Volume: 5, Issue: 6
Page: 33-44
2021

International Journal of
Science and Business
Journal homepage: ijsab.com/ijsb

IJSAB International

The Impact of Test Case Generation Methods on the Software Performance: A Review

Dathar A. Hasan, Bzar Kh. Hussan, Subhi R. M. Zeebaree, Dindar M. Ahmed, Omar S. Kareem & Mohammed A. M. Sadeeq

Abstract:
The software development in different fields leads to increase the requirements for effective, efficient and complicated software. Due to the huge amounts of software requirements, it is possible some errors to occur in the certain part of the programs and this means a real challenge for the software producer. The need for an effective test system is necessary for designing reliable programs and avoiding the errors that may appear during the software product. In this review, many techniques are discussed for the process of generating test cases which are a group of conditions that determine whether the designed programs are able to satisfy the user's requirements or not. Fuzzy logic utilizes an operational profile in the process of allocating test case to improve the software quality, as well as the design of fault propagation path to predicts the software defects during the test operation, also the automatic generation for PLC test cases that produce a new track through the program code in order to minimize the test cases needed for large size program.

IJSB
Accepted 13 March 2021
Published 19 March 2021
DOI: 10.5281/zenodo.4623940

Keywords: *Software test, test case, program evaluation, reliable software system, test suite.*

About Author (s)

Dathar A. Hasan (corresponding author), Duhok Polytechnic University, Shekhan Technical Institute, Kurdistan Region, Duhok, Iraq.
Email: dathar.hasan@dpu.edu.krd
Bzar Kh. Hussan, Erbil Polytechnic University, Kurdistan Region, Iraq.
Subhi R. M. Zeebaree, Duhok Polytechnic University, Kurdistan Region, Duhok, Iraq.
Dindar M. Ahmed, Duhok Polytechnic University, Kurdistan Region, Duhok, Iraq.
Omar S. Kareem, Duhok Polytechnic University, Kurdistan Region, Duhok, Iraq.
Mohammed A. M.Sadeeq, Duhok Polytechnic University, Kurdistan Region, Duhok, Iraq.

Introduction

Recently, software systems have a significant importance in the computer systems due to the rapidly increasing needs of various computer users for the software productions. As the profit is the most important factor, the software producers receive the massive new projects that are sometimes more than production capacity without hesitation to the probability of an error occurring in the software as a result of quick production mechanisms.

The process of test case designing and executing is the core of software testing. Therefore, whether effective test case can be designed to reach the test coverage of specified requirements will have a direct influence on the quality of procedure to be tested. To solve the defects of manual compiling test case for large software and raise the efficiency of testers, some people put forward automatic generation of test case (Jing et al., 2017). The software testing is an essential task that implements a program code on a series of test cases, the architecture and specifications of the software will specify these test cases. The results obtained from test case implementation are comparing with the ideal or expected results (Hooda & Chhillar, 2014). The software testing process is very necessary to reduce the efforts and assuring the program effectiveness. Software testing is a complicated and costly task that has to be performed by some automated tools under the supervision of software professionals (Gonçalves et al., 2017). Additionally, the manual test could be performed by the software developers via some special mechanism, the main disadvantages for this type of test is the long-time consumption as well as error occurring due to missing some test scenarios (Retna, 2012). The testing stage of a software system is the final stage of software construction lifecycle that satisfies the quality assurance requirements. However, to perform an efficient software testing many a professional laboratory has been prepared to build some standards for the testing task. These standards consist of the conditions and mechanisms of performing an effective software testing that meets the requirements of quality assurance (Sun, Qian, & Liu, 2015). Many types of research explain that some of the software attributes performance and interactions among them are the main reasons for software faults (Abdul Rauf & Reddy, 2015). As a result of complexity and scalability, software systems are divided into multiple modules each one performs its own task, the task execution for some modules may be more than the others. The reliability factor has a vital rule in a software system, it represents an operation without faults occurring under predefined states for a certain time (Amrita & Yadav, 2015).

