AUTOMATIC GENERATION OF TEST CASES FROM USE-CASE SPECIFICATION USING NATURAL LANGUAGE PROCESSING

MARKO PRIBISALIĆ[1]

[1] University of Rijeka, Department of Informatics, Rijeka, Croatia, e-mail: marko.pribisalic@gmail.com

Abstract Software testing often targets natural language specification documents. The initial assumption is that automation of creating test cases from natural language specification is of benefit to speed up testing enabling a better coverage of all possible test scenarios. The proposed enabling principle for automatic generation of test cases is automatic retrieval of logic for the interaction with an application. After retrieving, interaction logics is transformed into decision tables. Next, from decision tables it is possible to automatically generate test cases. Because in our research we target Croatian natural language, the assumption is that is it necessary to create a new approach to achieve the set goal. The main research questions posed in this paper are: "Is it possible to automatically generate test cases from use-case specifications written in the Croatian language?"; "Natural language processing tools for automatic test-generation save the tester's time and effort while improving the quality and coverage of the test cases?". The expected results are to defining the method that using test-case generation tools reduce the time and effort for software testers and improve the test coverage of requirements.

Keywords: test case, semantic analysis, logical specification, natural language processing, croatian language.

University of Maribor Press

DOI https://doi.org/10.18690/978-961-286-362-3.52
ISBN 978-961-286-362-3

1 Introduction

Software testing is a fundamental activity to ensure the quality of software systems. Test engineers conduct most (if not all) phases of software testing manually. One of those phases is test-case design in which the human tester uses written (formal) requirements (use-case specification), written often in natural language (NL), to derive a set of test cases (Garousi et.al, 2015). IEEE (1998) recommends the standard for software requirements specifications in praxis. The standard describes consideration for producing a good software requirements specification, parts of them and provide templates. Recommendations and standards intended to help: software users (customers) to accurately describe what they wish to obtain; software suppliers to understand exactly what the customer wants; individuals to develop and define the format and content of software requirements specification (SRS) outline for their organizations. Usually, test cases are created manually, so test case coverage depends upon each individual tester's skill, preferences and knowledge. Under these circumstances, it is difficult to create high coverage test cases. High coverage test cases systematically consider all functionalities described in corresponding specifications. The second characteristic stemming from leaning exclusively on tester's skills and experience is a demand to translate business rules specified in the free form of natural language info formal use-case tests. In order to cover all functionalities described in use-case specifications, it is important to find the correct technique to retrieve software logics needed to test all possible interactions between users and software. Semantic analysis technique facilitates the retrieval of interaction logics enabling automatic creating of test cases. Once the interaction logics are retrieved, they can be represented in the form of a decision table which subsequently enables the creation of the test cases (Masuda et.al, 2013). Previous research studies (Saeki (1989), Sneed (2007), Kim (2008), Uetsuki (2013)) consider software specifications written in the English language. Engaging languages other than English requires substantial adjustment of proposed techniques. For instance, if specifications are in the Croatian language, for automatic generation of test-case scenarios it is a prerequisite to include NLP techniques for the Croatian language. Finally, by including adequate semantic analysis technique and tools for automated test case generation, we believe that we will get a successful model for automated test case generation from specifications written in Croatian. Specifically, we aim of combining semantic analysis technique and tools for automated test case generation to create a model for automated test case generation from specifications written in Croatian.

2 Problem definition

Creating test cases is a challenging and time-consuming endeavour in the process of software testing which consists of test case generation, test execution, and test evaluation. In the software test life cycle, test case generation takes 40-70% of that process (Kulkarni & Joglekar, 2014). In the testing process, the practice is to manually write the test cases based on the provided functional requirements of the software. Under these circumstances, it is challenging to create high coverage test cases, while testing requires covering and testing all functionalities in software as described in use-case specifications. It is not uncommon that testers created test cases that do not match which the product owner's needs because not all the testers have prior knowledge of how the system is working (Broek et.al, 2014). Auto generation of test cases can contribute to saving money and time, improving the quality of testing and ensuring better test coverage. Automatically generated test case preferably ensure easier maintenance and reuse of test cases if deemed necessary.

