A Review of Automatic Generation of Test Cases from Use Case Specification

Surafiel Habib Asefa (✉ surafiel.habib@aau.edu.et)
Addis Ababa University  https://orcid.org/0009-0009-8946-957X

Systematic Review

Keywords: Software Testing, Automatic Test Case Generation, Use-Case Specification

Posted Date: June 2nd, 2023

DOI: https://doi.org/10.21203/rs.3.rs-2885830/v2

License: © This work is licensed under a Creative Commons Attribution 4.0 International License. Read Full License
Abstract

Software testing is an integral phase in the software development life cycle (SDLC). It refers to the process of verifying and evaluating the function of a software product. Among the testing activities, test case generation affects the effectiveness and efficiency of the testing process and requires a significant amount of effort and time. To reduce the effort and time involved in the manual test case generation process, significant research effort has been dedicated to the automatic generation of test cases. Although the majority of the proposed approaches are based on UML models, some works have presented a test case generation approach based on specifications. This literature review (LR) discusses use case specification-based automatic test case generation approaches and the methods used to validate them. Additionally, the review shows how the approaches differ in addressing some current issues in software testing.

1. Introduction

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. It involves executing a program or application with the intent of finding software bugs (errors or defects). Depending on the software methodology, testing can be implemented at any time in the development process. However, most of the test effort occurs after the requirements have been defined and the coding process has been completed. The testing stage is one of the most sensitive stages of software development and includes about 50% of the software development costs [1, 2].

The testing process includes test case generation, test case selection, test case execution, and test case evaluation. Among the activities, test case generation is the most intellectually demanding and critical, as it can have a strong impact on the effectiveness and efficiency of the whole testing process [3, 4]. Thus, the main focus during software testing is to reduce the number of test cases into a manageable test set and to take calculated risks about what is and what isn't important to test [3].

Over the past two decades, significant research effort has been devoted to automatic test case generation to reduce the cost and effort involved in manual test case generation and validation. Automatic test generation not only reduces the cost and effort of testing but also ensures that all requirements are properly covered in test cases [5, 6]. The majority of the automatic test case generation approaches presented require system specifications to be captured as UML behavioral models such as activity diagrams, state charts, and sequence diagrams [6,7]. Approaches that do not utilize behavioral modeling generate executable system test cases by exploiting the behavioral information implicitly described in use case specifications [8], a textual description of the functionality provided by the system that is widely used for communicating requirements among stakeholders and, facilitating communication with customers.
This literature review (LR) seeks to explore the approaches used for the automatic generation of test cases from use case specification and the method of validation for each approach. The rest of the paper is organized as follows. Section 2 discusses the methodology for the literature review. Section 3 presents the results of the review. Section 4 presents a discussion of the results and Section 5 provides a conclusion to the LR and gives future work suggestions.

2. Methodology

This study adopted an approach presented by Kitchenham [11] to conduct an SLR on the research topic. The process of the review consists of three phases: plan review, conduct review, and document review. In the first phase, the following research questions (RQs) are raised:

RQ1: What approaches have been introduced for automatically generating test cases from use case specification?

RQ2: How have the approaches been validated?

Five different online databases were used in this study to search for the most relevant papers published between 2012 and 2023. These include IEEE Xplore, Springer Link, Science Direct, ResearchGate, and Google Scholar. For the search, a Boolean search string was used on the databases to find the answers to the research questions above. The search term was described as follows: ((automatic OR automated) AND (“test case generation” OR “generating test case” OR “generate test case”)) AND (“use case specification” OR “use case”).

The inclusion criteria for research works were: the use of only English between the specified years in a recognized journal or conference and the exclusion criteria was if the generation approach is not based on use case specifications or is a manual approach. In the second phase, a paper selection process was conducted using title and abstract screening, free availability, duplicate removal, and full-text selection based on the inclusion and exclusion criteria. All papers that did not have any correlation with the specific domain were removed from the selected papers. Followed by a quality assessment (QA) process done to evaluate the selected papers’ quality. The QA questions for this review are: Does the study include use case specification-based automated test case generation? Is the study capable of answering the research questions? Is the approach well described so that the author or others can validate it in later research? and Is the paper well/appropriately referenced? A scoring technique was adopted, where three possible scores can be given to an answer to each QA question: “1” for “Yes”, “0.5” for “Partly”, and “0” for “No”. Three relevant papers were selected for this study based on the sum of the total score for all four QA questions. From the chosen papers: authors’ names, published year, paper’s title, objective, methodology, findings, and future works were extracted. The extracted data were synthesized to answer the research questions. Finally, in the third phase, the review was written.

3. Results
RQ1: What approaches have been introduced for automatically generating test cases from use case specification?

[12] Presents Use Case Specification Language Test Case Generation (USLTG), an approach for automatically generating system test cases from precise use case specification. The use case specification is modeled with use case specification language (USL). The model automatically generates other software artifacts by making model transformations. USLTG implements a scenario generation algorithm to generate a set of constrained use case scenarios from a USL model. Test input data suites are generated for each scenario by solving Object Constraint Language (OCL) constraints in the corresponding use case scenario. The use case scenarios and test data suites are transformed into a Test Case Specification Language (TCSL) model that provides notations for reusable test cases that are understandable for non-technical stakeholders and precise enough for system testing.