The process of writing a test case is an important task and it is a challenge at the same time because it has to be effective in software maintenance, the test has to deal with the various criteria and conditions not only focusing on the main functionalities of the software (Simon et al., 2015). The concept Test Case Selection (TCS) is necessary for the program size control that can be modified according to the needs, as well as choosing a set of most effective test case for the program after and before the modification process and representing it as test suite (Lawanna, 2016). The task of selecting suitable test cases to build a test suite for program updating consists of many stages such as preparing test codes, checking bugs, and choosing the best methods (Lawanna & Wongwuttiwat, 2016). In this review, the importance of software test cases has been discussed from various perspective, many new techniques have been proposed for building a reliable software by reducing the probability of program faults occurring.

The methods of producing the test cases

In order to improve the quality of the software, test cases are used to assure the reliability of the executed program. The function of selecting the best test cases is very useful to enable the testers to obtain the appropriate result in minimum time with less cost. Many types of research have been discussed in this review suggest various techniques that aim to select the best test cases which are:

Utilizing the Fuzzy logic for allocating software test cases

(Amrita & Yadav, 2015) proposed a fuzzy logic to deal with the Software Operational Profile (SOP). As the SOP has an important role in the prediction of system reliability and test case allocation for software, the professionals diagnosed many problems during the SOP related to the limitations in the resources of the data.

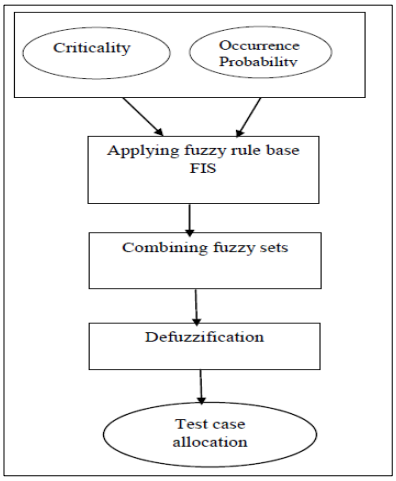

Figure 1: The Proposed Model Architecture by (Amrita & Yadav, 2015).

In the proposed model shown in figure1, in order to implement the fuzzy rules, two combined inputs are supposed to the fuzzy inference system, these are occurrence probability and severity. The process of test case allocation is performed after converting the two fuzzy inputs to crisp values (Note Narciso, Delamaro, & Nunes, 2014). There are four sets can be defined from the two inputs: firstly, infrequent and critical, secondly, infrequent and non-critical, thirdly, frequent and critical, and finally frequent and non-critical. Focusing on the frequent and critical state doesn't' means that there is no other probability. However, there are four possible steps for the two input test cases:

Membership function defining:

This function is important to convert the member variables to the fuzzy set and assigning the fuzziness level for the fuzzy set by using different shapes. (Panichella, 2019)There are many methods for identifying the membership value for fuzzy variables such as intuitive, inference, genetic algorithm, rank ordering, angular fuzzy set, and neural networks. For the intuitive method, the human intelligent skills are used to perform the assignment process, for inference, the deductive reasoning is produced based on the knowledge (Haji, Zeebaree, Jacksi, & Zeebaree, 2018). In a genetic algorithm, Darwin's theory is used which is a series of steps for assigning the values for fuzzy variables. In ranking process, the groups or individuals utilizing the polling idea for assigning the value, the linguistic variables of truth values based on the quantitative description is used for Angular functions.

Fuzzy rules Designing:

In the knowledge-based systems (Alkawaz & Silvarajoo, 2019), the fuzzy rules such as (AND) and (OR) with IF_THEN are considering to be the main basis for identifying the values for the variables in the proposed model. In the (IF_THEN) condition, the part (IF) is considered to be the metrics of the software input and the part (THEN) is supposed to be the overall numbers of faults before starting the test process. There is a value for each rule which is evaluated by the professional. In the proposed model there is only two input variables severity with 4 linguistic states and occurrence probability with five linguistic states. Therefore, the rules could be designed as follows:

Table.1: Rules of the two inputs.

