

A Systematic Mapping Study on Techniques for Generating Test Cases from Requirements

Alessandro Rodrigues, Jéssyka Vilela^a and Carla Silva^b

Centro de Informática, Universidade Federal de Pernambuco (UFPE), Av. Jornalista Aníbal Fernandes, s/n – Cidade Universitária, Recife-PE, Brazil

Keywords: Systematic Mapping Study, Test Cases, Requirements, Generation.

Abstract: Context: Software testing can be costly for organizations. Techniques and tools that deal with the automatic generation of test cases provide a way to reduce the efforts employed and the time-to-market, in addition to increasing the quality of the software. Objective: This work aims to investigate the literature regarding techniques used to generate test cases from requirements automatically. Method: We performed a Systematic Mapping Study (SMS) using the Snowballing technique to investigate these techniques, the information presented in the test plan/case, the languages used to specify the requirements, and finally, the steps proposed by the techniques. Results: techniques such as Model-based testing (MBT) and Natural Language Processing (NLP) are the most used, mainly based on requirements specified through Natural Language that can be structured or not, as well as UML (Unified Modeling Language) diagrams. We also extracted and presented a series of languages and tools developed, and some are under development that perform this generation.

1 INTRODUCTION

Software permeates the various spheres of the world, which creates a dependence on the proper functioning of these utilities in our society. Bank transactions, air travel, and automatic radiation doses are just a few examples of critical software in which failures and wrong software behavior can be disastrous. Thus, it is important to have ways to ensure the correct quality and functioning in developing a program.


Singhs and Singhs (2012) define software testing as “a process, or a series of processes designed to make sure computer code does what it was designed to do and that it does not do anything unintended” (Singh and Singh, 2012). Testing software does not mean there is no defect in its execution, but the controlled execution that test techniques provide aims to expose as many failures as possible with the minimum of resources invested, whether computational or human (Neto and Claudio, 2007).


Studies suggest that 50% of the time, cost, and effort of the software development process are spent on Software Testing (Hooda and Chhillar, 2014), which makes this task considered expensive for organiza-

tions. Using automatic test generation techniques directs the organization to improve quality, reduce rework, saving resources and also a significant amount of money (Fewster and Graham, 1999). In this context, automatic software testing tools are appropriate due to improved time-to-market and accuracy.

This work investigates the current state of techniques developed to generate test cases from requirements. We conducted a Systematic Mapping Study (SMS) (Kitchenham and Charters, 2007) using the snowballing technique (Wohlin, 2014) to find relevant studies to understand how this process has been conducted. We collected results from 41 articles directly related to the topic. We hope the results exposed here will bring insights to the evolution of automatic testing tools from requirements, such as Smartest.

This paper is organized as follows: Section 2 presents the background and related work. Section 3 presents the research methodology. The results and the analysis related to our research questions are presented in Section 4. Threats to validity are presented in Section 5. Finally, we present our conclusions and future works in Section 6.

^a  <https://orcid.org/0000-0002-5541-5188>

^b  <https://orcid.org/0000-0002-0597-3851>

2 BACKGROUND AND RELATED WORK

This section introduces a basis for understanding this work. Initially, we describe test case generation, as well as some common techniques used in the automatic generation of test cases, such as model-based testing (MBT) and natural language processing (NLP). Then, we present the related work.

2.1 Test Case Generation

A test case describes a particular condition to be tested and comprises a set of test inputs, execution conditions, and expected results developed for a specific objective (Craig and Jaskiel, 2002). These cases are elaborated from artifacts such as specifications, design, and implementations (Shanthi and Kumar, 2011) and are used to verify if the system developed corresponds to the requirements defined, contributing to project success and customer satisfaction.

The generation of test cases can be performed manually, automatically, or semi-automatically, the latter being preferable to save time and costs during the test cycle (Mustafa et al., 2021). It is common for this generation to occur based on requirements. In this case, the tests are derived from the requirements without considering the internal structure of the implementation, which may be functional and non-functional requirements (Vaysburg et al., 2002).

To find as many failures as possible, tests must be conducted systematically, and test cases must be designed using disciplined techniques. Some of these techniques used to generate test cases from requirements automatically are:

- **Model-Based Testing (MBT):** In this technique, the automatic generation of test artifacts is based on information extracted from the software models (El-Far and Whittaker, 2002), which include in their specification the characteristics of the software that will be tested.
- **Natural Language Processing (NLP):** it is concerned with giving computers the ability to understand text and spoken words in the same way as humans by combining computational linguistics - based on human language rules - with statistical, machine learning, and deep learning models (IBM, 2023).

