International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5
DOI: https://doi.org/10.17762/ijritcc.v11i5.6615
***Article Received**:05 March 2023 **Revised**: 02 April 2023 **Accepted**: 24 April 2023*

Finite State Testing of Graphical User Interface using Genetic Algorithm

Sumit Kumar[1], Nitin[2], Mitul Yadav[3]
[1]Department of Computer Science and Engineering
Veer Madho Singh Bhandari Uttarakhand Technical University
Dehradun, India
sumitnadar@gmail.com
[2]College of Engineering and Applied Science
University of Cincinnati
Cincinnati, USA
delnitin@gmail.com
[3]Department of Computer Science
Dev Bhoomi Institute of Technology
Dehradun, India
mitulyadav1905@gmail.com

Abstract— Graphical user interfaces are the key components of any software. Nowadays, the popularity of the software depends upon how easily the user can interact with the system. However, as the system becomes complex, this interaction is also complicated with many states. The testing of graphical user interfaces is an important phase of modern software. The testing of GUI is possible only by interacting with the system, which may be a time-consuming process and is generally automated based on the test suite. The test suite generation proposed in this paper is based on the genetic algorithm in which various test cases are generated heuristically. For performance validation of the proposed approach, the same has been compared with a variant of PSO, and it found that GA is slightly better in comparison to the PSO.

Keywords- Graphical User Interface (GUI), Software Testing Life Cycle (STLC), Software Development Life Cycle (SDLC), Genetic Algorithm.

I. INTRODUCTION

Most of the Software testing process requires the running of a significant quantity of test scenarios which requires a significant amount of time to complete. The effort required in software is measured in person months which means the testing process will also require a huge amount of time as it is not possible to increase the team size after a certain threshold. In the study, testing was performed to test the functionality of the software along with hardware has accounted for 79 billion euros. However, as the complexity of the software is increasing with the advancement of development, it was projected that by the end of 2014, the cost would grow to 100 billion euros[1]. In the software industry, with the advancement of development, software testing is now being automated; however, it totally depends on the project budget. When the manual evaluation is to be conducted, the analyzer usually analyzes the boundary values, which have a higher probability of errors, and the log of each case is maintained manually. Manual testing may contain a huge amount of error due to human intervention. In auto-testing, this interaction of humans with software is reduced. The most popular tool used for testing is JUnit. It has been observed that the total testing cost is reduced significantly if the procedure is well-designed and carried out [2]–[4]. In order to drive the research in the right direction based on the problem faced by the industry, a set of best practices has been defined by the action research methodology[5]. However, if the test cases generated in the case of automated testing are not aligned with the standard issues and do not cover the complete code, this may lead to a waste of time and resources, which may be higher than that of manual testing[6]. The ideal way to generate the test cases is to start from the initial phase of software testing, that is, requirement gathering and generating the model from the requirement. Testing based on a model is termed model-based testing. The different phases of the model-based tests are as follows:

- Using the requirement from the client to generate the model.
- Using the requirement, generate the test suite based on the expectations of the client.
- Compare the results for validation that the software is working correctly or not.

As we know, the testing companies of the two terms, that is, verification that the product developed meets the expectation of the stakeholders and validation does the behavior of the software is identical to the expectation of the client. The process of testing ensures that the quality of the software developed is high. In the table I, the summarization of the system is done based on layers. Initially, the system is viewed as front end and back end. The layers that describe the front end are GUI, and the backend is the System core. The component of each of these will be GUI source code architecture for the front end and software architecture for the back end. Now to test regression testing manually and acceptance testing is to be performed for the front end, while for backend unit and integration testing is performed and the same thing in an automated manner is to be performed, the tags will be used for GUI testing and Unit testing is used for testing the backend.

TABLE I. COMPARES THE DIFFERENT LAYERS OF SOFTWARE AND WAYS OF TESTING

System View	System Layers	System Components	Manual Testing	Automated Testing
Front End	GUI Model	GUI Source Code Architecture	Regression System and Acceptance Testing	Tag-based GUI Testing
Back End	System Core	S/W Architecture	Reviews Unit Testing and Integration Testing	Unit Testing

The Graphical user interface is the bridge between the user and backend logic that exploits the graphical capabilities of the computer to make easy interaction. The GUI has become complex as the functionality of the system is becoming complex. To ensure that the system works effectively, it is mandatory to ensure the software testing is done thoroughly. The most important thing about GUI testing is that the order of interaction also impacts the behavior of the system. The different interactions are to be covered, various test suites are written, and regression testing is performed. Figure 1 describes the calculator designed using MATLAB, which is considered the software under test.

