Literature Review on Test Case Generation Approach

Novi Setiani[1,2], Ridi Ferdiana[3], Paulus Insap Santosa[4]
Rudy Hartanto[5]
[1,3,4,5]Dept. Electrical Enginering and Informatics ,[2]Informatics Engineering
[1,3,4,5]Universitas Gadjah Mada, [2]Universitas Islam Indonesia
Yogyakarta, Indonesia
[1]novi.setiani@mail.ugm.ac.id, [2]novi.setiani@uii.ac.id, [3]ridi@ugm.ac.id, [4]insap@ugm.ac.id, [5]rudy@ugm.ac.id

ABSTRACT
Test case generation is a testing stage that requires the greatest resources among other stages so it has significant impact on the effectiveness and efficiency of software testing. Test case is a pair of input and output that will be executed by the tester whose aim is reveal the failures in software under test (SUT). For decades, this topic has become one of the most active topics in research on software testing. It has been proved by a variety of techniques and diverse tools proposed. In last decade, research in the field of test case generation experienced some progress. Nowadays, software testing is challenged to be able to test complex computation software that intensively interact with another system. The aim of this study is to give an up-to-date and overview of research in test case generation researches.

CSS Concepts
Software and its engineering → Software testing and debugging

Keywords
Test case; software testing; test case generation

1. INTRODUCTION
Testing consists of the test case design, test data generation, test data execution, analysis of the results of the execution and test results reporting [1]. Test case design is an activity to determine the software components to be tested and the design of inputs and outputs according to the specifications. In the generation of test data, the set of values is specified as input in executing the software. The test data will be executed in the software, then the output of the execution results will be analyzed to determine whether the test data generates failure in the software or not. All test results will be reported to the developer and manager team as a result of evaluation of software products that have been developed.

ICSIM 2019, January 10–13, 2019, Bali, Indonesia

ACM ISBN 978-1-4503-6642-7/19/01…$15.00
https://doi.org/10.1145/3305160.3305186

The test cases design and test data generation is a testing stage that requires the greatest resources among other stages [1] so that both have a significant impact on the effectiveness and efficiency of software testing. For decades, this topic has become one of the most active topics in research on software testing by producing a variety of techniques and diverse tools. Test case is a pair of input and output that will be executed in testing phase.. Automated test input generation techniques attempt to generate a set of input values for a program or program component, typically with the aim of achieving some coverage goal or reaching a particular state (e.g., the failing of an assertion).

Research in the field of test case generation experienced some progress that derived from the improvement of the computation ability and processing power in modern platform. In addition, there is a requirement in software development with complex computing, intensive system interaction through APIs, and the use of various libraries. Executing software testing on these complex software systems encourage automated test case generation became an important topic of research.

The aim of this study is to give an up-to-date and overview of research in test case generation approach. This research is conducted by following the procedures of performing systematic reviews in software engineering research from Kitchenham [2]. We reviewed test case generation approach that proposed by the previous researchers on the publication year since 2017 from reputable digital libraries.

This paper is further structured as follows: Section II presents research methodology in this literature review based on Kitchenham. Section III outlines our result after reviewing some updated research in test case generation topics. Section IV outlines the discussion about the result of our literature review. Finally, section V concludes the result of our literature review and identifies future work in test case generation research.

2. RESEARCH METHODOLOGY
2.1 Review Method
This literature review is performed in three stages: planning, conducting and reporting the literature review [2]. Planning is a process for defining the objectives of the review (Research questions determination) and developing the search strategy. Conducting is a process for selecting, extracting, and analyzing the data from primary studies. The result of data synthesizing is reported in the variants form, such as table, graphics, and description.

2.2 Research Questions

The research questions (RQ) were designed with the help of the Population, Intervention, Comparison, Outcomes, and Context (PICOC) criteria [2]. Table I shows the selected PICOC criteria for constructing the research questions. Research questions that will be answered in this review are described below:

RQ1: What kind of technique that has been adopted on test case generation?

RQ2: What are the challenges or issues raised on test case generation?

RQ3: What kind of testing level and domain that has been explored on test case generation research?

RQ4: Which journal is widely used for publishing the result of test case generation research?

Tabel I. PICOC Criteria

Criteria	Selected parameters
Populations	Software, application, system
Intervention	Test case generation
Comparison	-
Outcome	Issues arise on every approach
Context	Studies in industry and academia

2.3 Search Strategy

In this step, there are some activities involved, such as selecting the digital libraries, defining the search string, executing a search and retrieving an initial list of primary studies from digital libraries matching the search string. Before starting the search, an appropriate set of databases must be chosen to increase the probability of finding highly relevant articles. The most popular literature databases in the field are searched to have the broadest set of studies possible.