2.2 Related Work

The generation of tests cases or test plan from requirements specifications has been studied in some works.

Tahat et al. (Tahat et al., 2001) presented a new test generation-based approach that accepts individual requirements expressed in Specification Description Language (SDL) and textual formats automating the creation of system models through SDL requirements.

Nebut et al. (Nebut et al., 2004) provided a method, formal model and a prototype tool to automatically generate specific functional and robust test cases for a product from the requirements of a product line. This technique, in this study, is associated with a slight formalism to express the product family, in a declarative but unambiguous manner, the mutual facilities between requirements, in terms of pre/post conditions.

Júnior and Silva (de Santiago Júnior and da Silva, 2017) introduce a translation method called HiMoST, which generates software test cases through Model Checking. It starts with Harel's Statecharts for behavioral modeling and translates them into a general structure based on NuSMV language. CTL properties are formalized using specification patterns and a Combinatorial Interaction Testing algorithm. A cost-effectiveness evaluation comparing four patterns/pattern scopes revealed that the Precedence Chain pattern with Global scope performed the best.

Agile practices has also been investigated in the context of software testing (Coutinho et al., 2022)(Fazzolino et al., 2018). The objective of Coutinho, Andrade e Machado (Coutinho et al., 2022) was to understand how Requirements Engineering (RE) and Software Testing (ST) activities are performed in agile teams and their contribution to software quality. A survey was conducted with 72 software industry professionals experienced in RE and ST. The findings revealed that Agile RE and Agile ST activities still need improvement in coordinated implementation in projects. There were both similarities and differences between academic research and industry practices. Agile practices, specific to RE and ST, as well as other practices related to processes, approaches, and software development, were used in a complementary manner.

Fazzolino et al. (Fazzolino et al., 2018) was to test two enterprise systems, SISBOL and SISDOT, from the Brazilian Army, using different approaches and tools. The challenges in testing enterprise systems include considering user requirements, business rules, user interaction, legacy system integration, and database access. Test engineers face conflicting decisions regarding test levels and the use of mock objects. The study found that using executable specifications, as recommended in Behavior Driven Development, was valid for test-first scenarios and system requirements validation. Additionally, it helped identify new execution paths even after the source code had

been implemented, suggesting the adoption of BDD approach in later stages of development.

The study by Mustafa et al. (Mustafa et al., 2021) resembles ours by presenting an SRM on generation of test cases based on requirements from 2000 to 2018 (Mustafa et al., 2021). The study was carried out to acquire areas of knowledge about requirements-based test case generation and identify future research. As a result, the review showed that 53% of journal papers, 42% of conference papers and 5% of book chapters address requirements-based tests.

Additionally, Farooq and Tahreem (Farooq and Tahreem, 2022) conducted a systematic literature review (SLR) to evaluate the approaches and techniques that are used for automated test cases using user requirements and stories and find the best that is not just friendly but also useful for minimizing the Software cost and human effort. In their work, the taxonomy was also presented based on the generation of test cases with requirements-based techniques and tools.

As for the last two works (Mustafa et al., 2021) (Farooq and Tahreem, 2022), although they also performed a SLR on test case generation, they present some differences when compared to ours. The study by Mustafa et al. (Mustafa et al., 2021) also focuses on identifying existing challenges in generating test cases from requirements, in addition to presenting how the transformation of requirements from natural language to the formal model occurs. However, our research questions are more detailed, as well as the search string we used. Also, the most recent papers they selected were published in 2018. Likewise, Farooq and Tahreem (Farooq and Tahreem, 2022) focused on UML based approaches published until 2021 and in discovering challenges regarding these approaches. Our scope is a bit larger than theirs.

3 RESEARCH METHODOLOGY

The research method followed for this work is a Systematic Mapping Study (Galvão and Pereira, 2014). To conduct the SMS and to answer our research questions, presented in Table 1, we chose the Snowballing technique, which refers to the procedure in which, starting from an initial set of papers, a list of paper references (backward) or paper citations (forward) are used to identify additional papers (Wohlin, 2014).

The search string used in this first step derives from the topic of our interest. This string was also formed through the logical connectors AND, for aggregation of keywords and OR for inclusion of the respective synonyms (Kitchenham and Charters, 2007). As a result, we get the following search string:

Table 1: Research questions.