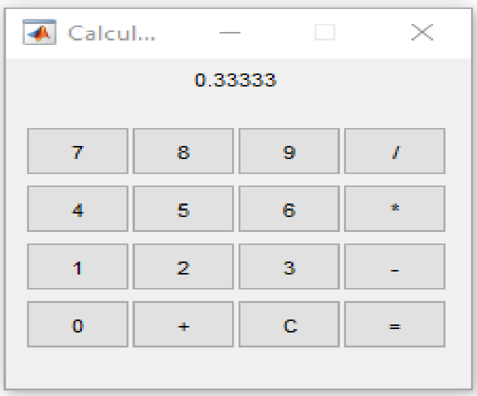

Figure 1. Describes calculator as software under test

A finite state machine is basically defined by the five tuples that are a set of states, the input symbols and the output, the transition function, and the final state. Input is given to a particular state and based on the input; the transition is made to another state.

II. RELATED WORK

In this section, various techniques used to generate the test cases have been discussed. In software testing, the various basic operation of GUI, like click, right-click, mouse-click, and many more, defines the GUI primitives of the software. The test suite basically defines the path from lower order to higher order for the formal language. In general, it could be imagined as the linear process of test case generation and testing on these test cases. Based on the case study, it has been identified that this method requires coverage of all paths, transitions, and all possible interactions in order to identify the faults effectively[7].

Another method suggested in [8] used UML based model, which comes in the category of partition method. In this approach, the test cases are generated based on activity control flow; using this approach helps in improving the efficiency of the testing. The basic idea is to first transform the GUI into UML-based models. The tool based on this approach, TDE/UML, was developed to generate the test cases. This approach covers the various graph coverage properties like a round trip, All paths, various activities, and all paths. The tools also have the capability to have sample data, choice-based coverage to test specific scenarios, and Complete coverage to perform thorough testing.

In [9] suggested balancing the cost of testing to the effectiveness of fault detection. The authors have suggested a framework that can generate the model and perform the analysis of the scenario to be feasible and cost-effective. Based on the analysis, the event interaction coverage is generated. This approach also ensures that various possible states of the GUI are covered or not.

The consolidation-based model was generated in[10][11]. The models are converted into event flow models, which are ripped from the software using the special module termed GUI ripper. The GUI ripper is one of the modules that are available in the first GUI testing tool that is GUITAR. Based on the study by the authors, it has been identified that the suggested technique is effective in identifying the faults in the software under test.

In another study, an approach based on the structure of the GUI is being studied[12]. This model is based on Hierarchical Predicate Transitions Nets (HPrTNs), which consider the events and states with equal weightage. The study of these techniques leads to the conclusion that this technique is capable of all the possible coverages like states, transitions, and threads. However, the main issue with this study is that no information about the automation tools has been provided.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5
DOI: https://doi.org/10.17762/ijritcc.v11i5.6615
***Article Received**:05 March 2023 **Revised**: 02 April 2023 **Accepted**: 24 April 2023*

III. Proposed Approach

The test case generation has been done using various nature-inspired techniques. These techniques that have been popularly used are genetic algorithm, differential evolutionary algorithm, and particle swarm optimization[13]–[18]. A genetic algorithm is one of the popular techniques used to explore the search space using Darwin's theory of survival of the fittest. In this paper, the proposed approach based on a genetic algorithm has been suggested. The reason for selecting the genetic algorithm is due to the reason of easy encoding for the binary problems, and secondly, it is easy to explore the huge search space easily using Heuristic methods rather than relying on the classical approach that may lead to NP-hard solutions. The detailed algorithm describing the proposed approaches is stated below:

A. Algorithm:

```
Input: Population size (N), Pc Probability of Crossover, Pm
        Probability of Mutation
  Output: The best set of Individuals
Step 1: Identify the total states for generating the model
Step 2: Group the states into different classes based on their
        frequency and allocate proportionate weight to them
Step 3: Initialize the random population of size in the form of
        chromosome to term as the parent population
Step 4: Evaluate the fitness value as the population generated
Step 5: Select the N good solution from the matting pool
Step 6: Apply crossover on the parent population to generate
        offspring
Step 7: Based on the probability of mutation, randomly mutate
        the population
Step 8: if the Stop criteria are met, step 8; else, apply selection
        and go to Step 4.
```

In order to understand the working of the proposed algorithm, let us consider the GUI of calculation; the numbers are 0 to 9.

