Volume 2, Issue 3, March 2012 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Hybrid Model: EDS Using Test Prioritization Strategies

P. Srinivasa Reddi*
Associate Professor
IT Department, SVEC, Tirupati

CH. Gouthami
M. Tech Student
IT Department, SVEC, Tirupati
gowthami.ch52@gmail.com

Abstract— **Now a day's Event-Driven Software application is playing a prominent role. EDS have rapidly become a crictical part of business for many organizations. All EDSs take sequences of events (e.g., messages and mouse-clicks) as input, change their state, and produce an output(e.g., events, system calls, and text messages); Common examples of EDS include graphical user interfaces (GUIs), web applications, network protocols, embedded software, software components, and device drivers. The term Events can be user actions such as clicking a mouse button or pressing a key or System occurrences. Most Modern EDS applications, particularly those that run in Macintosh and Windows environments, are said to be Event-Driven because they are designed to respond to events. The contributions of the work included: 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. The results of showed that GUI and web-based applications, when recast using the model, showed similar behavior. This paper extends the single model to hybrid prioritization criteria that combine several criteria that work well individually and evaluate whether the hybrid criteria result in more effective test orders.**

Keywords—**Event-Driven Software, test-suite prioritization, Web application testing, GUI testing, Interaction Testing, *User-session-based Testing***

I. Introduction

Event-Driven Software (EDS) is a class of software that is quickly becoming ubiquitous. It can change state based on incoming events. Events can be user actions such as clicking a mouse button or pressing a key. Examples include Web applications, graphical user interfaces, network protocols, device drivers, and embedded software. . In GUI applications the term GUI is the front end to a software's underlying back end code. In Web application the term Web is a set of static or dynamic web pages that are accessible by users through a browser over a network.

Many of today's EDS software applications are developed and maintained by multiple programmers often geographically distributed, who work on parts of the over all application code. Quality assurance tasks such as testing have become important for EDS as they are now being used in critical applications. Researchers have developed several models for automated GUI testing and web application testing. In GUI testing the DART (Daily Automated Regression Tester) [1], is used to test the GUI. It analyzes each widget in each of the windows of the project. It computes the total number of possible smoke tests and the test designer specifies the number of test cases that should be executed. For GUI smoke testing, a tester has to produce test cases that satisfy the following requirements: The smoke test cases should be generated and executed quickly. As the GUI is modified, many of the test cases should remain usable.

In web application ting a FSM (Finite State Machines) model is used to test the web application. Some testing criteria:

Page testing: every page in the site is visited at least once in some test case.

Hyperlink testing: every hyperlink from every page in the site is traversed at least once.

Definition-use testing: all navigation paths from every definition of a variable to every use of it, forming data dependence, is exercised.

All-uses testing: at least one navigation path from every definition of a variable to every use of it, forming data dependence, is exercised.

All-paths testing: every path in the site is traversed in some test case at least once.

Despite the above similarities of GUI and web applications, all the efforts to address their common testing problems have been made separately due to two reasons. First, is the challenge of coming up with a single model of these applications that adequately captures their event-driven nature, yet abstracts away elements that are not important for functional testing. Second, is the unavailability of subject applications and tools for researchers. We use the term GUI testing [1] to mean that a GUI-based software application is tested solely by performing sequences of events on GUI widgets and the correctness of the software is determined by examining only the state of the GUI widgets.

we will further generalize the model by evaluating its applicability and usefulness for other software testing activities, such as test generation. Our study 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.

II. BACKGROUND AND RELATED WORK

This section provides background on EDS applications i.e., GUI applications and web applications [2],[3],[4],.,

A. Event-Driven Software

The event-driven nature of GUIs presents the first serious testing difficulty. Because users may click on any pixel on the screen, there are many, many more possible user inputs that can occur .The user has an extremely wide choice of actions. At any point in the application, the users may click on any field or object within a window. They may bring another window in the same application to the front and access that. The window may be owned by another application. The user may choose to access an operating system component directly e.g. a system configuration control panel .The large number of available options mean that the application code must at all times deal with the next event, whatever it may be. In the more advanced development environments, where sophisticated frameworks are being used, many of these events are handled 'behind the scenes'. With less advanced toolkits, the programmer must write code to handle these events explicitly. Many errors occur because the programmer cannot anticipate every context in which their event handlers are invoked.

