DOI: http://dx.doi.org/10.26483/ijarcs.v13i2.6814

ISSN No. 0976-5697

Volume 13, No. 2, March-April 2022

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

# IMPROVING TEST AUTOMATION USING GENETIC ALGORITHM

Wumi Ajayi[1], Owolabi Bukola[2], Jolaosho Ahmed[3]
[1]Software Engineering Researcher,
[2,3]Department of Computer Sc, LCU, Nigeria ,

***Abstract:*** Software testing is an important step in the creation of software products. Automation is critical in the software industry because it enables software testing firms to increase their test efficiency. Researchers have worked on a variety of automated ways for producing test data to evaluate generated software with various disadvantages. This paper therefore, presented Genetic Algorithm (GA)-based test techniques to automate the development of structural-oriented test data. In this work, random test cases are first generated, then, mutates testing is applied to check it. If satisfied, then process stops. Genetic Algorithms are utilized, since they offered a technique of automatically generating test cases.

***Keywords:*** Algorithm, Automation, Industry, Genetic, Software, Technique, Test Cases, Test-Data.

## 1.0 INTRODUCTION

Software testing is a process that gathers data on the product or service being tested. Methods of testing include running programmes or apps to look for and fix flaws [1]. To get the most out of the system and to catch as many faults as possible early, the process of creating test cases is essential. Cost-wise, this is the best use of time and resources across the entire software development process. Automated testing does not have the capability to perform thorough testing. The Intelligent Security and Automation System (ISAS) architecture was recently utilised in the development of a new application. Robotic testing drastically reduces the cost and duration of trials, which is essential for keeping tabs on the progress of a project [3]. In order to build the finest test suites, it is important to follow the methodology that is being used to evaluate measurements. For any product, testing is the most critical stage. The testing phase is the final stop for any rejections or commission oversights. It's a little more difficult to test computer programmes than it is to practise a framework to guarantee it works properly. It's important to remember that any investigation, audit, survey or walkabout is a test when in doubt. There will be less difficulties with dynamic testing if more successful static attempts have been made. IT has repeatedly shown that the more quickly a problem is discovered and corrected, the lower the incremental cost of fixing it. The technique for testing begins with the depiction of an object [3]. In comparison to the Genetic Algorithm, all evolutionary algorithms achieve a near-optimal result. Tests are used to ensure that a product or service meets certain standards when exposed to a specific set of environmental conditions. This goal is made up of two parts. In this fragment, the primary goal is to verify that the item's requirements point of interest is accurate. The second section demonstrates that the settings and code are perfect matches for all of the criteria. All of the affirmation criteria must be met in order for a task to be considered precise [3].

Software testing, as defined by IEEE, is the process of executing and observing a system or component to assess its performance [3]. Manual or automated testing methods are also acceptable. Testing an application manually entails providing all possible inputs and producing the related outputs. As a result, it's both physically and mentally taxing. Automatic testing relies on design models, which allow testers to focus on more creative work instead of tedious tasks [4].

Effective automation provides a stronger link between the two than traditional testing methods, which separate verification and validation into separate tasks. As part of a comprehensive testing plan that includes both automated and human procedures, these real-time application automation techniques are essential. By using test cases, manual testing is a way to verify the design of the system and find problems [4]. A testing developer's expertise is required for this labor-intensive process. Automated testing environments are built with a wide range of diversity criteria and abstraction levels across a wide range of disciplines in order to overcome the problems inherent in manual testing. Automated testing's ultimate goal is to reduce the amount of time and money it takes to run a test. Figure 1 [4] depicts the entire software testing process.

Whether or whether a piece of software can be used in a variety of contexts is a measure of its quality. Customers' requirements guide the work of software engineers in the software development industry. The program's reliability, portability, usability, flexibility, testability, and efficiency are all evaluated and confirmed in a testing environment after it is designed utilising these characteristics [4]. A product's dependability is measured by how well it performs its functions. Customers and developers must have a mutually acceptable level of trust in a product's performance and reliability. A product's "testability" refers to how well it can be tested, while "usability" describes how well it can be used and how environmentally friendly it is. It is important to know if a product can be improved upon in the future by looking at how flexible it is. The product's efficiency is influenced by the aforementioned performance criteria.

