A Survey on Models and Test strategies for Event-Driven Software

[1]Mr.J.Praveen Kumar [2]Manas Kumar Yogi
Asst.Prof. CSE Dept.
Malla Reddy College of Engineering and Technology

Abstract
A Graphical User Interface (GUI) testing tool is one to test applications user
Interface and to detect the correctness of applications functionality. Event-Driven Software (EDS) can change state based on incoming events; common examples are GUI and web applications.
These EDS pose a challenge to testing because there are a large number of possible event sequences that users can invoke through a user interface. While valuable contributions have been made for testing these two subclasses of EDS, such efforts have been disjoint. This work provides the first single model that is generic enough to study GUI and web applications together. This paper presentsdetail survey of the existing GUI testing tools . This paperalso summarizes various existing automated GUI testing approaches such as PerformanceTesting and Analysis (PTA), Model Based Testing (MBT), Combinatorial InteractionTesting (CIT), (GUI)-based Applications (GAPs). The feasibility of using java GUI captureand replay tools for GUI performance test automation has been studied.
The severelimitations of GUI tools when used for recording and replaying realistic session of the realworld Java applications have been also addressed. Various GUI testing tool are comparedin terms of performance.In this we use the model to define generic prioritization criteria that are applicable to both GUI and web applications. Our ultimate goal is to evolve the model and use it to develop a unified theory of how all EDS should be tested.

Keywords: Graphical User Interface, Performance Testing, event driven software (EDS), *t*-way interaction coverage, test suite prioritization, user-session testing, web-application testing, GUI testing

1. Introduction

The GUI testing is a process to test application's user interface and to detect if application is
functionally correct. GUI testing involves carrying set of tasks and comparing the result of same withthe expected output and ability to repeat same set of tasks multiple times with different data input and same level of accuracy. GUI Testing includes how the application handles keyboard and mouse events, how different GUI components like menubars, toolbars, dialogs, buttons, edit fields, list controls,images etc. reacts to user input and whether or not it performs in the desired manner.

ImplementingGUI testing for your application early in the software development cycle speeds up developmentimproves quality and reduces risks towards the end of the cycle. GUI
Testing can be performed bothmanually with a human tester or could be performed automatically with use of a software program.

Every software organization tests its software's, still the end product always have some issues
left. Testing team tries their best to find all the bugs before release of the software but still there areissues left in the product and they often re-appear as new modules are added to the software. Even thebest of manual testing process struggle to deliver an effective, efficient, accurate and increased testcoverage.

Manual testing is often error prone and there are chances of most of the test scenarios left out.

Automated GUI Testing is a software program which is used to analyze whether the desktop
application is functionally correct. Automated GUI Testing includes automating manual testing taskswhich are mostly time consuming and error prone. Automated GUI Testing is a more accurate,efficient, reliable and cost effective replacement to manual testing. Automated GUI Testing involves carrying set of tasks automatically and comparing the result of same with the expected output andability to repeat same set of tasks multiple times with different data input and same level of accuracy.
Implementing GUI Testing for your application early in the software development cycle speeds updevelopment improves quality and reduces risks towards the end of the cycle .
Automated GUI Testing is a solution to all the issues raised with Manual GUI Testing. An
Automated GUI Testing tool can playback all the recorded set of tasks, compare the results of
execution with the expected behavior and report success or failure to the test engineers. Once the GUItests are created they can easily be repeated for multiple number of times with different data sets andcan be extended to cover additional features at a later time. Most of the software organizations considerGUI Testing as critical to their functional testing process and there are many things which should beconsidered before selecting an Automated GUI Testing tool. A company can make great strides usingfunctional test automation. The