B. GUI Testing

There are four stages for GUI Testing. They are:

- Low level - maps to a unit test stage.
- Application - maps to either a unit test or functional system test stage.
- Integration - maps to a functional system test stage.
- Non-functional - maps to non-functional system test stage.

The mappings described above are approximate. Clearly there are occasions when some"GUI integration testing" can be performed as part of a unit test. The test types in "GUI application testing" are equally suitable in unit or system testing. In applying the proposed GUI test types, the objective of each test stage, the capabilities of developers and testers, the availability of test environment and tools all need to be taken into consideration before deciding whether and where each GUI test type is implemented in your test process.

Stage	Test types
Low-Level	checklist testing, Navigation
Application	Equivalence partitioning, Boundary values
Integration	Desktop integration
Non-functional	Soak testing, compatibility testing, platform/environment

Fig 1: Four stages of Test types

C. Web Application Testing

Three main classes of testing techniques are used for web applications, [3] namely, functional testing, structural testing and user-session-based testing.

i. Functional Testing

Many of the current testing tools address web usability, performance, and portability issues. For example, link testers navigate a web site and verify that all hyperlinks refer to valid documents. Form testers create scripts that initialize a form, press each button and type preset scripts into text fields, ending with pressing the submit button. Compatibility testers ensure that a web application functions properly with in different browsers.

ii. Structural Testing

Ricca and Tonella [6] developed a high-level UML-based representation of a web application and described how to perform page, hyperlink, def-use, all-uses, and all-paths testing based on the data dependencies computed using the model browsers.

iii. User-session-based Testing

In user-session-based testing [6], data is collected from users of a web application by the web server. Each *user session* is a collection of user requests in the form of base request and name-value pairs (e.g., form field data). A base request for a web application is the request type and resource location without associated data (e.g., *GET /servlets/authentication/Login.jsp*). More specifically, a user session is defined as beginning when a request from a new IP address reaches the server and ending when the user leaves the web site or the session times out.

Tools such as WebKing and Rational Robot provide automated testing for webapplications by collecting data from users through few configuration changes to the web server.

iv. Test Prioritization Strategies For EDS

Given (T, Π, f), where T is a test suite[5],[6], Π is the set of all test suites that are prioritized orderings of T obtained by permuting the tests of T, and f is a function to evaluate the orderings from Π to the real numbers, the problem is to find a permutation, π ε Π such that . Prioritization [6] can be based on any criteria. Examples include code coverage, cost estimates, event coverage, and others. Test length based on number of base requests (Req-LtoS, Req-StoL): order by the number of HTTP requests in a test case Frequency-based prioritization (MFAS, AAS): order such that test cases that cover most frequently accessed pages/sequence of pages are selected for execution before test cases that exercise the less frequently accessed pages/sequences of pages. Unique coverage of parameter-values (1-way): order tests to cover all unique parameter-values as soon as possible. 2-way parameter-value interaction coverage (2-way): order tests to cover all pair-wise combinations of parameter-values between pages as soon as possible. Test length based on number of parameter values (PV-LtoS, PV-StoL): order by number of parameter-values used in a test case. Random: randomly permute the order of tests.we have developed additional criteria to prioritize GUI and web-based programs. Bryce and Memon prioritize pre-existing test suites[6].[7],[8] for GUI-based programs by the lengths of tests (i.e., the number of steps in a test case, where a test case is a sequence of events that a user invokes through the GUI), early coverage of all unique events in a test suite, and early event interaction coverage between windows (i.e., select tests that contain combinations of events invoked from different windows which have not been covered in previously selected tests). In half of these experiments, event interaction-based prioritization results in the fastest fault detection rate. The two applications that cover a larger percentage of interactions in their test suites (64.58% and 99.34% respectively) benefit from prioritization by interaction coverage. The applications that cover a smaller percentage of interactions in their test suites (46.34% and 50.75% respectively) do not benefit from prioritization by interaction coverage. We concluded that the interaction coverage of the test suite is an important characteristic to consider when choosing this prioritization technique. Similarly, in the web testing prioritize the user-session-based test suites for web applications. These experiments showed that systematic coverage of event-interactions and frequently accessed sequences improve the rate of fault detection when tests do not have a high Fault Detection Density (FDD), where FDD is a measure of the number of faults that each test identifies on average.

