A Literature Survey on Finite State Testing of Graphical User Interface

Sumit Kumar  
Ph.D. Scholar, Uttarakhand Technical University, Dehradun, Uttarakhand, India.

Nitin  
Associate Professor, Jaypee Institute of Information Technology, A-10, Sector-62, Noida-201307 Uttar Pradesh, India.

Abstract  
The automated graphical user interface (GUI) testing has been a challenging task. The manual testing done in the GUI requires huge amount of time, so in order to reduce this time, many automated GUI testing techniques have been proposed. The most common way testing with any software is based on the finite state machine (FSM). It has been used since we have the software that doesn’t have the GUI. So the various efforts have done to use the same finite state model in the graphical testing. In this paper, we have presented the survey of the various GUI testing techniques, mostly that are mostly based on finite state machines. The various approaches have described that may range from the combinatorial to artificial intelligence approach. There is still a gap that to be filled as the tools are still the part of academic research but not the part of industries.

Keywords: Finite state machines; Software testing; graphical user interface; test case generation

Introduction  
The testing is the most important part of the software development life cycle. As the software is becoming the graphical user interface have been introduced to the software. These GUI has helped in the easy interaction between the user and the client but has increased the search space for the testing. Now with GUI, there are various events associated with each element existing in the GUI. It will have the permutation along with the various inputs require performing the particular task. There have various ways to generate the test cases for the software under test (SUT). The most important method for the testing of the software is based on the finite state machine or finite state model (FSM).

Finite State Machines And Testing  
The finite state machine is a machine which consists of a set of states, various inputs and outputs. The input at particular state leads to the transition to the next state. In section 2 we have discussed the basics of finite state machine and how it has been used in the GUI testing. The paper has been divided into four sections. In section 2 we have discussed about the Finite state machines. In section 3 we have shown the detailed survey on the various FSM based on GUI testing techniques. In section 4 shows the conclusion of the paper.

Finite State Machines And Testing  
The finite state machine is a machine which consists of a set of states, various inputs and outputs. The input at particular state leads to the transition to the next state. This has been described in the Fig. 1. The states are described by the circles and the transition is described by the arrows directing from the current state to the next states.

Figure 1: Describing the Finite state machine

Input set in the case of the GUI will be objects like icons, buttons that can be controlled by the various input and output devices [6]. The output set can be divided into two parts [6].

A. Desirable events: If the response to the particular events is correct, then it is called desirable event.

B. Undesirable events: If the user gets unexpected result it is called as undesirable events.

When we have a chain of sequence from one state to another it is referred as the interaction sequence. The complete details of the finite state machines on testing GUI can found in the [6].

Literature Review  
Apfelbaum et al describes the model based testing. The traditional testing is the “Black Box Testing”. This technique is shown in Fig. 2. From the models we can understand the design of the system. Models are used to design the reusable...
framework for product development. Models are accepted in software engineering process for analysis and design process. Modeling is nothing but the knowledge about the system design and reuse the knowledge of the system grows. Apfelbaum et al states that the test designer is having good knowledge about the GUI design than the GUI tester. The GUI goals should be set by the designer and the objective of the testing is to meet the mentioned goals. The goal is the behaving the input and output is the sequence to perform particular event and is referred to as a test case. It uses the input and generates the hierarchal model based on them. The designing phases of model based testing are as follows [1]:

![Figure 2: Process of Black box testing [1]](image)

Provides a Stimulus to the system
In this phase the user can act like selecting a button on a GUI, invoking a function from the API or dialing a number on a telephone.

Verify that the System responded correctly
In this phase the verification of system will be done.

Set up the system testing environment
Testing process needs to be set up the environment. To set the environment a path will be followed by this phase.

Report and/or log the results
Finally the report of results will be reported to the system. The Shehady et al has generated a method for the automation of the testing. It is based on the formal model named as variable finite state machine (VFSM). The model can be converted into equivalent finite machines. The VFSM is converted into FST in five phases which are follows [2]:
1) Model Definition
2) Conversion
3) Test Generation
4) Test Execution
5) Analysis

We can explain all the five steps by seeing the Fig. 3. In the first step VFSM model will be constructed. The VFSM stands for Variable Finite State Machine.