Given: Number → 0-9,
Reset → C
Symbol → +, -, *, /

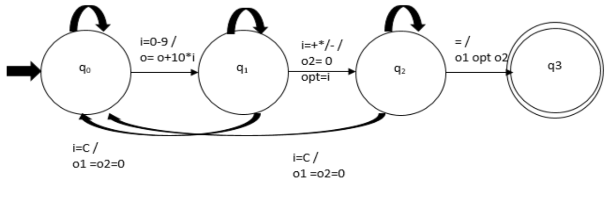

Figure 2. Describes software under test on the state machines

Figure 2 describes the finite state machine of the calculator. In figure 2, the initial state is represented by q_0. The various possible state transition from q_0 is either a number from 0 to 9 will be pressed, or C that is the reset button is pressed; thirdly, some operator might be pressed. If C is pressed, the state will reset to the initial state q_0. This has been described by the self-loop and back loops in the transition diagram. However, if a number is pressed state is transited to q_1, and the display will show the number or, every time, multiplied by ten and added to the number. Once the operator is pressed, the system again changes its state from q_1 to q_2 and receive another set of number. If an operator is pressed, the result will be displayed as output and can be considered the final state of the model.

This state model is coded that can be used to crosscheck the results generated by the actual software and the model. The test cases that are to be compared are generated with the help of a genetic algorithm once the anomaly is identified between the two models and software that is reported as a fault in the actual software.

B. Implementation

Figure 3 explains the flow chart for the genetic algorithm.

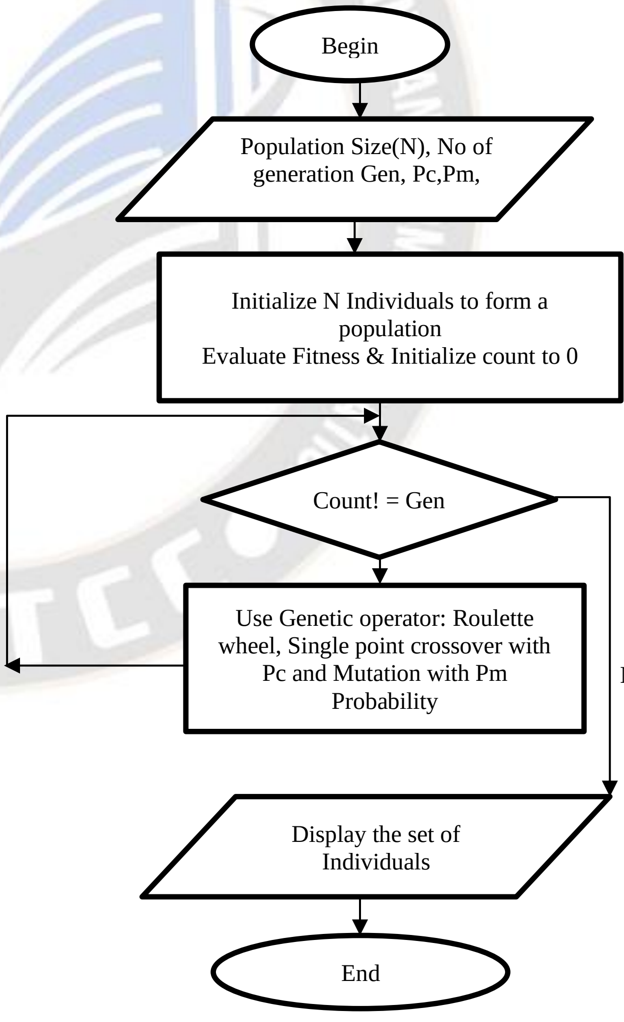
Begin
Population Size(N), No of generation Gen, Pc,Pm,
Initialize N Individuals to form a population
Evaluate Fitness & Initialize count to 0
Count! = Gen
Use Genetic operator: Roulette wheel, Single point crossover with Pc and Mutation with Pm Probability
No
Display the set of Individuals
End

Figure 3. Flowchart of Genetic algorithm

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5
DOI: https://doi.org/10.17762/ijritcc.v11i5.6615
***Article Received**:05 March 2023 **Revised**: 02 April 2023 **Accepted**: 24 April 2023*

