Cosine Similarity Reverse Report to measure them.

Over couple of years, there have been a slew of research on outlier detection. Material method and operational measurements, as well as other extraction database libraries methods, can be used to forecast a flaw. Researchers found several methods and approaches used to combine Artificial Intelligence with software testing and produce appropriate and good results throughout our study of past work in this area. This segment, as seen below, identifies the important sources in this context. Mostly on fault forecasting model, testing-related indicators had already been effectively implemented. Even so, in this article, we are more helpful in determining test case loss than fault forecasting. The biggest distinction is that I have been concentrating on test case-related functionality rather than generate relevant, believing that the software isn't usable. The scale of a test case, software and specification scope, and recent test case implementation outcomes are all commonly used research characteristics. And although our emphasis is on test case malfunction detection, the majority of the basic strategies and methods are based on fault prediction research.

Software testing is the most popular measurement of test case accuracy in the failed test sensing mechanism. For example, various reportage methods are often used to prioritise test cases. They

suggested a system that incorporates two main optimization techniques: test series and test case optimization. To improve regression testing, the results recommend a test case classification approach focused on k-means clustering. The research discovered that using the claim coverage criteria first improves the performance of the clustering-based method. They came to the conclusion that there is already a lot of work to be done to increase the efficiency of ML approaches used in software defect prediction. ML can also be utilized to know the viability of a test case. The study proved that it is difficult to describe test case viability. One benefit of language initiation, according to the research, is that the mediated comprehensives will show quality assurance analysts the types of sequences of words that result in prohibitively expensive test cases. Those methods, on the other hand, conclude that we have links to code relevant data for test cases, whether from prior implementations or from unit testing. In this report, I am focused on functional requirements that could only be derived using test cases in test automation environments.

In reality, the prior operation outcomes each case study are yet another common test case volatility forecasting function. The forecast approach assumes that the past outcomes of test case operations are documented and available. Test cases that have previously found flaws could be considered "tested candidates," and therefore may be allowed a higher chance of failing afterwards. As a result, a basic historiography criterion would also be whether or not someone test case since has lost. Good responses would be interpreted as failed assessments. This concept is commonly used in statistical research.

Message functionality are a modern classification that can operate from even the smallest repositories with no software, exposure, or implementation data. The written depiction of the testing ground is essentially the only prerequisite. For example, to describe manual test cases, the researchers had to use a based classification technology called LDA. Simulating test cases depending on their series of events is a popular method in test creation and prioritization in the field of incident applications such as web and smartphone apps.

In forecasting tests, classifiers like Linear and Non-Linear regression and machine learning techniques like Neural Network and Support Vector Machine (SVM) have been widely utilized. We are using the same concepts in this analysis and apply the most familiar strategies, such as Linear regression, Non-Linear regression, and Neural Networks, to learn a classifier model from the texture features per test case. Many methods focused on NLP have been suggested in the research to decrease the manual effort of translating NL specifications to test cases.

The linear regression method is one of the most widely used methods in analytics, and it is useful for precision of the results, especially because both the product class and the characteristics are quantitative. In a nutshell, the value class is a scaled version of the parameters and their values.

Due to various their ability to classify and recognize relevant data, Artificial Neural Networks have been used in a wide range of fields. Artificial Neural Networks are a collection of connected basic processing neurons whose structure is inspired by human mind neurons. The amounts of attached neurons, which are collected and modified by a classification algorithm from a series of class labels, preserve the network's computing power. Non-linear and multi-faceted input-output patterns can be classified by a qualified Artificial Neural Networks. A basic neuron sends information from n sources, each with its own weighted relation. The neuron then assesses the signal. The neuron then assesses the signals by applying the amount of each source and the corresponding relation quantities and comparing the total to its target value. The time domain F is then used to transform the measured vector. As a result, the transformed value is the neuron's contribution.

In the context of quality assurance, ML has different types of methods and algorithms. Learning algorithms used in Artificial Intelligence differ in terms of how they function, their mathematical and computational models, predictions, properties, precision, advantages and disadvantages, and the issue type they solve whether they solve grouping, regression, or other issues. MELBA (Machine Learning based improvement of Black box test specification), a partly automated recursive approach based on the C4.5 algorithm, is one of the recommended techniques highlighted. The test suites that resulted were dramatically more successful in terms of fault

identification despite only having a small increase in scale providing ten or more test method metrics and at least thirty documents. A system for valuation software test data creation using neural networks has been suggested in research. Here they try to determine which characteristics could be used to train a model that could forecast the exposure reached by automation systems using machine learning. SVR is by far the most reliable algorithm of the ones considered, according to the functionality for exposure predictive accuracy. The thesis also demonstrated that machine learning algorithms are a feasible alternative for predicting the future to some degree.