![Figure 3: Steps in Testing of User Interface [2]](image)

In the second phase conservation has been taking place. VFSM is taken as input and FSM produce as output. In the third phase test of input and expected output sequence has been generated from the FSM. In the fourth phase test sequences are executed by the interface and produce the actual output. In the final phase, expected output is compared to the actual output as shown in Fig. 3.

The VFSM model has been applied to Navigator U and found 3 errors that were not found in the months of debugging [2]. The VFSM was introduced to reduce the complexity of modeling interfaces.

Dalal et al. has proposed a model based approach. Automatic is also called model based approach. Model based approach is used to develop the data model to generate tests. The generic test generation system is shown in Fig. 4. Model based testing approach is used to reduce the cost of test generation. Effectiveness of the test is increased and also shortening the testing cycle by model based test approach. It generates the test cases which are generated in the early stages of the development based on the requirements. The test suite includes input and expected output. The model works in an automated manner to generate the test cases[3].

White et al. has addressed the Complexity of the Graphical User Interfaces (GUI) testing due to the large test space and its to with the various permutations. GUI testing is not an easy task the reason behind this is as follows [4]:
1) The GUI has an enormous number of states and every state should be tested.
2) The GUI input space is very large because of permutations of inputs and events which effect the GUI.
3) The complex dependencies exist in the GUI system.
4) The GUI system can cause external effects at any time.

The author resolves all these issues in this paper [4]. The author focuses on complete sequence iteration (CIS) which
utilizes the finite state model to generate the test cases. The CIS approach can reduce the number of test cases to be checked and approach is scalable. The authors have suggested the prioritization of the certain cases, if we have limited time for testing [4]. The example of the GUI testing method is shown in Fig. 5.

The testing of GUI is not an easy task. The Memon et al suggested the use of well defined technique of artificial intelligence that is planned. The main goal of planning is that the test designer should have better knowledge about the goals of GUI user and able to specify the goals into a sequence of events. Planning is nothing but the test case generation. The authors have suggested to design the hierarchical model based GUI structure. Authors have generated a tool named planning, assisted tester for graphical user interface system (PATHS). Paths can be scaled over larger GUI’s and favors the regression testing. The high level abstraction helps in fine tuning of the test cases. The main contribution of the author is as follows [5]:

1) The unique properties of the GUI are combined with planning. Automatically generating test cases will be demonstrated practically using planning.
2) The author suggested approach is useful to reduce the model size, complexity and also the approach is improved the efficiency of test case generation.
3) Regression testing will be easier using the hierarchical planning.
4) The portability of test suites will be increased using the test case generation process.
5) The suggested approach has allowed to reuse of operator definitions that appear across GUIs

The authors have also highlighted the drawback of the approach [5].
1. Test case generation is dependent on the task given to the planner.
2. To validate the authenticity of we have to check the test case with other software.

Belli et al described the holistic view of testing that should be carried out step by step along with system modeling. Author suggested the use of coverage matrix for the selection of test cases. The work done by the authors is an extension of ‘L’ white by considering both the desired and undesired situations. The advantage of the proposed approach of the author is its simplicity. Our approach will be adopted in more recent modeling as to state chart in further research [6].

Zhang et al describes fault-state detection approach for black box testing in two phases using the Non-deterministic finite state machine.

1) The Author used 2 trees, one is transfer tree (TT) to bring the software under test to the state at which testing is to be performed. The various path on tree leads to various states.
2) The another tree referred as Detection tree (DT) is used to identify the states. The authors have extended the Deterministic finite state model (DFSM) based specifications to Non-Deterministic Finite State model (NFSM) based specifications.