ID	Research Question	Motivation
RQ1	What techniques are used to create a test plan/case from requirements?	Know the possible techniques in the area.
RQ1.1	What information is present in the test plan/case?	Learn what information is most used in test cases.
RQ1.2	Which tool supports the generation of test plans/cases from requirements?	Get to know the existing tools and see the common points between them.
RQ1.3	How are requirements specified?	See how the requirements are defined.
RQ1.4	Are requirements represented graphically or textually?	Know which approaches are the most used to represent the requirements.
RQ1.5	How are test cases generated from requirements?	Knowing how to apply the creation of test cases, having as a starting point the requirements.

("requirement") AND ("software testing" OR "test case" OR "test plan") AND ("aligning" OR "derivation" OR "integration" OR "generation") AND ("method" OR "approach" OR "technique" OR "methodology" OR "tool")

By applying our search string, in December 2022, in the IEEE Xplore and ACM digital libraries, we selected the studies S1 to S5 considering their adherence to the theme, having a good number of references and citations, as listed in Table 2, with differences in the applicability of the techniques of automatic test cases generation from requirements, with different authors and year of publication, so that we have greater diversity.

A subsequent step was the choice of a paper, among the five selected, to start backward and forward snowballing. As the work S1 is appropriate to the theme, besides having many referenced papers and being one of the most cited, we decided to define it as the initial paper.

Table 2: Details of selected papers.

Work	Duplicate authors?	Year	# of ref	# of citations
Work 1 S1	No	2015	37	44
Work 2 S2	No	2020	85	9
Work 3 S3	No	2011	24	8
Work 4 S4	No	2003	6	46
Work 5 S5	No	2017	33	19

Once we had selected the work, we started the snowballing process. We applied the inclusion and exclusion criteria, which are used to reduce the probability of bias and select studies capable of answering the research questions (Kitchenham and Charters, 2007). The criteria adopted in this work were:

- Inclusion criteria: Primary studies; Research and papers from conferences and journals; Studies

from 2003 onwards; Studies reporting approaches to test plan/case creation; Studies that report approaches in creating an automated test plan;

- Exclusion criteria: Short Papers (< three pages); Articles not in English or Portuguese; Secondary studies; Duplicate studies; Redundant work by the same author; Studies whose focus is not on creating test cases; Studies irrelevant to the research, taking into account the research questions shown in Table 1.

We provide all references and citations in supplementary material: <https://doi.org/10.5281/zenodo.10724942>. After this step, we had 316 works in total. On the other hand, when applying the initial filter, 130 was subtracted from the total. In the supplementary material, an interested reader may find a figure depicting the number of papers excluded by criteria versus which paper it was derived from.

We analyzed the title and abstract in each iteration, considering the inclusion and exclusion criteria. 112 works were excluded from this stage. Although they are mostly related to generating test cases, they do not adequately fit the topic of this SMS. For example, some of them dealt with later stages of test case generation. In the supplementary material, an interested reader may find a figure depicting the distribution of these exclusions by source work.

Finally, as we noticed that there were still more than 70 papers left, we also read the paper's introduction and conclusion sections. We excluded another 32 studies according to the aforementioned inclusion and exclusion criteria. Then, we had a total of 41 papers since only 1 paper was excluded from the full reading. We assigned the code S# for each selected paper where S means selected and # the number of the paper. The selected papers are in Table 3.

4 RESULTS AND ANALYSIS

Regarding the distribution of the selected papers by digital libraries where they were published, IEEE (17 papers), ACM (10 papers) and Springer (9 papers) are the most common. In the next sections, we present the results for each research question.

4.1 RQ1: What Techniques are Used to Create a Test Plan/Case from Requirements?

In Table 4, the 1st column contains the technique or approach used, and the 2nd column presents the papers that mention it as a technique used in the test

Table 3: List of selected papers.