The research also demonstrated that machine learning algorithms are a feasible alternative for predicting exposure in automation tools to some degree. ML has also been used in the development of graphical user interfaces (GUI). They referenced how Artificial Intelligence can be used to evaluate GUIs instantly. In this case, hybrid genetic algorithms (HGA) had been utilized. To improve regression testing, the results recommend a test case classification approach focused on k-means clustering. The paper discovered that using the claim coverage criteria first improves the performance of the clustering-based method. They came to the conclusion that there is already a lot of work to be done to increase the efficiency of ML approaches used in software defect prediction. ML can also be used to determine the viability of a test case. The study proved that it is difficult to describe test case viability. One benefit of language initiation, according to the paper, is that the mediated comprehensives will depict quality assurance analysts the variety of sequences of words which result in prohibitively expensive test cases. The thesis has identified the most important review in the topic matter to learn and promote the relationship between the parameter defining object - oriented and the principle of shift predisposition.

Few researchers have also suggested a decline in the effort placed into software testing as a result of this strategy. They used SVM to learn a weighted classifier in research. When opposed to a randomized and systematic prioritization by research experts, the findings showed significant changes. The methodology was able to detect errors faster, making regression testing more effective. Models for forecasting the shift proneness of entity systems were built in research. Beginning of the software development phase, the built templates may be used to forecast transition classes. With a score of 0.877, Adaboost has the best accuracy.

For the preceding forms of testing, genetic algorithms have been used to generate test data: structure-based testing, temporal testing, practical testing, and protection testing. For single instance version systems, the research illuminated how to distinguish mere coincidence correct (CC) test cases, that execute the incorrect assertion but produce an accurate answer. The median recalling ratio and false positive ratio were eighty one percent and five percent, respectively, and the impact of CBFL was increased with improved processes in separate software iterations, according to the findings. Researchers have proposed test case prioritization methods in order to spot vulnerabilities in early stages, according to studies. TCP methods have been supported by the use of natural language processing. The findings indicate that each of these tactics will help increase software testing efficiency, with the risk.

Using an NLP technique, the thesis suggested a method for generating test cases from programme specifications articulated in linguistic form. The study concluded that the solution requires an automation approach and that the graphs created should be stored in the database such as Hadoop. They developed UnitTestScribe, an innovative methodology that incorporates static analysis, NLP, reverse splitting, and code summary strategies to produce descriptive NL explanations that concisely record the intent of system test methods. The goal was to see whether NLP tools could be used to further streamline the process. The goal was to see whether NLP tools could be used to better simplify the discovery of repeated fault files. The study looked at the detection capability of a massive fault control system and noticed that around forty percent of the labelled multiple copies could be identified. The Artificial Intelligence algorithms and techniques used in the identified studies and sources, as well as the test automation scope field and related parts.

3.Natural Language Processing and Quality Assurance

3.1 Classification of Test Cases using NLP:

The model followed here is language ignorant, meaning it can be extended to most languages with minor changes. The algorithm's purpose is to generate a numeric representation of a text that is constant in length independent of its length. Doc2Vec does also have an n-dimensional vector for each text, with each dimension translated as a function. Doc2Vec converts a non-fixed-length text into a vector and combines each word in the file. The Doc2Vec system has many variations. Here, we combine paragraph vectors with a decentralized bag of words model that employs unit-layer neural networks to predict when a word appears in a text. The PV-DBOW system avoids the background terms in the data, but it is forced to infer words automatically sampled from the paragraph in the output, as well as a paragraph ID. Furthermore, the Paragraph Matrix is a matrix in which each column contains a paragraph's vector. The descriptors for ‘‘seen" paragraphs for terms are stored in one Matrix. The algorithm is built by gradient descent to extract a document vector for ‘‘unidentified" paragraphs.[23]