We investigated the transfer-to-state phase of this approach. The transfer to state phase is represented by DFSM but it is very complex for NFSM. The NFSM approach has helped in designing a transfer tree with minimum height, minimum weight and also t-r distinguish tree with minimum weight and height for target [7]. The Fig. 6. shows the black box testing.
Hierons et al have suggested the adaptive test generation algorithm. In each stage of the algorithm produces the input sequence. In this paper an approach state counting is elaborated to adaptive state counting. The elaborated technique will reduce the size of the test suite. The additional advantage of adaptive state counting is that if the failures occurs, then testing will be terminated. The author also describes the use of adaptive test cases. A state counting actually refers to the problem of generating a checking experiment of NFSM. The approach suggested by the author is an adaptive state counting which used the inputs and outputs results used for performing the testing. The approach has an advantage that the size of the test suite is significantly reduced. Another advantage is that we can stop testing at the point where the failure has occurred and it is the important issue for Graphical User Interface testing [8]. The main contribution of the paper is as follows:

1) The adaptive test cases properties are explored.
2) Productive machine adapts to non-deterministic IUTs.
3) Finally the author proves that the test suites produced using the proposed algorithm.

Memon et al have highlighted the weakness of Smoke regression testing. DART process is used to automate GUI smoke testing. The DART process is described in Fig.7. The DART process contains the following steps:

1) AUT is identified by the test designer i.e. the location of the source file is identified by the developer. AUT is used for the DART setup called “baseline AUT”.
2) DART find out the baseline GUI structure using the GUI ripper.
3) The total smoke test cases are computed by the DART. The number of test cases is specified by the test designer.
4) The test case generator generates the smoke test cases using the DART’s automated.
5) The test case is automatically created by the test oracle generator. So the smoke test suite is ready.
6) The change request and bug report are used by the development team to modify the AUT.
7) DART is launches by the operating system task scheduler and AUT is launched by the DART. The AUT source code automatically instruments by DART using a code instrumenter.
8) The test case executor executes the test case and the output is compared with the stored expected output. The executed test cases are identified as successful or unsuccessful from the generated execution report. A coverage report is also generated which contains the details of executed statements and branches. All the details will be sent to the tester. The DART-Board is automatically developed which provides the summary of the detailed test results.
9) In this phase the tester examines the reports and also examine the unsuccessful test cases. The reason of unsuccessful test cases is:
   a) DUT crashed the software.
   b) The expected output is not matched with actual output.
   c) The event is changed in the test case.
10) Finally the tester submits the bug report.

The main weakness addressed by the author is that we can’t retest graphical user interface. The daily automated regression tester DART is used to perform the studies on different issues 1. Study of the space and time complexity of the testing suite and the amount of manual effort required. There are some test cases which are not covered by the test cases. The comprehensive test oracles may not have long test cases [9].
In this paper Paiva et al presents extensions to Spec Explorer developed by Microsoft research for the testing of API. In the proposed tool authors have extended the capability of gathering data regarding the physical objects and targeted action as a model. The authors have done the complete GUI modeling using this tool and have demonstrated the working on the Notepad application[10]. The contribution of the author is the GUI mapping tool [10].

1) To test an application GUI tool reduces the manual work through the GUI.
2) The gap is built between the high level modelling language and simulation of user events.
3) The GUI components are specified as reusable classes to promote a modelling pattern.

Cai et al addresses the problem of test case and also address that how the test case definition may affect the effectiveness of GUI testing. The problem of test cases are follows:

1) How to define the test data criteria for GUI testing. In the software testing the test data is well defined.
2) The other problem is related to the test case definition.

So in this paper author proposed a approach which defines the GUI test cases as a sequence execution and generation are done in a sequential manner. The test cases are defined using the meansly machine. It is an systematic approach which covers all path, transitions and events. Due to complete converge the technique is effective in finding the error [11].

In this Memon et al authors have given stressed on the model based testing. They highlighted that most of Graphical user interfaces (GUIs) Testing are resource intensive not model based due to the reasons

1. Models are expensive to create.
2. It has few applicability.

The author have consolidated the various models into one event flow model. The event and event interaction represents by event flow model. The all possible execution paths are represents by control flow model. The all possible definitions and uses of memory location represents by data flow model.