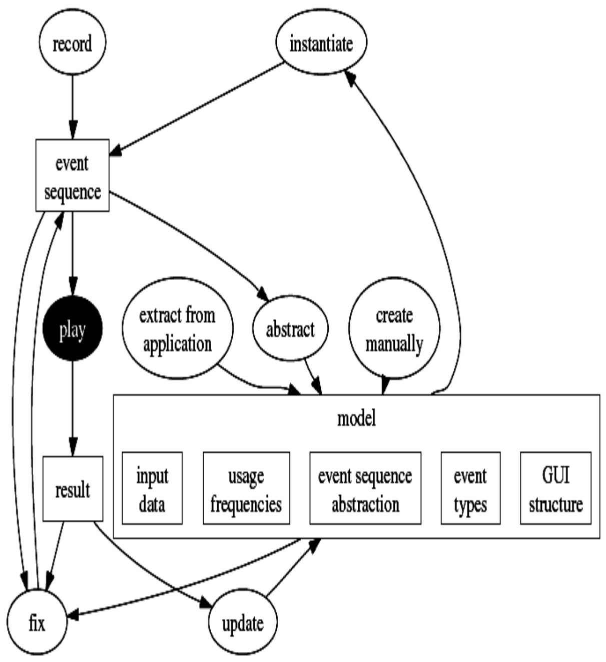
important benefits include, higher test coverage levels, greaterreliability, shorted test cycles, ability to do multi user testing at no extra cost, all resulting in increasedlevels of confidence in the software.

2. Existing models :

Testing for functional correctness of EDS such as stand-alone GUI and web-based applications is critical to many organizations. These applications share several important characteristics. Both are particularly challenging to test because users can invoke many different sequences of events that affect application behavior. Earlier research has shown that existing conventional testing techniques do not apply to either GUIs or web applications, primarily because the number of permutations of input events leads to a large number of states, and for adequate testing, an event may need to be tested in many of these states, thus requiring a large number of test cases (each represented as an event sequence). Researchers have developed several models for automated GUI testing and web application testing.

2.1. Performance Testing and Analysis (PTA)

It is practical to automatically test the performance of interactive rich-client Java applications when thefollowing two issues are addressed.
1) A metric need a measurement approach to quantify the performance of an interactive
application, and
2) A way to automatically perform realistic interactive sessions on an application, without
perturbing the measured performance .
This kind of GUI performance test automation has two key requirements that go beyond
traditional GUI test automation: (a) the need to replay realistically complex interactive sessions and (b)the minimal perturbation of the measured performance by the tool [1].
To find performance problemsin real applications, the length of the event sequences played during testing is important. Sequencesrepresenting only one or two events are often used for functional testing. They represent a form of unittest. Slightly longer sequences could be considered integration tests, as they often cover someinteractions between components. To find performance problems, however, event sequences need to besignificantly longer, so that the underlying system can reach the steady-state behavior that is normal inreal world usage. Using the GUI testing tools for performance testing is their use of harnesses andmock objects. Those artifacts represent deviations from the real-world setup and thus can affect theobserved performance [1].



2.2. Model Based Testing (MBT)

The new feedback-based technique has been used in a fully automatic end-to-end process for a specific type of GUI testing. The seed test suite (in this case, the smoke tests) is generated automatically using an existing event interaction graph model of the GUI, which represents all possible sequences of events that may be executed on the GUI.[2]. It utilizes runtime information as feedback for model-based GUI test case generation. However, runtime information has previously been employed for various aspects of test automation, and model-based testing has been applied to conventional software as well as *event driven software (EDS)*.It presents an overview of related research in the areas of model-based and EDS testing, GUI testing, and the use of runtime information as feedback for test generation.
Model-based testing automates some aspect of software testing by employing a model of the
software. The model is an abstraction of the software's behavior from a particular perspective (e.g.,software states, configuration, values of variables, etc.); it may be at different levels of abstraction,such as abstract states, GUI states, internal variable states, or path predicates, State machine models.

The most popular models used for software testing are state machine models. They model the
software's behavior in terms of its abstract or concrete states; they are typically represented as state transition diagrams. Several types of state machine models have been used for software testing .
The main inclusions in this test are:

- extension of work on automated, model-based ,systematic GUI test case generation.