In our past work, we have developed different criteria to prioritize GUI and Web-based programs. Prioritize the preexisting test suites for GUI-based programs by the lengths of tests (i.e., the number of steps in a test case, where a test case is a sequence of events that a user invokes through the GUI), early coverage of all unique events in a test suite, and early event interaction coverage between windows (i.e., select tests that contain combinations of events invoked from different windows which have not been covered in previously selected tests) [5]. In half of these experiments, event interaction-based prioritization results in the fastest fault detection rate. The two applications that cover a larger percentage of interactions in their test suites (64.58 and 99.34 percent, respectively) benefit from prioritization by interaction coverage. The applications that cover a smaller percentage of interactions in their test suites (46.34 and 50.75 percent, respectively) do not benefit from prioritization by interaction coverage. We concluded that the interaction coverage of the test suite is an important characteristic to consider when choosing this prioritization technique.

These experiments showed that systematic coverage of event interactions and frequently accessed sequences improve the rate of fault detection when tests do not have a high Fault Detection Density (FDD), where FDD is a measure of the number of faults that each test identifies on average.

III. HYBRID MODEL

To develop the hybrid model, we first review how GUI and Web applications operate. For GUI applications, action listeners are probably the easiest—and most common—event handlers to implement. The programmer implements an action listener to respond to the user' s indication that some implementation-dependent action should occur. When the user performs an event, e.g., clicks a button, chooses a menu item, an action event occurs. The result is that (using the Java convention) an actionPerformed message is sent to all action listeners that are registered on the relevant component. To develop the hybrid model we can generate no. of test cases from the given application.

A test case is modeled as a sequence of actions. For each action, a user sets a value for one or more parameters. Fig. 2 shows an example window from a GUI application entitled "Replace." We use the term window to refer to GUI windows such as this Replace window. The window has several widgets. A user typically sets some properties of these widgets (e.g., checking a check-boxes, adding text to a text field) and "submits" this information. Underlying code then uses these settings to make changes to the software state. Because of how widgets are used in the GUI, we refer to them as parameters in this paper. We refer to the settings for the widgets as values. We refer to the pair <parameter name; value> as parameter-values. For instance, in Table 1, the "Find what" Combo box is a parameter with the value "software"; the "Match case" check-box is a parameter with the value "false"; these parameters are used by actions.

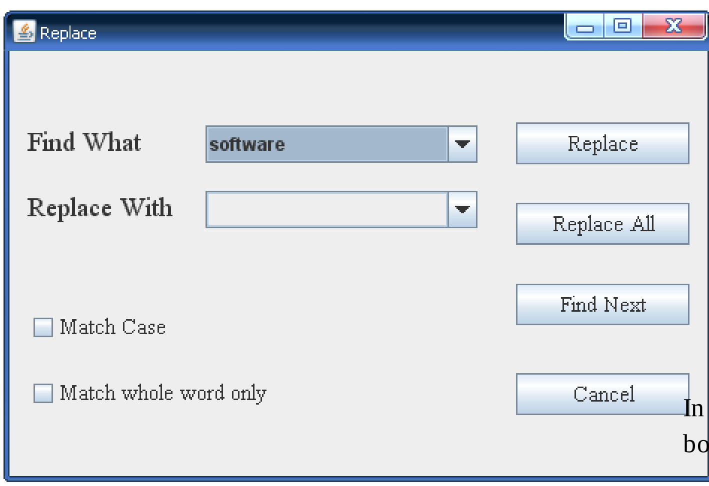

Fig. 2. GUI Application Window

Parameter ,Value
1. <"Find What" combo box, set text>
2. <"Find What" combo box, left click dropdown>
3. <"Replace With" combo box, set text>
4. <"Replace With" combo box, left click dropdown>
5. <"Match case" checkbox, left click select >
6. <"Match case" checkbox, left click unselect >
7. <"Match whole word only" checkbox, left click select >
8. <"Match whole word only" checkbox, left click unselect >
9.<"Replace" button, left click>
10.<"Replace All" button, left click>
11.<"Find Next" button, left click>
12. <"Cancel" button, left click>

Fig. 3. Twelve Parameter values on the GUI Window

Fig 3 shows all possible parameter-values for the window shown in Fig 2. The consecutive sequence of user interactions on a single window as an action. An example of an action for the Replace window is the sequence "enter 'software' in text-box," "check 'Match case' check-box," and "click-on 'Find Next' button."

Similarly, for Web applications, we refer to a Web application page as a window. As with GUIs, widgets in a window are referred to as parameters, and their settings as values. Fig 4 shows the "Login" text field is a parameter that is set to the value "guest" and their Parameter, values[7] are shown that Table 2 and Table 3 shows that a sample GUI Test cases and their Parameter values.