These difficulties have led to the incorporation of different sub-fields/disciplines of NLP to be ported and tested for automatic generation of test cases form specifications written in natural language. Though natural language processing tasks are closely intertwined, they are frequently subdivided into sub-fields/categories for convenience. In this study, we are mainly tasked with information extraction (IE). Information extraction (IE) is the task of automatically extracting structured information from unstructured and/or semi-structured machine-readable documents and other electronically represented sources. (Berti-Equille & Borge-Holthoefer, 2015). In planned work focus will be on automatic test-case generation from use-case specifications written in the Croatian language. The reason stems from the present situation in Croatian IT companies that are focused on the Croatian market, hence services, software and specifications as well are written in Croatian. Croatian is different from the English language, so it is worth noticing some major differences. "Croatian is a highly flective Slavic language and words can have seven different cases for singular and seven for plural, genders and numbers. The Croatian word order is mostly free, especially in non-formal writing. These features place Croatian among morphologically rich and mostly free word-order languages. English grammar has minimal inflection compared with most other Indo-European languages, therefore it is considered to be analytic. English word order is almost exclusively subject-verb-object. Both languages are characterized by an accentuation system developed on syllables" (Martinčić-Ipšić et.al, 2016).

The research questions we put forth are:

RQ 1 - Is it possible to automatically generate test cases from use-case specifications written in the Croatian language?

Sneed (2007) presented the automated testing of software against natural language requirements. The approach was to analyze requirements and automatically extract test cases. The tool is the text analyzer developed by the author. The text analyzer scans through the requirements text to pick out potential test cases based on keywords and sentence structure. The Sneed (2007) approach has an error discovery rate of 89%. It is a much cheaper and more efficient way of exposing errors than a pure manual test case selection process. In this case, over 95% of the potential functions were covered, which means that the approach achieved higher functional test coverage. Saeki et.al (1989) presented a software development process from natural language specification. That was an approach to solve problems about natural language specification by the process, which was defined as "design" and "elaborate". The approach was not applied for automatic derivation of test cases from specifications in natural language processing. Kim et.al (2008) presented a measurement of the level of quality control activities in software development. Overall quality control score can be calculated by evaluation each of fourteen quality metrics which are adopted as a key performance index constituting of quality control level. As quality metrics are suggested the measurements of specification documents, for examples, document defect density, document reusability and so on. Natural languages are inherently ambiguous which makes the requirements documented in use-case specification document unclear. This unclear requirement causes that developers develop software which is different from the specification and discrepant with customer needs (Sabriye & Zainon, 2017). Uetsuki et.al (2013) presented an efficient software testing method by decision table verification. They propose a knowledge creation method of software logic extracted automatically from the programme source code. All possible programme paths are extracted from source code, then converted into a decision table. The logic verification can be performed in a short time by comparing the decision table with a specification of software. Matsuodani (2012) showed that retrieving logics from specification documents into decision tables is beneficial and suggested the opportunities for future use of decision table. The semantic analysis technique can detect ambiguity in the logic of specification documents and feedback measurements for document quality. The semantic analysis technique can feedback the analyst to write manually specification documents more precise. The more exact we can describe logic in the specification documents in advance, the less workload to fix of incorrect logic will be needed. When experts did not understand the logic of the sentences, it must be something incorrect in the sentences, it is called incorrect logic (Masuda et.al, 2015).

