Survey on Graphical User Interface and Machine Learning Based Testing Techniques

1B. Uma Maheswari and 2S. Valli
1Department of Master of Computer Application, St. Joseph’s College of Engineering, Old Mamallapuram Road, Chennai, 600119, Tamilnadu, India
2Department of Computer Science and Engineering, College of Engineering Guindy, Anna University, Sardar Patel Road, Guindy, Chennai, 600 025, Tamilnadu, India

Corresponding Author: B. Uma Maheswari, Department of Master of Computer Application, St. Joseph’s College of Engineering, Old Mamallapuram Road, Chennai, 600119, Tamilnadu, India

ABSTRACT
The Graphical User Interface (GUI) is used in building interactive and web based applications. Several components are available for building the front end of the software. The interaction among components is accomplished through their corresponding events. This survey explains the testing concepts in detail by addressing the various testing techniques, test model structure and detected fault category. Various testing techniques such as data flow testing, object oriented testing, model based testing, web applications testing, user interaction testing, user interface testing, machine learning based testing, state based testing, test suite reduction and specification based testing are discussed in this survey. The survey also presents the parameters used for evaluation. Machine learning algorithms, features and the data set used for classification in the this study are analyzed in survey. Considering of role of direct and indirect metrics in software testing is also addressed in this survey. The testing tools and frameworks used for testing the Graphical User Interface (GUI) applications and their issues are also handled.

Key words: Web applications, graphical user interface, machine learning, defect prediction, model based testing

INTRODUCTION
Developer’s challenge is to ensure that the intended functionality of the GUI applications is achieved. Automatic failure identification was done by Travison and Stanef (2008) using the test instrumentation and pattern matching. Numerous model based testing (Neto et al., 2007) approaches are used in developing various software projects. The quality of the testing approach depends on project characteristics (Dias Neto and Travassos, 2008) such as the structural model, inputs to the test model, complexity level, support tools, test coverage criteria and the level of automation. These project characteristics are identified and characterized by the machine learning based testing through qualitative and quantitative analysis. The logical and runtime errors that are present in the software are detected using the translators, whereas, the runtime errors generally occur during the execution of the code. There are numerous existing researches for finding the issues in the software. The association and categorization among the faults and the failures was analysed by Hamill and Goseva-Popstojanova (2009). The deviation from the behaviour and the violation with the functional constraints were identified. Requirement faults,
coding faults and data faults cause the system to fail and hence the software practitioner detect the faults and the failures are uncovered by clients after the software is delivered. Fault detection in the GUI applications is the act of locating bugs in the components or events of a program.

**GRAPHICAL USER INTERFACE TESTING TECHNIQUES**

Testing involves input and output verification. But GUI testing should also test data and events. In order to improve the programmer productivity, testing and quality assurance activities should be carried out in the software life cycle. GUI testing is not an individual activity, it should be conducted from different perspectives which include test coverage, test case generation, test oracle and regression testing. There are different testing techniques for GUI applications in view of its various aspects and features. The GUI is the front end design of any software system.

Therefore, it is essential to verify its components and their interactions. All the GUI applications have interactive components. The input actions, entering text, mouse clicks, selecting a graphical object, selecting the menu items and closing a window are the triggering events. Test scripts should be written to verify the user interactions in GUI testing. Since, there are various approaches for GUI testing, each has its own advantages and disadvantages. For this reason, it is essential to evaluate such techniques to know their effectiveness. Most of the web applications are built with GUI components. So, it is essential to find the correct testing technique to verify them. A collection of testing techniques, to verify the logic and the functionalities of the graphical applications, are discussed in this survey.

The graphical user interface testing techniques are illustrated in Fig. 1. The survey of various white box testing techniques, such as data flow testing, object oriented testing, model based testing, user interface testing and machine learning based testing are conferred in the described. Model

![Diagram](image)
based web application testing and user interaction testing are discussed. Machine learning based testing using Support Vector Machines (SVM), J48 Decision Tree and Genetic Algorithm are also elucidated in this survey. Bearing out the software functionality using the state based testing, test suite reduction and specification based testing of black box techniques are also elucidated.

DATA FLOW TESTING OF GRAPHICAL USER INTERFACE APPLICATIONS

Structural data flow testing, as shown in Fig. 2, was performed by Liu (2006) for testing the Java Server Pages (JSP) in the web applications. The structural data flow testing was performed between the functions, the group of functions, objects and the applications. The computational use of the variables was walked around by the control flow graph. The logical errors in the JSP web pages were identified by the data flow paths. They navigated the Session Control Flow Graph (SCFG) to detect the errors.