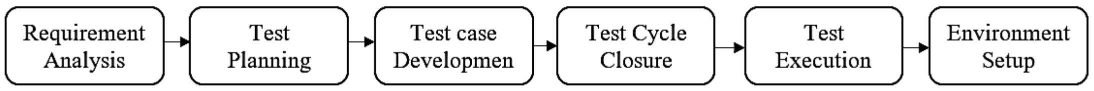

**Figure 1: Software test cycle [4]**

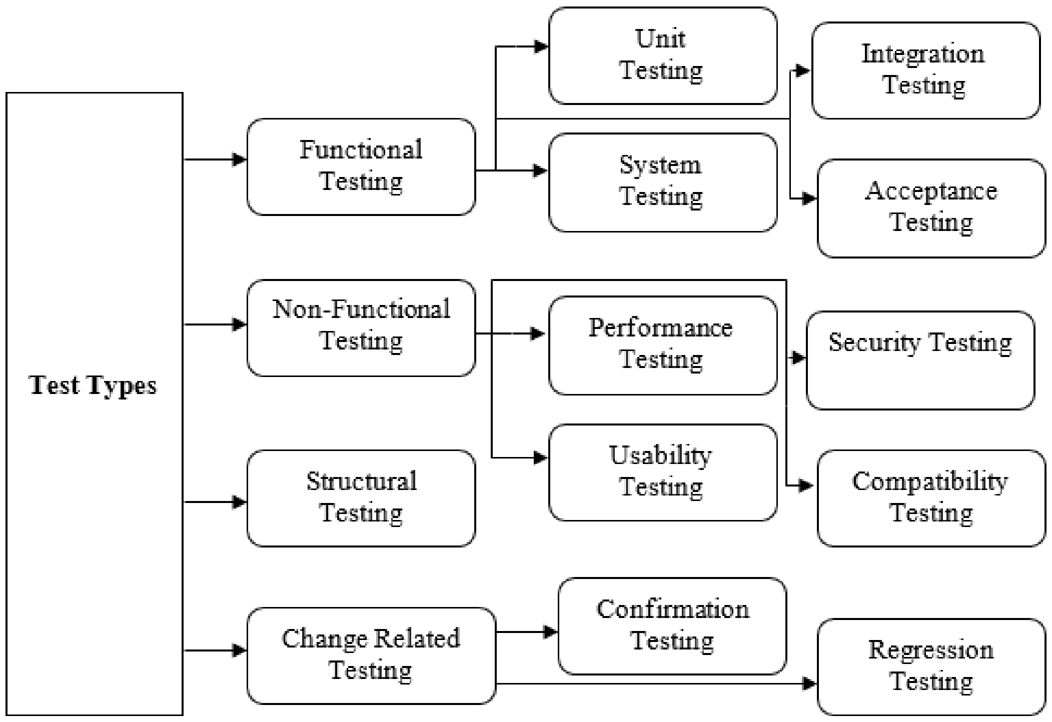

**Figure 2: Software Testing Types[4]**

Software testing enables product design to be improved via standard verification in both static and dynamic environments. Verification is carried out using formal methods and techniques in a static environment, but in a dynamic context, verification is carried out at any point using test cases. In comparison to static analysis, dynamic analysis is far more efficient and effectively detects design flaws. The numerous kinds of testing processes used in software test engineering are shown in Figure 2. Early stage testing minimizes product difficulties by finding flaws and minimizing product failure during implementation. Software testing refers to simple test and coding processes that help eliminate mistakes at each step of the application's development. In the case of verification and validation, the verification process analyzes the appropriate product selection, whilst the validation process analyzes the product attributes. It enables the user or client to validate the produced application in order to ensure that it meets their requirements by resolving any errors.