The all possible sequences of events which is executed on GUI represents by event flow model. A GUI is split in to hierarchy of model dialogues. The hierarchy and model dialogue is represented by integration tree and event flow graph. The model is capable of generating the test cases generation, test oracle generation and coverage calculation based by defining the event space exploration strategies. The author have successfully demonstrated the application of ESES on two applications [12].

Jin et al highlighted the importance of Finite State Machine in the software development life cycle. The authors have proposed as novel automatic software testing using the FSM methodology. In the present paper authors have discussed about the state analytical method that is used in software development. The another major thing is the event space transfer sequences which is helpful for state space analysis and is able to conduct testing on the previous profiles. The author have validate that GUI testing using FSM proposed by author is effective in finding logical error in the programs.

The operational profile developments is tested by FSM as shown in Fig. 8. So the operational profile is represented in the form of FSM model. The operational sequences are expressed by state transition diagram as shown in Fig. 8. The interactive sequence are also shown in the digram. The events are represented in more general way. The actions such as Cargo Storage/Distribution referred as the transition function [13].

Miao et al have suggested the automated testing based on an innovative model and the finite state machines. It is compared with event flow graph based model. The author has declared that there model is quite efficient in the terms of storage.[14]

Kalaji et al have suggested a powerful approach for modeling state based system extended finite state machine (EFSM) for the testing. The main drawback of using EFSM for testing is that there may exist an infeasible path as test case. The reason for the generation of infeasible is due to the guard that reference a counter variable which depends on the previous state. The authors have suggested an approach based on genetic algorithm to guide for searching the feasible path [15]. This paper contains the approach based on control and data analysiz which extend to TP. The contribution of the paper are as follows: [15]

1) The proposed method is used to bypass the counter problem.
2) The proposed approach is more effective to generate FTPs and EFSMs.

In this paper Yuan et al have suggested an approach in which the feedback from the "seed test suite" for the generation of the test suites. The seed test suite is generated using structural event interaction graph. When these test suite is executed the effect of the event on all other events is monitored and generates the Event Semantic iteration relation. This can be used to generate the new test cases. The author have verified the results on 8 application and lead to conclusion that ESI relationship help in capturing the faults than remaining counterparts [16]. The main contribution of the paper is as follows:

1) Systematic GUI test case generation and the model based is extended.
2) GUI events come among with definition of new relationship.
3) To explore a larger input space run time state is explored.
4) The certain faults require well-crafted combinations of test cases and oracles.
In this paper Yuan et al have suggested to use the combinatorial approach based on coverage criteria that can cover the most of the search space. The combinatorial technique are helpful in testing due to the reason the context can be used on various event combinations, in the length of sequence and ans different position of the events. The authors claimed to find the faults in the 8 applications under test which were undiscovered in the case of other testing approaches [17]. The limitation of the current are as follows:

1) The current approaches lack the GUI modeling methods that can capture the long event sequences using context.
2) The context aware GUI interaction testing on fault detection using systematic exploration is also lacking by the current approaches.
3) The current approaches also lacking the test adequacy criteria.

The main contribution of the author as follows:

1) The new coverage criteria has been proposed which contains event combination strength, sequence length and all possible starting and ending point of each event.
2) The stateful GUI domain allows the recast of the GUI test case generation problem.
3) The evaluation of the new cast criteria has been done.

In this case study Alsmadi et al have highlighted the work based on the Mutation testing on the GUI interface. The authors have described the seven mutation techniques for the GUI testing and presented the results with their effectiveness [18]. The mutation testing is the AI technique and having the advantage like to expose the software errors and detect the bugs. Here the author has developed the mutation technique for GUI testing. Author reverses the traditional mutation approach for GUI testing. The authors classify the expected behaviours using test case mutations in GUI testing in to three levels.

1) In the first level the muted test cases are killed. The muted test case produce invalid test cases and they would not execute successfully.
2) The validation process will be passed by some mutation and produce a valid test case. The valid test case produces the results or behaviors that differ from original test cases.
3) The some mutation is not killed because they will be proved as valid by producing results identical to the original test case.