- definition of new relationships among GUI events based on their execution.
- utilization of runtime state to explore a larger input space and improve fault detection.
- Immersion of the feedback-based technique into a fully automatic end-to-end GUI
- testing process and demonstration of its effectiveness on fielded and fault-seeded
- applications.
- Empirical evidence tying fault characteristics to types of test suites.
- Demonstration that certain faults require well crafted combinations of test cases and

oracles.

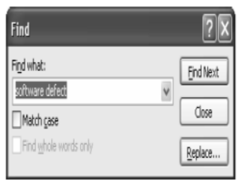
2.3. Combinatorial Interaction Testing (CIT)

Combinatorial Interaction Testing is a method which focuses on test prioritization techniques for GUI. The specific contributions of this work include: the first single model for testing stand-alone GUI and Web-based applications, a shared prioritization function based on the abstract model, and shared prioritization criteria. We validate the usefulness of these artifacts through an empirical study. The results show that GUI and Web-based applications, when recast using the model, showed similar behavior, reinforcing our belief that these classes of applications should be modeled and studied together. Other results show that GUI and Web applications behave differently, which has created opportunities for evolving the model and further experimentation.The generalize the model by evaluating its applicability and usefulness for other software testing activities, such as test generation.
It also makes contributions toward test prioritization research. Many of our prioritization criteria
improve the rate of fault detection of the test cases over random orderings of tests. We also develop hybrid prioritization criteria that combine several criteria that work well individually and evaluate whether the hybrid criteria result in more effective test orders.

3. Proposed model:

3.1. Modeling Test Cases

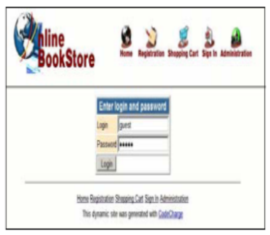
A test case is modeled as a sequence of actions. For eachaction, a user sets a value for one or more parameters.We provide examples of test cases for both GUI and webapplications next.



(a) Example GUI application window

Parameter, Value
1. <"Find what" drop-box, settext>
2. <"Find what" drop-box, leftclick dropdown>
3. <"Match case" checkbox, leftclick select>
4. <"Match case" checkbox, leftclick unselect>
5. <"Find whole words only" checkbox, leftclick select>
6. <"Find whole words only" checkbox, leftclick unselect>
7. <"Find Next" button, leftclick>
8. <"Close" button, leftclick>
9. <"Replace" button, leftclick>

(b) Nine parameter-values on the GUI window



(c) Example web application window

Parameter and value descriptions for Login.jsp
1. < Login text field, guest >
2. < Password text field, guest >
3. < FormAction, Login >
4. < FormName, Login >

(d) Four parameter-values in the Web Application window

Start of TC No. of Actions	<Testcase> <Length>4</Length>

Action 1	<Menu>
	    <Window>TerpWord</Window>
	    <Nonterminal>File</Nonterminal>
	    </Menu>
	    <Menu>
	    <Window>TerpWord</Window>
	    <Nonterminal>Save</Nonterminal>
	</Menu>

Action 2	<Component>
	    <Window>Save</Window>
	    <Nonterminal>File Name text field</Nonterminal
	    <Eventtype>SETTEXT</Eventtype>
	    <Eventvalue>exampleFile</Eventvalue>
	</Component>
	<Component>
	    <Window>Save</Window>
	    <Nonterminal>Files of Type drop-down
	      box</Nonterminal>
	    <Eventtype>LEFTCLICK SELECT</Eventtype>
	    <Eventvalue>Plain Text File (*.txt)</Eventvalue>
	</Component>
	<Component>
	    <Window>SAVE</Window>
	    <Nonterminal>OK button</Nonterminal>
	    <Eventtype>LEFTCLICK</Eventtype>
	</Component>

