American Journal of Software Engineering, 2015, Vol. 3, No. 1, 1-5
Available online at http://pubs.sciepub.com/ajse/3/1/1

DOI:10.12691/ajse-3-1-1

A Combined Model with Test Prioritizing for Testing an Event Driven Software

Baskaran Periyasamy*, R. Ashok kumar

[1]SNS College of Engineering, Coimbatore
*Corresponding author: baskarcse06@gmail.com

Received December 19, 2014; Revised January 15, 2015; Accepted January 20, 2015

Abstract Event-Driven Software (EDS) can transform the state based on incoming events; common examples are GUI, Web and Embedded applications. These EDSs pose a confront to testing because there are a large number of promising event sequences that users can raise through a user interface. This system provides the single model that is generic enough to study Graphical User Interface (GUI), Web and Embedded applications collectively. It uses the model to describe general prioritization criteria that are appropriate to EDS applications. The ultimate goal is to evolve the model and use it to extend a unified theory of how well all EDS should be tested. The project shows that the GUI, Web-based and Embedded applications, when recast by means of the new model, show related performance. This criterion that gives precedence to all pairs of event contacts did well for GUI, Web and Embedded applications; another condition that gives priority to the minimum number of parameter value settings did weakly for all. In this system by considering the prioritization criteria the order of test cases that are to be executed for the EDS application will be generated. These results emphasize the principle that these three subclasses of applications should be modeled collectively.

Keywords: *Event-Driven Software, GUI, EDS, user interface, test prioritization, parameter, test case, test sequence*

Cite This Article: Baskaran Periyasamy, and R. Ashok kumar, "A Combined Model with Test Prioritizing for Testing an Event Driven Software." *American Journal of Software Engineering*, vol. 3, no. 1 (2015): 1-5. doi: 10.12691/ajse-3-1-1.

1. Introduction

Event-Driven Software (EDS) is a group of software that is rapidly presents in every where. All EDSs obtain sequences of events (e.g., messages and mouse-clicks) as input, change their state, and turn out an output (e.g., events, system calls, and text messages). Examples include Web applications, graphical user interfaces, network protocols, device drivers, and embedded software. The challenge of coming up with a single model of these applications that sufficiently confines their event-driven nature, yet abstracts left elements that are not important for functional testing. The unlucky deficiency of such a model has kept the advancement of imparted testing procedures and calculations that may be utilized to test the classes of uses. It has likewise kept the advancement of an imparted set of measurements that may be utilized to assess the test consequences of these sorts of uses. Second is the inaccessibility of subject applications and devices for scientists.

On focusing the first challenge; i.e., try to develop a single abstract model for GUI, Web and Embedded application testing. To provide focus, restrict the model to extend the previous work on test prioritization techniques for GUI, Web and Embedded applications testing. This allows to adapt the replica to prioritization-specific problems as well as to recast the earlier prioritization criteria in a form that is general enough to influence the single model. In the future, this model can be extended to other testing problems that are shared by EDS applications. Ultimate goal is to generalize the model and to extend a theory of how EDS should be tested. The specific extensions of this work include: the first single model for testing stand-alone GUI, Web-based and Embedded applications, a joint prioritization function based on the abstract model, and shared prioritization criteria. The results show that GUI, Embedded and Web-based applications, when recast by means of the model, showed comparable performance, emphasizing the principle that these modules of applications should be modeled and studied collectively. Other results show that EDS applications perform in a different way, which has twisted opportunities for evolving the model and further experimentation. In future work, further generalize the model by assessing its applicability and helpfulness for other software testing actions, such as test creation. This work also makes extensions to test prioritization study. Many of the prioritization criteria progress the rate of error detection of the test cases over arbitrary orderings of tests. The future model also build up hybrid prioritization criteria that merge numerous criteria that work fine independently and assess whether the hybrid criteria effect in more efficient test orders.

2. Test Prioritization

Because of the client driven nature, GUI, Embedded and Web frameworks routinely experience changes as a component of their upkeep process. New forms of the applications are frequently made as an aftereffect of bug fixes or prerequisites adjustment. In such circumstances, countless cases may be accessible from testing past forms of the application which are frequently reused to test the new form of the application. Because of time obligations, an analyzer should regularly select and execute a subset of these experiments. Experiment prioritization is the procedure of planning the execution of experiments as indicated by some rule to fulfill an execution objective.