ID	Reference
S1	(Yue et al., 2015)
S2	(Liu and Nakajima, 2020)
S3	(Giannakopoulou et al., 2011b)
S4	(Nebut et al., 2003)
S5	(Ahsan et al., 2017)
S6	(Ali and Hemmati, 2014)
S7	(Ali et al., 2010)
S8	(Barros et al., 2011)
S9	(Hametner et al., 2012)
S10	(Hesari et al., 2013)
S11	(Wang et al., 2015)
S12	(Zhang et al., 2014)
S13	(Bernardino et al., 2017)
S14	(Wang et al., 2020)
S15	(Kriebel et al., 2018)
S16	(Elghondakly et al., 2015)
S17	(Gupta and Mahapatra, 2021)
S18	(Gröpler et al., 2021)
S19	(Zhang et al., 2018)
S20	(Santos et al., 2019)
S21	(Mustafa et al., 2017)
S22	(Jorge et al., 2018)
S23	(Arruda et al., 2020)
S24	(Liu and Nakajima, 2010)
S25	(Zhang and Liu, 2013)
S26	(Legard et al., 2004)
S27	(Bouquet et al., 2006)
S28	(Giannakopoulou et al., 2011a)
S29	(Gligoric et al., 2010)
S30	(Pickin et al., 2007)
S31	(Aniculaesei et al., 2019)
S32	(Vani et al., 2015)
S33	(Reales et al., 2012)
S34	(Mateo et al., 2009)
S35	(Lamancha et al., 2010)
S36	(Verma and Beg, 2013)
S37	(Dwarakanath and Sengupta, 2012)
S38	(Wang et al., 2015)
S39	(Malekzadeh and Ainon, 2010)
S40	(Riebisch and Hubner, 2005)
S41	(Carvalho et al., 2013)

generation plan. This data was extracted from the explicit mention by the authors. Below, we show some quotes that confirm the use of these techniques.

"[...] 'Vibration-Method' or simply 'V-Method', for automatic generation of test cases and test oracle from model-based formal specifications [...]." S2

"[...] automated framework for model-driven testing that can be applied in MDE and SPL [...]." S39

"[...] uses text mining and symbolic execution for test case generation." S20

MBT is the most used technique in the papers with a total of 10 citations. It is related to Software Product Line. Some of them were derived from a paper S4 of our seed set, which may explain such presence. We

Table 4: Techniques used to create a test plan/case from requirements.

Technique/Approach	#citations
MBT (Model-Based Testing)	10
NLP (Natural Language Processing)	7
UMTG (Use Case Maps-based Test Generation)	4
Symbolic Execution	3
RTCM (Requirements Traceability for Change Management), Synthesis, RUCM (Rational Unified Process Use Case Model)	2 each
Model Checking	2
V-method, Keyword-driven, Text-Mining, Specification-based Test Case Generation Process, MBTCC (Model-Based Test Case Classification), TCG (Test Case Generation, SOFL (Software Functional Language), BZ-TT, Algorithms, Reuse of test, CEG (Constraint-Driven Exploratory Testing), Transformation algorithms, Metamodeling	1 each

also have that some of the aforementioned techniques make use of another technique presented in Table 4. These are the cases of Text Mining and RTCM that use NLP. Algorithms are explicitly mentioned in S36, but it is expected in others, mainly those that propose some tool. In addition, we have works S20 S38 that mention more than one technique.

4.1.1 RQ1.1: What Information is Present in the Test Plan/Case?

To answer RQ 1.1 from the extraction of data stored in the online spreadsheet, we built a word cloud with the purpose of showing the most used fields of the requirements. This information was extracted from the figures and tables contained in the papers. We noticed that fields such as “name”, “number”, “precondition” are the most used in the templates and tables presented. For the sake of space, the word cloud is presented only in the supplementary material.

4.1.2 RQ1.2: Which Tool Supports the Generation of Plan/Cases from Requirements?

Table 5 shows the tools used in the selected works. Of the 41 papers, only 8 do not refer to the tools used.

4.1.3 RQ1.3 How are Requirements Specified?

Natural language with a pattern was the most used approach to specify requirements as input for test case generation, with 18 references. In the studies, it was mentioned as “Restricted Natural Language”, “Structured Natural Language”, SRC, among others.

The second most used approach to specify the requirements is the Natural language without a pattern, with 11 references. In those that use these requirements as input, these requirements appeared

Table 5: Tools used for the generation of plan/cases from requirements (RQ 1.2).