Action 3	<Menu>
	    <Window>TerpWord</Window>
	    <Nonterminal>Edit</Nonterminal>
	</Menu>
	<Menu>
	    <Window>TerpWord</Window>
	    <Nonterminal>Find...</Nonterminal>
	</Menu>

Action 4	<Component>
	    <Window>Find</Window>
	    <Nonterminal>Find what drop-box</Nonterminal
	    <Eventtype>SETTEXT</Eventtype>
	    <Eventvalue>software defect</Eventvalue>
	</Component>
	<Component>
	    <Window>Find</Window>
	    <Nonterminal>FindNext button</Nonterminal>
	    <Eventtype>LEFTCLICK</Eventtype>
	</Component>

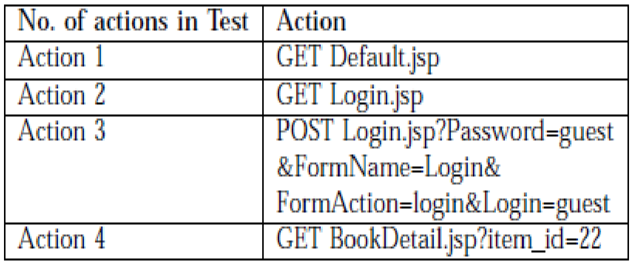
End of TC	</Testcase>

(a) Sample GUI test case

Window name	P-V No.	P-V description (<parameter,value>)
TerpWord	PV.1 PV.2	<File,null> <Save,null>
Save	PV.3 PV.4 PV.5	<File name text field, SETTEXT="exampleFile"> <Files of Type drop-down box, LEFTCLICK SELECT="Plain Text File (*.txt)"> <OK button, LEFTCLICK>
TerpWord	PV.6 PV.7	<Edit, null> <Find..., null>
Find	PV.8 PV.9	<Find what drop-box, SETTEXT="software defect"> <FindNext button, LEFTCLICK>

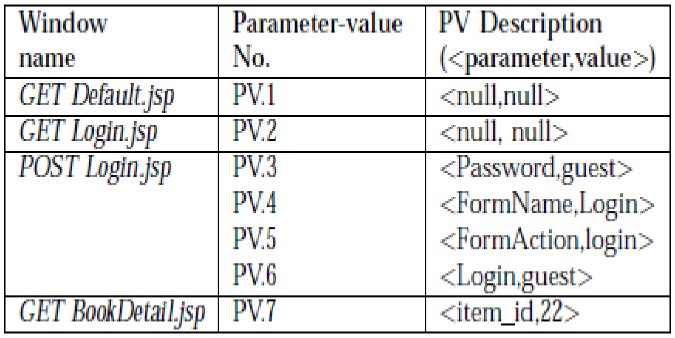
(b) Windows and User Interactions in test case

TABLE 2: Example GUI test case



No. of actions in Test	Action
Action 1	GET Default.jsp
Action 2	GET Login.jsp
Action 3	POST Login.jsp?Password=guest &FormName=Login& FormAction=login&Login=guest
Action 4	GET BookDetail.jsp?item_id=22

