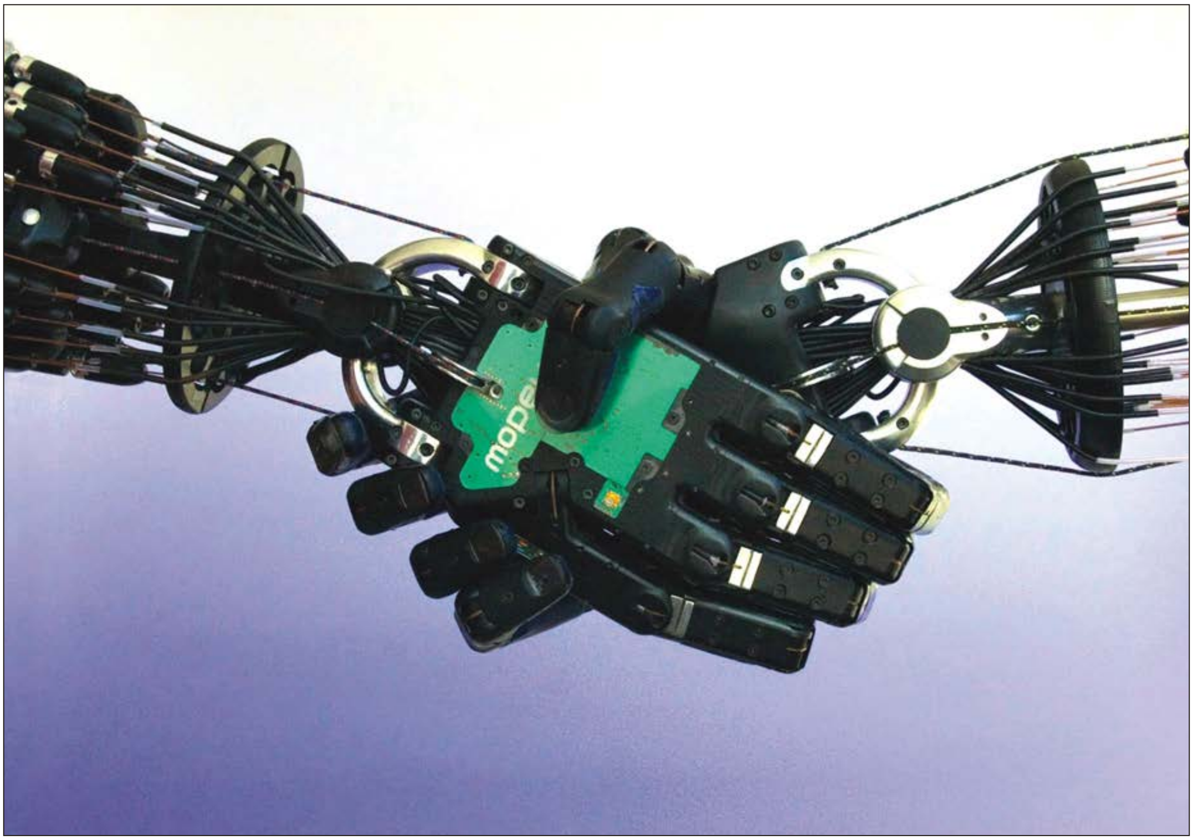

Journal of Artificial Intelligence

ISSN 1994-5450

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Artificial Intelligence 7 (3): 94-112, 2014
ISSN 1994-5450 / DOI: 10.3923/jai.2014.94.112

Survey on Graphical User Interface and Machine Learning Based Testing Techniques

[1]B. Uma Maheswari and [2]S. Valli
[1]Department of Master of Computer Application, St. Joseph's College of Engineering, Old Mamallapuram Road, Chennai, 600119, Tamilnadu, India
[2]Department 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 Staneff (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