3. Related Works

Programming is progressively being created/ kept up by numerous, regularly topographically disseminated engineers working simultaneously. Therefore, fast criticism based quality confirmation systems, for example, every day constructs and smoke relapse tests [1], which help to discover and kill abandons right on time amid programming improvement and support, have ended up critical. Here addresses a significant shortcoming of current smoke relapse testing methods, i.e., their powerlessness to naturally (re)test graphical user interfaces (Guis).

A few commitments are made to the range of GUI smoke testing [1]. Initially, the necessities for GUI smoke testing are recognized and a GUI smoke test is formally characterized as a specific succession of occasions. Second, a GUI smoke relapse testing procedure called Daily Automated Regression Tester (DART) that computerizes GUI smoke testing is introduced. Third, the transaction between a few attributes of GUI smoke test suites including their size, flaw identification capacity, and test prophets is exactly concentrated on. The results demonstrate that: 1) the whole smoke testing procedure is doable regarding execution time, storage room, and manual exertion, 2) smoke tests can't cover certain parts of the application code, 3) having far reaching test prophets may compensate for not having long smoke experiments, and 4) utilizing certain prophets can compensate for not having substantial smoke test suites [1].

An UML model of Web applications is proposed for their abnormal state representation. Such a model is the beginning stage for a few investigations, which can help in the appraisal of the static site structure. Besides, it drives Web application testing, in that it can be misused to characterize white box testing criteria and to semi-naturally create the related experiments.

The proposed procedures were connected to a few certifiable Web applications [2]. Results propose that a programmed backing to the check and approval exercises can be greatly advantageous. Indeed, it promises that all ways in the site which fulfill a chose model are appropriately practiced before conveyance. The abnormal state of robotization that is attained in experiment era and execution builds the quantity of tests that are led and rearranges the relapse checks.

Event driven software (EDS) is a broadly utilized class of programming that takes groupings of occasions as information, changes state, and yields new occasion arrangements. Dealing with the extent of tests suites for EDS is troublesome as the quantity of occasion mixes and arrangements become exponentially with the quantity of occasions [3]. Another testing strategy that develops programming connection testing. Customary programming collaboration testing deliberately analyzes all t-path cooperations of parameters for a project.

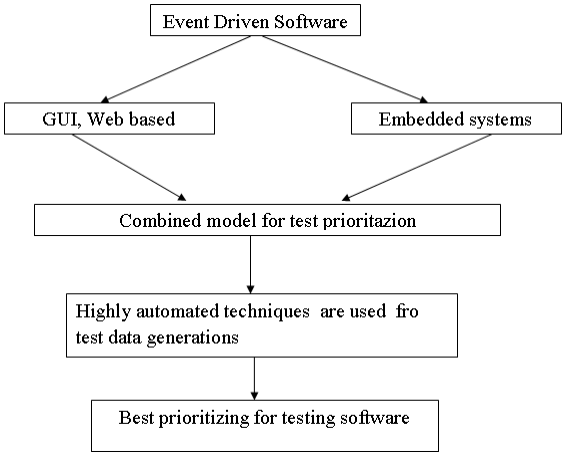
Here, expands the idea to t-path cooperations over successions of occasions. The method applies to numerous classes of programming; that concentrate on that of EDS. As an evidence of-idea, prioritize existing test suites for four GUI based projects by t-way collaboration scope.

By Comparing the rate of fault detection with that of several other prioritization criteria. Results show that prioritization by interaction coverage has the fastest rate of fault detection in half of our experiments, making the most impact when tests have high interface coverage.

Web applications have quickly turned into a basic piece of business for some associations. Be that as it may, expanded utilization of web applications has not been responded with comparing increments in dependability. Special attributes, for example, speedy turnaround time, coupled with developing fame spur the requirement for proficient and viable web application testing methods [4]. A few new test suite prioritization methods for web applications and look at whether these techniques can enhance the rate of issue location for three web applications and their previous test suites. Prioritize test suites by test lengths, recurrence of appearance of appeal groupings, and orderly scope of parameter-qualities and their associations. Trial results demonstrate that the proposed prioritization criteria frequently enhance the rate of flaw location of the test suites when contrasted with arbitrary requesting of experiments. As a rule, the best prioritization measurements either (1) consider recurrence of appearance of arrangements of appeals or (2) deliberately cover mixes of parameter-values as right on time as would be prudent.

The fundamental reason of cooperative testing [5] and examination is that devices and apparatus clients regularly have their individual qualities and shortcomings, and comparably diverse apparatuses ordinarily have their particular qualities and shortcomings; empowering co-operation [10] among these substances can give opportunities to improve their qualities and assuage their shortcomings, separately.