We may obtain the learned values to construct feature vectors after training certain neural networks. These vectors are meant to reflect the document's definitions. Each component of the vector abstracts a number of words from the repository. Given a broad enough collection, the distance between vectors referring to grammatically identical documents is less than the distance between vectors referring to grammatically dissimilar documents. Doc2Vec is focused about utilizing neural networks to know vector representations of words. It is trained utilizing stochastic gradient descendent, with back-propagation to achieve the gradient. As a result, we get a wide variety of vectors every time we run the algorithm on a collection of documents. Nonetheless, between run to run, the comparative gap between grammatically similar documents is completely indistinguishable. This implies that the vectors again from two separate classes should attain a good distance with each pass, whereas the vectors from the very same class should maintain a smaller distance, which is the main thing for the classification stage. As a result, we operate Doc2Vec at about the same period on both training and testing documents in order to maintain the vectors compared. Using Doc2Vec and a clustering algorithm, we can build a connection between test case grammatic relatedness and computationally efficient.[23]

3.2 Generation of Test Cases using NLP:

Firstly, in this method, we have input in the form of requirement document. Then the input is fed to natural language processing which examines each and every case. Then examines each situation within the instance. Then we obtain a conjunctive phrase containing the keywords "if" and "then." Now we remove the noun from the conjunctive phrase and enter it into the test case table's 'Test Data' area. String a sentence in the ‘Expected Result' field that includes the word ‘then.' Change the ‘Status' Pass/Fail by comparing the real and predicted results.[24] By using this method we can generate test cases using natural language processing.

3.3 Predicting Test Case Failure using NLP:

In this method firstly we calculate history-based measures that is firstly, If the test cases passed in the latest iteration, add value ONE to this version's test cases; otherwise, assign value ZERO to recently failed or recent test cases. This is known as Simple History. Then, delegate the average of all prior iterations' simple history values. Instead of focusing on the most recent edition, this metric includes the entire past. This is called All History. Lastly designate a weighted average of simple history values from all past iterations, with latest editions receiving more weight. This is called Weighted History.[22]
Then the coverage of nouns is measured by using a POS tagger and confining only to nouns. Here each noun is used as a unit and the entire test suite is traversed there by calculating TF/IDF for each unit in every test case. Allocate every test case a metric equal to the amount of the TF-IDF of all the test case's units separated also by number of units in the test case. The TFIDF of

all specific nouns in test case is then computed as we go through all the test cases. The TF-IDF for certain nouns in each test case is then added together and divided by the total number of nouns throughout the test case. Eventually, we utilize the history-based and TF-IDF methods to estimate the PASS or FAIL with each update's test cases using the LR, NLR, and NN algorithms. [22]
As we previously stated and addressed, incorporating AI into software testing would unleash tremendous strength, moving the software testing and development sector in a different direction a period marked by ingenuity and adaptability.

4. Conclusion

Manual testing is a costly and time-consuming software testing operation that is necessary for gathering feedback from consumers on recently released functionality. NLP has the potential to efficiently interpret structured data collaborating with smart models and algorithms. NLP has also shown that it can improve app testing performance. In the not-too-distant future, NLP-driven testing will usher in a new age of QA practice. It can monitor and operate the majority of the testing sites, adding significant benefit to the testing outcome and delivering more reliable outcomes in a timely manner. There is no question that NLP will dominate and guide the QA and testing industry in the coming years. Here we discussed three scenarios or cases where NLP is being used or can be used in software testing industry, which is classification of test cases using NLP, predicting test case failure using NLP and generation of test cases using NLP which clearly depicts that usage of NLP in software testing not only decreasing the time but also increases the efficiency and quality of testing. There is a possibility in near future where, NLP based testing would bind to emerging innovations (such as Cloud, IoT, Big Data). This would help us derive the best practices that complement the customer application in order to produce more reliable and smart test cases.

Furthermore, the Natural Language Based approach will increase the precision of forecasts much further than the know only using history. To my understanding, Natural Language Processing has been used on manual test case files for failed test forecasting, classification, generation of test cases and the findings have been positive. I wish to enhance this research on other frameworks and build on various Natural Language Processing-based functionality to derive functionality parameters much more efficiently from test cases.

References:

[1] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, “Test case prioritization: an empirical study,” in Software Maintenance, 1999. (ICSM ’99) Proceedings. IEEE International Conference on, 1999, pp. 179–188.

[2] A. Hudaib, B. Hammo, and Y. Alkhader, “Towards software requirements extraction using natural language approach,” in Proceedings of the 6th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems, ser. SEPADS’07. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS), 2007, pp. 155–160. [Online]. Available: http://dl.acm.org/citation.cfm?id=1353801.1353829

[3] N. E. Fenton and M. Neil, “A critique of software defect prediction models,” IEEE Transactions on Software Engineering, vol. 25, no. 5, pp. 675–689, Sep 1999.