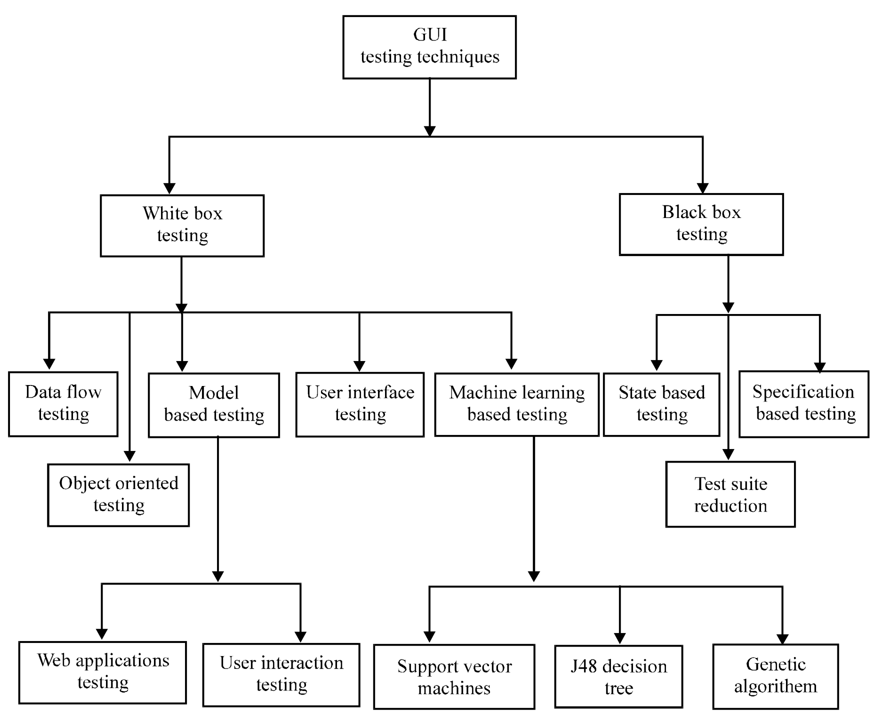

Fig. 1: Graphical user interface testing techniques

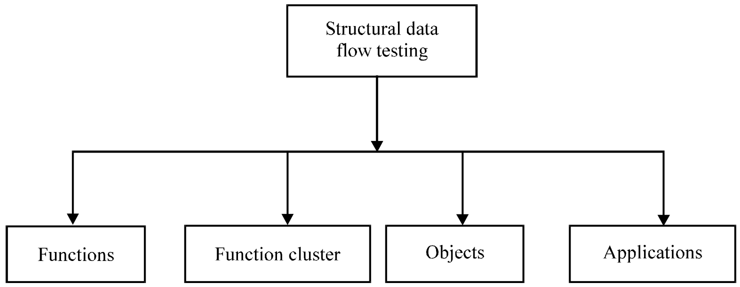

Fig. 2: Structural data flow testing between objects

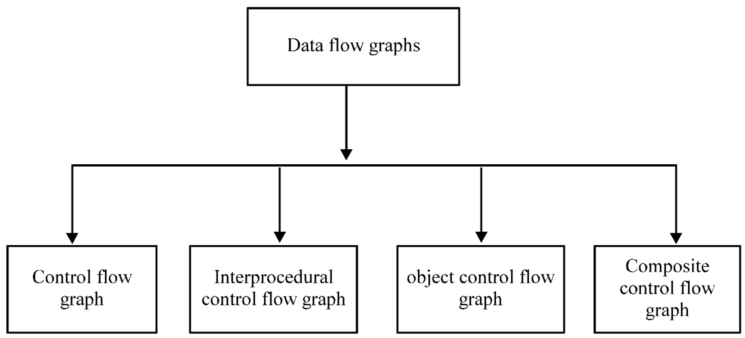

Fig. 3: Data flow graph for testing web applications

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 isillustrated 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 aAutoBlackTest tool to generate test cases automatically by navigating the GUI for software verification process.

The "Q-Learning Agent act" together with software and the automaticblack 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, Toggle 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 exists. The unrestricted input control text box was used for testing. Valid and invalid values were set for the properties of the control and the behaviour of the textbox 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 textbox 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. Sprenkle *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. Strecker 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