The model used in this study is developed using the MATLAB script. The input to this script will be the set of inputs that are generated by the genetic algorithm. The various customization of genetic algorithms in view of the problem is stated below:

1) Encoding Process

Encoding describes the process of representing the problem of the real world to the search space defined by the algorithms. In the current problems, the possible events in the case of the calculator are pressing the buttons of the calculator. These events can be represented by a unique number which can be easily coded into binary values; hence each event will have a unique id that can be represented by four bits. Another challenge is what should be the size of the test case to keep it small and reasonable; keep it at 5. Hence the five events will be represented by twenty-length chromosomes. The initial chromosome was generated randomly. The mathematical formulation for the fitness values is:

$$f = \sum_{i=1}^{|L|} W(i) \tag{1}$$

In the above equation, the weight values of each event are computed based on the number of events that is L. The computation of the weight matrix is described in table II. In this table II, the events that are the button pressed have been classified into four categories. The first and third category is the input in the form of numbers that will be from 0 to 9. Hence total weight will be the probability of occurrence of the event that is 1/10. Similarly, the other weights are calculated. However, Operator C is considered to have a weight of 0 as this reset the state to initial and should be avoided.

TABLE II. DETAILS ABOUT THE DIFFERENT EVENTS AND THEIR WEIGHTS

S. No	Category	Possible event	Weight assigned
1	Operand A	0,1,2,3,4,5,6,7,8,9	0.1
2	Operator	*,-,+,\	0.25
3	Operand B	0,1,2,3,4,5,6,7,8,9	0.1
4	E.val	=,or +,*, \, -	0.20

2) Selection Operator:

After the creation of the test cases using the random population, their fitness is calculated using the fitness function defined for the evaluation. Now to differentiate the good solution from the bad solution, the selection algorithm of the roulette wheel is proposed. In this approach, relative fitness is taken into consideration while assigning spaces on the wheel. The good solution will have higher space on the wheel in comparison to the bad solution. As the area on the wheel has increased for a good solution, this ensures a higher probability of selection of the good solution in comparison to the bad solution. The same has been described in figure 4.

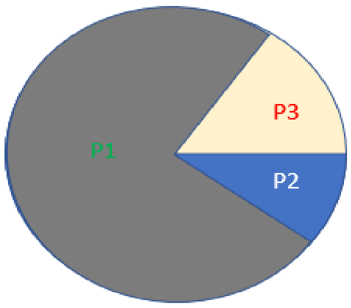

Figure 4. Describes the roulette wheel with a good solution having a large area of the wheel

3) Crossover Operator:

The set of two good solutions that represent parents are selected. To generate the offspring, the single-point crossover is used. In this approach, random crossover sites are selected, and the chromosome values of the mixed to generate a pair of offspring. In general, the generation of offspring from parents is kept high and is usually kept in the range of 0.7 to 0.9. In the current study, A value of 0.8 has been entered.

4) Mutation Operator:

As for the evolutionary algorithms, there is always some external environmental factor, and there is some factor of adaptation. This adaptation is provided using a mutation operator. However, this change is slow in nature but can help when the solution is a struct in the local minima. In the current study, the rate of mutation is set to 0.2.

5) Termination:

There are two possible grounds for terminating the genetic algorithm:

a) Using the threshold on fitness function: As the algorithm explores the new solution, the value of the fitness function also improves. After each generation, the best individual fitness is recorded, and once the defined threshold is reached, the algorithm will exit from the generation cycle and report the solutions.

b) Using the generation count: In some cases, there is a probability that the threshold of the fitness function may not be achieved, which may lead to infinite execution. In order to avoid such scenarios, the generation number is fixed in advance. In the current study, this value is set to 20.

IV. RESULT AND DISCUSSION

Following 20 rounds of the genetic algorithm with a 100-size population, the results of the last iteration have been recorded; these results will contain the chromosome values, which are the test cases for testing the software under test. However, in order to test the software with GUI, the test cases generated will be provided as the input to another software that will generate the events on the software under test; the software used here is Autoit. Once the test case is executed, the results obtained are recorded. At the same time, results are also generated by the model designed by the tester. The example of the test case generated for size five chromosome is as follows

0	1	14	2	12

This test case can be read as follows:

0 1 14 2 12

The expression in the calculator is processed if any of the operators is pressed on the calculator. The results generated from the calculator will be generated will be recorded in the log files for further verification of the software behavior. These logged results are compared with the model. In case of fault, the test cases with such scenarios will get identified. The auto script used for GUI testing is given below:

```
Win_Activate("GUI_Calculator")
Mouse_Click (“left," 31,81)
Mouse_Click (“left," 89,190)
Mouse_Click (“left," 84,149)
Mouse_Click (“left," 188,183)
```

In the above script, the first command used to start the application in the windows environment that is GUI_Calculator. The second command is mouseclick from the left corner to the mentioned coordinates. The complete script is autogenerated by the program.