Various techniques (Di Lucca and Fasolino, 2006) were examined for testing the functional and non-functional metrics. Unit testing was performed to verify the individual components of the web design. The flow of data in web applications is illustrated in Fig. 3. The data flow between the individual functions is captured by the Control Flow Graph (CFG).

The information passed between the functions was confined by inter-procedural graph. The object functions which are triggered by GUI events were integrated using the object control graphs. The data flow information passed in the web pages were identified by composite flow graphs. An object model (Zeng and Miao, 2007) describes the navigation behaviour and object structure of the web applications. Node coverage and edge coverage were exercised by trapping the properties of the object model.
Kervinen et al. (2006) developed a Labelled Transition System (LTS), which examines the GUI components in mobile device functioning with the Symbian operating system. Various performance problems in the network like web server, database server and application server, while transferring the content from the server to the clients, were tested. The non-functional requirements (Iyer et al., 2005) such as performance, scalability and reliability issues in multi-tier web applications were also identified.

OBJECT ORIENTATION TESTING

In the object orientation testing, the Flex Rules was developed (Sarala and Valli, 2004, 2006a, b) for finding the missing new operator. The algorithm checks the exact match of the data type in actual and formal parameters. The testing is accomplished for detecting defects in C# console based applications. Errors due to unintended characters and the missing argument indicator, leading to execution errors were detected. Wrong usages of formal parameters leading to logical errors are addressed. The authors have identified defects in hybrid inheritance, runtime polymorphism, member functions, conditional statement and function overriding. They also handled the dangling reference problem and identified the missing of address and new operators in C++ applications. Data exchange in business applications can be performed using the Extensible Markup Language which provides a convenient format.

The same data is converted in XML and relational database. Verifying the accuracy between these objects is done by Raha and Jadhav (2008) through object mapping. The discussed works detect defects which lead to logical and execution errors in object oriented languages. This encourages identifying the defects leading to logical errors in GUI applications. Geetha et al. (2008) performed static testing for finding the inheritance related bugs and property errors in the product. The code analysis extracts the code for testing and the data flow testing was based on the bug analysis of the program.

MODEL BASED TESTING (MBT) TECHNIQUES

The actions of the GUI applications were represented by Chen et al. (2008) as a tree and each sub tree denotes one or more test cases. A group of strongly related components are macro components which has high level operations. A GUI Testing Tool (GTT) was developed using GUI Testing Modelling Language (GTML) with an Extended Backus Naur Form (EBNF) grammar for testing JAVA swing based applications. They tested the swing, classes, visual macro model editor and word processor components. The test cases were generated and executed by the GUI testing framework (GUITAR). Mariani et al. (2012) developed a AutoBlackTest tool to generate test cases automatically by navigating the GUI for software verification process.

The "Q-Learning Agent act" together with software and the automatic black box testing performed much better than the GUITAR. It performed code coverage for the complex actions, such as File Chooser, Color Chooser, Fill Form, HandleList and Compound statements. It verifies the GUI frameworks of NET applications and also reveals the faults in widgets, such as Label, ToolTip button, Taggle Button, Checkbox, Radio Button etc. of JAVA applications. The AutoBlackTest detects failures such as crashes, hangs and uncaught exceptions in the JAVA programs. If the target application is not available, the system crashes and if the target application does not respond to actions, it hangs.

Identifying the undetected defects, which lead to logical errors, has been attempted by Maheswari and Valli (2011). Structural testing is performed to identify defects which produce
unexpected results. All the GUI controls have several properties assigned to them. These assignments of the properties can be done during the design time or compile time of the software. Even though, several properties and actions are assigned for the GUI controls, there are some errors that exist. The unrestricted input control text box was used for testing. Valid and invalid values were set for the properties of the control and the behavior of the text box control was examined. The tokenizer in the Visual Basic Control Testing (VBCT) analyses the GUI code, extracts the tokens of the text box control and the property assignment checker scans the tokens for submitting them to the error handler. Several algorithms were written to detect the defects leading to logical errors in text box control.