Our focus will be on applying the results of Croatian natural language processing into software testing. We plan to propose a semantic analysis for testing logics retrieval from Croatian use-case specifications. Garousi et.al (2018) points out that many authors in previous research as dominant language use English for requirements specification. Yet several research studies have been reported for other languages. Masuda et al. in (Masuda et.al, 2013; Masuda et.al, 2015) have focused on automatic test-case generation in Japanese (Masuda et.al, 2013; Masuda et.al, 2015) while Yang et al. in (Yang et.al, 2017) develop testing from Chinese specification documents. It is worth noticing that each natural language has specifics requiring adjusted of semantic analysis techniques for logics retrieval. The semantic analysis technique, besides facilitating automatic testing procedures can provide feedback on improvement of the manually written specification. The more exact we can express software logic in specification documents, the less workload will be in testing and subsequently fixing of incorrect logic. To prove that the semantic technique works properly, we plan to do a comparative analysis, which will be focused on automatic test-case generation from English and Croatian use-case specifications.

RQ 2 - NLP tools for automatic test-generation save the tester's time and effort while improving the quality and coverage of the test cases?

Ther are several possibilities for evaluation of the quality of the test cases generated from natural language. The first is based on an activity diagram and activity graph. Activity diagrams, also known as control flow and object flow diagrams, are one of the UML (unified modelling language) behavioural diagrams. These diagrams are suitable for business process modelling and can easily be used to capture the logic of a single use case, the usage of a scenario, or the detailed logic of a business rule (OMG, 2020). Activity graph is a graphical method for showing dependencies between tasks (activities) in a project (Oxford, 2016).

The parameters to assess the quality of the test cases generated from natural language or activity diagram are the number of test cases created and the effort required to create the test cases. Activity Diagrams are converted into activity graphs for the purpose of visualization and to implement the algorithms. Since the test cases are generated from the activity graph, the following criteria determine where the testing process terminates and how good a test coverage does the generated activity graph provide (Mingsong, 2006):

- **Activity Coverage** - The generated test cases must ensure that all the activity states in the diagram are covered sequentially, from the initial node to the final node, at least once;
- **Path Coverage** - The generated test cases must ensure that all possible paths from the initial node to the final node are covered at least once in the activity graph.
- **Transition Coverage** - The generated test cases must ensure that all possible transitions/edges from the initial node to the final node are covered at least once in the activity graph.
- **Predicate Coverage or Branch Coverage -** In the case of a decision node, the generated test cases must cover the true and false logic paths of the condition.

If the generated test cases are executed and linked to defects in the software workflow, parameters such as rework ratio, defect detection percentage, and test execution rates can be used to verify the quality of the test cases.

The **effort required** to generate the test cases is the average time taken to generate the test cases (Elghondakly et.al, 2015).

Test Case Productivity (TCP) is defined as the ratio of the number of test steps/test case generated to the effort (in hours) taken to generate these test steps (Gulechha, 2009).

3 Importance of research (why it is worthy of doctoral research)

The focus of our research is the automatic generation of test cases from use-case specifications using natural language processing to Croatian documents. Software testing according to natural language specification documents is the standard approach for system and acceptance testing (Sneed, 2007). Since we are targeting the Croatian natural language, this is currently an unexplored problem. In this stage of the research, we are aiming to find the right methodology, methods and techniques. At this point, it is not clear to conclude with certainty whether the studied problem will require a creation of a novel technique or will be possible to adapt the existing technique. From previous research dealing with other languages, eg Japanese (Masuda et.al, 2013), we see that this technique needs to include the testing logics retrieval from harmonization between natural language processing techniques and software testing. So our first step will include the study of portability of existing technique to new software testing domain and in Croatian natural language, preferably followed by the development of new technique. Harmonization between natural language processing techniques and software testing is crucial to ensure success to develop test cases.