[13] Proposes Use Case Modelling for System Tests Generation (UMTG) to automatically generate test cases by generating behavioral information and high-level operation descriptions from use cases. Requirement specifications are used to manually create a domain model as a UML class diagram which is checked for completeness to ensure the generation of correct and complete test input. UMTG further processes the use cases with the OCL constraints to generate a Use Case Test Model. An algorithm covers all the paths of the Use Case Test Model and generates a list of scenario and object Diagram pairs. UMTG searches for the attributes in the object diagram that appear in the constraints of the scenario to identify the whole set of input values for the test case. Using a mapping table, UMTG parses each high-level operation description and automatically generates executable test cases.

[14] Proposes an automatic approach to generate test cases from the use case description model. Based on [15] as a template to create a complete description of a use case by including all the relevant details which case will be used as input to the Algorithm of Control Flow Diagram (ACFD) to generate a control flow diagram. The control flow diagram is used as input for the Proposed Tool of Generating Test Paths (PTGTP) which generates test paths (test cases). The algorithm of Test Paths Optimization (ATPO) is executed to determine the test path achieving the maximum coverage and a genetic algorithm was used to optimize the test paths and evaluate the adequacy of generated test cases.

RQ2: How have the approaches been validated?

The review finds that the methods used for the validation of approaches are experiment (33.3%) and comparative analysis (66.6%).

In [12] an experiment was used to generate system test cases for a use case specification. The approach generated five test scenarios and eight test input data suites, which combined to form eight test cases.

In a case study [13], UMTG is used to generate test cases from the use cases that correspond to the main functionalities of BodySense, a safety-critical automotive software developed by IEE. The results of the
case study indicate that the approach is more effective in terms of time consumption and considers the
approach feasible in an industrial context.

In [14], a case study is used to demonstrate the feasibility and efficiency of the proposed approach. To
evaluate the results of the case study, the authors compare the proposed approach with the approach in
[16]. The paper finds that the proposed approach is more efficient in terms of time consumption and
achieves 100% transition coverage.

4. Discussion

The presence of ambiguity is one of the defining characteristics of natural language. A common cause of
software project failure is inadequately produced specifications commonly written in plain English or
other natural languages [17]. UMTG [13] uses Restricted Use Case Modeling (RUCM) which provides
restriction rules and specific keywords constraining the use of natural language in use cases so that test
cases cover every specification. The use of a modeling language reduces the ambiguities of natural
language and the risk of project failure resulting from ambiguous software specifications. According to
[18] modeling specifications can reduce the time and effort spent in test case generation, thus the use of
models in the approach [13] also decreases the time and effort required by software engineers to add
missing specifications into new or existing test cases. Likewise, the approach in [14], uses a template [15]
to create a complete description of use cases by including all the relevant details to eliminate manual
intervention to ensure the coverage of essential requirements by test cases.

UMTG [13] requires the involvement of software engineers for half of the ten steps in the approach: to
elicit use cases, model the domain, refine the model, specify constraints and specify the mapping table.
Manual intervention is required by the software engineer between nearly every automated step. On the
contrary, the approaches proposed in [12, 14] are fully automatic and do not require manual work.
Nonetheless, UMTG [13] can generate test cases parallelly in a multicore platform, decreasing the
performance time which would be increased due to manual intervention. Because the approach is
developed for an industrial context, parallel execution is a practical solution to the increase in
performance time caused by the required manual intervention [19] because of the tools used in industrial
software development.

UMTG [13] focuses on the automation of functional system testing, which requires the generation of
parameter values (additional input data) to guarantee the coverage of all the use case scenarios. The
approach is an important contribution to software testing since one of the biggest challenges in test case
generation is maintaining high levels of coverage to ensure that increasingly complex products meet
quality standards [20]. USLTG [12] also takes this challenge into account by generating a set of test cases
and providing the test agent with the best test case that achieves the highest transition coverage (100%
coverage). [14] also reports that the generated test cases reached an activity path coverage criterion
which is used for both loop testing and concurrency among activities of activity diagrams and that the
validity of test paths was evaluated by Cyclomatic complexity, which indicates that the test cases
generated from the proposed approach will be compatible with the complexity of the software that will represent the case of the research

5. Conclusion and Recommendation for Future work

In order to minimize the time and effort involved in manual test case generation, automatic test case generation approaches have been developed. This paper presents a literature review of the works in use case specification-based automatic test case generation.

The results of the review show that Use Case Specification Language Test Case Generation (USLTG), Use Case Modelling for System Tests Generation (UMTG), and an un-named test case generation approach have been introduced and validated. Although there are variations in the implementation, the steps in the identified approaches remain consistent: the use case specification is represented as a model, and the model is used to generate constrained use case scenarios which are then transformed into test cases. The approaches are validated by an experiment (33.3%) and comparative analysis (66.6%). In the approaches that used comparative analysis, a case study was conducted. In one approach, the case study applied use case specifications identical to those of a selected software system, while another utilized use case specifications similar to those of the approach it used for comparison. In addition, the review finds that the approaches have addressed one or more issues in software testing such as the considerable amount of time and effort required for test case generation, the risk of project failure due to ambiguous software specifications, and the coverage of all use case scenarios in test cases.

From the review, we have found that further studies are necessary to address the most significant issues in automated software testing. Future research could provide an approach to eliminate the manual intervention required to refine models in automated test case generation solutions. Such research could contribute to decreasing the time and cost of the software testing process.

References