The databases were searched by title and content. The search was limited by the year of publication in range of 2017 until early 2018. The publication selection only involved papers from the selected library: IEEExplore, Sciencedirect, and ACM Digital Library. For simplification reason, the process of executing search is implemented by Advanced Search and command search feature in every digital library. The search is excluded the article that contains term review and literature because in this paper, we want to search a case study research.

2.4 Study Selection

Papers resulted from search execution are selected according to the exclusion and inclusion criteria shown in Table II. After this study selection process, there are some papers that eliminated from the resulted list. Total number papers that involved in data extraction is 37 papers.

Tabel II. Inclusion and Exclusion Criteria

Criteria	Description
Inclusion criteria	Studies in academic and industry For duplicate publications of the same study, only the most complete and newest one will be included Studies in test case generation topic that proposing an approach and evaluating in some case studies
Exclusion criteria	Studies not written in English Studies that conducting the literature review Studies not published in relevant topic

2.5 Data Extraction and Analysis

To answer the RQ1 about the technique that has been implemented on test case generation and RQ2 about the challenges arise on test case generation, we analyzed the articles as described at section "RQ1 and RQ2: Techniques and Challenges". The techniques are categorized on seven group found. Some aspects from every group are analyzed to answer the questions. To answer RQ3 about testing level and domain that has been explored in test-case generation research, we described at section "RQ3: Testing Level and Domain". To answer RQ4 about the channel publication widely used to publish the result of software testing research, we described at section "RQ4 Publication Distribution".

3. RESULT

3.1 RQ1 and RQ2: Techniques and Challenges

By analyzing 37 selected papers, test case generation approaches and challenges can be categorized as described in Table III.

Tabel III. Test Case Generation Techniques

Technique	Description	Challenge	List of Papers
Model based testing (MBT)	semi-formal method that exploited software specifications as sources for test case generation.	Measuring test case quality and combining some models that describe different aspect of software as merged sources for test cases.	scenario oriented [5, 9, 10], and state oriented [3, 6, 7, 8, 11, 12, 13]
Search based software testing (SBST	utilizing optimization algorithm for searching set of test data that satisfy a certain fitness function	Generating test case that can be easily understood by the tester and reducing computation complexity issue	Particle Swarm Optimization [14, 15] , Genetic Algorithm [16, 17], and Hybrid Algorithm [18, 36]
Random testing (RT	generating test case by randoming its value based on input specification	combining RT with other method, i.e, using evolution strategies for identifying regions of the input domain	Random Testing [19, 20] and Random Testing for GUI Testing [21]
Combinatorial	searching optimum test data	reducing search time search time	CT in IoT systems [22],

testing (CT)	that cover subset of input elements combinations called t-way covering array.	complexity that exponentially increase along with length of array.	CT using search based algorthim [23]
Metamorphic Testing (MT)	Exploiting metamorphic relation in the program to generate follow up test case from source test case	identifying and selecting metamorphic relation automatically	Generating test cases[25], Identifying failed test case [26]
Concolic Testing	combining random testing and symbolic dynamic execution	handling scalability problem for complex data structure and reducing execution time	Concolic testing on Java Programs [27, 28, 29]

3.2 RQ3: Testing Level and Testing Domain

Based on software development activities, testing level can be divided into five levels [1]: acceptance testing, system testing, integration testing, module testing, and unit testing. From Figure 1 it can be concluded that majority software testing research focus on unit testing and system testing. Unit testing is usually conducted in search based software testing and concolic testing. Whereas system testing is usually explored in model based testing, random testing, combinatorial testing, and metamorphic testing.

Software testing researchs have been conducted in variant domains. The most research is done on object oriented paradigm [31, 36] by implementing automatic testing for Java or C++ program language [26].

Furthermore, some researchs focus on solving software testing problems in embedded software [4, 6, 7, 8, 22], and avionic industry [13]. There are a few papers studying software testing in web-based application [38, 39], web services [32], and mobile application [11]. Almost papers focus on functional testing, but there are some papers examining problems in GUI testing [5, 21].