Miao et al have suggested the use of genetic algorithm and fuzzy logic for the optimization of the GUI testing. The major challenge for this type of optimization technique is the cost or fitness function that is terms of the time once the cost has been identified the genetic algorithm is used to find the optimal sequence. The obtained results by the authors highlight the effectiveness of cost estimation and sequence generation [19].

The Fig. 9. shows the test case automation. The proposed consists of following steps show in Fig. 10. [19].

1) The software system is described in terms of procedures and associated graphical components.
2) The critical condition of each component and associated module has been identified.
3) The test cases of software component have identified.
4) The cost of execution of particular test case using criticality vector is identified.
5) The all possible path of execution is identified.
6) To identify the optical test sequence apply the genetic with fuzzy.
Aho et al have proposed the study on gaps in academic approaches for testing. These gaps are making it infeasible for the industries to adopt. The tools developed by the academicians to automate the software testing. The author have discussed the seven gaps[20].

- Scaling with accuracy.
- Coverage in reasonable time.
- Validating the correctness of extracted model.
- The tools are not Generalized.
- The introduction and adoption effort.
- Reducing the manual effort.
- Reducing the maintenance after changes.

The author analyzes the problem domain with the help of wider perspective. The author is trying to identify the challenges and provide the possible solution.

![Fuzzy System](image)

**Figure 10: Fuzzy System [19]**

Fujiwara et al developed the test selection method based on finite state models. The name of developed method is “partial W method”. Many test selection methods have been developed by many authors. The some best known methods are: [21]

1) Transition Tour
2) W-Method
3) Distinguishing Sequence Method
4) Unique Input-Output Method

All the four methods detect the output error while implementation using test suits. All the method detects the output error during execution time. The “partial W method” is finding the all errors provided. The number of states remains in a certain bound.

The purpose of the paper is as follows:
1) The proposed method is the binding element which allows the comparison of various test selection methods.
2) Author also provide the discussion of other issues which are:
   - Usability
   - Effectiveness
   - Fault coverage of test suites.

Hermadi et al proposed the genetic algorithm based approach. The proposed approach is generating a set of test data which covers the set of target paths. The proposed approach achieved the path coverage and improves the performance of genetic algorithm in terms of [22]:

- Search space exploitation
- Exploration
- Allows faster convergence

Hermadi et al exposes the challenges of path testing and analyze the parameters which are affecting the performance of genetic algorithm. The author analyzed the path testing based on complexity and automation. Experimental results show the running GA based path testing. The parameter used in the paper is [23]:

- Population size
- Number of generations
- Allele range
- Mutation rate

Qureshi et al surveys the GUI test case generation techniques. The GUI test case generation techniques are classified on the basis of fault models. The evaluation criteria of all the surveyed techniques are based on following parameters [24]:

1) Input representation of GUI under test
2) Intermediate representation
3) Coverage criteria
4) Automation
5) Tool support
6) Case study
7) Fault model
8) Fault Injection
Esabella et al. have analyzed the test case generation using GUI testing. The testing are of following types [25]:

- GUI testing
- Logical testing
- Integration testing

Author focuses on the GUI testing as it is very popular. It interacts with software in a more sophisticated way. Testing should be done in the way that it provides these features [25]:

- Effectiveness
- Efficiency
- Increased fault tolerance rate
- Good path coverage

This paper also covers the techniques used for test case generation and GUI based software applications. The Fig. 11. Shows the test case generation using covering arrays. White et al. suggested the use of complete CIS to test in constrained time in the year of 2000. In this paper the same author extends the previous work by investigating the use of memory tools. The memory tools used to detect missing effects and CIS sequences [26]. The contribution of the author is more in mainly three areas.