Software testing's objective is to minimise problems at the lowest feasible cost, as the majority of the testing process consumes almost 45-60 percent of the development budget. Software testing is frequently divided into human and automated testing, depending on the circumstances. The automated process cycle begins with a decision regarding the technique of product evaluation. Following that, an acquisition tool such as automation studio, Experitest, or Testcomplete is used to specify the testing tool. After selecting the suitable tool, the complete script is integrated into the tool, which provides a design plan for constructing the testing method. The final level of automation is execution; it runs the scripts defined in the design and assesses the results. Software testing is further classified into black box and white box testing based on the developers' knowledge of the scripts. When testing is undertaken within the script itself, it is referred to as white box testing, and it requires the testing engineer to be familiar with the scripts. White box testing utilises a number of elements to evaluate the underlying structure of test case data. Figure 3 depicts a categorization structure for white box testing, an extensively used approach in business.

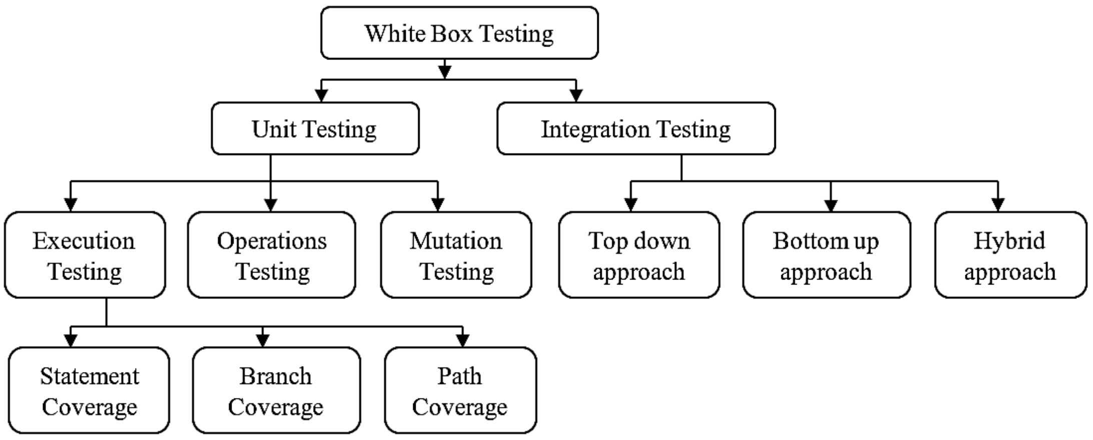

**Figure 3: White Box Testing Classification [4]**

Execution testing and its many variants are commonly used in white box testing. A key benefit of branch coverage is that it offers comprehensive test results for each branch, allowing for a rapid evaluation of the final programme. Tests of the path coverage kind cover all of the steps in the path, making them ideal for small-scale projects. Accurate results are achieved when the script's statements are covered. Due to the fact that it is static, white box testing raises the cost function, offers formal assessment, ignores developer requirements, and is not ideal for all testing engineers.

While white box testing requires a deep understanding of the testing script, black box testing may be performed by anyone. A variety of third-party technologies are used to run tests in different applications. These include Selenium, UI test builder, and applitools eyes Analyzing a vast number of box values, the Black Box only outputs the ones that are necessary. The main testing technique is the correlation between the black box interface and the engineer's expectations. Figure 4 [4] depicts the methods used in black box testing.