Fig. 4. Web Application Window

Table 1
Four parameter values on web application

Parameter, Value
1. <FormName, Login>
2.<Login text field, guest>
3. <Password text field, guest>
4.<FormAction, Login >

In Hybrid model we can use different no. of criterions for both GUI applications and web based applications.

A. Parameter-value interaction coverage-based criteria

In this module 1-way and 2-way parameter-value interaction coverage techniques select tests to systematically cover parameter-value interactions between windows.

- *1- Way*

Table 3 shows that a 1-way criterion selects a next test to maximize the number of parameter-values that do not appear in previously selected tests.

Table 3
1-Way

Test	Parameter-values	Windows visited
t 1	1->2->5->6->15->8->4->8	W1->W2->W4->W2->W1->W2
t 2	1->3->6->17	W1->W2->W5
t 3	2->3->6->8->10->11->12->9->13->16	W1->W2->W3
t 4	3	W1

The first selected test is t1 because it covers because it covers 6 parameter values (1, 2, 4, 2, 1, 2).The next test selected is t2 because it covers 3 parameter values. The final prioritized sequence is t1, t2, t3, t4.

- **2- Way:**

Table 4 shows the criterion selects a next test to maximize the number of 2-way.

Table 4
2-way

Test No.	No. of 2-way Interactions	List of 2-way interactions
t 1	13	(1,6), (1,15), (1,8), (2,6), (2,15), (2,5) ,(5,6), (5,15), (5,7), (6,14), (4,8), (4,6), (4,8)
t 2	15	(1,6), (1,15), (1,8), (2,6), (2,15), (2,5) ,(5,6), (5,15), (5,7), (6,14), (4,8), (4,6), (4,8), (6,15),(6,8).

B. Count based Criteria

In this criterion we count the number of windows, actions, or parameter-values that they cover.

Window coverage

In this criterion, we prioritize tests by giving preference to test cases that cover the most *unique windows* that previous tests have not covered.

Action count-based

In this criterion, we prioritize tests by the number of actions in each test (duplicates included). The prioritization includes selecting the test cases with preference given to those that include the most number of actions.

Parameter-value count-based

Test cases contain settings for parameters that users set to specific values. We prioritize tests by the number of parameters that are set to values in a test case (duplicates included). This includes selecting those tests with the largest number of parameter value settings in a test first.

C. Frequency-based Criteria

In this module we prioritize the test case based on frequency.

- **Most-frequently present sequence of windows (MFPS)**

Table 5 shows that the criterion, MFPS, we first identify the most frequently present sequence of windows, s_i, in the test suite and order test cases in decreasing order of the number of times that s_i appears in the test case. Then, from among the test cases that do not use s_i even once, the most frequently present sequence, s_j is identified, and the test cases are ordered in decreasing order.

Table 5
Frequence of Presence Table

Sequence name	Totalnoof occurrences	Test cases with maximum
W1->W2	**4**	t 1,t2, t3
W2->W4	**1**	t 1
W4->W2	**1**	t 1
W2->W5	**1**	t 2
W2->W3	**1**	t 3

- **All present sequence of windows (APS)**

In APS, the frequency of occurrence of all sequences is used to order the test suite. For each sequence, s_i, in the application, beginning with the most frequently present sequence, test cases that have maximum occurrences of these sequences are selected for execution before other test cases in the test suite.

- **Weighted sequence of windows (Weighted-Freq)**

Table 6 shows that the weighted technique assigns each test case a weighted value based on all the window sequences it contains, and the importance (the weight of a sequence of windows is measured by the number of times the sequence appears in the suite) of the window sequence. Initially, we identify the frequency of appearance of each unique sequence of windows in the test suite and build a weighted matrix for each unique window sequence. This frequency of appearance is the weight of the unique sequence of window.

Table 6
weighted sequence of windows

Test	Parameter values	Windows visited
t1	1->2->5->6->15->8->4->8	W1->W2->W4->W2->W1->W2
t2	1->3->6->17	W1->W2->W5
t3	2->3->6->8->10->11->12->9->13->16	W1->W2->W3
t4	3	W1

D. Subject Applications

In Hybrid model we have taken four GUI and three Web-based applications

1. Calc
2. Paint
3. SSheet
4. Word
5. Book
6. CPM
7. Masplas

Based on the subject applicatons by applying the hybrid model we can calculate the Fault detection rate.