TABLE III. FITNESS-BASED CHROMOSOME RESULT

Ind. No	G.1	G.2	G.3	G.4	G.5	Fitness	Model Value	GUI Value
11	7	6	15	5	2	0.4	52	52
95	4	12	15	7	5	0.5	75	75
29	4	12	6	9	9	0.6	2796	2796
38	8	3	2	12	1	0.6	832	832
94	14	1	5	2	0	0.6	1520	1520
13	7	8	13	3	9	0.6	2	2
99	13	8	13	9	6	0.7	0	0
58	15	8	9	2	13	0.7	1	1
47	13	4	11	13	0	0.8	E	NaN
40	12	6	14	8	12	0.8	64	64

From table III, the best value from the output produced by the model and GUI are displayed together with the model, and for division by zero, NaN is displayed. However, the output of the model is E, which is the opposite of what was expected.

Table III describes the result obtained from the GA-based approach, which is compared with model-based results. To understand this table, consider the numbers as the exact number, and 10 to 14 represents +, -, *, /, =.

In order to compare the performance of the proposed algorithm, we have implemented the PSO algorithm as stated in [19]. Both the algorithms were executed on a population size of 100 for 20 iterations. As the heuristic algorithms may not give the best result in one execution there for ten runs of each of the algorithm has been executed. The table described below describes the performance of the algorithm in each run.

TABLE IV. COMPARE THE PROPOSED GA-BASED APPROACH WITH PSO BASED APPROACH

Run Number	Generation 5		Generation 10		Generation 15		Generation 20	
	GA	PSO	GA	PSO	GA	PSO	GA	PSO
R1	0.5	0.6	0.6	0.6	0.9	0.8	0.8	0.6
R2	0.7	0.6	0.8	0.8	0.7	0.8	0.9	0.8
R3	0.6	0.5	0.7	0.6	0.7	0.7	0.8	0.7
R4	0.6	0.5	0.8	0.8	0.7	0.8	0.9	0.7
R5	0.6	0.6	0.8	0.6	0.8	0.7	0.8	0.9
R6	0.5	0.7	0.7	0.6	0.8	0.8	0.9	0.9
R7	0.7	0.6	0.6	0.6	0.7	0.7	0.8	0.7
R8	0.7	0.6	0.7	0.8	0.8	0.8	0.8	0.7
R9	0.5	0.7	0.7	0.7	0.7	0.7	0.9	0.8
R10	0.6	0.6	0.6	0.7	0.8	0.7	0.9	0.7
Average	**0.6**	**0.6**	**0.7**	**0.68**	**0.76**	**0.75**	**0.85**	**0.75**
Standard Deviation	**0.08**	**0.07**	**0.08**	**0.09**	**0.07**	**0.05**	**0.05**	**0.10**

V. CONCLUSION

GUI testing is one of the crucial tasks for the software development life cycle. In this paper, we have compared the performance of GA and PSO using the weighted fitness assigned to the events. This makes an alternative solution to code coverage. Similarly, to make the execution fast, we can limit the number of events by chromosome size. On comparing the solutions based on the fitness of the genetic algorithm and PSO, we can observe that the Genetic algorithm is on a higher edge in comparison to particle swarm optimization. In the future, we can tune the fitness function to evaluate many events and integrate the Hadoop environment to perform parallel execution to make testing faster.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 5
DOI: https://doi.org/10.17762/ijritcc.v11i5.6615
***Article Received**:05 March 2023 **Revised**: 02 April 2023 **Accepted**: 24 April 2023*

REFERENCES

[1] "Software Testing Spends to Hit b[1],100bn by 2014 | ITOnews.eu." https://itonews.eu/software-testing-spends/ (accessed Dec. 14, 2022).

[2] Y. Amannejad, V. Garousi, … R. I.-2014 I. S., and undefined 2014, "A search-based approach for cost-effective software test automation decision support and an industrial case study," ieeexplore.ieee.org, pp. 302–311, 2014, doi: 10.1109/ICSTW.2014.34.