These algorithms in the VBCT verify the properties of the text box control which are not detected by the interpreter. The algorithms identify the invalid assignments of the source code of the GUI applications. The algorithm detects the issues and provides suggestions to modify the source code and properties. The GUI control properties, such as Appearance, Border Style, Dragmode, Forecolor, Index, Mouse Icon, Mouse Pointer and Tabindex were also analyzed. The performance analysis of the VBCT algorithmic tool and the Visual Basic compiler were compared for detecting the defects leading to logical errors.

The DART (Daily Automated Regression Tester) framework was developed by Memon et al. (2003) for testing GUI applications. It uses GUI ripping for opening all the windows and extracts all the widgets and their properties. The MS WordPad software was used for evaluating the DART framework. The “File” Menu is considered as one of the widgets for evaluation. Code coverage for the properties that is open, save and print was accomplished, by generating test cases of different lengths. With the use of event flow graphs and the integration tree, regression testing has been attempted by verifying all the objects and properties of all the windows in the application.

The survey on models (Kumar and Yogi, 2012) explains the Performance Testing Analysis (PTA). Testing the software with capture and playback tool was analysed. Xie and Memon (2007) designed a system with six instances of the test oracle to ensure the software execution by comparing the actual output with the expected output of the Application Under Test (AUT). A test oracle was used in checking the correctness of the database operations. GUI testing was attempted by McMaster and Memon (2008). The active calls of the space application were examined.

The WEB application fault detection Visualization with ORacles (WEBVIZOR) is an open source tool. Sprengle et al. (2008) enables visualization and is used in the performing comparative analysis of the test results of web applications. It visualizes the actual and expected results of the program and also the output from the oracle comparators for fault detection. Streckert and Memon (2008) accomplished the relationship between the errors and test cases with event flow testing. The coverage function detects the faults in the mutation and branch statements and also verifies the mutant type statements in the byte code. The GUI-event coverage identifies the faults in the components of the web applications.

TECHNIQUES FOR TESTING WEB APPLICATIONS

Wide ranges of web applications are used by governments, businesses and consumers. Changes in the business rules lead to software modification, so it should be verified to avoid loss in software development. Web applications handle HTTP requests, generate dynamic content for the user and interact with the other components. Most of the JSP pages are not checked by the compiler hence, it is essential to verify the data flow information between the web pages. Bugs in web applications
Fig. 4: Web application models

were identified by Artzi et al. (2010). They designed the Apollo architecture, an algorithm that generates test cases for evaluating dynamic web applications. The architecture checks whether the web application issues and the output of the application.

Invariant based testing was performed by Mesbah et al. (2012). They proposed a method for testing Asynchronous JAVASCRIPT and XML (AJAX) applications. AJAX specific faults such as the Document Object Model (DOM) validity, discoverability and back button compatibility were identified. Ceri et al. (2000) proposed four types of models for representing the web applications, as shown in Fig. 4.

A multi agent system MAEST, for software testing, had been attempted for providing assistance to testers in the testing process. Some of the agents, such as the administrator, testing, interface and helping agents are used to organize and design the test cases in MAEST. They discussed the use of ontology for software testing (Eisenbarth et al., 2003) distinguished the specific and computational part of the source code, where the features occur. They used the concept analysis to investigate the binary relations between the features and the computational units, using the execution profiles of different scenarios. The cyclomatic complexity (Deng et al., 2004) computed from the generated application graph. The relational database content is tested using the tool AGENDA.

COMBINATORIAL INTERACTION TESTING FOR GUI

Klaib et al. (2010) reduced the number of test cases by the tree generation strategy. The maximum cost was computed for the test cases for the inclusion of testsuite. Interaction testing was performed by Yuan et al. (2011). Array based test cases were generated, by examining the sequence of events of GUI widgets and their corresponding states in the event interaction graphs. They found event faults in the edit window. The interaction among the events, cut, copy and paste are evaluated by generating all possible permutations of the events.

The test suite size is reduced using the event flow graph. Undetected faults are revealed with the help of the starting and ending positions of the events. The size of the test suite is the efficiency and the fault detection ability is the effectiveness. GUI and Web applications states change, depending on the events. Bryce et al. (2011) developed a design for graphical and web applications. Combinatorial interaction testing was performed, by verifying the event sequences of the “Find” GUI window and “Online Bookstore” web application. Faults were seeded by modifying the relational, arithmetic and logical operators in the JAVA programs.
USER INTERFACE TESTING TECHNIQUES

Test suites (Ames and Jie, 2004) from the capture/replay testing tool creates the call graph for regression testing. The test cases were written in the XML format and the redundant test cases were identified with the weights assigned in the call graph. The correctness of the states was verified in the critical paths. User interface testing by Maheswari and Valli (2013) detects the defects leading to logical errors in the various GUI widgets. The defects in the Listbox, Combobox, Checkbox, Option Button, Command Button, Label and Drive List Box were detected. The appearance faults in the picture box and the data store faults in data control were also identified.