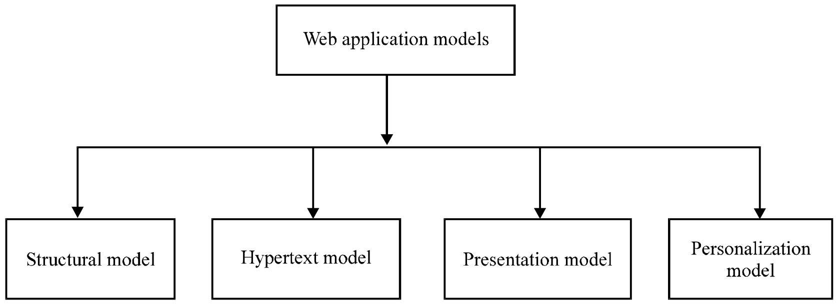

Fig. 4: Web application models

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 A_{i+1} 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

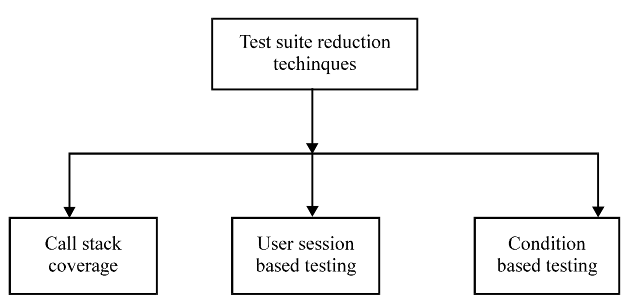

Fig. 5: Test suite removal techniques

Table 1: Coding window environment

Code editor window
Private Sub Form_Load( )
List1.BackColor = a + B
Combo1.BackColor = White
Check1.BackColor = C + D
Option1.BackColor = Blue
Text1.ToolTipText = Textbox
Option1.ToolTipText = Optionbox
List1.ToolTipText = Fruitslistbox
Combo1.Caption = Combobox
Picture1. Caption = Picturebox
Option1. Caption = Optionbox
List1.ToolTipText = Laplacetransform, 4
List1.AddItem = Fruitslistbox
Combo1.AddItem = Expertsystem
Picture5.Align = AlignRight
Data1.RecordSource = "Book Details"
End Sub

Table 2: Fault types and fault actions

Fault types	Fault actions
Data store	Errors in the code which interacts with the database
Logical	Logical errors in data flow
Form	Changes in the web pages
Appearance	Errors which change the user view
Link	Errors which change the web page location.

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":

$$\text{APFD} = 1 - \frac{\text{TF1}+\text{TF2}+\text{TF3}+\ldots+\text{TFn}}{\text{mn}} + \frac{1}{2\text{n}} \tag{1}$$

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 wasthe 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.

Table 3: Widgets and their verifications

Widgets	Verifications
JTextField, JTextArea	Compare the text displayed in text field and in text area
JComboBox, JList	Verifies the value of the combo and list boxes
JCheckBox or JRadioButton	Evaluate the selection of check box and radio button
JLabel	Verify the display of the text

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:

$$\text{fom} = \text{redux}\times\text{cvg}\times\text{fd} \tag{2}$$

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

Table 4: Case Studies conducted for graphical user interface testing techniques

Reference	Testing performed	Data set
McMaster and Memon (2008)	Call stack based test suite reduction	Space application (Antenna steering system) terpoffice suite
Sampath *et al*. (2007)	User session based testing	Online book Store, CPM, MASPLAS, Dspace
Alshraideh (2008)	Unit testing of JAVA Scripts	Triangular
Gross *et al*. (2012)	Search based testing	Address book, Calculator
Sun and Jones (2004)	Specification driven testing	Currency convertor
Ames and Jie (2004)	Critical paths for GUI	Numerical chameleon
Liu (2006)	Data flow analysis for JSP based, Applications	Session.jsp Login.jsp
Kervinen *et al*. (2006)	Model based testing	Microsoft windows
Jones and Harrold (2003)	Test suite reduction using condition testing	Space
Deng *et al*. (2004)	Testing on static web pages	Online bookstore
Kumar and Yogi (2012)	Regression testing of web pages	Jakartaregex, NanoXML, JABA
Sprenkle *et al*. (2008)	Analysis tool for JAVA programs	Masplas, dspace
Raha and Jadhav (2008)	Automation method for testing XML layers	EPCIS

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, filelistbox 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 (Gove 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, listbox, 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:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5}$$

$$\text{F} - \text{measure} = \frac{(2 \times \text{Precision} \times \text{Recall})}{(\text{Precision} + \text{Recall})} \tag{6}$$

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 Li *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 5: Parameters used in machine learning based testing