4 Methodology

In the first phase, we plan to conduct a survey in the form of a systematic literature mapping (classification) and systematic literature review. Objectives are to summarize the state-of-the-art in NLP-assisted software testing. It could open potential directions in utilizing NLP-based techniques for Croatian language and providing an overview of the research areas. For data analysis, we plan to use the automated content analysis approach, which is based on algorithms that use probabilistic topic models (Blei, 2012). Additionally, for the literature review we plan to use a text analytics tool, which automatically analyses text documents to identify high-level concepts and provide key ideas and insights from the text, eg "Leximancer" (Leximancer, 2020). We believe that the literature review will help us to find methods to achieve the previously defined aims.

Based on relevant papers, we plan to propose a semantic analysis technique of logics retrieval for Croatian use-case specifications. The analysis technique can feedback on how we write manually specification documents precisely. The more correct we can describe the logic on specification documents in advance, the less workload to fix incorrect logic.

After the definition of the analysis technique, we plan to verify their relevance by using NLP tools. The success of generating test cases from requirements strongly depends on the right selection and usage of an appropriate NLP method. Initially, we plan to use the Stanford Parser, which offers a broad variety of NLP-related functionalities (Manning et.al, 2014). Stanford Parser supports Croatian (human) languages and it is a frequently used parsing tool. Afterwards, there is a possibility of adding another NLP tools like "NLTK" (NLTK, 2020)).

Ultimately, we plan to carry out the experiment in the Croatian Financial Agency FINA - Informatics Sector / IT Service Development Department / Department of verification solutions who is in charge of public service testing on a daily basis. Software requirements which testers using in everyday work are defined in use-case specifications written on Croatian natural language. We plan to verify that the defined model provides answers to our research questions, and what are the specific benefits of our approach.

5 Preliminary/Expected results

The expected results of using test-case generation tools are: (1) reducing the time and effort for testers and (2) improving the test coverage of requirements. We plan to conduct the testing experiment in two groups of testers. The first group of 2 testers will be given 10 use-case specification for manual test case generation, and the same use-case specification will be given to the other set of 2 testers for automatic test generation, such as proposed in the doctoral dissertation. The results obtained from both groups of testers will be compared with ground truth which will be prepared in advance. Expected results are that the automatic method takes less time as compared to the manual method (quantified by the number of test cases and test steps). We opt to ensure that the increased number of test cases, test steps, and Test Case Productivity generated using the automatic method provide more coverage of the functionality than the manual method. We will evaluate the coverage of each of the test cases written by the user manually and generated by the user automatically. Also, we expect that we will show that the semantic analysis technique could retrieve testing logics from Croatian natural language specification documents. Finally, by using the correct semantic analysis technique and NLP tools for automated test case generation, we believe that we will develop a successful model for automated test case generation from a use-case specification written in Croatian.

References

Berti-Equille, L., Borge-Holthoefer, J. (2015). Veracity of Data: From Truth Discovery Computation Algorithms to Models of Misinformation Dynamics, Synthesis Lectures on Data Management, Morgan & Claypool Publishers https://doi. org/10.2200/S00676ED1V01Y201509DTM042

Blei, D. M. (2012). Probabilistic topic models. Communications of the ACM. https://doi.org/10.1145/2133806.2133826.

Broek, R., Bonsangue, M., Chaudron, M., and Merode, H. (2014). Integrating testing into agile software development processes," in Model-Driven Engineering and Software Development (MODELSWARD), 2014 2nd International Conference on, pp. 561–574, IEEE

Elghondakly, R., Moussa, S., Badr, N. (2015). Waterfall and agile requirements-based model for automated test cases generation," in Intelligent Computing and Information Systems (ICICIS), 2015 IEEE Seventh International Conference on, pp. 607-612, IEEE

Garousi, V., Bauer, S., Felderer, M. (2018) NLP-assisted software testing: A systematic mapping of the literature Retrieved March 02, 2020, from https://www.researchgate.net/publication/325557382_NLP-assisted_software_testing_a_systematic_review

Garousi, V., Coşkunçay, A., Can, A.B., Demirörs, O., (2015). A survey of software engineering practices in Turkey, *Journal of Systems and Software,* vol. 108, pp. 148-177