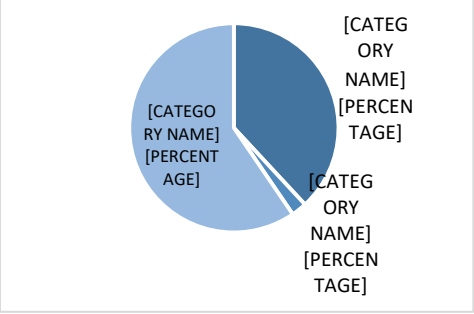

Figure 1. Research Distribution Based on Testing Level

3.3 RQ4:Publication Distribution

From Figure 2, it is concluded that 61% papers come from conference publication. The rest comes from journal publication which the majority of test case generation papers were published in Journal of Information and Software Technology. This journal covered all aspects of software development practices, includes methods and techniques.

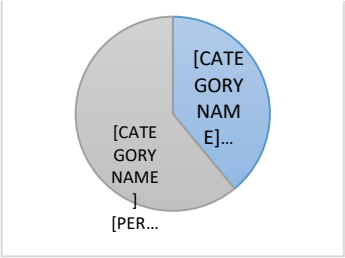

Figure 2. Publication Distribution

4. EXPERIMENT

An experiment to analyze two test case generation methods in unit testing level is conducted. From blackbox testing method, random testing technique is selected and from whitebox testing, SBST technique is selected. Two automatic test case generator tools are used, Randoop[40] and Evosuite [41]. 10 Java programs from two open source projects are used as case studies. Data collected from this experiment are time execution and the number of test cases generated from these tools as described at Table IV.

SBST tool generate fewer test cases than RT tool and need more execution time. The average time needed by SBST tool for generating test cases on one unit class is 60 second, meanwhile RT tool only need 3 second. On the other hand, test cases resulted from SBST tool are more understandable than RT's. SBST tool not only generate test cases, but also code coverage measurement, so it will give tester a valuable information. Test case generated from SBST tool reach 68.3% code coverage in average, with range 20%-96%

5. RESULT AND DISCUSSION

Based on source code utilization for generating test cases, the approaches usually categorized as whitebox and blackbox testing. Model based testing, combinatorial testing and random testing are examples of black box testing where testing process didn't need source code implementation. MBT used specification model for generating test cases, and random testing only utilized input specification for resulting set of random input value to execute the program. Search based software testing is an implementation of white box testing because they need to read the program implementation in generating the test cases. Concolic testing is a hybrid method which combining between black box (random testing) and white box testing (dynamic symbolic execution). The latest testing method, metamorphic testing can be used in conjunctional with both blackbox and whitebox testing.

From level testing point of view, there are few efforts in integration, module and acceptance testing. These result are obtained by the facts that it is difficult to collect architectural and detail design of software as sources for running integration and module test. Apply automated effort on acceptance testing is also still a challenge because it must involve users with multi-preferences and background.

Tabel IV. Experiment Results

Class	Number of test cases	
	RT	SBST
A4J	52	24
A4jUtil	49	23
RPCMessageReceiv	101	1
HTTPWorker	101	5
RIFInvoker	100	2
RIFClassLoader	52	5
RIFManager	100	5
RIFImplementation Manager	61	4
RIFService	77	2
WebServiceDescrip	49	19

6. CONCLUSION AND FUTURE WORK

From this review, we can conclude that the specification-based approach is still dominated in software testing research, especially in automated test case generation approach. The specification-based that driven by the model of System Under Test is explored by most software testing researchers by using model based testing. But there are still open challenges for researchers to explore another method such as in concolic testing and metamorphic testing.

There are two types problems found from this literature review: problems in research methodology and problems in proposed approaches. Problem in research methodology related to limited empirical result and experiment. Most researches used only few case studies and the method is unable to be replicated because the software under test wasn't shared publicly. Problem in test case generation method is related to computation complexity, limited ability when handling complex data structure, and low coverage output. To handle last problem, we will propose an oracle driven test case generation method. This method will generate test cases based on an assertion of metamorphic relation.

Most of test case generation research focused on functionality aspect of the system that using object oriented paradigm. Test case generated mostly in unit and system level, so there is still future works for research on integration, module and acceptance testing level. The outlet of test case generation research publication in reputable journal are quite diverse, so there are still many opportunities to explore this topic in the future.

7. REFERENCES

[1] T. Dalgleish *et al.*, *Introduction to Software Testing*, vol. 136, no. 1. 2007.

[2] T. O. F. Contents, "Journal of Systems and Software." pp. 1–12, 2011.

[3] S. Groning, C. Rosas, and C. Wietfeld, "Validating electric vehicle to grid communication systems based on model checking assisted test case generation," *2017 IEEE Int. Symp. Syst. Eng. ISSE 2017 - Proc.*, 2017

[4] S. Rösch and B. Vogel-Heuser, "A Light-Weight Fault Injection Approach to Test Automated Production System PLC Software in Industrial Practice," *Control Eng. Pract.*, vol. 58, no. October 2016, pp. 12–23, 2017..