When the code fragment (Table 1) is interpreted, it is free of compilation errors. On execution, the background color for the listbox, combobox, checkbox and option box is black, which is incorrect even though the white color is assigned in the property window. The tooltip message and the caption text are not associated with the controls, as they are not enclosed in double quotes.

The data items are not inserted in the list and combo boxes. The picture is not aligned and the navigation of the records is not possible. All the above violations, which are not detected by the GUI compiler, are identified by the Graphical User Interface Widget Testing (GUIWT) tool. From the experimental results, it is evident that the designed test scripts detect the faults in a robust manner.

Test suite reduction techniques: Technology improvement in reusability and development in multiple languages requires regression testing. Regression testing ensures that no new errors are introduced after the modification of the software. During this testing, redundant and obsolete tests were identified and they were deleted from the original test suite. Figure 5 provides some of the test cases removal techniques.

TEST SUITE REDUCTION USING CALL STACK COVERAGE

The active calls were collected for the executing application (McMaster and Memon, 2008), when the methods called they were inserted in the stack and retrieved when they return the value. It was tested for the space program. In the test suite reduction, techniques by Kumar and Yogi (2012), the test set Tᵢ test the application Aᵢ. The Tᵢ which is a subset of Tᵢ test modified application Aᵢ⁺, using the control flow graphs. Control flow graphs were constructed using Java Architecture for the Byte code Analysis (JABA). Depth First Traversal is used in identifying the dangerous edges.

TEST SUITE REDUCTION USING USER SESSIONS

It is essential to detect the obsolete tests from the original set. Sampath et al. (2007, 2008), Sampath and Bryce (2012) performed regression testing using the concept analysis for web

Fig. 5: Test suite removal techniques
programs. The URL requests were given to the lattices for reducing the test suite for various web sites. Most frequently used web sites are examined for failures. The various faults which were inserted in the original program were listed in Table 2.

The original and fault inserted programs were evaluated with the oracle comparators. The structure, content and the output were compared with “diff and oracle” comparators. The reduction of test suites for the web applications was done by Sampath et al. (2007). The fault detection was increased by prioritizing the test cases in the test suite. The fault detection was computed using Eq. 1 where, T is test cases “n” and F is faults “m”:

$$\text{APFD} = 1 - \frac{T_{F1} + T_{F2} + T_{F3} + \ldots + T_{Fn}}{\sum_{i=1}^{m} \frac{1}{2n}}$$  \hspace{1cm} (1)

During the test suite reduction, the fault detection ability was upgraded by selectively keeping the test cases.

TEST SUITE REDUCTION USING CONDITION BASED TESTING

All the conditional and decision statements (Jones and Harrold, 2003) were collected for the test suite. The uncovered modified decision statements were removed from the test suite which waste the weakest test cases. The highest entity coverage was obtained by prioritizing test cases. The program dependence graph detects the modified functions and the statements where the relational and arithmetic operators were interchanged.
Automated session data repair removes the session data which are potentially obsolete (Harman and Alshahwan, 2008). The Reweb and Testweb tools were used for creating the UML graph for test suite reduction. The changes in the structure of the pages and the parameters of the forms and files were analysed.

SPECIFICATION BASED TESTING

The reliability and scalability of the applications was increased. Hence, testing the specification of the applications with respect to SRS was mandatory. The testing technique (Sun and Jones, 2004) uses the Jemmy Test Engine to generate the GUI events and to capture the event responses. The Jemmy Test Engine tests the GUI and the verification performed on these widgets as shown in Table 3.

The GUI based test specifications verified the handlings of the GUI controls and their responses. The N-Version condition based testing was conducted in an identical manner, since each version satisfied the same specification. Testing with customized test requirements was done by Sampath et al. (2007). The data flow requirements were used for designing the test case. The cost effectiveness of the requirements was defined by the figure of merit (fom) and it is calculated using Eq. 2:

\[
\text{fom} = \text{redux} \times \text{cvg} \times \text{fd}
\]

where, redux is reduction of the test suite (%), cvg is coverage (%) and fd is fault detection (%).