Reference	Features	Machine learning algorithm used for classification	Data set
Briand *et al.* (2008)	Category partitions	C4.5 decision tree	Space program
Elish and Elish (2008)	Software metrics (Mccabe complexity, basic halstead, derived halstead, line count and branch count)	Vector machines (SVM)	NASA (CM1,PC1, Support KC1,KC3)
Gove and Faytong (2011)	GUI events	Support Vector Machines (SVM) and induced grammars	Java Editor
Mair *et al.* (2000)	Cost estimations	Neural networks, reasoning and induction.	Desharnais
Boetticher (2003)	GUI controls	Neural networks	E-Commerce
Koru and Liu (2005)	Module measures (Size, coupling, cohesion, inheritance and complexity)	C4.5 decision tree	NASA

Table 6: Evaluation parameters for testing graphical user interface applications

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

Table 7: Comparison of GUI testing using evaluation parameters

Reference	GUI representation	Testing techniques	Level of testing	Test case model	Software domain
Alshraideh (2008)	Data dependency graph and Control dependency graph	White box testing	Unit testing	Data flow testing	JAVA
Chen *et al*. (2008)	Event flow graph	White box testing	Integration testing	Machine learning	JAVA based testing
Gross *et al*. (2012)	State transition model	White box testing	Unit testing	Machine learning	JAVA based Testing
Sun and Jones (2004)	Event based test specification	Black box testing	Integration testing	Specification based Testing	JAVA
Ames and Jie (2004)	Call graph	White box testing	System testing	User interface testing	XM, JAVA
Liu (2006)	Control flow graph	White box testing	Integration testing	Dataflow testing	JAVA
Di Lucca and Fasolino (2006)	Control flow graph	White box testing	System testing	Data flow testing	HTM, JSP
Kervinen *et al*. (2006)	State transition model	Black box testing	Integration testing	User interface testing	VBS cript
Paiva *et al*. (2005)	State transition model	Black box testing	Integration testing	State based testing	Microsoft office
Travison and staneff (2008)	Call Stacks	Black box testing	Integration testing	User interface testing	HTML
Jones and Harrold (2003)	Program dependence graph	White box testing	Unit testing	Data flow testing	C++
Deng *et al*. (2004)	Web application graph	White box testing	Integration testing	Data flow testing	XM, JAVA
Kumar and Yogi (2012)	Control flow graph	White box testing	Integration testing	Data flow testing	XML
Harman and Alshahwan (2008)	URLGraph	White box testing	Integration testing	DFT	XML

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

Referance	Coverage conditions	Tool support	Automated	Case study	Fault injection
Alshraideh (2008)	Data coverage, Branch coverage	Java script compiler	Semi	Yes	Yes
Chen *et al*. (2008)	Transitions coverage	Full	Yes	Yes	No
Gross *et al*. (2012)	Transitions coverage	Full	Yes	Yes	No
Sun and Jones (2004)	Functional coverage	Jemmy test engine	Semi	Yes	No
Ames and Jie (2004)	Transitions coverage	Abbot	Semi	Yes	No
Liu (2006)	Data coverage	Full	Yes	Yes	No
Di Lucca and Fasolino (2006)	Data and transition coverage	Reweb, testweb	Semi	Yes	No
Kervinen *et al*. (2006)	Transition coverage	QTP	Semi	Yes	No
Paiva *et al*. (2005)	Transition coverage	SPEC	Semi	Yes	No
Travison and Staneff (2008)	Transition coverage	-	Yes	Yes	No
Jones and Harrold (2003)	Code coverage	-	Yes	Yes	No
Deng *et al*. (2004)	Transition coverage	AGENDA	Semi	Yes	No
Kumar and Yogi (2012)	Transition coverage	DEJAVOO	Semi	Yes	No
Harman and Alshahwan (2008)	Transition coverage	JSpider	Semi	Yes	No

Table 9: Direct metrics for testing graphical user interface applications

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

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.