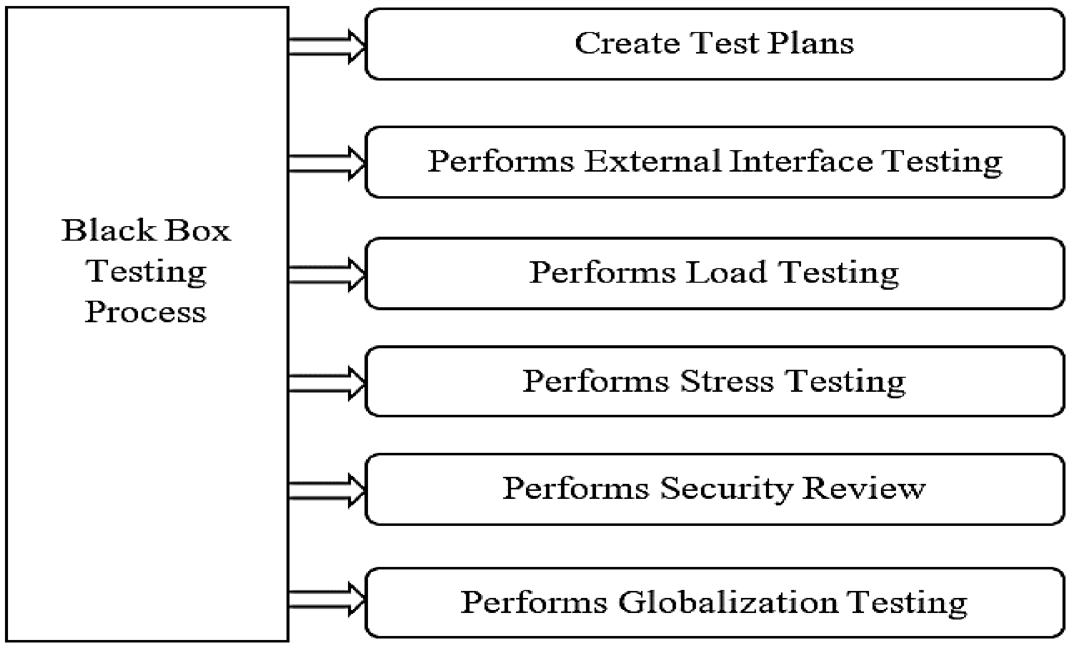
Black Box Testing Process
Create Test Plans
Performs External Interface Testing
Performs Load Testing
Performs Stress Testing
Performs Security Review
Performs Globalization Testing

**Figure 4: Black Box Testing [4]**

## 2.0 AIM AND OBJECTIVES

The aim of this study is to improve software test automation using Genetic Algorithm. The objectives is to;

i. Generate random test cases
ii. Apply mutates testing to check the generated test cases.
iii. If (ii) is satisfied, then stop

## 3.0 LITERATURE REVIEW

### 3.1 Genetic Algorithm

This is the natural selection process through which the best entities are picked to reproduce or develop progeny. Natural selection begins with the selection of the best and fittest individuals within a population. They produce offspring that inherit their parents' characteristics and are incorporated into the next generation [5]. If the parents are more fit, their children will fare better than the parents. Five stages are taken into account in a genetic algorithm:

1. Initial population
2. Fitness function
3. Selection

4. Crossover
5. Mutation

a) **Initial Population:** The operation is initiated by a population of entities. Each entity provides a solution to the problem on ground. Each entity is defined by a collection of gene-related parameters (variables). A chromosome is constructed by stringing together genes (solution). The genetic algorithm of an entity is its alphabet string. Binary values that are frequently utilised (string of 1s and 0s). This is referred to as chromosomal gene transcription [5],[6].

**b) Fitness Function:** This term refers to an entity's fitness level (the ability of an individual to compete with other individuals). Each individual is assigned a fitness score. The fitness level of an individual has an effect on his or her likelihood of being selected for reproduction**.**

**c) Selection**: The selection stage's objective is to identify the most suitable individuals and allow them to pass on their genes to the next generation. Two pairs of persons (parents) are chosen based on their fitness levels. Fitness-oriented entities are more likely to be chosen [5].

**Crossover:** A genetic algorithm's main phase is crossover. For each pair of parents to be mating, a crossover point is selected randomly from inside the DNA.

**d) Mutation:** Certain genes, particularly those in newly generated individuals, can be mutated with a low random frequency. This indicates that some bits in the bit string can be flipped**.**

e) **Termination:** If the population converges, the algorithm terminates (it produces offspring that are not materially different from the preceding generation) [6].

Genetic algorithms' benefits are as follows: They have a random pattern that can be statistically evaluated but cannot be anticipated exactly; (i.e stochastic). Big data is another area where it excels. Probabilistic and non-deterministic rules function well with it. It's a great tool for optimising numerous goals at once. There are some drawbacks to this approach. It can be time consuming and expensive in terms of computer resources, and creating objective functions can be challenging.