If the fom is higher, the reduced test suite is cost effective. By using this technique, redundant test cases are identified for elimination.

CASE STUDIES FOR GUI TESTING

Researchers have conducted numerous case studies for proving their testing techniques and methodologies. Some of the testing techniques and the experimental applications are listed in Table 4.

MACHINE LEARNING BASED TESTING TECHNIQUES

GUI testing is very expensive and takes more time. So, automating the testing process provides good solutions for complex and bigger applications. Most of the defects were found only in less number of modules in the software (Fenton and Ohlsson, 2000). These modules with the defects lead to software failures. Defect prediction is a quality assurance activity, which assists the software developer to allocate effort and resources efficiently (Koru and Liu, 2005). Recently, machine learning techniques were used to automate the software testing process. A framework has been designed by Noorian et al. (2011) to perform software testing, using machine learning. Briand et al. (2008) applied machine learning technology for evaluating the test suites. The test
suite and test specifications were given as inputs using the Category-Partition (CP) strategy. These test suites were evaluated using the C4.5 machine learning algorithm. The weaknesses and redundancies of test specifications and test suites were overcome by adding or deleting the test cases from the test suites.

Mair et al. (2000) used neural networks reasoning and induction techniques to build a effort prediction model. They compared the prediction system using accuracy, explanatory value and configurability. The investigation proved that ANN methods are better than rule induction. Briand et al. (2007) in another study have identified the failure conditions of the statements, using the failing test cases of the program. Statements with faults were considered as suspicious and caused the system to fail. The Space program written in “C” language was tested.

The Tarantula and RUBAR algorithms ranked the faulty statements. The Category-Partition black box testing technique identified the invalid, empty, non-existent files and the wrong number of parameters, invalid inputs and missing values, using C4.5 decision trees. Supervised binary classification is used for fault identification using the factorization algorithm (Zhang, 2011). It extracts the external features of the software. Product and process metrics, such as McCabe complexity, Basic Halstead, Derived Halstead and Line Count, were the features used for training the model. They proved that the NMF algorithm performs the classification with a high F-Measure.

MACHINE LEARNING BASED TESTING USING SUPPORT VECTOR MACHINES

Since software testing is a quality assurance activity, to allocate the effort and resources efficiently, Elish and Elish (2008) have recommended a mechanism using support vector machines to detect the error. Various machine learning and statistical models were used to evaluate the SVM performance for identifying the defects. The defect prone modules were predicted using module level features such as cyclomatic complexity, essential complexity, design complexity, effort estimate and Halstead measure. The prediction was used in evaluating and comparing the prediction models. They proved that the defect evaluating performance of the SVM is superior to other machine learning models. Boetticher (2003) developed a predictive effort estimation model using the neural networks machine learner in formulating the estimation model. Inputs were derived from the GUI interface specification documents.
The program unit with different types of widgets, such as labels, edits, boxes, check boxes, radio buttons, list boxes, memo boxes, file list box buttons, charts, combo boxes, grids, menus, navigational bars, and trees is the input for the neural network machine learner. The output measure was the actual effort spent for developing the program unit. The four major subsystems of E-Commerce organization were the data sets used in the experiments. The program events were used to detect faults in the applications (Gohe and Faytong, 2011, 2012). If one or more of the events in the event sequence are disabled or inaccessible, the test case is infeasible to detect the faults. The researchers identified such infeasible test cases using the SVM and Grammar induction.

The SVM and MartiRank algorithms, designed by Murphy et al. (2007), used for software testing, where test cases are created by analyzing the problem domain, the corresponding data sets, the algorithm and the implementation's runtime options. These algorithms do not address the negative class labels. In this study, the class label is a Boolean variable (Defect/No-Defect). A hot method prediction model for compiler optimization has been developed by Johnson and Valli (2008, 2011) using the Support Vector Machine. Programs written in "C" language were taken for training the prediction model and various static features were used in identifying the program method as hot or cold. The SPEC and UTDSP benchmark suites were used in validating the prediction system.