Gulechha, L. (2009). Software Testing Metrics, available at: https://fdocuments.in/document/software-testing-metrics-558444fda73ef.html, Retrieved March 20, 2020, from https://fdocuments.in/document/software-testing-metrics-558444fda73ef.html

IEEE , I. C. S. S. E. S. Committee, & I.-S. S. Board, (1998). IEEE Recommended Practice for Software Requirements Specifications, in Institute of Electrical and Electronics Engineers

Kim, C., Kim, S.-M. Song, K.-W. (2008) Measurement of Level of Quality Control Activities in Software Development [Quality Control Scorecards]", in IEEEConvergence and Hybrid Information Technology, 2008. ICHIT'08. International Conference on, pp.763-770

Koehn, P. (2009). Statistical machine translation: Cambridge University Press

Kulkarni, P., Joglekar, Y. (2014). Generating and analyzing test cases from software requirements using nlp and hadoop, International Journal of Current Engineering and Technology (INPRESSCO)

Lee, D., Y. (2008). Corpora and discourse analysis in Advances in discourse studies, ed: Routledge, pp. 86-99.

Leximancer. (2020). https://info.leximancer.com/. Retrieved March 18, 2020, from https://info.leximancer.com/

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

Martinčić-Ipšić, S., Margan, D., Meštrović, A. (2016). Multilayer network of language: A unified framework for structural analysis of linguistic subsystems Physica. A, Statistical mechanics and its applications 457 pp.117-128

Matsodani, T. (2012). Application of Decision Table to Software Logic with Designing and Testing", in Reliability Enginerring Association of Japan, pp.397-404

Masuda, S., Matsuodani, T., Tsuda, K. (2013). A Method of Creating Testing Pattern for Pair-wise Method by Using Knowledge of Parameter Values, in Procedia Computer Science, pp.521-528

Masuda, S., Futoshi I., Nobuhiro, H., Tohru, M., Kazuhiko, T. (2015). Semantic analysis technique of logics retrieval for software testing from specification documents. International Conference on Software Testing, Verification and Validation Workshops, pp. 1-6

Mingsong, C., Xiaokang, Q., Xuandong, L. (2006). Automatic test case generation for UML activity diagrams, in Proceedings of the 2006 international workshop on Automation of software test, pp. 2-8, ACM

NLTK (2020). Natural Language Toolkit — NLTK 3.5 documentation, available at: https://www.nltk.org/ Retrieved March 22, 2020, from https://www.nltk.org/

OMG (2020). Unified Modeling Language, , available at: http://www.uml.org/, Retrieved March 28, 2020, from http://www.uml.org/

Oxford (2016). A Dictionary of Computer Science (7th edition), edited by Andrew Butterfield and Gerard Ekembe Ngondi, Oxford University Press

Sabriye A. O. J. and Zainon, W. M. N. W. (2017). "A Framework for Detecting Ambiguity in Software Requirement Specification" in 2017 8th International Conference on Information Technology (ICIT), Amman, Jordan, 2017, pp. 209–2013. https://doi.org/10.1109/ICITECH.2017.8080002

Saeki, M., Horai, H., Enomoto, H. (1989). Software development process from natural language specification, in ACMProceedings of the 11th international conference on Software engineering, pp.64-73

Sneed, J H., M. (2007). Testing against natural language requirements, in IEEEQuality Software, 2007. QSIC'07. Seventh International Conference on, pp.380-387

Uetsuki, K., Matsuodani, T., Tsuda, K. (2013). An efficient software testing method by decision table verification, in International Journal of Computer Applications in Technology, pp.54-64

Yang, Y., Huang, X., Hao, X., Liu, Z., Chen, Z. (2017). An Industrial Study of Natural Language Processing Based Test Case Prioritization," IEEE International Conference on Software Testing, Verification and Validation, pp. 548-549