	(IF) Occurrence Probability and Severity states	**(THEN)** Number of test cases
1	If (occurrence probability is very low) and (severity is minor)	Very low.
2	If (occurrence probability is very low) and (severity is major)	Low
3	If (occurrence probability is very low) and (severity is critical)	Medium
.	-	
.	-	
19	If (occurrence probability is very high) and (severity is critical)	High
20	If (occurrence probability is very high) and (severity is catastrophic)	Very High

Defuzzification

It is the process of converting the fuzzy quantities to the crisp quantities and it is the inverse of the fuzzification process. There are many methods for implementation of defuzzification processes such as centroid method, mean max, and center of the largest area. Centroid method is one of the most used due to its physical appealing. This method produces the number of test cases that are required for testing a certain software by normalization of the defuzzied value to obtain the overall number of test cases as shown in figure2.

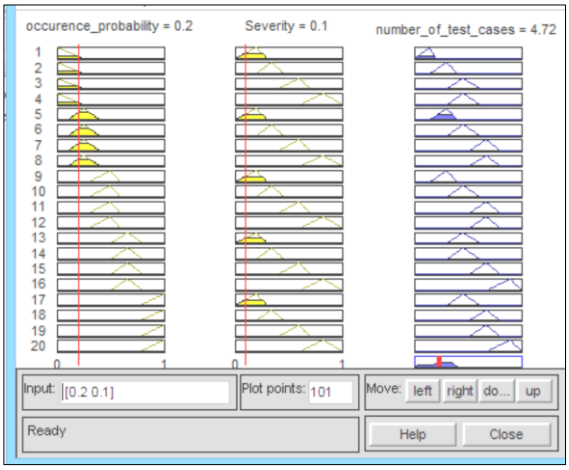

Figure 2: Defuzzied value for Operation.

The process of model validation for the proposed model is implemented by giving an equal importance for both occurrence probability and severity. As shown in table 2, the number of test cases is minimized as a result of applying this model (Amrita & Yadav, 2015).

Table 2: No. of test cases for different models.

No. of operations	Occurrence Probability	Severity	Leung Model [2]	Sravana Model [7]	Proposed Model
1	0.3	0.3	20	19	16
2	0.4	0.2	17	14	11
3	0.1	0.4	9	9	13
4	0.2	0.1	4	8	5
∑	1		50	50	45

Using Fault propagation path coverage for test cases generation

Kun & Yichen (2016) focuses on the describing the fault assessment process and applying fault injection method by using an error propagation model. This model is constructed depending on two basic concepts, the first one is concerning with testing stage in which any correlated defects is possible to activate if a defect is triggered into a fault and then represented as a failure. The second concept is the process of finding of the correlated defects requires activating the defects at the beginning of triggering the other related defects. These two concepts are the basis for building the method of software testing system that uses the fault propagation path coverage. The process of detecting a new defect begins by producing a seed-defect, test case design, and detecting all-new defects via using fault injection method. The process of typical defects cultivating is the principle of fault injection by error conjecture that's necessary for designing of the test case (S. R. M. Zeebaree, Sallow, Hussan, & Ali, 2019). The defects that have a strong ability for generating new defects as well as having consistent types related to features of the measured software are called typical defects. The past experiences of the software testing are the source of these features. Depending on the currently used forms of software modeling like complex network, state diagram, and program slicing it is possible to specify the propagation path by creating reliable, suitable, and accurate models for under test software (S. Zeebaree, Zebari, & Jacksi, 2019). In the proposed model, the representation of the software modeling is carried out using control flow diagram due to focusing on introducing a testing method for path coverage. The algorithm of this model uses the function to automatically generate test cases, specifying the defects and applying fault propagation path coverage to discover more defects than other models.

During test process, there is a need for continuous inputs and performing defects injection process into the fault. The need for checking whether the inputs represent the fault propagation path or not, as well as detecting the new defects. The intelligent algorithm generates a large number of test cases to be used through optimization iterations towards the desired direction. The algorithm then reduces the tester's load. Figure 3 shows the mechanism of the adopted algorithm (Zebari, Zeebaree, Jacksi, & Shukur, 2019).