1) Memory tools are used to detect undesirable GUI effects.
2) The testing of sequences with CIS sequences is used for testing required. The testing can be analytically or empirically.
3) Empirical study is also considered the main work in this paper. It contains the five GUI systems which are follows [26]:
   - RealNetwork Suite
   - Adobe Suite
   - Inter WinDVD
   - Multimedia Database System
   - Java Firewall Application

Reza et al. proposed a model based testing method. The proposed approach is used to test the structural representation of GUIs. The GUIs also known as Hierarchical Predicate Transaction Nets (HPtTNs). To detect the design faults and the benefits from HPtTNs, original criteria proposed for HPtTNs. Model Based Software Testing (MBST) works with HPtTNs. The working process having following points [27]:

1) GUI model is constructed using HPtTNs.
2) The path and transition execution sequences have been created from HPtTNs. The reachability graph is used to cover HPtTNs.
3) The expected output will be generated.
4) The execution of adequacy of all transitions, all states and all threads play token game is used.
5) The actual and expected output has been evaluated. The expected outputs contains the accepted and un-accepted behaviors.
6) The proper action has been taken based on the behavior of the model. The behavior model can be [27]:
   - To accept
   - To stop
   - To modify a model

The proposed coverage criteria using HPtTNs using three types of simple testing:

1) All Transitions
2) All States
3) All Threads

All transition testing has been extended with additional criteria. The additional criteria are following [27]:

- The single event and single outcome criteria : In this criteria single event and outcome should be performed at least once.
- Many events and single outcome criteria: In this many events leading to single outcome should be performed once.
• Event-to-event criteria: In this criteria every event performs a particular task.
• Event-to-component criteria: In this criteria each event leads to a component to be performed at least once.
• Component-to-event criteria: In this criteria each component leads to an event to be performed at least once.
• Component-to Component criteria: In this criteria each component leads to another component to be performed at least once.

Paiva et al proposed a new approach in order to model and test Hierarchical Graphical User Interfaces (GUIs). The model in the sense that reduces the structure of Hierarchical Finite State Machines (HFSMs) to the number of states Finite State Machines (FSMs). First, identify the independent dialogs and then HFSM is build from FSM. The dialogs are described by the by FSM is reduced. The model of notepad application is considered to elaborate the approach. Notepad application is the part of Microsoft Windows. The model is written in the form of Spec#. The model is converted into FSM using Spec Explorer. Spec Explorer Tool is developed at Microsoft Research. The total number of states of the FSM is defined by HFSM. The number of states is reduced. The test cases are generated from the FSM by Spec Explorer [28].

Darvish et al presents an automated blackbox framework. GUI invariant consists state based event constraints that are rivaled by blackbox framework. The core algorithm of blackbox framework is GUIDiVa. This is an iterative algorithm and useful for all possible constraint violation and test case failure. A human oracle is used to validate the accuracy of the searched constraints [30]. The author also proposed the GUI Invariant Discovery and Validation Framework. These frameworks detect the all the types of state-based event constraints. In GUI application state-based event constraints occurs frequently. The novelty of the paper is to use the iterative algorithm that is GUIDiVa. The assumption of this algorithm is

• A combinatorially coverage adequate test case. The constraints on failed events will be identified by GUIDiVa. The failed events will be found by the algorithm any time in order to generate new test cases. The total number of feasible test cases under the test suite that contain the failed event and also violate the same constraint. The test suite contains the infeasible test cases with event combinations. The infeasible test cases violates the final discovered constraint. In the next iteration of the algorithm a new test suite is used. If there are no infeasible/new test cases left then GUIDiVa stop.

Figure 12: Action-Event Framework [29]
Memon et al. focus on coverage criteria of GUIs. The important role of GUIs provides the test quality. The new coverage criteria is presented to determine the GUI is adequately tested. To specify, measure of test adequacy coverage criteria used the event and event sequences. The total event permutations are large in any non-trivial GUI. To identify the important event sequences the GUI’s hierarchical structure is exploited to be tested. The GUI can be further divide into GUI components. The GUI components used to basic unit of testing. The GUI components can be represented by event flow graph. To evaluate the adequacy of tests on these events, intra-component-criteria are used. The event flow graph and an integration tree for a given GUI are given to the algorithms. The important contribution of the paper is as follows [31]:

1) Author provide the definition of GUI component. The structural GUI testing is used. The GUI is decomposed into a hierarchy of interacting components.
2) GUI component is represented by an event flow graph. The event flow graph captures the flow of events within a component and this representation is called integration tree. The integration tree is used to identify the interactions among components.
3) The intra-component and inter-component GUI testing have a class of criteria and also have a technique in order to compute the coverage of a given test suite.
4) The usefulness of event based coverage and a correlation between coverage a case study is given in terms of events and code.