Tools
aToucan4Test, TRUST, SysML, Litmus, TGV, BZ-TT, SOFL, Java Pathfinder tool, Umlaut, TaRGeT, Fitness, UMTG plugin, NLReq2TC, Zen tool, TestLink, WIKI, CASE tool, LTS-BT, DNF, Automatic Specification Based Test Case Generation prototype, S29, QVT, SPF, UDITA, PralinTool, T-VEC, JPL, ANSYS SCADE KCG, ANSYS SCADE Test Environment e ANSYS SCADE Design Verifier, Mutant Generation System, Trace2TestCase, LTL2SCADE and LTL2Trap, Reform, ModGen and TCG, Style Parser, Link Grammar parser, GATE, NLTK, ADL translator, TestSpec, Randoop, EvoSuite, T-VEC tool, CNL Parser, Automated test case generation based on state charts, Test Generation with Verification Technology, TestGen-Intermediate Format (IF), Test Case Generation (TCG) and TaRGeT

as “Freestyle Natural Language”, “Predicate Expression”, and others. In third place appears “UML”, with 10 references. In this case, we identify these by expressions such as “Class Diagram”, “Sequence Diagram” and “UML Diagram”. Below we show some quotes that confirm the use of these requirements.

“... the input to the automation process consists of freestyle NL documents.” S27

“Our approach consists of translating ADEPT models into Java programs that can be symbolically executed by SPF.” S3 “Since TRUST generates test cases from UML state machines.” S11

DSL, Domain specific data structure, Models extracted from ADEPT tool and Model-oriented notations such Z and B has only one citation each.

4.1.4 RQ1.4: Are Requirements Represented Graphically or Textually?

From the answers of RQ1.3, it is expected that “Textually” would appear more often (70,7%) than “Graphically” (29,3%). The answer was often not explicit in the paper, so we derived it from other information presented, such as a screenshot of a tool showing an SRS document as being the input to generate the test cases, thus being classified as “Textually”.

4.1.5 RQ1.5: How are Test Cases Generated from Requirements?

RQ1.5 concerns how to achieve automatic generation of test cases from requirements. The results of this research question is presented in the supplement material. Some of the responses extracted from the papers are described below.

From UMTG Technique: *“...five steps: 1. Test scenario description and user stories were being processed as inputs through a NL parser. 2. Dependencies procured from the parser were being utilized in the creation of UML activity diagram, which were aimed to demonstrate the functionality flow. 3. Ac-*

tivity graph was generated from activity diagram that comprised of all the activities delineated as nodes. 4. An algorithm entitled as depth first search (DFS) is deployed for traversing the activity graph from initial to final activity state, in order to discover all possible authentic paths. 5. Possible functional test cases were then generated by utilizing the test paths (final outcome).” S21

From MBT Technique: “...is made up of 5 different steps: 1. Step 1: Product Line modelling; 2. Step 2: Sequence diagram enrichment; 3. Step 3: domain-level test scenario generation; 4. Step 4: domain level test case generation; 5. Step 5: product level test case generation and automation.” S37

From NLP Technique: “The tool works on each requirement sentence and generates one or more Test Cases through a five step process 1) The sentence is analyzed through a syntactic parser to identify whether it is testable; 2) A compound or complex testable sentence is split into

5 THREATS TO VALIDITY

(Wohlin et al., 2000) pointed out four types of threats to validity: Construct, Conclusion, Internal, External. In this section, we discuss the threats to validity of our study based on these four types.

Construct: it refers to the relationship between the theory behind the experiment and the observations. The data came from 9 research bases, such as IEEE, ACM and SPRINGER, providing diversity. Even so, we understand that there are other databases of qualities that could add even more value to our work.

Conclusion: it deals with how certain we can be that the technique we use in our investigation really is related to the actual result we observe. To mitigate the effects of this threat, the SMS process was carefully designed and discussed among the authors to minimize the risk of exclusion from relevant studies.

Internal: it contemplates the fact that there may be other factors that caused the result. In this sense, SMS and the Snowballing technique aim to minimize internal validity. Additionally, the selection process was carried out in such a way that when there were doubts in the application of some criterion, the study was not eliminated and moved on to the next step.

External: it contemplates the point about whether we can generalize the results outside the scope of our study. This threat was mitigated with the use of selection criteria and the diversity of the selected studies. Even so, more studies need to be conducted.

6 CONCLUSIONS AND FUTURE WORK

Test case generation techniques can increase product quality, reduce product costs and the time-to-market (Fewster and Graham, 1999). This process can be done based on requirements and may involve the use of tools and procedures. In this sense, the present work investigated the state of the art to identify which techniques, tools, requirements and procedures have been used. The objective lies in the fact of being able to bring relevant information to the academic community and assist in the evolution of tools for the automatic test cases generation from requirements, such as SmarTest, because it can reveal paths to follow through the information we have in the literature.