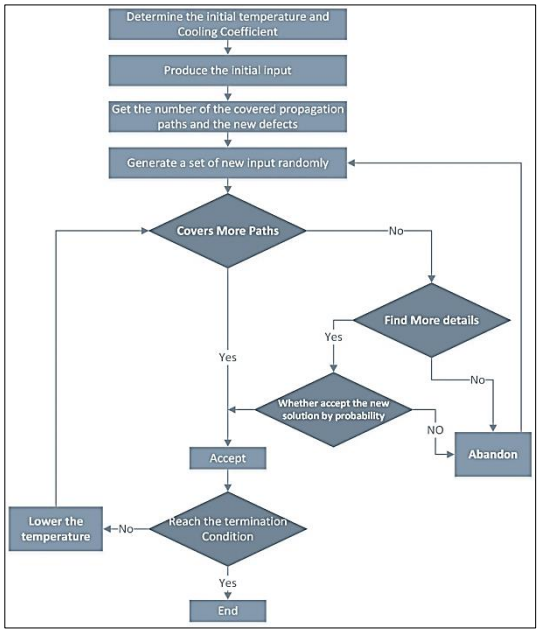

Figure 3: The simulated annealing algorithm(Kun & Yichen, 2016).

Improving test case selection via Test case design Based Technique

Lawanna (2016) thinks that the increase of the test cases number has a significant impact on the performance of software updating through adding new code to the previous program. As a result, extra time is needed for executing and testing the added code. The best solution is to reduce the test suite by utilizing the design-based technique. This technique includes four algorithms, they are testing test case, classification, deletion, and selection. These algorithms control the large-scale programs using test case selection. (Hardt, 2020) The main two motivations of the proposed test code design (TCD) are achieving minimum size test cases and focusing on the software faults reduction. In order to obtain an identical test model for the selection method, the irrelative test cases can be removed from test suite using the classification technique, figure 4 shows the conceptual model in which the steps of test case improvement selection are described.

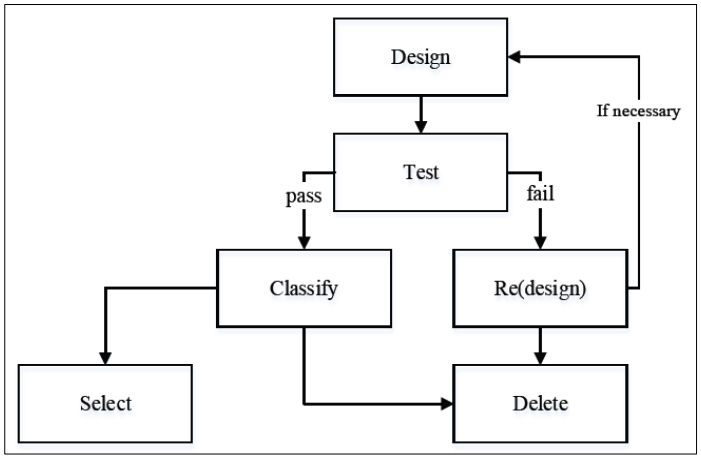

Figure 4: The overview Test Case Design(Lawanna, 2016).

Test Case Design

In order to implement the proposed (TCD) model, four steps have to be performed as follow:

A. It is very important to list all test cases for the program as:

$$T = \{t_1, t_2, t_3, \quad , t_c\}$$

B. For each test case, we have to specify the steps of test operation as:

$$\mathbf{t= \{s_1, s_2, s_3, \ , s_c\}}$$

these steps help the developers to design the test cases.