FRRAME WORK FOR EVENT DRIVEN SOFTWARE TESTING



4. Prioritization Criteria

Parameter-Value Interaction Coverage Technique

The 1-way and 2-way parameter value interface coverage techniques, select tests to scientifically cover parameter value interactions between windows.

1-way:

In this way, a next test to maximize the number of parameter values that do not appear in previously selected tests is selected. We can assure that the faster systematic coverage of parameter settings may expose faults earlier. For OrderSuite, we instantiate f(x) to return the set of parameter values in test case x, F(S) to return the set of parameter values accessed by all test cases in sequence S; ⊕ is the function discussed earlier.

2-way:

The 2-way criterion selects a next test to maximize the number of 2-way parameter value interactions between windows. We hypothesize that interactions of parameters set to values on dissimilar windows may render faults. For OrderSuite, we instantiate f(x) to return the set of 2-way parameter value interactions among windows accessed by test case x; F(S) is similar, apart from that it works on the sequence S; ⊕ is the function used earlier.

Input Parameters:

Suite: Test suite to be prioritized (symbolizeed as a set);

f: Function returns criteria essentials of a single test case;

F: Function returns criteria elements in sequence of test cases;

⊕: Operation combines results of f and F; returns number;

Output:

OrderedSequence: Priority controlled sequence containing all tests;

Computation:

S← EMPTY;
T← Suite;
REPEAT
t←BestNextTestCase(*S*,*T*,*f*,*F*,⊕);
S←InsertAtEnd(S,t);
T←T – t;
UNTIL (T= = Ø);
OrderedSequence←S;

Ordersuite Function

Input Parameters:

S:Priority controlled sequence of test cases selected so far;

T:Set of remaining test cases;

f: Function returns criteria elements in a single test;

F: Function returns criteria elements in sequence of tests;

⊕:Operation combines results of *f* and *F*; returns number;

Output:

t: a test case from *T*;

Computation:

Bestnexttestcase Function

Max←*MININT;*
fs←*F(S);*
FORALL *x* ϵ *T* { *y*←*f(x)* ⊕ *fs;*
IF((*Max*<*y*)||(*Max*= =*y*)&&(RANDOM() ≤ 0.5)){
Max←*y;*
t←*x;* } }
RETURN(*t*);

Count-Based Criteria

Another factor essential to test cases for event-driven systems is the inherent enslavement between the variety and number of window artifacts it accesses and the amount of code covered on executing these test cases.

Unique Window Coverage:

Here, we prioritize tests by giving preference to test cases that cover the most unique windows that previous tests have not covered. We hypothesize that faults will be exposed when we visit windows and that we should visit all windows as soon as possible. For OrderSuite, we instantiated f(x) to return the set of windows accessed by test case x; F(S) is similar, except that it operates on the sequence S; ⊕ is the function used earlier.

Action Count-Based:

In this rule, we prioritize tests by the quantity of activities in each one test (copies included). An activity is a grouping that sets one or more parameter values in a solitary window. The prioritization incorporates selecting the experiments, with inclination given to those that incorporate the most number of activities, Action-Ltos. For Ordersuite, we instantiated f(x) to furnish a proportional payback of activities (additionally including copies) experiment x; in light of the fact that this rule does not think about experiments that have as of now been chosen,

F(S)=0; ⊕ returns its first parameter, i.e., the value of f(x). Action-StoL gives priority to test cases with the smallest number of actions. For OrderSuite, f(x)= Negative of the f function used in Action-LtoS.

Parameter-Value Count-Based:

Experiments contain settings for parameters that clients set to particular qualities. We prioritize tests by the quantity of parameters that are situated to values in an experiment (copies included). We theorize that experiments that set more parameters to values are more prone to uncover flaws. This incorporates selecting those tests with the biggest number of parameter quality settings in a test initially, called PV-Ltos.

For OrderSuite, we instantiated f(x) to return the number of parameters that are set to values (also counting duplicates) in test case x; again, F(S)=0 and ⊕ returns its first parameter, i.e., the value of f(x). We also prioritize in the reverse manner by selecting those tests with the smallest number of parameter value settings first, called PV-StoL. Here too, f(x)= Negative of the f function used in PV-LtoS.

Frequency-Based Criteria

The subsequent three criteria differ in how they view the frequency of the occurring of window sequence in a test case, and thus produce different prioritized orders.

Most Frequently Present Sequence (MFPS) of Windows:

In this we have to categorize the most regularly present sequence of windows, si, in the test suite and order test cases in diminishing order of the number of times that si appears in the test case. Then, from among the test cases that do not exercise si even once, the most frequently present sequence, sj, is identified, and the test cases are ordered in diminishing order of the number of times sj appears in the test case.

