ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology
Volume 1, Issue 6, August 2012

Improving the Performance of a Single Model and Test Prioritization Strategy for Event Driven Software

Polani Sri Gnana Kiran[1], M.R.Rajaramesh[2]
[1]M.Tech, CSE, Sri Vasavi Engineering College, Pedhatadepalli, Tadepalligudem
W.G.Dt., A.P. India.
[2] Associate Professor, Dept of IT, Sri Vasavi Engineering College, Pedhatadepalli, Tadepalligudem.
W.G.Dt. A.P., India.

Abstract— **Event-driven software is very diverse, e.g., in form of Graphical User Interfaces (GUIs), Web applications, or embedded software. All EDS 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) Regardless of the application, the challenges for testing event-driven software are similar. Most event-driven systems allow a huge number of possible event sequences, which makes exhaustive testing infeasible. In this paper, we propose several new test suite prioritization strategies for web-based applications and GUI applications. Further, examine whether these strategies can improve the rate of fault detection for web-based and GUI applications and their preexisting test suites. We prioritize test suites by test lengths, frequency of appearance of request sequences, and systematic coverage of parameter-values and their interactions. This paper extends the single model to generalized prioritization criteria that combine several criteria and result more effective test orders**

Index Terms—**Combinatorial interaction testing, covering arrays, event-driven software (EDS),test suite prioritization, user-session testing, Web application testing, GUI testing.**

I. INTRODUCTION

An EDS takes internal/external events (e.g., commands, messages) as input (e.g., from users, other applications), changes its state, and sometimes outputs an event sequence. Common examples of EDS include graphical user interfaces (GUIs), web applications, network protocols, embedded software, software components, and device drivers.

An EDS is typically implemented as a collection of event handlers designed to respond to individual events. Nowadays, EDS is gaining popularity because of the advantages this "event-handler architecture" offers to both developers and users. From the developer's point of view, the event handlers may be created and maintained fairly independently; hence, complex system may be built using these loosely coupled pieces of code. In interconnected/distributed systems, event handlers may also be distributed, migrated, and updated independently. From the user's point of view, EDS offers many degrees of usage freedom. For example, in GUIs, users may choose to perform a given task by inputting GUI events (mouse clicks, selections, typing in text-fields) in many different ways in terms of their type, number and execution order.

Event-driven software plays an important role in today's software systems. For example, most end-user software has a Graphical User Interface (GUI) that communicates with the user via events, e.g., mouse events. Another example is the Service Oriented Architecture (SOA), where service providers and service consumers communicate with each other using events. Other types of event-driven software include Web applications in general, network protocols, and embedded software. These software types are very diverse and are, therefore, seldom considered together when it comes to quality assurance. From a quality assurance point of view, these systems are quite similar. Therefore, event-driven testing methodologies can be applied to all of them.

In this paper, we 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 strategies. Many of prioritization strategies improve the rate of fault detection of the test cases over random orderings of tests. We also develop Generalized prioritization criteria that combine several criteria that work well individually and evaluate whether the Generalized criteria result in more effective test orders.

II. RELATED WORK

A) *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 test process.

B) *Web Application Testing*

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

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.

Structural Testing: Ricca and Tonella 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.

User-session-based Testing : In user-session-based testing , 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). 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 web applications by collecting data from

users through few configuration changes to the web server.

III. Test Case Prioritization Strategies

To improve the performance of testing model, test case prioritization can be integrated into the event layer. Given (T, π, f), where T is a test suite,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. Prioritization can be based on any criteria. Examples include code coverage, cost estimates, event coverage, and others. In this work, we exploit the characteristics of user-session-based test cases and examine the following functions (or criteria) for web-based applications.

- 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 pairwise 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.

IV. GENERALIZED MODEL

To develop the generalized model, we want to know 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 action Performed message is sent to all action listeners that are registered on the relevant component. To develop the generalized 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. 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.

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

A) Test Lengths

Prioritization of length by base requests selects a next test with the maximum count of base requests, counting duplicates. Since the number of requests in a test case partially determines how much of the application code is exercised by the test case, ordering test cases based on their length can affect the rate of fault detection of the ordered test suite. We order test cases in descending (Req-LtoS) and ascending (Req-StoL) order of length, where length of a test case is defined as the number of base requests the test case contains.

B) Frequency based Prioritization

In this prioritization methodology, we prioritize test cases based on the count of the most-frequently accessed sequences of pages that appear in the test case. Since failures in frequently accessed application components have more impact on user-perceived reliability of the application, we hypothesize that by favoring test cases that exercise frequently accessed application components, the rate of fault detection of the ordered test suite can be improved.