For the selection of studies that we used to extract the data, we used the Snowballing technique, where we performed an iterative procedure with the references and citations of a base set of five chosen papers. This returned us 316 papers, which from readings and use of inclusion and exclusion criteria, gave us a total of 41 references to be used in extracting information.

As for the results, we noticed that there is a preference for the use of techniques and approaches involving MBT and NLP. In addition, we concluded that “Natural Language with a pattern”, “Natural Language without a pattern” and “UML” are the most used requirements as input, which also favored the Textual type to be found more often. Additionally, there are not as many template information fields and other artifacts revealed in the works we investigated. Finally, we can say that there is no great intersection between the tools used, given that most of them are tools proposed by the authors.

As future work, we intend to perform a deeper analysis of NLP based tools, as they have a great potential in a software industry that specifies requirements using natural language.

REFERENCES

- Ahsan, I., Butt, W. H., Ahmed, M. A., and Anwar, M. W. (2017). A comprehensive investigation of natural language processing techniques and tools to generate automated test cases. In *Proc. of the Second Intl. Conf. on Internet of things, Data and Cloud Computing*, pages 1–10.
- Ali, S. and Hemmati, H. (2014). Model-based testing of video conferencing systems: challenges, lessons learnt, and results. In *2014 IEEE Seventh Intl. Conf. on Software Testing, Verification and Validation*, pages 353–362. IEEE.
- Ali, S., Hemmati, H., Holt, N. E., Arisholm, E., and Briand,

- L. C. (2010). Model transformations as a strategy to automate model-based testing-a tool and industrial case studies. Technical report, Simula Research Lab.
- Aniculaesei, A., Vorwald, A., and Rausch, A. (2019). Using the SCADE toolchain to generate requirements-based test cases for an adaptive cruise control system. In *ACM/IEEE 22nd Intl. Conf. on Model Driven Eng. Languages and Systems Companion (MODELS-C)*, pages 503–513.
- Arruda, F., Barros, F., and Sampaio, A. (2020). Automation and consistency analysis of test cases written in natural language: An industrial context. *Science of Computer Programming*, 189:102377.
- Barros, F. A., Neves, L., Hori, É., and Torres, D. (2011). The ucsCNL: A controlled natural language for use case specifications. In *SEKE*, pages 250–253.
- Bernardino, M., Rodrigues, E. M., Zorzo, A. F., and Marchezan, L. (2017). Systematic mapping study on MBT: tools and models. *IET Software*, 11(4):141–155.
- Bouquet, F., Dadeau, F., and Legeard, B. (2006). Automated boundary test generation from jml specifications. In *FM 2006: Formal Methods*, volume 4085, pages 428–443. Springer.
- Carvalho, G., Falcão, D., Barros, F., Sampaio, A., Mota, A., Motta, L., and Blackburn, M. (2013). Test case generation from natural language requirements based on SCR specifications. In *Proc. of the 28th Annual ACM Symp. on Applied Computing (SAC '13)*, pages 1217–1222.
- Coutinho, J., Andrade, W., and Machado, P. (2022). A survey of requirements engineering and software testing practices in agile teams. In *Proc. of the 7th Brazilian Symp. on Systematic and Automated Software Testing*, pages 9–18.
- Craig, R. D. and Jaskiel, S. P. (2002). *Systematic software testing*. Artech House.
- de Santiago Júnior, V. and da Silva, F. (2017). From state-charts into model checking: A hierarchy-based translation and specification patterns properties to generate test cases. In *Proc. of the 2nd Brazilian Symp. on Systematic and Automated Software Testing*, pages 1–10.