MACHINE LEARNING BASED TESTING USING THE J48 DECISION TREE

Automating the testing process in GUI applications improves the quality of the software. Machine learning-based testing identifies the defects leading to logical errors. The user interface widgets such as textbox, list box, combobox, checkbox, option button, command button and label controls were considered. The features were extracted using the Graphical User Interface Widget Testing (GUIWT) tool. This training data set builds the defect prediction model. The testing instances without class labels were the input to the defect prediction model which classifies the GUI statements into the "Defect/No-Defect" categories. The defect prediction model was evaluated using the confusion matrix. The confusion matrix comprises of:

- TP—Correctly classified as defects
- FN—Misclassification of defects as no-defects
- TN—Correctly classified as no-defects
- FP—Misclassification of no-defects as defects

The prediction performance measures were calculated using following equations:

\[
\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}
\]

\[
\text{Precision} = \frac{TP}{TP + FP}
\]

\[
\text{Recall} = \frac{TP}{TP + FN}
\]
By using this automation process, more than 90% defect classification rate has been achieved.

MACHINE LEARNING BASED TESTING USING GENETIC ALGORITHMS

New solutions can be searched by improving the testability of the applications. A search based testing done by Gross et al. (2012) for the highest code coverage was conducted by applying the genetic algorithm over the test suites. The genetic operators were used to improve the fitness value of the individuals in populating the candidate solutions. In this approach, the set of GUI interaction sequences is the search and the crossover creates the offspring. The interaction sequences were added, removed or changed with the help of the mutation process. The branch coverage of the program evaluates the fitness of the test suite. Alshraideh (2008) performed the unit testing of JAVA script programs. An automated test data generation tool was used in performing unit testing in JAVA script functions. The conclusions were, that 50% developmental cost is applied for testing and 15% time is scheduled for regression testing. The testing automation technique helps the testers to detect the bugs in the JAVA script programs. The tester annotates the file to be tested. The JAVA script code is parsed, to generate the dependency graph for the function under test.

Regression test suite removal was analyzed by La et al. (2007) using genetic Algorithms. The test cases positions were exchanged for ordering of the test suite. Test cases were selected with Baker’s linear ranking algorithm. Huang et al. (2010) designed the framework for performing the black box testing of the JAVA editor window and tested the File Menu properties. The functionality of the software was verified, using the genetic algorithm and combinatorial interaction testing was performed to test the event sequences of the GUI. The metamorphic relation proposed by Xie et al. (2011) between the input and output of the applications in bioinformatics and computational linguistic domain was analyzed. Metamorphic testing was conducted to validate the machine learning classifiers. There is proof that cross validation is not sufficiently effective to detect faults in a supervised classification program. Some of the features of the machine learning algorithms and the data set used for classification in machine learning based testing are mentioned in Table 5.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Features</th>
<th>Machine learning algorithm used for classification</th>
<th>Data set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Briand et al. 2008</td>
<td>Category partitions</td>
<td>C4.5 decision tree</td>
<td>Space program</td>
</tr>
<tr>
<td>Elish and Blish 2008</td>
<td>Software metrics (Mecabe complexity, basic halstead, derived halstead, line count and branch count)</td>
<td>Vector machines (SVM)</td>
<td>NASA (CM1,PC1, Support KCl,KC5)</td>
</tr>
<tr>
<td>Gove and Faytong 2011</td>
<td>GUI events</td>
<td>Support Vector Machines (SVM) and induced grammar</td>
<td>Java Editor</td>
</tr>
<tr>
<td>Mair et al. 2009</td>
<td>Cost estimations</td>
<td>Neural networks, reasoning and induction</td>
<td>Desharnais</td>
</tr>
<tr>
<td>Boetticher 2003</td>
<td>GUI controls</td>
<td>Neural networks</td>
<td>E-Commerce</td>
</tr>
<tr>
<td>Koru and Liu 2005</td>
<td>Module measures (Size, coupling, cohesion, inheritance and complexity)</td>
<td>C4.5 decision tree</td>
<td>NASA</td>
</tr>
</tbody>
</table>
Table 6: Evaluation parameters for testing graphical user interface applications

<table>
<thead>
<tr>
<th>Evaluation Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI Representation</td>
<td>How is the GUI software modelled for the testing?</td>
</tr>
<tr>
<td>Testing techniques</td>
<td>Which testing technique is used either logical or functional?</td>
</tr>
<tr>
<td>Level of testing</td>
<td>Which kind of system testing is performed?</td>
</tr>
<tr>
<td>Test case model</td>
<td>What type of test model has been used?</td>
</tr>
<tr>
<td>Software domain</td>
<td>What kind of software domain does the testing program belong to?</td>
</tr>
<tr>
<td>Coverage conditions</td>
<td>Which kind of coverage criteria is adopted by the technique?</td>
</tr>
<tr>
<td>Tool support</td>
<td>Does the technique use the support of other tools?</td>
</tr>
<tr>
<td>Automated</td>
<td>Is the software fully automated or semi-automated?</td>
</tr>
<tr>
<td>Case study</td>
<td>Whether the technique is evaluated or not?</td>
</tr>
<tr>
<td>Fault injection</td>
<td>Whether the fault has been seeded manually or not?</td>
</tr>
</tbody>
</table>

