2011 27th IEEE International Conference on Software Maintenance (ICSM)

# A Tool for Combinatorial-based Prioritization and Reduction of User-Session-Based Test Suites

Sreedevi Sampath
Information Systems
UMBC
Baltimore, MD 21250
sampath@umbc.edu

Renee C. Bryce
Computer Science
Utah State University
Logan, Utah 84341
renee.bryce@usu.edu

Sachin Jain
Information Systems
UMBC
Baltimore, MD 21250
sajain1@umbc.edu

Schuyler Manchester
Computer Science
Utah State University
Logan, Utah 84341
schuyler.manchester@usu.edu

***Abstract*—Test suite prioritization and reduction are two approaches to managing large test suites. They play an important role in regression testing, where a large number of tests accumulate over time from previous versions of the system. Accumulation of tests is exacerbated in user-session-based testing of web applications, where field usage data is continually logged and converted into test cases. This paper presents a tool that allows testers to easily collect, prioritize, and reduce user-session-based test cases. Our tool provides four contributions: (1) guidance to users on how to configure their web server to log important usage information, (2) automated parsing of web logs into XML formatted test cases that can be used by test replay tools, (3) automated prioritization of test cases by length-based and combinatorial-based criteria, and (4) automated reduction of test cases by combinatorial coverage.**

## I. Introduction

In test prioritization, the goal is to order test cases such that tests locate faults early in the test execution cycle. Test prioritization is important because a large number of test cases can accumulate over several life cycles of software systems. Running all the test cases is expensive, and therefore, prioritization methods are proposed that order the test cases based on certain criteria with the goal of detecting faults quickly. Several program coverage-based and similarity-based test prioritization criteria exist for traditional systems.

While several empirically validated strategies are proposed for test prioritization, tools and frameworks that enable easy adaptation of the test prioritization strategies are limited. In a recent survey on regression test selection techniques, Yoo and Harman [1] observe that there is a need for easily available testing tools that implement prioritization techniques. In this paper, we focus on enabling easy access to test prioritization methods for web applications. In particular, we target one important area of web application testing called user-session-based testing. In user-session-based testing, usage logs collected during web system deployment are converted into test cases [2]. Strategies to convert the web usage log into test cases have been previously proposed [3], [4], [5]. We have studied and evaluated the effectiveness of black-box test prioritization criteria for web-based systems [6], [7]. We include the most effective prioritization criteria from this prior work in the tool.

Our tool, **C**ombinatorial-based **P**rioritization for **U**ser-Session-Based **T**esting (CPUT), makes three significant contributions beyond the state of the art:

1) **A new logger for Apache**: The logger is a primary component of the framework. It enables capture of usage accesses, such as HTTP GET and POST requests and associated data. Prior tools for user-session-based testing [3], [4] developed a logger for the web server, Resin, which has a smaller user population. Our logger is designed to capture HTTP requests and associated data for the Apache web server, the most popular open-source web server.
2) **Conversion of web usage logs into a new, extensible XML test case format**: Previously proposed techniques for user-session-based testing [3], [4] use test cases that are in the format of a sequence of HTTP requests. In our framework, we create tests in an XML format. An XML format allows for easy extensibility as needed by organizations with custom usage data. Also, replay tools can be written using publicly available APIs that parse XML data.
3) **Prioritization criteria**: Our prior work has empirically shown that new black-box test prioritization of length-based and combinatorial-based criteria create effective test orders [7]. Prior frameworks and tools have not implemented these criteria for web-based systems.

Section II presents an overview of CPUT, including details of each component and an example that walks through how to use each component. Section III presents the conclusions.

## II. CPUT: Test Prioritization Framework

This paper presents our tool, CPUT: **C**ombinatorial-based **P**rioritization of **U**ser-session-based **T**estsuites. The main components are the **logger**, **test creation engine**, and the **test prioritization engine**. The remainder of this section describes the issues and our design choices for each of these components.

### A. Logger for Apache

One of the fundamental components to enable user-session-based testing is a web logger. Some important characteristics of the logger are that it should (a) be able to log all relevant pieces of data, (b) introduce minimum performance overhead, and (c) be easy to deploy and integrate with the rest of the web server. With these characteristics in mind, we developed the logger component for CPUT.

978-1-4577-0664-6/11/$26.00 ©2011 IEEE

We select Apache as the web server for which we implement a logger, since it commands 59.13% of the market share [8]. Programming for Apache is implemented by abstracting server operations into separate modules. The modules allow server administrators to keep only the modules that are relevant to their operations and remove the unneeded modules. Apache provides a rich API to develop these modules and modules allow easy addition of functionality to Apache without the need for patches. Given these advantages of modules, we implemented the logger for Apache as a module.