[3] V. Garousi and F. Elberzhager, "Test Automation: Not Just for Test Execution," IEEE Softw., 2017, doi: 10.1109/MS.2017.34.

[4] V. Garousi, E. Y.-2018 I. I. C. on, and undefined 2018, "Introducing automated GUI testing and observing its benefits: an industrial case study in the context of law-practice management software," ieeexplore.ieee.org, 2018, doi: 10.1109/ICSTW.2018.00042.

[5] "5 Reasons Why Test Automation Can Fail | Thoughtworks."

[6] V. Garousi and E. Yildirim, "Introducing automated GUI testing and observing its benefits: An industrial case study in the context of law-practice management software," Proc. - 2018 IEEE 11th Int. Conf. Softw. Testing, Verif. Valid. Work. ICSTW 2018, no. April, pp. 138–145, 2018, doi: 10.1109/ICSTW.2018.00042.

[7] I. G. Harris, "Fault models and test generation for hardware-software covalidation," IEEE Des. Test Comput., vol. 20, no. 4, pp. 40–47, 2003, doi: 10.1109/MDT.2003.1214351.

[8] M. Vieira, J. Leduc, B. Hasling, R. Subramanyan, and J. Kazmeier, "Automation of GUI testing using a model-driven approach," Proc. - Int. Conf. Softw. Eng., pp. 9–14, 2006, doi: 10.1145/1138929.1138932.

[9] Q. X.-P. of the 28th international conference on and undefined 2006, "Developing cost-effective model-based techniques for GUI testing," dl.acm.org, Accessed: Dec. 14, 2022. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/1134285.1134473

[10] A. M. Memon, "An event-flow model of GUI-based applications for testing," Softw. Test. Verif. Reliab., vol. 17, no. 3, pp. 137–157, 2007, doi: 10.1002/stvr.364.

[11] A. M.-P. of T. O. 2004 workshop on and undefined 2004, "Developing testing techniques for event-driven pervasive computing applications," cs.umd.edu, Accessed: Dec. 14, 2022. [Online]. Available: http://www.cs.umd.edu/~atif/pubs/MemonBSPC2004.pdf

[12] H. Reza, S. Endapally, and E. Grant, "A model-based approach for testing GUI using hierarchical predicate transition nets," Proc. - Int. Conf. Inf. Technol. Gener. ITNG 2007, no. April, pp. 366–370, 2007, doi: 10.1109/ITNG.2007.9.

[13] D. Berndt, J. Fisher, L. Johnson, J. Pinglikar, and A. Watkins, "Breeding software test cases with genetic algorithms," in 36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the, 2003, p. 10 pp. doi: 10.1109/HICSS.2003.1174917.

[14] A. Sheta, "Reliability Growth Modeling for Software Fault Detection Using Particle Swarm Optimization," in 2006 IEEE International Conference on Evolutionary Computation, pp. 3071–3078. doi: 10.1109/CEC.2006.1688697.

[15] H. S. H. Khin, C. YoungSik, and S. P. Jong, "Applying particle swarm optimization to prioritizing test cases for embedded real time software retesting," Proc. - 8th IEEE Int. Conf. Comput. Inf. Technol. Work. CIT Work. 2008, pp. 527–532, 2008, doi: 10.1109/CIT.2008.Workshops.104.

[16] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon, "Using GUI ripping for automated testing of android applications," 2012 27th IEEE/ACM Int. Conf. Autom. Softw. Eng. ASE 2012 - Proc., pp. 258–261, 2012, doi: 10.1145/2351676.2351717.

[17] P. Godefroid et al., "Automating Software Testing Using Program Analysis," IEEE Softw., vol. 25, no. 5, pp. 30–37, Sep. 2008, doi: 10.1109/MS.2008.109.

[18] R. L. Becerra, R. Sagarna, and X. Yao, "An evaluation of differential evolution in software test data generation," 2009 IEEE Congr. Evol. Comput. CEC 2009, pp. 2850–2857, 2009, doi: 10.1109/CEC.2009.4983300.

[19] K. Senthil Kumar and A. Muthukumaravel, "Optimal test suite selection using improved cuckoo search algorithm based on extensive testing constraints," Int. J. Appl. Eng. Res., vol. 12, no. 9, pp. 1920–1928, 2017.