Table 7: Comparison of GUI testing using evaluation parameters

<table>
<thead>
<tr>
<th>Reference</th>
<th>GUI representation</th>
<th>Testing techniques</th>
<th>Level of testing</th>
<th>Test case model</th>
<th>Software domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aleksandreh (2008)</td>
<td>Data dependency graph and Control dependency graph</td>
<td>White box testing</td>
<td>Unit testing</td>
<td>Data flow testing</td>
<td>JAVA</td>
</tr>
<tr>
<td>Chen et al. (2008)</td>
<td>Event flow graph</td>
<td>White box testing</td>
<td>Integration testing</td>
<td>Machine learning</td>
<td>JAVA based testing</td>
</tr>
<tr>
<td>Gross et al. (2012)</td>
<td>State transition model</td>
<td>White box testing</td>
<td>Unit testing</td>
<td>Machine learning</td>
<td>JAVA based Testing</td>
</tr>
<tr>
<td>Ames and Jie (2004)</td>
<td>Call graph</td>
<td>White box testing</td>
<td>System testing</td>
<td>User interface testing</td>
<td>JAVA, XML</td>
</tr>
<tr>
<td>Liu (2006)</td>
<td>Control flow graph</td>
<td>White box testing</td>
<td>Integration testing</td>
<td>Dataflow testing</td>
<td>JAVA</td>
</tr>
<tr>
<td>Di Lucca and Fassino (2006)</td>
<td>Control flow graph</td>
<td>White box testing</td>
<td>System testing</td>
<td>Dataflow testing</td>
<td>HTM, JSP</td>
</tr>
<tr>
<td>Kervinen et al. (2006)</td>
<td>State transition model</td>
<td>Black box testing</td>
<td>Integration testing</td>
<td>User interface testing</td>
<td>VBScript</td>
</tr>
<tr>
<td>Palva et al. (2005)</td>
<td>State transition model</td>
<td>Black box testing</td>
<td>Integration testing</td>
<td>State based testing</td>
<td>Microsoft Office</td>
</tr>
<tr>
<td>Travis and stanef (2008)</td>
<td>Call Stacks</td>
<td>Black box testing</td>
<td>Integration testing</td>
<td>User interface testing</td>
<td>HTML</td>
</tr>
<tr>
<td>Jones and Harrold (2008)</td>
<td>Program dependence graph</td>
<td>White box testing</td>
<td>Unit testing</td>
<td>Dataflow testing</td>
<td>C++</td>
</tr>
<tr>
<td>Deng et al. (2004)</td>
<td>Web application graph</td>
<td>White box testing</td>
<td>Integration testing</td>
<td>Dataflow testing</td>
<td>JAVA XML</td>
</tr>
<tr>
<td>Kumar and Yogi (2012)</td>
<td>Control flow graph</td>
<td>White box testing</td>
<td>Integration testing</td>
<td>Dataflow testing</td>
<td>XML</td>
</tr>
<tr>
<td>Harman and Alishahwan (2008)</td>
<td>URLGraph</td>
<td>White box testing</td>
<td>Integration testing</td>
<td>DFT</td>
<td>XML</td>
</tr>
</tbody>
</table>

EVALUATION CRITERIA FOR VARIOUS TESTING TECHNIQUES

It is mandatory to identify the issues in the GUI testing techniques. For this purpose, the following parameters in Table 6 were used by several authors for testing GUI applications.

The parameters in Table 6 are identified in the study and listed in Table 7 and 8.