[4] N. Nagappan, L. Williams, M. Vouk, and J. Osborne, “Early estimation of software quality using in-process testing metrics: a controlled case study,” in ACM SIGSOFT Software Engineering Notes, vol. 30, no. 4. ACM, 2005, pp. 1–7.

[5] S. Elbaum, G. Rothermel, and J. Penix, “Techniques for improving regression testing in continuous integration development environments,” in Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014, pp. 235–245.

[6] T. B. Noor and H. Hemmati, “A similarity-based approach for test case prioritization using historical failure data,” in Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on. IEEE, 2015, pp. 58–68.

[7] H. Hourani, A. Hammad and M. Lafi, "The Impact of Artificial Intelligence on Software Testing," *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, Amman, Jordan, 2019, pp. 565-570, doi: 10.1109/JEEIT.2019.8717439.

[8] H. Hemmati, Z. Fang, M. V. Mntyl, and B. Adams, “Prioritizing manual test cases in rapid release environments,” Software Testing, Verification and Reliability, vol. 27, no. 6, pp. e1609–n/a, 2017, e1609 stvr.1609. [Online]. Available: http://dx.doi.org/10.1002/stvr.1609

[9] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, “Featurerich part-of-speech tagging with a cyclic dependency network,” in Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1, ser. NAACL ’03. Stroudsburg, PA, USA: Association for Computational Linguistics, 2003, pp. 173–180. [Online]. Available: https://doi.org/10.3115/1073445.1073478

[10] G. Salton, A. Wong, and C. S. Yang, “A vector space model for automatic indexing,” Commun. ACM, vol. 18, no. 11, pp. 613–620, Nov. 1975. [Online]. Available: http://doi.acm.org/10.1145/361219.361220

[11] D. Y. Lee, "Corpora and discourse analysis," in Advances in discourse studies, ed: Routledge, 2008, pp. 86-99.

[12] P. Koehn, Statistical machine translation: Cambridge University Press, 2009.

[13] L. Berti-Equille and J. Borge-Holthoefer, "Veracity of Data: From Truth Discovery Computation Algorithms to Models of Misinformation Dynamics," Synthesis Lectures on Data Management, 2015.

[14] J. L. Leidner, "Current issues in software engineering for natural language processing," in Proceedings of workshop on Software engineering and architecture of language technology systems, 2003, pp. 45-50.

[15] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. Mcclosky, "The stanford core-NLP natural language processing toolkit," in ACL System Demonstrations, 2014.

[16] D. Graham and M. Fewster, Experiences of Test Automation: Case Studies of Software Test Automation: Addison-Wesley, 2012.

[17] V. Garousi, S. Taşlı, O. Sertel, M. Tokgöz, K. Herkiloğlu, H. F. E. Arkın, et al., "Experience in automated testing of simulation software in the aviation industry," IEEE Software, In Press, 2017.

[18] V. Garousi and F. Elberzhager, "Test automation: not just for test execution," IEEE Software, In Press, 2017.

[19] V. Garousi and M. V. Mäntylä, "When and what to automate in software testing? A multi-vocal literature review," Information and Software Technology, vol. 76, pp. 92-117, 2016.

[20] M. Shahbaz, P. McMinn, and M. Stevenson, "Automated discovery of valid test strings from the web using dynamic regular expressions collation and natural language processing," in International Conference on Quality Software, 2012, pp. 79-88.

[21] Garousi, V., S. Bauer and M. Felderer. "NLP-assisted software testing: A systematic mapping of the literature." *Inf. Softw. Technol.* 126 (2020): 106321.

[22] H. Hemmati and F. Sharifi, "Investigating NLP-Based Approaches for Predicting Manual Test Case Failure," *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, Västerås, Sweden, 2018, pp. 309-319, doi: 10.1109/ICST.2018.00038.

[23] Sahar Tahvili, Leo Hatvani, Enislay Ramentol, Rita Pimentel, Wasif Afzal, Francisco Herrera, "A novel methodology to classify test cases using natural language processing and imbalanced learning, Engineering Applications of Artificial Intelligence", Volume 95,2020,103878,ISSN 0952-1976,https://doi.org/10.1016/j.engappai.2020.103878.

[24] A. Ansari, M. B. Shagufta, A. Sadaf Fatima and S. Tehreem, "Constructing Test cases using Natural Language Processing," *2017 Third International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB)*, 2017, pp. 95-99, doi: 10.1109/AEEICB.2017.79723

[25] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of Duplicate Defect Reports Using Natural Language Processing," 29th International Conference on Software Engineering (ICSE07),2007.