[5] L. Zhao and D. Gao, "GUI test case generation based on Activity-Flow Graph," *Proc. IEEE Int. Conf. Softw. Eng. Serv. Sci. ICSESS*, no. 91118007, pp. 738–741, 2017.

[6] R. Darwish, L. N. Gwosuta, and R. Torkar, "A Controlled Experiment on Coverage Maximization of Automated Model-Based Software Test Cases in the Automotive Industry," *Proc. - 10th IEEE Int. Conf. Softw. Testing, Verif. Validation, ICST 2017*, pp. 546–547, 2017.

[7] A. Aniculaesei, P. Denecke, F. Howar, and A. Rausch, "Automated Generation of Requirements-Based Test Cases for an Adaptive Cruise Control System," pp. 11–15, 2018

[8] O. Olajubu, S. Ajit, and S. Turner, "Automated test case generation from high-level logic requirements using model transformation techniques," *Proc. 9th Comput. Sci. Electron. Eng. Conf. (CEEC 17)*, pp. 178–182, 2017

[9] S. Preeti, B. Milind, M. S. Narayan, and K. Rangarajan, "Building Combinatorial Test Input Model from Use Case Artefacts," *Proc. - 10th IEEE Int. Conf. Softw. Testing, Verif. Valid. Work. ICSTW 2017*, pp. 220–228, 2017.

[10] S. Yimman, S. Kamonsantiroj, and L. Pipanmaekaporn, "Concurrent test case generation from UML activity diagram based on dynamic programming," *Proc. 6th Int. Conf. Softw. Comput. Appl. - ICSCA '17*, pp. 33–38, 2017.

[11] L. Panizo, A. Salmerón, M.-M. Gallardo, and P. Merino, "Guided test case generation for mobile apps in the TRIANGLE project: work in progress," *Proc. 24th ACM SIGSOFT Int. SPIN Symp. Model Checking Softw. - SPIN 2017*, pp. 192–195, 2017

[12] P. K. Arora and R. Bhatia, "Mobile agent-based regression test case generation using model and formal specifications," *IET Softw.*, vol. 12, no. 1, pp. 30–40, 2018.

[13] G. Philip, "Model Based Safety Analysis : Automatic Generation of Safety Validation Test Cases," 2017

[14] S. Rösch, D. Tikhonov, D. Schütz, and B. Vogel-Heuser, "Model-based testing of PLC software: Test of plants' reliability by using fault injection on component level," *IFAC Proc. Vol.*, vol. 19, pp. 3509–3515, 2014.

[15] B. S. Ahmed, L. M. Gambardella, W. Afzal, and K. Z. Zamli, "Handling constraints in combinatorial interaction testing in the presence of multi objective particle swarm and multithreading," *Inf. Softw. Technol.*, vol. 86, pp. 20–36, 2017.

[16] W. Jing, Z. Yikun, Z. Ming, C. Hao, H. Xinhong, and S. Jianxiong, "A method for test case generation by improved genetic algorithm based on static structure of procedure," *2017 12th IEEE Conf. Ind. Electron. Appl.*, pp. 1499–1504, 2017.

[17] R. Khan, M. Amjad, and A. K. Srivastava, "Generation of automatic test cases with mutation analysis and hybrid genetic algorithm," *2017 3rd Int. Conf. Comput. Intell. Commun. Technol.*, pp. 1–4, 2017.

[18] K. Z. Zamli, F. Din, G. Kendall, and B. S. Ahmed, "An experimental study of hyper-heuristic selection and acceptance mechanism for combinatorial t-way test suite generation," *Inf. Sci. (Ny).*, vol. 399, pp. 121–153, 2017.

[19] H. Zheng *et al.*, "Automated test input generation for android: Towards getting there in an industrial case," *Proc. - 2017 IEEE/ACM 39th Int. Conf. Softw. Eng. Softw. Eng. Pract. Track, ICSE-SEIP 2017*, pp. 253–262, 2017.

[20] G. Grieco, M. Ceresa, A. Mista, and P. Buiras, "QuickFuzz testing for fun and profit," *J. Syst. Softw.*, vol. 134, pp. 340–354, 2017

[21] R. Ramler, G. Buchgeher, and C. Klammer, “Adapting automated test generation to GUI testing of industry applications,” *Inf. Softw. Technol.*, vol. 93, pp. 248–263, 2018.