An algorithm based on the idea that the best chromosomes have a "menu" of options is known as a genetic algorithm. [3],[5]. The population of chromosomes can be represented using binary, real integers, and permutation in genetic algorithms. Chromosomes can be manipulated via genetic operators (such as selection, crossover, and mutation) to generate more identical copies of themselves. Large and adequate goal functions define a chromosome's fitness. Genetic algorithms, a subclass of stochastic methods, are formed from random searches. In contrast to a genetic algorithm, random actions comprised of iterative and basic random search stages are more likely to yield an answer to a given problem. [7, 8].

In order to identify the problems with the current software testing method, a complete analysis focused on dependability and categorization accuracy. However, due to its computational power, machine learning has received increased attention in software testing. [7] Predicted software flaws using various machine learning algorithms. In the suggested survey, object-oriented criteria such as the learning function, features, and efficiency are used to evaluate machine learning algorithms. It explains how software engineering uses machine learning models.

[8] also studies software engineering automation in great detail. Traditional methods to modern machine learning approaches are covered in this discussion of software engineering automation.

Using an artificial intelligence model, [9] examined the drawbacks of the traditional software testing approach. Traditional models have a lower level of reliability compared to the recommended model. However, there are some drawbacks to the proposed method, such as the time it takes to examine complex script errors.

In a discussion of the difficulties in forecasting software flaws using a hybrid machine learning technique[10]. An efficient and reliable failure prediction model can be created by combining relational association rules and artificial neural networks. Defect avoidance is much easier to compute than software engineering defect prediction.

Defect avoidance in software testing is based on notions of human error, as shown in a paper by [11]. Preventative measures and guidance based on the most common human errors are used in the study to improve the product's performance.

Multi-objective optimization of software performance was devised by [12]. The proposed model improves on previous models in terms of dependability, efficiency, and classification accuracy, to name a few multi-objective criteria.

By [13], they developed a study model for software testing that relies on fault tree analysis. This fault tree analysis is more clear and efficient than standard methods, resulting in a faster calculation time. When compared to fault tree models, the proposed method performs better than machine learning models, however it falls short.

RNN encoder and decoder [14] discussed the difficulties with software reliability models.. The proposed model is more reliable when dealing with large and complex datasets.

Based on component-based software, [15] set up a paradigm for reliability analysis. The issues in the product are found by dissecting the script to its component parts and looking for errors in the suggested model. Because of the division operation, this method gives more accurate results but takes longer to calculate.

Failure prediction using neurofuzzy data has been proposed by [16]. In order to increase testing efficiency, the proposed Inter-version and Inter-project assessment models are included in the research model [17] used a feedback-based prediction technique to study software testing issues. The shortcomings are based on personal experience and are summarised by comparing the product's features to a specific group of people. Using their feedback, we are able to identify and fix bugs more quickly. To summarise and evaluate the feedbacks, the proposed model requires more time. Additionally, the feedbacks themselves may contain uncertainties that could have an adverse effect at times. There are a variety of hybrid methods used in software testing.

A number of approaches to software defect prediction were discussed in [18].

[19] outlined a model for locating software bugs. For software defect prediction, [20] found that a rule mining-based technique is more accurate and efficient. According to

the poll, researchers are actively working on ways to improve reliability.

## 4.0 METHODOLOGY

In this work, random test cases are first generated. Then, mutates testing is applied to check it. If satisfied, then stop [3].

**Optimized algorithm**

1. Mutant is injected into the programme
2. Create test cases of random sample
3. Use mutation score= (number of mutants observed) / (total number of mutants).
4. If the mutant score is Maximum stop, else go to step 5
5. Refine the test case using the mutation score. If a test case have mutation score of 20% or less drop it
6. Using Genetic Algorithm, generate new experiments operations on the remaining test cases. Go to step 3.