Table 10: Indirect metrics for testing graphical user interface applications

Metrics	Components to be evaluated
Functionality	Links, forms, cookies, database
Usability	Navigation, content, help
Interface	Webservers, application servers, database servers
Compatibility	Browsers, operating systems, mobile applications
Performance	Load testing, stress testing
Security	Authorization, authentication

Table 11: Testing tools and frameworks

Testing tools and frameworks	Usage of the tool
Cactus, JUnit	Unit testing the JAVA programs
HTTP	Find the correctness of theretrieved web pages
ParasoftWebking, Rational Robot, CanooWebTest	Automated testing tool
AGENDA	Test relational database applications
ColdFusion	Enables to build JAVA-EE applications for the enterprise
Groovy	High productivity web framework for the JAVA platform
PHP	Produce dynamic web pages
Python	Integrates object-oriented programming language, functional programming and imperative programming
Ruby	Dynamic object-oriented programming language.
Scala	Multi-paradigm language
AJAX	A framework to build dynamic web pages

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

Alshraideh, M., 2008. A complete automation of unit testing for Javascript programs. J. Comput. Sci., 4: 1012-1019.

Ames, A.K. and H. Jie, 2004. Critical paths for GUI regression testing. University of California, Santa Cruz, USA. http://users.soe.ucsc.edu/~sasha/proj/gui_testing.pdf

Artzi, S., A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar and M.D. Ernst, 2010. Finding bugs in web applications using dynamic test generation and explicit-state model checking. IEEE Trans. Software Eng., 36: 474-494.

Boetticher, G.D., 2003. Applying Machine Learners to GUI Specifications in Formulating Early Life Cycle Project Estimations. In: Software Engineering with Computational Intelligence, Khoshgoftaar, T.M. (Ed.). Vol. 731, Springer, New York, ISBN-13: 9781461504290, pp: 1-16.

Briand, L.C., Y. Labiche and X. Liu, 2007. Using machine learning to support debugging with tarantula. Proceedings of the 18th IEEE International Symposium on Software Reliability, November 5-9, 2007, Trollhattan, pp: 137-146.

Briand, L.C., Y. Labiche and Z. Bawar, 2008. Using machine learning to refine black-box test specifications and test suites. Proceedings of the 8th International Conference on Quality Software, August 12-13, 2008, Oxford, pp: 135-144.

Bryce, R.C., S. Sampath and A.M. Memon, 2011. Developing a single model and test prioritization strategies for event-driven software. IEEE Trans. Software Eng., 37: 48-64.

Ceri, S., P. Fraternali and A. Bongio, 2000. Web Modeling Language (WebML): A modeling language for designing web sites. Comput. Networks, 33: 137-157.

Chen, W.K., Z.W. Shen and C.M. Chang, 2008. GUI test script organization with component abstraction. Proceedings of the 2nd International Conference on Secure System Integration and Reliability Improvement, July 14-17, 2008, Yokohama, pp: 128-134.

Deng, Y., P. Frankl and J. Wang, 2004. Testing web database applications. ACM SIGSOFT Software Eng. Notes, 29: 1-10.

Di Lucca, G.A. and A.R. Fasolino, 2006. Testing Web-based applications: The state of the art and future trends. Inform. Software Technol., 48: 1172-1186.

Dias Neto, A.C. and G.H. Travassos, 2008. Surveying model based testing approaches characterization attributes. Proceedings of the 2nd ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, October 9-10, 2008, Kaiserslautern, Germany, pp: 324-326.

Eisenbarth, T., R. Koschke and D. Simon, 2003. Locating features in source code. IEEE Trans. Software Eng., 29: 210-224.

Elish, K.O. and M.O. Elish, 2008. Predicting defect-prone software modules using support vector machines. J. Syst. Software, 81: 649-660.

Fenton, N.E. and N. Ohlsson, 2000. Quantitative analysis of faults and failures in a complex software system. IEEE Trans. Software Eng., 26: 797-814.

Geetha, B.G., V. Palanisamy, K. Duraiswamy and G. Singaravel, 2008. A tool for testing of inheritance related bugs in object oriented software. J. Comput. Sci., 4: 59-65.

Gove, R. and J. Faytong, 2011. Identifying infeasible GUI test cases using support vector machines and induced grammars. Proceedings of the IEEE 4th International Conference on Software Testing, Verification and Validation Workshops, March 21-25, 2011, Berlin, Germany, pp: 202-211.

Gove, R. and J. Faytong, 2012. Machine learning and event-based software testing: Classifiers for identifying infeasible GUI event sequences. Adv. Comput., 86: 109-135.