Our logger module is designed to be generic so it can be deployed on Apache running on Windows or Linux platforms. The module is written in C and enables the logging of HTTP GET and POST requests. HTTP GET requests are logged by default in most web server systems. With HTTP POST requests however, form data is transmitted as part of the HTTP request body instead of being appended to the URL in the request header. Additional methods are required to gather the data associated with an HTTP POST request. We implemented this feature to enable POST logging into the logger module. To deploy the module, a server administrator places the module in the same directory as other Apache modules and enables it in Apache's configuration file. We designed the module such that the administrator can either choose to enable or disable logging after loading the module. The administrator can also specify the path and file name to the log file.

The data format is a sequence of HTTP requests as received by the web server. Similar to previous work [9], the web server records the time stamp, IP address, the URL, any associated parameter-values, the referrer URL and cookie information. The log is written to a text file and subsequently stored in a PostGreSQL database and fed into the test case creation engine that creates the test cases in XML format.

### B. XML-format Test case creation engine

When creating user-session-based test cases from a usage log the following characteristics play an important role. Web application logs can be very large depending on the usage of the web system. Therefore, the web log and test cases should be processed such that they can be efficiently stored and retrieved in the future. The format of the test cases should be extensible and generic so that replay tools can process and execute the test cases efficiently.

To handle large usage logs and a large number of test cases, we made the design decision to store the web log and the test cases in PostGreSQL database tables. Storing the logs and test cases in a database allows for efficient storage and retrieval. The test cases are initially stored in a database table and eventually printed out to file in an XML format.

When storing the usage log into a database table, the user has three options—create new table, append to existing table, or overwrite existing table. The *create new* option is useful when a user has a new web log that they want parsed into test cases. The *append* option is useful when the user has logs from two different days wherein the log from the first day has already been parsed, and the user would like to augment the test cases from the first log with the usage log generated on the second day. The *overwrite* option is useful when a user wants to overwrite or erase a previously existing database table containing the usage log from a previous run.

The usage log is first stored in a database table and then parsed and converted into test cases. Traditionally, test cases for web systems are created in the form of a sequence of HTTP requests [3], [4]. While this format is convenient for replay with tools like the Unix utility `wget` which easily operate on HTTP requests, the format cannot be easily extended. For example, when representing a test case as a sequence of URLs, the data is usually appended to the end of the base URL, which is the notation for HTTP GET requests. HTTP POST requests cannot be represented using the sequence of URLs notation, because the parameter-values associated with the URL should be included as part of the request body and not the request itself. With an XML format, the POST data can be represented using the XML notation, and can then be parsed in the future during replay. We address this problem by using an XML notation for our test cases. An XML format has several advantages, primarily the test case format can be standardized and extended. Extensibility is a rather important characteristic, because, with the rapidly changing nature of web systems, new features may need to be incorporated into the tests to capture all relevant information. Further, the XML format test cases can easily be parsed by using the publicly available XML parsers and processed by replay tools.

The test case creation engine of CPUT uses previously proposed heuristics [3] to convert a usage log into test cases. Namely, the cookie information, the IP address, and the time stamp of each request is used to classify a request as belonging to a test case [3], [4].

The test cases are stored in an XML format. The important tags in this format include: *testSuite* to denote the test suite, *session id* represents the unique ID of a test case within the test suite, *url* represents a page that the test case accesses and has an associated request with a *request type* of GET or POST. Within a request, a *baseurl* includes the specific page that is accessed and parameters (denoted as *param*) which have *name* and *value* that is assigned to the parameter.

### C. Test prioritization engine

The main goal of this tool is to make test prioritization accessible to researchers and practitioners. Our tool supports applications with unique URLs (i.e., JSP is one example where the URL changes, such as login.jsp, browse.jsp, payment.jsp) and non unique-URLs (i.e., PHP is one example where the URL may remain the same despite content changes on the page, such as using index.php as the base URL but the contents of the page change based on parameters passed to functions in the code).