Chander et al. focus on the problem generating a set of test input sequences. The coverage criteria are identified by this problem. The coverage criteria are used to cover all transition levels or cover all length-n transition level. The author has shown in the paper that optimal test input generation problems reduced to the integer linear programming (ILP) problems. The author also proves that the optimal test input generation problem is NP-Complete. The prototype implementation is used for finite state machines [32].

Xie has proposed the state-of-the-art in GUI testing. He studied the GUI faults and interaction between GUI events. The result of this cost effective study is to develop the cost effective model based GUI testing techniques. The novelty of this research is to reduce a model of GUI’s event integration space. The model is derived from GUI. The model is used to generate GUI test cases that effects to detect the GUI faults. The model is also extended to develop [33]:

- The new test oracles
- The new coverage criteria for GUIs
- The new regression testing techniques.

The proposed framework has shown in Fig. 13. And have the following characteristics.

- It is automatic, so the tester work will be simple. Using automatic framework GUI model is obtained automatically.
- The practitioners use the framework even in the tight deadlines as it is efficient.
- The framework is extensible because of this quality new techniques can be implemented. The new framework will be packaged due to the new techniques.
- This framework is enough and applicable to a wide range of GUIs.

Hooda et al. gives a review of study of test case generation techniques. The paper consists the review on [34]:

- The various test case generation method.
- Test case minimization
- Selection technique
- Prioritization technique
- Evaluation technique

![Figure 13: Overview of the framework [33]](image1)

![Figure 14: Test Case Life Cycle [34]](image2)
The test case life cycle has shown in Fig. 14. The software test case generation has the following steps:

- In the first step all possible constraints have been found out from start to finish node. The constraints are nothing but the pair of algebraic expressions. The constraints are also dictates the condition of variables between start and end nodes.
- The variable having maximum value is assigned by highest value to reduce the test cases. The lowest value is assigned to the variable having minimum value within its given range.
- The variable is given to the user is assigned by the constant value in the defined path.
- Finally the tables created that includes the all possible test cases.