C. Test setup preparation begins by grouping the similar and different test cases into two set as:

$$\mathbf{T_S= \{t_1, t_2, t_3, \ , t_x\}}$$
$$\mathbf{T_D= \{t_1, t_2, t_3, \ , t_y\}}$$

D. Listing all valid test cases as a suitable test suite for each program.

In fact, there are two algorithms used in test case development for software modification as shown below:

i. Producing dissimilar test cases algorithm as:
   - Input T= {t_1, t_2, t_3, …, t_C} and t= {s_1, s_2, s_3, …, s_S}
   - If c ∈ C then T= {t1, t2, t3, …, t_C}
     Else if $s \in t$ then t= {s_1, s_2, s_3, …, s_S}
     Else if $s_1 \cap s_S = \emptyset$ then T_D= {t_1, t_2, t_3, …, t_y}

ii. Discovering t- and t+ algorithm is working as follow:
   - Let $\boldsymbol{t'_1}$ be the index of dependency among test cases, and $\boldsymbol{t'_2}$ be the steps minimum explanation test cases and $\boldsymbol{t'_3}$ is test cases with poor idea explanation.
   - If $t_x \propto t_y$ then $t_- = t'_1$
   - Else if $(s_x \cap s_y) \neq \emptyset$ then $t_- = t'_2$
   - Else if $t_- - (t'_1 + t'_2)$ then $t_- = t'_3$
   - Else if $t - (t'_1 + t'_2 + t'_3)$ then $t+ = t - \Sigma t_-$
   - End

Redesigning test case

The test cases that are not suitable will be produced after applying the previous two algorithms by redesigning the test cases. The cost parameter of the t specifies whether it can be redesigned or not. As the cost increases the complexity of redesigning is also increased, this can be checked by calculating the cost of all steps of the test case.

Deleting test cases

One of the most useful functions that maintain the software is to delete all irrelative test cases, it helps to minimize the test suite but on the other hand, the maintenance of the software will increase. The deletion algorithm is:

- If $t' = t'_1$ then Delete t'_1
- Else if $t' = t'_2$ then then Delete t'_2
- Else if $t' = t'_3$ then Reject t'_3
- End

Selecting test cases

The algorithm of this process is:

- If $t = t+$ then Select $t+$
- End

The main aim of designing TCD model is to select efficient and minimized test suite for software maintenance requirements(Lawanna, 2016).

Utilizing coverage metrics in test case generation for PLC programs

Simon et al. (2015) proposed a model for generating automatic test cases that perform the testing operation in Programmable Logic Controller (PLC) programs depending on the IEC61131-3 standard. The model continuously generates new program traces to represent a part of the program in the form of coverage metrics. The generated test cases are then converted to understandable program language for IEC61131-3 to be implemented on the PLC hardware. In this technique, the model has generated automatically by using the build in program code in the PLC module. (Kumar & Kumar, 2016)A witness is designed using the model checker that is able to check the reachability of any line inside the program, this process can be performed for all necessary inputs. In case of the witness is not able to reach the line, it will be assigned as unreachable. ARCADE PLC is used in this model, which is a framework for the verification of programs for programmable logic controllers. ARCADE PLC is able to support the programs that are written as instruction list, structured text, and functions block diagram as well as supporting PLCopen XML format programs. These capabilities of PLC ARCADE help to verify some critical properties by creating a state space automatically, these state spaces express the configurations of the PLC programs stored in the memory chip. Two spaces s0, s1 are assumed to be the inputs if there is a transition between the inputs in the state space which means that s0 is reachable in one cycle from s1 as shown in figure 5.

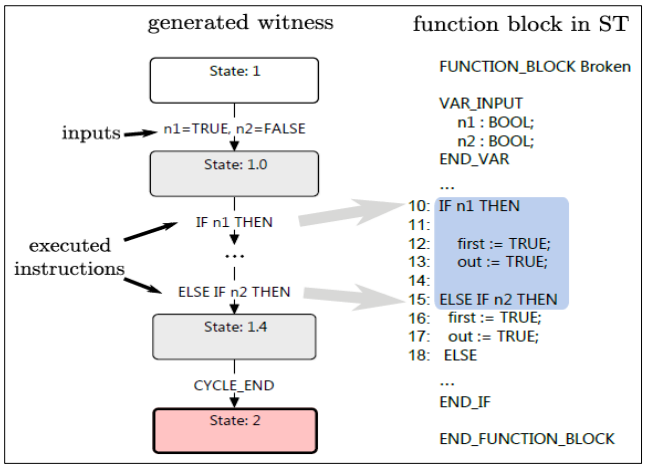

Figure 5: A witness including required input values is generated by the model checker. The corresponding program trace covers a part of the actual code(Simon et al., 2015).