We identify the total number of times that each unique sequence of pages is accessed in the entire test suite and construct a frequency table. We consider sequences in terms of base requests (i.e., ignoring the parameter-value pairs) and only sequences that involve interactions between JSP and Java servlet pages, i.e., we do not include sequences that contain static HTML pages. We assume that faults will exist in the application code and thus consider only sequences between pages that access application code. Using the frequency table to identify the most frequently accessed sequences, we then prioritize the test cases in two ways.

- Most Frequently Accessed Sequence (MFAS). This approach identifies the most frequently accessed request sequence, *si*, in the test suite and orders test cases in decreasing order of the number of times that *si* appears in the test case.
- All Accessed Sequences (AAS). In AAS, the frequency of access of all sequences is used to order the test suite. For each sequence, *si*, in the application, beginning with the most frequently accessed sequence; test cases that have maximum occurrences of these sequences are selected for execution before other test cases in the test suite.

C) Systematic Prioritization by Parameter Values

Web applications contain *pages* that contain *parameters* for which users may specify *values*.

Parameter-value Interaction Coverage. The 2-way criterion selects a next test that maximizes the number of *t*- way parameter-value interactions between pages that occur in a test.

Length by parameter-value counts. During a user session, a user may specify any number of parameter values. We prioritize tests by the number of parameter values in a test case (duplicates included). This includes selecting those tests with the largest number of parameter values in a test first, called PV-LtoS. We also prioritize in the opposite manner by selecting those tests with the smallest number of parameter-values first, called PV-StoL.

D) Random

Prioritization by Random selects a next test at random. In this work, we randomly select test cases for the prioritized test suite until there are no more test cases to select.

V. CONCLUSION

Event-driven software plays an important role in today's software systems. Event-driven software is very diverse, e.g., in form of Graphical User Interfaces (GUIs), Web applications, or embedded software. In this paper, we presented a generalized model for web-based and GUI-based applications of event-driven software systems. This model result more effective test orders. We have developed the prioritization criteria that improve the rate of fault detection of the test cases over random orderings of tests. We prioritize test suites by test lengths, frequency of appearance of request sequences, and systematic coverage of parameter-values and their interactions.

VI REFERENCES

[1] J. A. Jones and M. J. Harrold. Test-suite reduction and prioritization for modified condition / decision coverage. Trans. on Software Engineering, 29(3):195–209, Mar. 2003.

[2] 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.

[3] A. M. Memon, "An event-flow model of GUI-based applications for testing: Research Articles," Software Teststing, Verification and Reliability, vol. 17, no. 3, pp. 137–157, 2007.

[4] S. Elbaum, A. G. Malishevsky, and G. Rothermel. Test case prioritization: A family of empirical studies. IEEE Trans. On Software Engineering, 28(2):159–182, Feb. 2002.

[5] R. C. Bryce and A. M. Memon. Test suite prioritization by interaction coverage. In the Workshop on Domain-Specific Approaches to Software Test Automation, pages 1–7, Sep. 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] D. Jeffrey and N. Gupta, "Test Case Prioritization Using Relevant Slices," *Proc. Int'l Computer Software and Applications Conf.,* pp. 411-418, Sept. 2006.
[9] J. Lee and X. He, "A Methodology for Test Selection," *J. Systems and Software,* vol. 13, no. 3, pp. 177-185, Nov. 1990.
[10] J. Offutt, J. Pan, and J.M. Voas, "Procedures for Reducing the Size of Coverage-Based Test Sets," *Proc. Int'l Conf. Testing Computer Software,* pp. 111-123, June 1995.
[11] 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.
[12] M. Grindal, J. Offutt, and S. Andler, "Combination Testing Strategies: A Survey," *Software Testing, Verification and Reliability,* vol. 15, pp. 167-199, Mar. 2005.
[13] D.R. Kuhn, D.R. Wallace, and A.M. Gallo, "Software Fault Interactions and Implications for Software Testing," *IEEE Trans. Software Eng.,* vol. 30, no. 6, pp. 418-421, Oct. 2004.
[14] C.J. Colbourn, "Combinatorial Aspects of Covering Arrays," *Le Matematiche,* vol. 58, (Catania), pp. 121-167, 2004.

VII. About The Authors

Polani Sri Gnana Kiran is pursuing his M.Tech degree in Computer Science Engineering from Sri Vasavi Engineering College, Tadepallegudem, B.Tech from Computer Science Engineering from Chirala Engineering College, Chirala in 2008. His research interest Software Engineering.

M.R.Raja Ramesh is working as an Associate Professor in the Department of Computer Science Engineering at Sri Vasavi Engineering College, Pedatadepalli, Tadepallegudem. He has Completed M.Tech in Computer Science Engineering from Andhra University, Vishakapatnam, 2004. His MCA pursued in University of Hyderabad, Hyderabad, 2001. He has a total teaching experience (UG and PG) of 11 years. His main research interest is in Software Engineering.