All Present Sequence (APS) of Windows:

In APS, the frequency of occurrence of all sequences is used to order the test suite. For each sequence, si, in the application, beginning with the most frequently present sequence, test cases that have highest occurrences of these sequences are chosen for execution before other test cases in the test suite. So, that we can find the best sequence of tests for an application.

Weighted Sequence of Windows (Weighted-Freq):

We count the number of times each unique sequence of windows appears. The test case has a impacted value based on the summing up of the product of the amount of times each distinctive sequence of windows emerges in the test case.

Test Suite Prioritization:

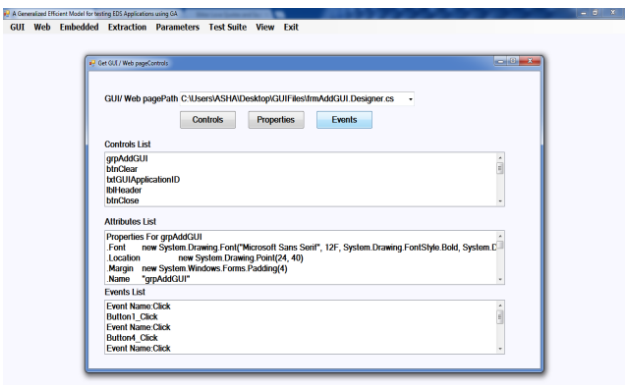
The function for the test selection process is presented as follows. Order Suite takes four parameters:

- The suite to be ordered—note that this is a set.
- A function f that takes a single test case as input and returns a set of elements that are of interest to the criterion being used as the basis for prioritization.
- Another function F (related to f above) operates on the sequence of test cases, S, selected thus far. For the example discussed in the above paragraph, F(S) returns the set of all windows covered by the test cases in sequence S. In this example, F(S) essentially applies the above f to each element in S and takes a set-union of the results.
- An operation assigns a “strength” value to the present test case. For the above example, T is the composed function (SetCardinality 0 SetDifference), i.e., “cardinality of the set difference.” Hence, a test case that covers up the maximum number of unique windows not yet covered by the test cases selected thus far will have the largest value for this function’s output and hence, “most fit”; it will be selected next to be inserted in the ordered sequence. If two or more test cases share the top place for selection, then a arbitrary choice is made using the RANDOM() (returns a random real number between 0 and 1) function in BestNextTestCase.

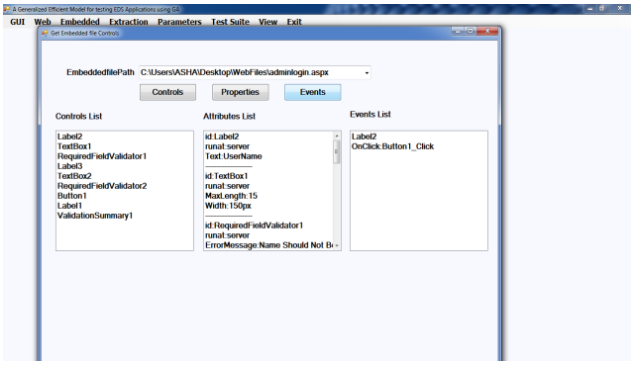
Function OrderSuite begins with an unordered sequence and invokes BestNextTestCase until all of the test cases have been ordered. We will instantiate f, F, and T for each of the prioritization criteria.

The output will be the creation of test cases in a order that are to be executed in an application so that the tester can test the application without problems. This order of test cases shows us the better way of testing an application for competent results.