The intermediate states between State 1 and State 2 denote the states that execution during the cycle (which are not observable). The witness shown in this Figure has executed the desired line after one cycle and covers additional lines that were executed in the trace. Note that with the same technique we can check the reachability of multiple lines at once, which will be of further interest in the next section.

The previous explanation is used to build an algorithm that creates a group of traces in the tested programs by the following steps: -

Line Coverage:

The basic aim of this step is to produce a set of traces to enable each line in the program to be implemented via one of these traces. However, there is no need to check each line reachability to finish the coverage, but in the case of checking the last line, the program has to pass all the beforehand instructions. The mechanism of the process of line coverage is shown in figure 6.

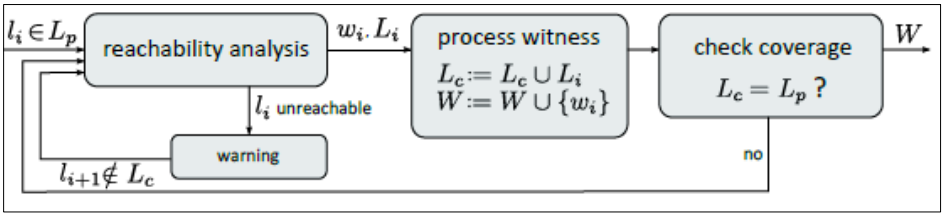

Figure 6: Workflow of the line coverage algorithm (Simon et al., 2015).

The first step is to choose a start line (lS) that belongs to the set of program lines (LP) as lS ∈ LP, if the line lS is reachable the covered lines LS can be extracted from the created witness wi. The covered line can be updated to get Lc=Lc ∪ Li, then saving the results wi. in the set W. At the final step, the coverage is checked using Lc = LP, this represents to the completed coverage and obtaining the set W which refers to the number of traces for each line in the program. The algorithm ends its operation with the ending of the last step and any line is not reachable then it will be signed as unreachable.

Branch Coverage

The work of the branch coverage algorithm is similar to that of the line coverage algorithm except that the branch coverage begins with a pair and evaluating each element with respect to the other. The process of generating a witness for this pair is performed by the model checker. The evaluation of each trace can be extracted in the witness since the traces consist of concrete input values(Abdulazeez, Zeebaree, & Sadeeq, 2018).

The two previous techniques algorithms are embedded with the proposed framework for test case generation of function blocks in order to support line and branch coverage. The execution of the PLCs program testing using simulator may give inappropriate program functionality results due to the working in the critical environment. As shown in figure 3, the coverage algorithm is executed to produce a group of witness W that refer to the program traces with the necessary input value, each witness is represented as a state variable in a function block test.

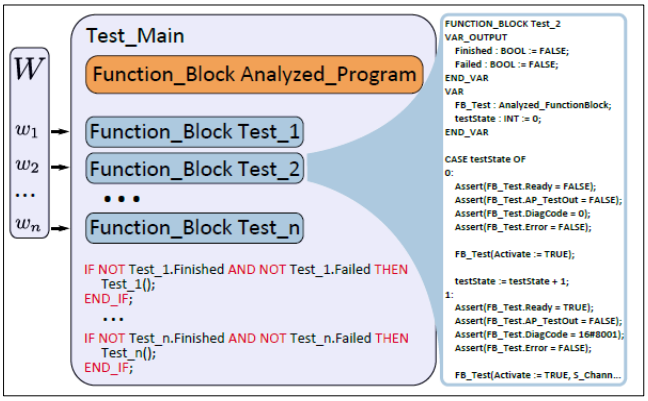

Figure 7: A witness including required input values is generated by(Simon et al., 2015).