(a) Sample user-session-based web test case

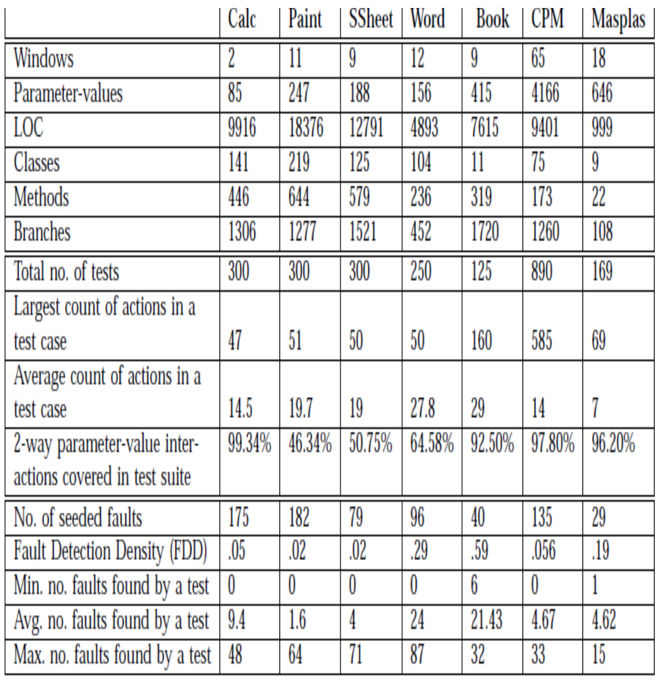


Window name	Parameter-value No.	PV Description (<parameter,value>)
GET Default.jsp	PV.1	<null,null>
GET Login.jsp	PV.2	<null, null>
POST Login.jsp	PV.3 PV.4 PV.5 PV.6	<Password,guest> <FormName,Login> <FormAction,login> <Login,guest>
GET BookDetail.jsp	PV.7	<item_id,22>

(b) Windows and Parameter-Values in test case

TABLE 3: Example Web test case

Previous work treats stand-alone GUI and web-based applications as separate areas of research. However, these types of applications have many similarities that allow us to create a single model for testing such event driven systems. This model may promote future research to more broadly focus on stand-alone GUI and web-based applications instead of addressing them as disjoint topics. Within the context of this model, we develop and empirically evaluate several prioritization criteria and apply them to four stand-alone GUI and three web-based applications and their existing test suites. Our empirical study evaluates the prioritization criteria. We present our threats to validity in this section because several opportunities for future research are created by the threats to validity of the results of our empirical study. Threats to construct validity are factors in the study
design that may cause us to inadequately measure concepts of interest.



	Calc	Paint	SSheet	Word	Book	CPM	Masplas
Windows	2	11	9	12	9	65	18
Parameter-values	85	247	188	156	415	4166	646
LOC	9916	18376	12791	4893	7615	9401	999
Classes	141	219	125	104	11	75	9
Methods	446	644	579	236	319	173	22
Branches	1306	1277	1521	452	1720	1260	108
Total no. of tests	300	300	300	250	125	890	169
Largest count of actions in a test case	47	51	50	50	160	585	69
Average count of actions in a test case	14.5	19.7	19	27.8	29	14	7
2-way parameter-value inter-actions covered in test suite	99.34%	46.34%	50.75%	64.58%	92.50%	97.80%	96.20%
No. of seeded faults	175	182	79	96	40	135	29
Fault Detection Density (FDD)	.05	.02	.02	.29	.59	.056	.19
Min. no. faults found by a test	0	0	0	0	6	0	1
Avg. no. faults found by a test	9.4	1.6	4	24	21.43	4.67	4.62
Max. no. faults found by a test	48	64	71	87	32	33	15

Test suit

3.2. Test suites

Models of the TerpOffice applications, called event-flow graphs [1], were used to generate test cases. The test-case generation algorithm has also been described earlier [1]; in summary, the algorithm is based on graph traversal;
starting in one of the events in the application's main window,the event-flow graphs were traversed, outputting the encountered event sequences as test cases. In all, 300 test cases were generated for each application.
The suites for web applications are based on usage of the application, also referred to as user-session-based testsuites.
A total of 125 test cases were collected for Book, by asking for volunteer users by sending emails to local newsgroups and posting advertisements in the University of Delaware's classifieds. For CPM, 890 test cases were collected from instructors, teaching assistants, and students using CPM during the 2004-05 and
2005-06 academic years at the University of Delaware. A total of 169 test cases were collected when our third subject application, Masplas, was deployed for the Mid-Atlantic Symposium on Programming Languages .
Table shows the characteristics of the test casesused in our study, such as the total number of testcases for each application, and statistics on the lengths
of the test cases. We also report the total numberof unique parameter-values and the percentage of 2-way parameter-value interactions covered in the test suites. We compute the percentage of 2-way parametervalueinteractions by counting the number of uniqueparameter-values on each window that can be selectedin combination with unique parameter-values on otherwindows within the application.