METRICS USED IN GRAPHICAL USER INTERFACE TESTING

There are various measurements and metrics to evaluate the system. The test metrics are used to verify the effectiveness of the testing techniques. The direct metrics measure the GUI
Table 8: Comparison of evaluation for testing parameters

<table>
<thead>
<tr>
<th>Reference</th>
<th>Coverage conditions</th>
<th>Tool support</th>
<th>Automated</th>
<th>Case study</th>
<th>Fault injection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aleishabedh (2008)</td>
<td>Data coverage,</td>
<td>Java script compiler</td>
<td>Semi</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Branch coverage</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chou et al. (2008)</td>
<td>Transitions coverage</td>
<td>Full</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Gross et al. (2012)</td>
<td>Transitions coverage</td>
<td>Full</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Sun and Jones (2004)</td>
<td>Functional coverage</td>
<td>Jemmy test engine</td>
<td>Semi</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Ames and Jie (2004)</td>
<td>Transitions coverage</td>
<td>Abbot</td>
<td>Semi</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Liu (2006)</td>
<td>Data coverage</td>
<td>Full</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Di Lucca and</td>
<td>Data and transition coverage</td>
<td>Reweb, testweb</td>
<td>Semi</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Pasinato (2006)</td>
<td>Transition coverage</td>
<td>QTP</td>
<td>Semi</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Paiva et al. (2005)</td>
<td>Transition coverage</td>
<td>SPEC</td>
<td>Semi</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Travis et al.</td>
<td>Transition coverage</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Jones and Harrold</td>
<td>Code coverage</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Deng et al. (2004)</td>
<td>Transition coverage</td>
<td>AGENDA</td>
<td>Semi</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Kumar and Yogi</td>
<td>Transition coverage</td>
<td>DEJAVAQ</td>
<td>Semi</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>(2012)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Harman and Alishawwan (2008)</td>
<td>Transition coverage</td>
<td>JSpider</td>
<td>Semi</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 9: Direct metrics for testing graphical user interface applications

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reduced test suite size</td>
<td>Size difference between the original and modified test suite.</td>
</tr>
<tr>
<td>Program coverage</td>
<td>Whether the testing verifies codes and the data flows?</td>
</tr>
<tr>
<td>Fault detections</td>
<td>Number of faults detected by the testing technique.</td>
</tr>
<tr>
<td>Space requirement</td>
<td>How much memory space is required by the test suite?</td>
</tr>
<tr>
<td>Classes, methods</td>
<td>Are all the classes and methods verified in the application?</td>
</tr>
<tr>
<td>Conditions</td>
<td>Whether all the conditions are tested for their true and false values</td>
</tr>
<tr>
<td>NLOC</td>
<td>Do the test cases verify all the lines in the software?</td>
</tr>
<tr>
<td>User sessions</td>
<td>Are all the user sessions evaluated for web applications?</td>
</tr>
<tr>
<td>Uniform resource locators</td>
<td>Whether all the URLs are verified</td>
</tr>
</tbody>
</table>

applications quantitatively and the indirect metrics are used to evaluate the quality of the software. The direct metrics and their description are given in Table 9.

The list of indirect metrics and the components to be evaluated for those metrics are described in Table 10.

TESTING TOOLS USED IN THE GRAPHICAL USER INTERFACE TESTING

There are numerous testing tools and frameworks for verifying the GUI applications. The usage of testing tools and frameworks is presented in Table 11. Most of these tools are used to test the JAVA and C++ applications and the testing techniques in [17,33,55] test the GUI based applications.
There are many methods and play backing tools for preparing test scripts. Capture and playback tools capture the input and store it in the test log. They prepare the test scripts to evaluate the system by recording the user actions which are replayed for comparing the actual and expected results.

CONCLUSION

Since graphical user interface testing is critical in the testing field, this study addresses various testing technologies performed by various researchers. It explains the test models, metrics, software domain used for testing and issues in the existing studies. This survey gives an idea of testing GUI applications and using graphical user interface testing techniques. The direct and indirect metrics, to evaluate the user interface and web applications, are discussed. The regression techniques and the reduction of the test suites are studied. The extraction of the GUI components and the events using various tools has been discussed. The usage of machine learning algorithms, such as Support Vector Machines (SVM), J48 Decision Tree and Genetic Algorithms (GA) and the various data sets used for evaluation are also addressed. Some of the testing tools are spotted and the limitations of the play back tools are listed.

REFERENCES


Travison, D. and G. Stanefoff, 2008. Test instrumentation and pattern matching for automatic
failure identification. Proceedings of the 1st International Conference on Software Testing,
Yuan, X., M.B. Cohen and A.M. Memon, 2011. GUI interaction testing: Incorporating event
Zhang, L., 2011. Software defect prediction using non-negative matrix factorization. J. Software,
6: 2114-2120.