These variables are:

- fb_test: analyzed function block
- teststate: int
- finished; failed: bool

These three variables are very important in function block test, teststate variable works as a counter for the PLC current cycle, while finished/failed variable refers to the test completed without error and failed variable means error occurring during the test. For every test there is an instance of an analyzed function block as shown in right side of figure 3, the procedures of generating code for each cycle in W is performed by checking the matching between the expected behavior and the state variable value, then FB_Test with the given input values. Finally, testState is increased by one. If one of the assertions in the first step fails, Failed is set to TRUE. However, when the complete program trace of W could be simulated without an error, the test succeeded and Finished is set to TRUE.

Discussion

The main goal of program testing process is to find bugs in the software system or other application. Test case generation is the process of creating test suites for checking and detecting the software system faults. A test suite is a set of relevant test cases bundled together. Test case generation is the most important and fundamental process of software testing. Many techniques are proposed for generating the best test suite toward obtaining an effective and efficient software. As software operational profile (SOP) has an important role in software case test generation but it comes with some problems related to limitations in data resources, the fuzzy logic is used to overcome this problem. During the testing process, there is the probability of potential faults occurring which are difficult to be detected, the best solution is the error propagation model that use fault injection method for discovering seed-defects. However, the large numbers of test suite influence on the program updating process, this can be overcome using a model consisting of four algorithms that reduce the time needed for updating programs. The process of test case generation can be increased using traces generation for each function blocks in PLC software.

Table 3: The techniques used for test case generation.

Parameter	(Amrita & Yadav, 2015)	(Kun & Yichen, 2016)	(Lawanna, 2016)	(Simon et al., 2015)
Problem	Limited data resources cause difficulties in conversing collected data into point estimate.	Lack to find correlated defects through the test process.	The low performance caused by a large amount of test suite used for updating programs	Testing Function blocks require long times using old test generation techniques.
Technique	Fuzzy logic based on operational profile.	Designing error propagation model that use an intelligent algorithm to generate test cases.	Designing a technique that consists of four algorithms	Designing a method for building new program traces as test cases.
Result	More convenient, reliable and eliminating ambiguities by getting a greater number of linguistic variables for inputs.	All correlated defects are being discovered in paths of propagation.	Improving the test case selection by minimizing the test case suite for program updating.	Increasing the rate of case test generation for checking function blocks.

Conclusion

The wide increase of the software in any organization requires some additional efforts to build a free fault software, one of the most important steps towards achieving the stability in software functionality is the process of generating and allocating the program test case. We

focused on many approaches that are used for test case generation. Fuzzy logic utilizes an operational profile in the generating of the test cases to improve the software quality, as well as predicting the software faults through the test process by designing the fault propagation path. The automatic test cases allocating for PLC that designs a new track through the program code. In order to minimize the test cases needed for large size program, the test-based technique.

References

Abdul Rauf, E. M., & Reddy, E. M. (2015, 5-7 March 2015). *Combinatorial testing: A case study approach for software evaluation.* Paper presented at the 2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT).

Abdulazeez, A. M., Zeebaree, S. R. M., & Sadeeq, M. A. M. (2018). Design and Implementation of Electronic Student Affairs System. (3), 66-73%V 67. doi:10.25007/ajnu.v7n3a201

Alkawaz, M. H., & Silvarajoo, A. (2019, 13-14 Dec. 2019). *A Survey on Test Case Prioritization and Optimization Techniques in Software Regression Testing.* Paper presented at the 2019 IEEE 7th Conference on Systems, Process and Control (ICSPC).

Amrita, & Yadav, D. K. (2015). A Novel Method for Allocating Software Test Cases. *Procedia Computer Science, 57*, 131-138. doi:https://doi.org/10.1016/j.procs.2015.07.389

Gonçalves, W. F., Almeida, C. B. d., Araújo, L. L. d., Ferraz, M. S., Xandú, R. B., & Farias, I. d. (2017, 21-24 June 2017). *The influence of human factors on the software testing process: The impact of these factors on the software testing process.* Paper presented at the 2017 12th Iberian Conference on Information Systems and Technologies (CISTI).