- Dwarakanath, A. and Sengupta, S. (2012). Litmus: Generation of test cases from functional requirements in natural language. In *Natural Language Processing and Inf. Systems*, volume 7337, pages 58–69. Springer.
- El-Far, I. K. and Whittaker, J. A. (2002). *Model-based software testing*.
- Elghondakly, R., Moussa, S., and Badr, N. (2015). Waterfall and agile requirements-based model for automated test cases generation. In *Seventh Intl. Conf. on Intelligent Computing and Inf. Systems (ICICIS)*, pages 607–612. IEEE.
- Farooq, M. S. and Tahreem, T. (2022). Requirement-based automated test case generation: Systematic literature review. *VFAST Transactions on Software Eng.*, 9(4):133–142.
- Fazzolino, R., de Faria, H. M., Amaral, L. H. V., Canedo, E. D., Rodrigues, G. N., and Bonifácio, R. (2018). Assessing agile testing practices for enterprise systems: A survey approach. In *Proc. of the III Brazilian Symp. on Systematic and Automated Software Testing*, pages 29–38.
- Fewster, M. and Graham, D. (1999). *Software test automation*. Addison-Wesley.
- Galvão, T. F. and Pereira, M. G. (2014). Revisões sistemáticas da literatura: passos para sua elaboração. *Epidemiologia e serviços de saúde*, 23:183–184.
- Giannakopoulou, D., Bushnell, D., Schumann, J., et al. (2011a). Formal testing for separation assurance. *Annals of Mathematics and Artificial Intelligence*, 63:5–30. Pages 1–26.
- Giannakopoulou, D., Rungta, N., and Feary, M. (2011b). Automated test case generation for an autopilot requirement prototype. In *2011 IEEE Intl. Conf. on Systems, Man, and Cybernetics*, pages 1825–1830. IEEE.
- Gligoric, M., Gvero, T., Jagannath, V., Khurshid, S., Kuncak, V., and Marinov, D. (2010). Test generation through programming in UDITA. In *ACM/IEEE 32nd Intl. Conf. on Software Eng.*, pages 225–234.
- Gröpler, R., Sudhi, V., García, E. J. C., and Bergmann, A. (2021). NLP-based requirements formalization for automatic test case generation. In *CS&P*, volume 21, pages 18–30.
- Gupta, A. and Mahapatra, R. P. (2021). A circumstantial methodological analysis of recent studies on nlp-driven test automation approaches. *arXiv preprint arXiv:2104.04843*.
- Hametner, R., Winkler, D., and Zörtl, A. (2012). Agile testing concepts based on keyword-driven testing for industrial automation systems. In *IECON 2012-38th Annual Conf. on IEEE Industrial Electronics Society*, pages 3727–3732. IEEE.
- Hesari, S., Behjati, R., and Yue, T. (2013). Towards a systematic requirement-based test generation framework: Industrial challenges and needs. In *21st IEEE Intl. Requirements Eng. Conf. (RE)*, pages 261–266. IEEE.
- Hooda, I. and Chhillar, R. (2014). A review: study of test case generation techniques. *Intl. Journal of Computer Applications*, 107(16):33–37.
- IBM (2023). What is natural language processing (nlp)? topic page about natural language processing. Technical report.
- Jorge, D., Machado, P., Alves, E., and Andrade, W. (2018). Integrating requirements specification and model-based testing in agile development. In *26th Intl. Requirements Eng. Conf. (RE)*, pages 336–346. IEEE.
- Kitchenham, B. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE-2007-01, School of Computer Science and Math., Keele University.
- Kriebel, S., Markthaler, M., Salman, K. S., Greifenberg, T., Hillemacher, S., Rumpe, B., and Richenhagen, J. (2018). Improving model-based testing in automotive software engineering. In *Proc. of the 40th Intl. Conf. on Software Eng.: Software Eng. in Practice*, pages 172–180.
- Lamancha, B., Usaola, M., and Velthuis, M. (2010). An automated model-driven testing framework-for model-