The ab algorithm, where a and b are positive integers.

```
1. Start (a, b)
2. Is (a==1)
3. Return 1
4. Is (b==1)
5. Return a
6. P=1, I=1
7. While (i<=b)
8. {P=P*a
9. i++}
10. Return P
```

Four mutants injected into this programme. Now, the algorithm appears as follows.

```
1. Start(a, b)
2. Is (a=1),
3. Return 1
4. Is (b=1)
5. Return a
6. P=1, I=1
7. While (i<=b)
8. {P=P*a
9. i++}
10. Return P
```

The number of mutations was estimated by this algorithm using the flow diagram presented in Figure 1. To begin, a programme had specific mutants introduced. The optimal test case was discovered with the help of mutations [3]. Then, the test case's performance is reviewed, and the first step is taken. There were no other mutations found after that. 'How many mutants have been found?' was the question. The next step is to determine how many mutations have been detected. Defects were missed if the number of mutations found was less than what was required. For a true value of fitness, one must first determine how many mutations there have been. There are two stages to this process: if the number of identified mutations is greater than or equal to 50%, we'll present a best fit; otherwise, the mutant number will be displayed. There is no need to continue looking for defects if the number of mutations found in the second phase is zero. As a result, the test case provided the best answer [3].

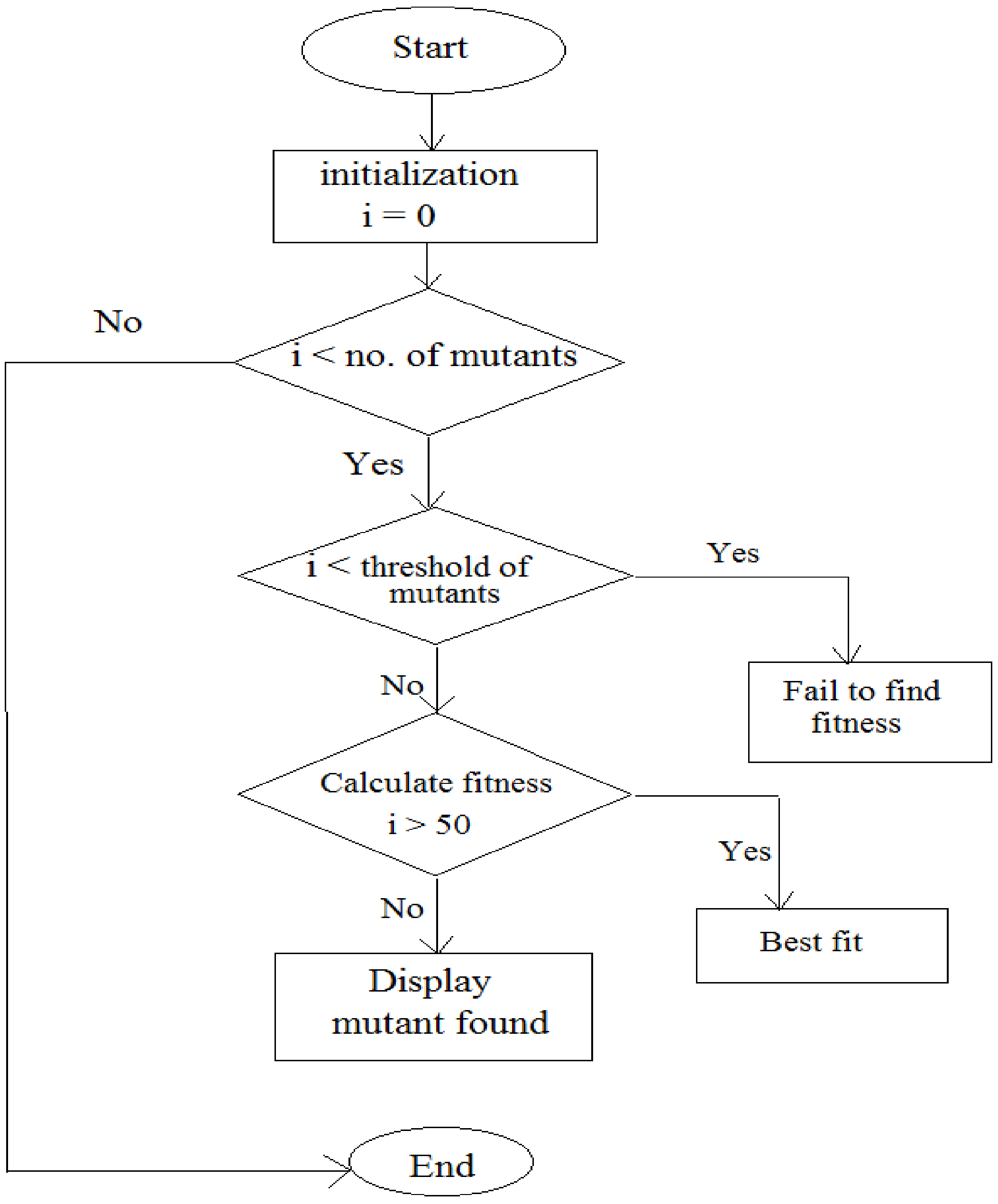