Haji, L., Zeebaree, S., Jacksi, K., & Zeebaree, D. (2018). A State of Art Survey for OS Performance Improvement. *Science Journal of University of Zakho, 6*, 118-123. doi:10.25271/sjuoz.2018.6.3.516

Hardt, R. (2020, 28 Sept.-2 Oct. 2020). *A toolset to support a software maintenance process in academic environments.* Paper presented at the 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME).

Hooda, I., & Chhillar, R. (2014). A Review: Study of Test Case Generation Techniques. *International Journal of Computer Applications, 107*(16), 5.

Jing, W., Yikun, Z., Ming, Z., Hao, C., Xinhong, H., & Jianxiong, S. (2017, 18-20 June 2017). *A method for test case generation by improved genetic algorithm based on static structure of procedure.* Paper presented at the 2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA).

Kumar, N. S., & Kumar, G. S. (2016, 16-18 Dec. 2016). *Modeling and verification of timed automaton based hybrid systems using spin model checker.* Paper presented at the 2016 IEEE Annual India Conference (INDICON).

Kun, W., & Yichen, W. (2016, 25-28 Jan. 2016). *Software test case generation based on the fault propagation path coverage.* Paper presented at the 2016 Annual Reliability and Maintainability Symposium (RAMS).

Lawanna, A. (2016, 20-23 Sept. 2016). *Test case design based technique for the improvement of test case selection in software maintenance.* Paper presented at the 2016 55th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE).

Lawanna, A., & Wongwuttiwat, J. (2016, 22-25 Nov. 2016). *Test case selection: Vital model for software maintenance.* Paper presented at the 2016 IEEE Region 10 Conference (TENCON).

Note Narciso, E., Delamaro, M., & Nunes, F. (2014). Test Case Selection: A Systematic Literature Review. *International Journal of Software Engineering and Knowledge Engineering, 24*, 653-676. doi:10.1142/S0218194014500259

Panichella, A. (2019, 26-27 May 2019). *Beyond Unit-Testing in Search-Based Test Case Generation: Challenges and Opportunities.* Paper presented at the 2019 IEEE/ACM 12th International Workshop on Search-Based Software Testing (SBST).

Retna, I. a. E. (2012). STUDY PAPER ON TEST CASE GENERATION FOR GUI BASED TESTING. *International Journal of Software Engineering & Applications (IJSEA), 3*(1), 9. doi:10.5121/ijsea.2012.3110

Simon, H., Friedrich, N., Biallas, S., Hauck-Stattelmann, S., Schlich, B., & Kowalewski, S. (2015, 8-11 Sept. 2015). *Automatic test case generation for PLC programs using coverage metrics.* Paper presented at the 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA).

Sun, Y., Qian, H., & Liu, X. (2015, 1-4 Dec. 2015). *Evaluation and Measurement of Software Testing Process Quality Applicable to Software Testing Laboratory.* Paper presented at the 2015 Asia-Pacific Software Engineering Conference (APSEC).

Zebari, R., Zeebaree, S., Jacksi, K., & Shukur, H. (2019). E-Business Requirements for Flexibility and Implementation Enterprise System: A Review. *International Journal of Scientific & Technology Research, 8*, 655-660.

Zeebaree, S., Zebari, R., & Jacksi, K. (2019). Security Approaches For Integrated Enterprise Systems Performance: A Review. *International Journal of Scientific & Technology Research, 8*, 2485-2489.

Zeebaree, S. R. M., Sallow, A. B., Hussan, B. K., & Ali, S. M. (2019, 2-4 April 2019). *Design and Simulation of High-Speed Parallel/Sequential Simplified DES Code Breaking Based on FPGA.* Paper presented at the 2019 International Conference on Advanced Science and Engineering (ICOASE).

Cite this article:

Dathar A. Hasan, Bzar Kh. Hussan, Subhi R. M. Zeebaree , Dindar M. Ahmed, Omar S. Kareem & Mohammed A. M. Sadeeq (2021). The Impact of Test Case Generation Methods on The Software Performance: A Review. *International Journal of Science and Business, 5*(6), 33-44. doi: https://doi.org/10.5281/zenodo.4623940

Retrieved from http://ijsab.com/wp-content/uploads/744.pdf

Published by

IJSAB International