Gross, F., G. Fraser and A. Zeller, 2012. EXSYST: Search-based GUI testing. Proceedings of the 34th International Conference on Software Engineering, June 2-9, 2012, Zurich, Switzerland, pp: 1423-1426.

Hamill, M. and K. Goseva-Popstojanova, 2009. Common trends in software fault and failure data. IEEE Trans. Software Eng., 35: 484-496.

Harman, M. and N. Alshahwan, 2008. Automated session data repair for web application regression testing. Proceeding of the 1st International Conference on Software Testing, Verification and Validation, April 9-11, 2008, Lillehammer, pp: 298-307.

Huang, S., M.B. Cohen and A.M. Memon, 2010. Repairing GUI test suites using a genetic algorithm. Proceedings of the 3rd International Conference on Software Testing, Verification and Validation, April 6-10, 2010, Paris, pp: 245-254.

Iyer, L.S., B. Gupta and N. Johri, 2005. Performance, scalability and reliability issues in web applications. J. Ind. Manage. Data Syst., 105: 561-576.

Johnson, S. and S. Valli, 2008. Hot method prediction using support vector machines. Ubiquitous Comput. Commun. J., 3: 67-73.

Johnson, S. and V. Shanmugam, 2011. Effective feature set construction for SVM-based hot method prediction and optimisation. Int. J. Comput. Sci. Eng., 6: 192-205.

Jones, J.A. and M.J. Harrold, 2003. Test-suite reduction and prioritization for modified condition/decision coverage. IEEE Trans. Software Eng., 29: 195-209.

Kervinen, A., M. Maunumaa, T. Paakkonen and M. Katara, 2006. Model-Based Testing through a GUI. In: Formal Approaches to Software Testing, Grieskamp, W. and C. Weise (Eds.). Springer, New York, pp: 16-31.

Klaib, M.F.J., S. Muthuraman, N. Ahmad and R. Sidek, 2010. Tree based test case generation and cost calculation strategy for uniform parametric pairwise testing. J. Comput. Sci., 6: 542-547.

Koru, A.G. and H. Liu, 2005. Building effective defect-prediction models in practice. IEEE Software, 22: 23-29.

Kumar, M.J.P. and M.K. Yogi, 2012. A survey on models and test strategies for event-driven software. Int. J. Comput. Eng. Res., 2: 1087-1091.

Li, Z., M. Harman and R.M. Hierons, 2007. Search algorithms for regression test case prioritization. IEEE Trans. Software Eng., 33: 225-237.

Liu, C.H., 2006. Data flow analysis and testing of JSP-based web applications. Inform. Software Technol., 48: 1137-1147.

Maheswari, B.U. and S. Valli, 2011. Algorithms for the detection of defects in GUI applications. J. Comput. Sci., 7: 1343-1352.

Maheswari, B.U. and S. Valli, 2013. Algorithms for detecting defects in user interface widgets. Eur. J. Sci. Res., 94: 261-272.

Mair, C., G. Kadoda, M. Lefley, K. Phalp, C. Schofield, M. Shepperd and S. Webster, 2000. An investigation of machine learning based prediction systems. J. Syst. Software, 53: 23-29.

Mariani, L., M. Pezze, O. Riganelli and M. Santoro, 2012. AutoBlackTest: Automatic black-box testing of interactive applications. Proceedings of the IEEE 5th International Conference on Software Testing, Verification and Validation, April 17-21, 2012, Montreal, QC., pp: 81-90.

McMaster, S. and A.M. Memon, 2008. Call-stack coverage for GUI test suite reduction. IEEE Trans. Software Eng., 34: 99-115.

Memon, A., I. Banerjee, N. Hashmi and A. Nagarajan, 2003. DART: A framework for regression testing nightly/daily builds of GUI applications. Proceedings of the International Conference on Software Maintenance, September 22-26, 2003, Amsterdam, The Netherlands, pp: 410-419.

Mesbah, A., A. van Deursen and D. Roest, 2012. Invariant-based automatic testing of modern web applications. IEEE Trans. Software Eng., 38: 35-53.

Murphy, C., G.E. Kaiser and M. Arias, 2007. An approach to software testing of machine learning applications. Proceedings of the 19th International Conference on Software Engineering and Knowledge Engineering, July 9-11, 2007, Boston, USA., pp: 167-172.