- driven development and software product lines. In *ENASE*, pages 112–121.
- Legéard, B., Peureux, F., and Utting, M. (2004). Controlling test case explosion in test generation from formal models. *Software Testing, Verification and Reliability*, 14(2):81–103.
- Liu, S. and Nakajima, S. (2010). A compositional approach to automatic test case generation based on formal specifications. In *Fourth Intl. Conf. on Secure Software Integration and Reliability Improvement*, pages 147–155.
- Liu, S. and Nakajima, S. (2020). Automatic test case and test oracle generation based on functional scenarios in formal specifications for conformance testing. *IEEE Transactions on Software Eng.*, 48(2):691–712.
- Malekzadeh, M. and Ainon, R. N. (2010). An automatic test case generator for testing safety-critical software systems. In *2nd Intl. Conf. on Computer and Automation Eng. (ICCAE)*, pages 163–167.
- Mateo, P., Polo, M., and Lamancha, B. (2009). Automatic generation of test cases in software product lines. In *Intl. Conf. on Enterprise Inf. Systems*.
- Mustafa, A., Wan-Kadir, W. M., and Ibrahim, I. (2017). Comparative evaluation of the state-of-art requirements-based test case generation approaches. *Intl. Journal on Advanced Science, Eng. and Inf. Technology*, 7(4-2):1567–1573.
- Mustafa, A., Wan-Kadir, W. M., Ibrahim, N., Shah, M. A., Younas, M., Khan, A., and Alanazi, F. (2021). Automated test case generation from requirements: A systematic literature review. *Computers, Materials and Continua*, 67(2):1819–1833.
- Nebut, C., Fleurey, F., Le Traon, Y., and Jézéquel, J. M. (2004). A requirement-based approach to test product families. In *Software Product-Family Eng.: 5th Intl. Workshop, PFE 2003*, pages 198–210. Springer Berlin Heidelberg.
- Nebut, C., Pickin, S., Le Traon, Y., and Jézéquel, J. M. (2003). Automated requirements-based generation of test cases for product families. In *18th IEEE Intl. Conf. on Automated Software Eng., 2003. Proceedings*, pages 263–266. IEEE.
- Neto, A. and Claudio, D. (2007). Introdução a teste de software. *Engenharia de Software Magazine*, 1:22.
- Pickin, S., Jard, C., Jeron, T., Jezequel, J.-M., and Le Traon, Y. (2007). Test synthesis from UML models of distributed software. *IEEE Transactions on Software Eng.*, 33(4):252–269.
- Reales, P., Polo, M., and Caivano, D. (2012). Model based testing in software product lines. In *Enterprise Inf. Systems*, volume 102, pages 270–283. Springer.
- Riebisch, M. and Hubner, M. (2005). Traceability-driven model refinement for test case generation. In *12th IEEE Intl. Conf. and Workshops on the Eng. of Computer-Based Systems (ECBS'05)*, pages 113–120.
- Santos, E., Costa, L., Aragão, B., Santos, I., and Andrade, R. (2019). Extraction of test cases procedures from textual use cases to reduce test effort: Test factory experience report. In *Proc. of the XVIII Brazilian Symp. on Software Quality*, pages 266–275. ACM.
- Shanthi, A. V. K. and Kumar, D. G. M. (2011). Automated test cases generation for object oriented software. *Indian Journal of Computer Science and Eng.*, 2(4):543–546.
- Singh, S. K. and Singh, A. (2012). *Software testing*. Vandana Publications.
- Tahat, L. H., Vaysburg, B., Korel, B., and Bader, A. J. (2001). Requirement-based automated black-box test generation. In *25th Annual Intl. Computer Software and Applications Conf. - COMPSAC 2001*, pages 489–495. IEEE.
- Vani, V., Mahalakshmi, G., and Antony, J. (2015). Generation of patterns for test cases. *Indian Journal of Science and Technology*, 8(24):1.
- Vaysburg, B., Tahat, L. H., and Korel, B. (2002). Dependence analysis in reduction of requirement based test suites. In *Proc. of the 2002 ACM SIGSOFT Intl. Symp. on Software Testing and Analysis*, pages 107–111.
- Verma, R. and Beg, M. (2013). Generation of test cases from software requirements using natural language processing. In *6th Intl. Conf. on Emerging Trends in Eng. and Technology*, pages 140–147.
- Wang, C., Pastore, F., Goknil, A., Briand, L., and Iqbal, Z. (2015). Automatic generation of system test cases from use case specifications. In *Proc. of the Intl. Symp. on Software Testing and Analysis*, pages 385–396.
- Wang, C., Pastore, F., Goknil, A., and Briand, L. C. (2020). Automatic generation of acceptance test cases from use case specifications: an NLP-based approach. *IEEE Transactions on Software Eng.*, 48(2):585–616.
- Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proc. of the 18th Intl. Conf. on Evaluation and Assessment in Software Eng.*, pages 1–10.
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2000). *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA.
- Yue, T., Ali, S., and Zhang, M. (2015). Rtcn: a natural language based, automated, and practical test case generation framework. In *Proc. of the Intl. Symp. on Software Testing and Analysis*, pages 397–408.
- Zhang, H., Wang, S., Yue, T., et al. (2018). Search and similarity based selection of use case scenarios: An empirical study. *Empirical Software Eng.*, 23:87–164.
- Zhang, M., Yue, T., Ali, S., Zhang, H., and Wu, J. (2014). A systematic approach to automatically derive test cases from use cases specified in restricted natural languages. In *8th Intl. Conf. on System Analysis and Modeling: Models and Reusability, SAM*, pages 142–157. Springer Intl. Publishing.
- Zhang, W. and Liu, S. (2013). Supporting tool for automatic specification-based test case generation. In *Structured Object-Oriented Formal Language and Method. SOFL 2012*, volume 7787, pages 12–25. Springer.