4. Conclusion

stand-alone GUI and web-based applications as separate areas of research. However, these types of applications have many similarities that allow us to create a single model for testing such eventdriven systems. This model may promote future research to more broadly focus on stand-alone GUI and webbased applications instead of addressing them as disjoint topics. Other researchers can use our common model to apply testing techniques more broadly. their existing test suites. Our empirical study evaluates the prioritization criteria. Our ability to develop prioritization
criteria for two types of event-driven software indicates the usefulness of our combined model for the
problem of test prioritization. Our results are promising as many of the prioritization criteria that we use improve
the rate of fault detection over random ordering of test cases. We learn that prioritization by 2-way and PV-LtoS
generally result in the best improvement for the rate of fault detection in our GUI applications and one of our
web applications. However, for our web applications ,frequency-based techniques provide the best rate of fault
detection in 2 out of the 3 subjects. We attribute this to the source of the test cases. The test suites for the
web applications come from real user-sessions, whereas the GUI test cases were automatically generated without
influence from users.

REFERENCES

[1] A. M. Memon and Q. Xie, “Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software,” *IEEE Trans. Softw. Eng.*, vol. 31, no. 10, pp. 884–896, Oct. 2005.

[2] A. Andrews, J. Offutt, and R. Alexander, “Testing web applications by modeling with FSMs,” *Software and Systems Modeling*, vol. 4, no. 3, pp. 326–345, Jul. 2005.

[3] G. D. Lucca, A. Fasolino, F. Faralli, and U. D. Carlini, “Testing web applications,” in *the IEEE Intl. Conf. on Software Maintenance*. Montreal, Canada: IEEE Computer Society, Oct. 2002, pp. 310–319.

[4] F. Ricca and P. Tonella, “Analysis and testing of web applications,” in*the Intl. Conf. on Software Engineering*. Toronto, Ontario, Canada: IEEE Computer Society, May 2001, pp. 25–34.

[5] R. C. Bryce and A. M. Memon, “Test suite prioritization by interactioncoverage,” in *Proceedings of The Workshop on Domain-Specific Approaches to Software Test Automation (DoSTA 2007); co-located with The 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. Dubrovnik, Croatia: ACM, Sep. 2007, pp. 1–7.

[6] S. Sampath, R. Bryce, G. Viswanath, V. Kandimalla, and A. G. Koru, “Prioritizing user-session-based test cases for web application testing,” in *the International Conference on Software Testing, Verification and Validation*. Lillehammer, Norway: IEEE Computer Society, Apr. 2008, pp. 141–150. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 37, NO. 1, JAN/FEB 2011 IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. X, NO. X, JANUARY 2010 18

[7] P. Brooks, B. Robinson, and A. M. Memon, “An initial characterization of industrial graphical user interface systems,” in *Proceedings of the International Conference on Software Testing, Verification and Validation*, 2009, pp. 11–20.

[8] L. White, “Regression testing of GUI event interactions,” in *Proceedings of the International Conference on Software Maintenance*. IEEE Computer Society, Nov. 1996, pp. 350–358.

[9] “Web site test tools and site management tools,” accessed on <http://www.softwareqatest.com/qatweb1.html>, accessed on Apr. 5, 2009.

[10] D. C. Kung, C.-H. Liu, and P. Hsia, “An object-oriented web test model for testing web applications,” in *The First Asia-Pacific Conf. on Quality Software*. Singapore: IEEE Computer Society, Oct. 2000, pp. 111–120.

Authors Description

Manas Kumar Yogi
Pursuing Mtech.(CSE Dept.)
Malla Reddy College of Engineering and Technology, Hyderabad.

Mr.J.Praveen Kumar
Asst.Prof. CSE Dept.
Malla Reddy College of Engineering and Technology, Hyderabad.