Neto, A.C.D., R. Subramanyan, M. Vieira and G.H. Travassos, 2007. A survey on model-based testing approaches: A systematic review. Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies: Conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering, November 5-9, 2007, ACM, Atlanta, GA, pp: 31-36.

Noorian, M., E. Bagheri and W. Du, 2011. Machine learning-based software testing: Towards a classification framework. Proceedings of the International Conference on Software Engineering and Knowledge Engineering, July 7-9, 2011, Boston, USA., pp: 225-229.

Paiva, A.C.R., J.C.P. Faria, N. Tillmann and R.A.M. Vidal, 2005. A model-to-implementation mapping tool for automated model-based GUI testing. Proceedings of the 7th International Conference on Formal Methods and Software Engineering, Manchester, UK., November 1-4, 2005, Springer-Verlag Berlin, Heidelberg, pp: 450-464.

Raha, D. and M.K. Jadhav, 2008. Automation method for testing XML/DB/XML layers. Proceedings of the 1st International Conference on Software Testing, Verification and Validation, April 9-11, 2008, Lillehammer, pp: 458-464.

Sampath, S., S. Sprenkle, E. Gibson, L. Pollock and A.S. Greenwald, 2007. Applying concept analysis to user-session-based testing of web applications. IEEE Trans. Software Eng., 33: 643-657.

Sampath, S., R.C. Bryce, A.G. Koru, V. Kandimalla and G. Viswanath, 2008. Prioritizing user-session-based test cases for web application testing. Proceedings of the International Conference on Software Testing, Verification and Validation, April 9-11, 2008, Washington, DC, USA., pp: 141-150.

Sampath, S. and R.C. Bryce, 2012. Improving the effectiveness of test suite reduction for user-session-based testing of web applications. Inform. Software Technol., 54: 724-738.

Sarala, S. and S. Valli, 2004. A tool to automatically detect defects in C++ programs. Proceedings of the 7th International Conference on Information Technology, December 20-23, 2004, Hyderabad, India, pp: 302-314.

Sarala, S. and S. Valli, 2006a. Algorithms for defect detection in console-based applications in C#. ICFAI J. Syst. Manage., 4: 46-55.

Sarala, S. and S. Valli, 2006b. Algorithms for defect detection in object oriented programs. Inform. Technol. J., 5: 876-883.

Sprenkle, S., H. Esquive, B. Hazelwood and L. Pollock, 2008. WebVizOr: A visualization tool for applying automated oracles and analyzing test results of web applications. Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques, August 29-31, 2008, Windsor, pp: 89-93.

Strecker, J. and A.M. Memon, 2008. Relationships between test suites, faults and fault detection in GUI testing. Proceedings of the 1st International Conference on Software Testing, Verification and Validation, April 9-11, 2008, Lillehammer, pp: 12-21.

Sun, Y. and E.L. Jones, 2004. Specification-driven automated testing of GUI-based Java programs. Proceedings of the 42nd Annual Southeast Regional Conference, April 2-3, 2004, Huntsville, AL., USA., pp: 140-145.

Travison, D. and G. Staneff, 2008. Test instrumentation and pattern matching for automatic failure identification. Proceedings of the 1st International Conference on Software Testing, Verification and Validation, April 9-11, 2008, Lillehammer, pp: 377-385.

Xie, Q. and A.M. Memon, 2007. Designing and comparing automated test oracles for GUI-based software applications. ACM Trans. Software Eng. Methodol., 16: 1-35.

Xie, X., J.W.K. Ho, C. Murphy, G. Kaiser, B. Xu and T.Y. Chen, 2011. Testing and validating machine learning classifiers by metamorphic testing. J. Syst. Software, 84: 544-558.

Yuan, X., M.B. Cohen and A.M. Memon, 2011. GUI interaction testing: Incorporating event context. IEEE Trans. Software Eng., 37: 559-574.

Zeng, H. and H. Miao, 2007. Model checking-based testing of web applications. Wuhan Univ. J. Nat. Sci., 12: 922-926.

Zhang, L., 2011. Software defect prediction using non-negative matrix factorization. J. Software, 6: 2114-2120.