**Figure 5: Optimized Algorithm Flow[5]**

## 4.1 Applications of Genetic Algorithm

There are a number of applications for genetic algorithms, including solving difficult issues like NPC and NP-hard. For a complex problem, they are an excellent way to quickly identify a viable solution. It is more efficient and effective to use genetic algorithms in undiscovered search areas. When used in a real-world situation, genetic algorithms differ dramatically from the results they provide in a lab setting [21]. When it comes to difficult-to-calculate and-analyze circumstances, evolutionary algorithms can be used to come up with answers.

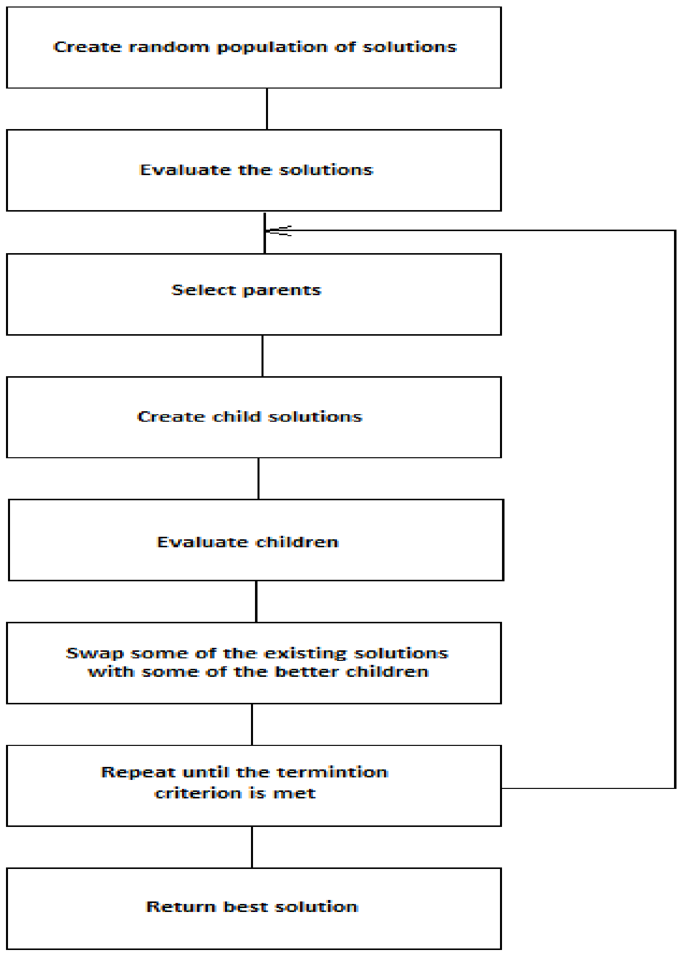
Create random population of solutions
Evaluate the solutions
Select parents
Create child solutions
Evaluate children
Swap some of the existing solutions with some of the better children
Repeat until the termintion criterion is met
Return best solution

**Figure 6: Flow Chart of the Workflow of Genetic Algorithm Used for Test Case Generation In Software Testing [21].**

## 5.0 CONCLUSION

GAs, for example, were examined in this article to see if they helped with software testing. According to the findings, genetic algorithm-based software testing becomes more efficient as the number of test cases grows. Using Random Testing Methods is inefficient because the data points are not time-dependent. As a result, Genetic Algorithms are used to speed up and increase the efficiency of software testing since they provide a method of automatically generating test cases. It is possible to use evolutionary generation of test cases, which has been proven to be more efficient and cost-effective than random testing.