Table 1: Summary Of The Various Articles

<table>
<thead>
<tr>
<th>Author</th>
<th>Year</th>
<th>Proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fujiwara et al.</td>
<td>1991</td>
<td>Partial W Method</td>
</tr>
<tr>
<td>Apfelbaum et al.</td>
<td>1997</td>
<td>A goal based testing</td>
</tr>
<tr>
<td>Shehady et al.</td>
<td>1997</td>
<td>Models based testing (VFSM)</td>
</tr>
<tr>
<td>Dalal et al.</td>
<td>1999</td>
<td>Design test case based on requirements</td>
</tr>
<tr>
<td>White et al.</td>
<td>2000</td>
<td>Suggested the Use of complete CIS to test in constrained times</td>
</tr>
<tr>
<td>Memon et al.</td>
<td>2001</td>
<td>Coverage criteria of GUIs</td>
</tr>
<tr>
<td>Memon et al.</td>
<td>2001</td>
<td>Presented a tool planner</td>
</tr>
<tr>
<td>Belli et al.</td>
<td>2001</td>
<td>Holistic view over selection test cases based on coverage metric</td>
</tr>
<tr>
<td>White et al.</td>
<td>2001</td>
<td>Suggested the use of complete CIS to test in a constrained time in the year of 2000. The extend the previous work by investigating the use of memory tools.</td>
</tr>
<tr>
<td>Zhang et al.</td>
<td>2003</td>
<td>Used optimal trees and NFSM derived from FSM</td>
</tr>
<tr>
<td>Hermadi et al</td>
<td>2003</td>
<td>Genetic Algorithm based Approach</td>
</tr>
<tr>
<td>Herons et al.</td>
<td>2004</td>
<td>Suggested adaptive state counting problem</td>
</tr>
<tr>
<td>Memon et al.</td>
<td>2005</td>
<td>Highlighted the week ness of smoke testing</td>
</tr>
<tr>
<td>Paiva et al.</td>
<td>2005</td>
<td>Extended the tool Spec Explorer</td>
</tr>
<tr>
<td>Cai et al.</td>
<td>2005</td>
<td>Suggested testing in sequential manner</td>
</tr>
<tr>
<td>Paiva et al.</td>
<td>2005</td>
<td>The new approach in order to model and test Hierarchical Graphical User Interfaces (GUIs)</td>
</tr>
<tr>
<td>Xie</td>
<td>2006</td>
<td>The state-of-the-art in GUI testing</td>
</tr>
<tr>
<td>Memon et al.</td>
<td>2007</td>
<td>Coverage calculation based by defining the event space exploration strategies</td>
</tr>
<tr>
<td>Reza et al.</td>
<td>2007</td>
<td>The model based testing methods</td>
</tr>
<tr>
<td>Jin et al.</td>
<td>2009</td>
<td>Used the event space transfer sequences</td>
</tr>
<tr>
<td>Miao et al.</td>
<td>2010</td>
<td>Model based on FSM with efficient in storage</td>
</tr>
<tr>
<td>Kalaji et al.</td>
<td>2010</td>
<td>Extended finite state machine (EFSM) with genetic Algorithm</td>
</tr>
<tr>
<td>Hermadi et al.</td>
<td>2010</td>
<td>Path testing affecting the performance of</td>
</tr>
<tr>
<td>Yuan et al.</td>
<td>2010</td>
<td>Feedback based test case generation using seed test suite</td>
</tr>
<tr>
<td>Hoai et al.</td>
<td>2010</td>
<td>Action Event Framework (AEF). MGT framework is introduced</td>
</tr>
<tr>
<td>Yuan et al.</td>
<td>2011</td>
<td>Suggested the combitorial approach based on coverage criteria</td>
</tr>
<tr>
<td>Chander et al.</td>
<td>2011</td>
<td>Focus on the problem generating a set of test input sequences</td>
</tr>
<tr>
<td>Esabella et al.</td>
<td>2012</td>
<td>Test case generation using GUI testing</td>
</tr>
<tr>
<td>Qureshi et al.</td>
<td>2013</td>
<td>Survey the GUI test case generation techniques</td>
</tr>
<tr>
<td>Alsmadi et al.</td>
<td>2013</td>
<td>Presented a case study on various mutation testing in GUI</td>
</tr>
<tr>
<td>Hooda et al.</td>
<td>2014</td>
<td>Gives a review of study of test case generation techniques</td>
</tr>
<tr>
<td>Chhillar et al.</td>
<td>2014</td>
<td>Fuzzy logic and GA based optimization of GUI testing</td>
</tr>
<tr>
<td>Darvish et al.</td>
<td>2014</td>
<td>Automated blackbox framework</td>
</tr>
<tr>
<td>Aho et al.</td>
<td>2015</td>
<td>Suggested a gaps and their solutions between academic tool and industrial work</td>
</tr>
</tbody>
</table>

Conclusion

After the detail analysis of various approaches that have been used in various articles. We have come across the various ideas that have shown the use of mathematical models to the AI techniques. The thing described as research gaps may require the many modern tools like genetic algorithm and the no approach has been highlighted which have used the self learning approach except in [16] which discussed about learning from past experiences so there is scope for writing algorithm which can have neural network that can learn and modify the test cases accordingly. In all we can say as still we can find the way to implement the things so still the testing using FSM will be required, but they are needed to fast and easy adaptable to the new problems and acceptable to the industries. We thus suggest to design a tool based on FSM which fast, self learning, and generalized.

References