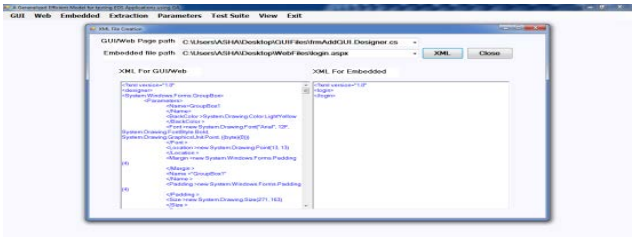
For **GUI/Web Page Controls**



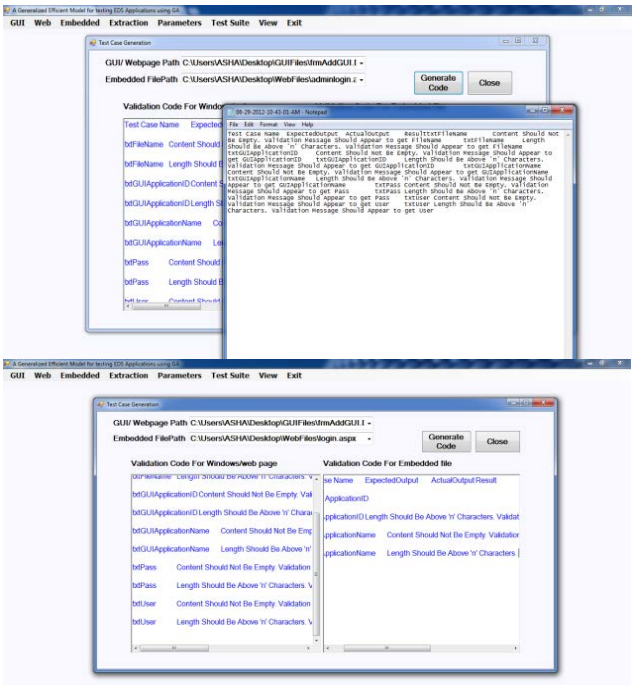
Embedding file controls :



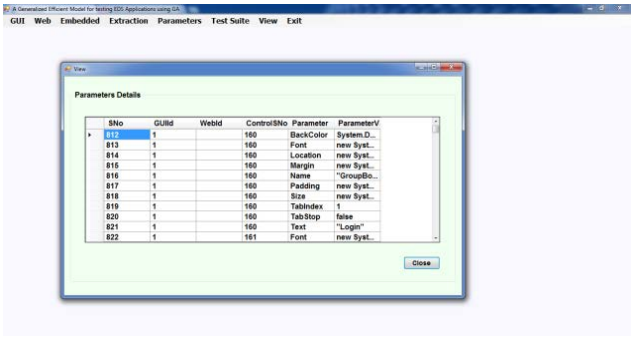
XML for EDS applications :



Testcase Generatioon



To View details of Parameters for GUI, Web & Embedded files



5. Conclusion

EDS applications have many comparisons that allow to create a single model for testing such event-driven

systems. It may support future research to more generally focus on stand-alone GUI, Web-based and Embedded applications as an alternative of addressing them as disjoint topics. Other researchers can use the general model to apply testing techniques more generally. This ability to increase prioritization criteria for three types of event-driven software shows the usefulness of the combined model for the problem of test prioritization.

The first threat is the validation of the unified model. Validate the model through the application of test suite prioritization by using numerous prioritization criteria and three controls applied to seven applications. While work contributes an initial validation of the model, the domains of both testing and EDS are much larger. For instance, broader testing activities such as test generation and test suite reduction can further validate the unified model in the future.

The next major risk to external validity is that running the data collection and test suite prioritization process on seven programs and their existing test suites, which we chose for their availability.

References

[1] A.M. Memon and Q. Xie, “Studying the Fault-Detection Effectiveness of GUI Test Cases for Rapidly Evolving Software,” IEEE Trans. Software Eng., vol. 31, no. 10, pp. 884-896, Oct. 2005.

[2] W. Wang, S. Sampath, Y. Lei, and R. Kacker, “An Interaction-Based Test Sequence Generation Approach for Testing Web Applications,” Proc. IEEE Int’l Conf. High Assurance Systems Eng., pp. 209-218, 2008.

[3] 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.

[4] 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.

[5] Hao, D., Zhang, L., Zhang, L., Rothermel, G., & Mei, H. A Unified Test Case Prioritization Approach.

[6] Renée C. Bryce, Sreedevi Sampath, Atif M. Memon, "Developing a Single Model and Test Prioritization Strategies for Event-Driven Software", IEEE Transactions on Software Engineering, vol.37, no. 1, pp. 48-64, January/February 2011.

[7] Herbold, S.Grabowski, J. ; Waack, S., “A Model for Usage-Based Testing of Event-Driven Software, Secure Software Integration & Reliability Improvement Companion (SSIRI-C), 2011 5th International Conference.

[8] C. Kallepalli and J. Tian, "Measuring and Modeling Usage and Reliability for Statistical Web Testing," IEEE Trans. Softw. Eng., vol. 27, no. 11, pp. 1023-1036, 2001.

[9] 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. (Pubitemid 47354557).

[10] Ashokkumar.R, Baskaran P - A Co-Operative Cluster Based Data Replication Technique for Improving Data Accessibility and Reducing Query Delay in MANET - published at: "International Journal of Scientific and Research Publications (IJSRP), Volume 3, Issue 11, November 2013 Edition".