## 6.0 REFERENCES

[1]. Maheshwari V, Prasanna M. Generation of test case using automation in software systems–a review. *Indian Journal of Science and Technology*. 2015 Dec 24;8(35):1-9.

[2]. Pandey M, RajasekharaBabu M, Manasa J, Avinash K. Mobile based automation and security systems. *Indian Journal of Science and Technology*. 2015 Jan; 8(S2):12–6

[3]. Mateen A, Nazir M, Awan SA. Optimization of test case generation using genetic algorithm (GA). arXiv preprint arXiv:1612.08813. 2016 Dec 28.

[4]. Shakya S, Smys S. Reliable automated software testing through hybrid optimization algorithm. *Journal of Ubiquitous computing and communication technologies (UCCT)*. 2020 Aug 4;2(03):126-35.

[5]. https://www.analyticsvidhya.com/blog/2017/07/introduction-to-genetic-algorithm/. Accessed online December 7, 2021

[6]. https://www.section.io/engineering-education/the-basics-of-genetic-algorithms-in-ml/. Accessed online December 7, 2021

[7]. Singh A, Bhatia R, Singhrova A. Taxonomy of machine learning algorithms in software fault prediction using object oriented metrics. Procedia computer science. 2018 Jan 1;132:993-1001.

[8]. Haghighatkhah A, Banijamali A, Pakanen OP, Oivo M, Kuvaja P. Automotive software engineering: A systematic mapping study. *Journal of Systems and Software. 2017* Jun 1;128:25-55.

[9]. Elmishali A, Stern R, Kalech M. An artificial intelligence paradigm for troubleshooting software bugs. Engineering Applications of Artificial Intelligence. 2018 Mar 1;69:147-56.

[10]. Miholca DL, Czibula G, Czibula IG. A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks. Information Sciences. 2018 May 1;441:152-70.

[11]. Huang F, Bin LI. Software defect prevention based on human error theories. *Chinese Journal of Aeronautics*. 2017 Jun 1;30(3):1054-70.

[12]. JakubovskiFilho HL, Ferreira TN, Vergilio SR. Preference based multi-objective algorithms applied to the variability testing of software product lines. *Journal of Systems and Software*. 2019 May 1;151:194-209.

[13]. Li HW, Ren Y, Wang LN. Research on software testing technology based on fault tree analysis. Procedia Computer Science. 2019 Jan 1;154:754-8.

[14]. Wang J, Zhang C. Software reliability prediction using a deep learning model based on the RNN encoder–decoder. Reliability Engineering & System Safety. 2018 Feb 1;170:73-82.

[15]. Kaliraj S, Bharathi A. Path testing based reliability analysis framework of component based software system. Measurement. 2019 Oct 1;144:20-32.

[16]. Juneja K. A fuzzy-filtered neuro-fuzzy framework for software fault prediction for inter-version and inter-project evaluation. Applied Soft Computing. 2019 Apr 1;77:696-713.

[17]. Xiao P, Liu B, Wang S. Feedback-based integrated prediction: Defect prediction based on feedback from software testing process. *Journal of Systems and Software*. 2018 Sep 1;143:159-71.

[18]. Rhmann W, Pandey B, Ansari G, Pandey DK. Software fault prediction based on change metrics using hybrid algorithms: An empirical study. *Journal of King Saud University-Computer and Information Sciences.* 2020 May 1;32(4):419-24.

[19]. Zhang XY, Zheng Z, Cai KY. Exploring the usefulness of unlabelled test cases in software fault localization. *Journal of Systems and Software*. 2018 Feb 1;136:278-90.

[20]. Shao Y, Liu B, Wang S, Li G. Software defect prediction based on correlation weighted class association rule mining. Knowledge-Based Systems. 2020 May 21;196:105742.

[21]. Sharma A, Patani R, Aggarwal A. Software testing using genetic algorithms. International Journal of Computer Science & Engineering Survey. 2016 Apr;7(2):21-33.