[22] K. Cui, K. Zhou, T. Qiu, M. Li, and L. Yan, “A hierarchical combinatorial testing method for smart phone software in wearable IoT systems,” *Comput. Electr. Eng.*, vol. 61, pp. 250–265, 2017.

[23] S. Esfandyari and V. Rafe, “A tuned version of genetic algorithm for efficient test suite generation in interactive t-way testing strategy,” *Inf. Softw. Technol.*, vol. 94, no. January 2017, pp. 165–185.

[24] A. Giantsios, N. Papaspyrou, and K. Sagonas, “Concolic Testing for Functional Languages,” pp. 137–148

[25] J. Y. S. Tang, A. Yang, T. Y. Chen, and J. W. K. Ho, “Harnessing multiple source test cases in metamorphic testing: A case study in bioinformatics,” *Proc. - 2017 IEEE/ACM 2nd Int. Work. Metamorph. Testing, MET 2017*, pp. 10–13, 2017.

[26] Z. W. Hui, S. Huang, T. Y. Chen, M. F. Lau, and S. Ng, “Identifying failed test cases through metamorphic testing,” *Proc. - 2017 IEEE 28th Int. Symp. Softw. Reliab. Eng. Work. ISSREW 2017*, pp. 90–91, 2017.

[27] S. Godboley, A. Dutta, D. P. Mohapatra, and R. Mall, “GECOJAP: A novel source-code preprocessing technique to improve code coverage,” *Comput. Stand. Interfaces*, vol. 55, pp. 27–46, 2018.

[28] S. Godboley, A. Dutta, D. P. Mohapatra, and R. Mall, “J3Model: A novel framework for improved Modified Condition/Decision Coverage analysis,” *Comput. Stand. Interfaces*, vol. 50, no. March 2016, pp. 1–17, 2017.

[29] S. Godboley, A. Dutta, D. P. Mohapatra, and R. Mall, “Scaling modified condition/decision coverage using distributed concolic testing for Java programs,” *Comput. Stand. Interfaces*, vol. 59, no. February, pp. 61–86, 2018.

[30] S. S. Bodiwala and D. C. Jinwala, “Optimizing test case generation in glass box testing using Bio-inspired Algorithms,” *Proc. IEEE Int. Conf. Softw. Eng. Serv. Sci. ICSESS*, pp. 40–44, 2017.

[31] H. Huang, F. Liu, X. Zhuo, and Z. Hao, “Differential Evolution Based on Self-Adaptive Fitness Function for Automated Test Case Generation,” *IEEE Comput. Intell. Mag.*, vol. 12, no. 2, pp. 46–55, 2017.

[32] A. Arcuri, “RESTful API automated test case generation,” Proc. - *2017 IEEE Int. Conf. Softw. Qual. Reliab. Secur. QRS 2017*, pp. 9–20, 2017.

[33] L. Yu and W. T. Tsai, “Test Case Generation for Boolean Expressions by Cell Covering,” *IEEE Trans. Softw. Eng.*, vol. 44, no. 1, pp. 70–99, 2018..

[34] M. Boucher and G. Mussbacher, “Transforming Workflow Models into Automated End-to-End Acceptance Test Cases,” *Proc. - 2017 IEEE/ACM 9th Int. Work. Model. Softw. Eng. MiSE 2017*, pp. 68–74, 2017.

[35] Q. Xin and S. P. Reiss, “Identifying test-suite-overfitted patches through test case generation,” *Proc. 26th ACM SIGSOFT Int. Symp. Softw. Test. Anal. - ISSTA 2017*, pp. 226–236, 2017.

[36] A. Panichella, F.M. Kifetew, P Tonella, "Automated Test Case Generation as A Many-Objective Optimisation Problem with Dynamic Selection of the Targets". *IEEE Trans. Softw. Eng*., vol 44, no. 2, February 2018.

[37] Meiliana, et.al., "Automated Test Case Generation from UML Activity Diagram and Sequence Diagram using Depth First Search Algorithm", Procedia Computer Science, Vol. 116, 2017 pp 629-637.

[38] L. Nagowah, and K. Kora Ramiah. "Automated Complete Test Case Coverage for Web Based Application", *Proc - Int. Conf. on Infocom Technologies and Unmanned Systems 2017*.

[39] J.L. de Moura, et.al. "Test Case Generation from BPMN Models for Automated Testing of Web-Based BPM Application". *Proc - 17th Int. Conf. on Computational Science and Its Application 2017*.

[40] https://randoop.github.io/randoop

[41] http://evosuite.org