IV. RESULTS

Table 7 shows the hybrid model of CPM for 3 criteria i.e.' APS, 2-way and MFPS

Table 7
CPM:Hybrid-Average Percentage Fault Detected

% of test suite run	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
APS , 2-way and MFPS										
2-way	93.22	93.22	93.22	93.22	94.69	94.69	95.62	95.62	95.62	95.62
APS	**93.74**	**94.19**	**95.88**	**95.88**	**95.88**	**96.11**	96.11	96.11	96.11	96.18
APS-2-way-10%-no-APFD-increase	**93.74**	93.74	93.74	93.74	95.13	95.13	96.08	96.08	96.08	96.08
APS-2-way-20%-no-APFD-increase	**93.74**	**94.19**	**95.88**	**95.88**	**95.88**	95.88	**96.92**	**96.92**	**96.92**	**96.92**
MFPS	**93.33**	**93.33**	**93.33**	**93.33**	93.33	94.28	94.28	94.28	94.49	94.69
MFPS-2-way-10%-no-APFD-increase	**93.29**	**93.29**	**93.29**	**93.29**	**94.73**	**94.73**	**95.69**	**95.69**	**95.69**	**95.69**
MFPS-2-way-20%-no-APFD-increase	**93.29**	**93.29**	**93.29**	**93.29**	**94.73**	**94.73**	**95.69**	**95.69**	**95.69**	**95.69**

V. CONCLUSION AND FUTURE WORK

This paper extends the single model to hybrid prioritization criteria that combine several criteria that work well individually and evaluate whether the hybrid criteria result in more effective test orders. In future work we have developed the prioritization criteria that improve the rate of fault detection of the test cases over random orderings of tests.

REFERENCES

[1]. A.M. Memon and Q. Xie, "Studying the Fault-Detection Effectiveness of GUI Test Cases for Evolving Software," IEEE Trans. Software 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, July 2005.

[3]. G.D. Lucca, A. Fasolino, F. Faralli, and U.D. Carlini, "Testing Web Applications," Proc. IEEE Int'l Conf. Software Maintenance, pp. 310- 319, Oct. 2002.

[4]. F. Ricca and P. Tonella, "Analysis and Testing of Web Applications," Proc. Int'l Conf. Software Eng., pp. 25-34, May. 2001.

[5]. R.C. Bryce and A.M. Memon, "Test Suite Prioritization by Interaction Coverage," Proc. Workshop Domain-Specific Approaches to Software Test Automation in Conjunction with Sixth Joint Meeting of the European Software Eng. Conf. and ACM SIGSOFT Symp. Foundations of Software Eng., pp. 1-7, Sept. 2007.

[6]. S. Sampath, R. Bryce, G. Viswanath, V. Kandimalla, and A.G. Koru, "Prioritizing User-Session-Based Test Cases for Web Application Testing," Proc. IEEE Int'l Conf. Software Testing, Verification, and Validation, pp. 141-150, Apr. 2008.

[7]. S. Sampath, S. Sprenkle, E. Gibson, L. Pollock, and A.S. Greenwald, "Applying Concept Analysis to User-Session-Based Testing of Web Applications," IEEE Trans. Software Eng., vol. 33, no. 10, pp. 643-658, Oct. 2007.

[8]. S. Sprenkle, L. Pollock, H. Esquivel, B. Hazelwood, and S. Ecott, "Automated Oracle Comparators for Testing Web Applications," Proc. Int'l Symp. Software Reliability Eng., pp. 253-262, Nov. 2007.

P.Srinivasa Reddi received the B.Tech. degree in Information Technology from the Nagarjuna university,Guntur,Andhra pradesh in 2002 and M.Tech. degree in computer science from the university of JNTU Htderabad in 2006. From 2002 to 2003,he worked as software engineer and later joined as a lecturer in Sree Vidyanikethan Engineering College, tirupathi worked from 2003 to 2004.Since 2006,he had been an Assistant professor and then associate professor in Sree Vidyanikethan Engineering College, Tirupathi.His resaerch interests include computer vision and Image processing.He is a life member of ISTE.

CH.Gouthami received the B.Tech Computer Science and Engineering from JNTUK, Kakinada, India in 2009 and pursuing her Master's degree in Software Engineering from the JNTUA, Anantapur, India. Her research areas are Software Engineering, Data warehousing and Data Mining, Database Management Systems and Cloud Computing.