In CPUT, we include experimentally verified prioritization methods that have shown to be effective in a large study on event-driven software [7]. Our prior empirical studies showed that 2way and the length-based criteria have high effectiveness across all the subject applications, therefore, we implement them in CPUT. A user can choose one of four prioritization criteria to order their test suite. All the prioritization criteria in

| Log file | Component name | Component output | Execution time | Output Space requirements |
|---|---|---|---|---|
| 15 days | Test case creation engine | XML format tests | 42.3s | 2.3 MB (173 tests) |
| | Test prioritization (Length) | Order file | 0.000s | 2.2 KB |
| | Test prioritization (Number of parameters) | Order file | 0.015s | 2.2 KB |
| | Test prioritization (2way) | Order file | 3.401s | 2.2 KB |
| | Test reduction (2way) | Order file | 3.401s | 1.75 KB (138 tests) |

TABLE I
TIME AND SPACE RESULTS FOR COMPONENTS OF CPUT

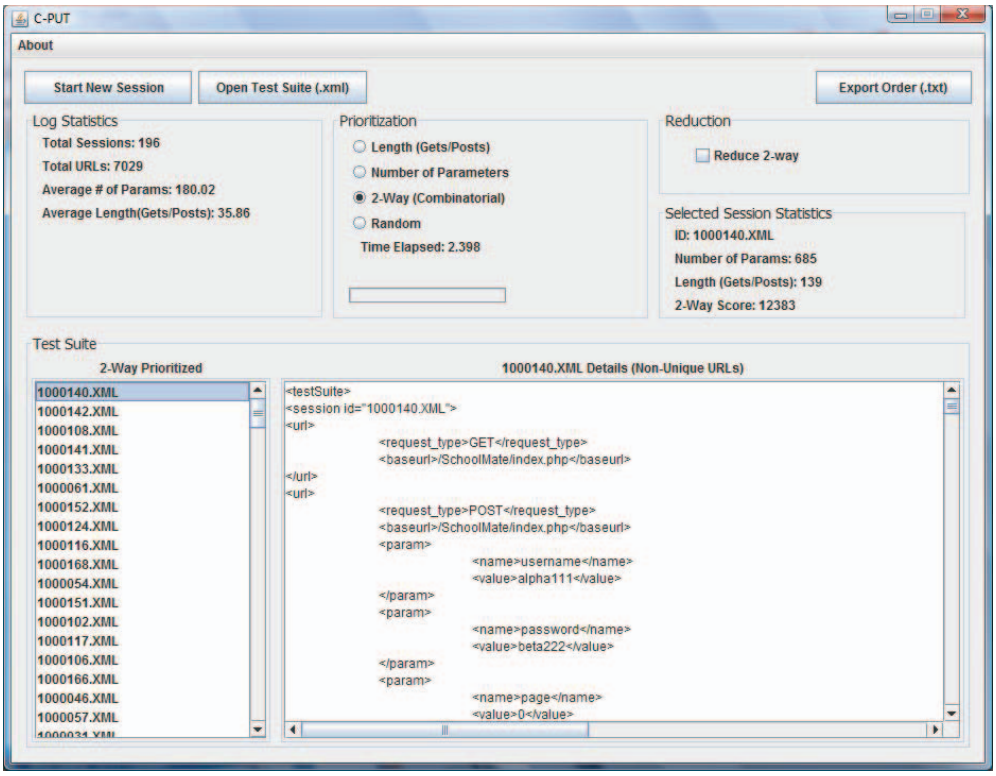


Fig. 1. Overview of main screen of CPUT

CPUT are implemented in Java. CPUT allows users to select a test prioritization criterion and export to file the order of the tests generated by the test criterion. The four criteria that are included in this version of CPUT are:

**1. Length (Gets/Posts):** order test cases in descending order of the number of GET/POST requests in each test case.

**2. Number of parameters:** order test cases in descending order of the number of parameters that are assigned values in each test case. A parameter is set to a value either by the user or by the web application itself, especially in the case of hidden form values. We record both types of parameters.

**3. 2way combinatorial:** order test cases in descending order of the number of unique 2-way parameter-value interactions (pairs) between pages in each test case. A pair is a combination of two parameter-values that occur on different pages. Once a pair is covered in a test, we mark the pair as "covered" and only count unique pairs that have not been covered in previously selected tests. (Due to space limitations, we refer the reader to [7] for a detailed example that walks through the computation.)

**4. Random:** prioritize using the random function in Java. The tool will produce a different random ordering each time that the user chooses to prioritize at random. This option is only recommended for use as a baseline in empirical studies.

### *D. Reduction*

Another test selection methodology that is often used in the regression testing phase is test suite reduction. We provide a preliminary reduction technique in CPUT. In CPUT, we provide the functionality to reduce a test suite by 2way combinatorial coverage of inter-window interactions. The reduction technique removes tests that do not provide additional coverage of uncovered 2way interactions. The reduced test suite can also be exported to file.

### *E. Example walk though*

Figure 1 shows the main screen of CPUT. This is the window the user sees when they launch CPUT.

**Step 1: Create a new test suite or open an existing one** The user should select one of the two buttons at the top of the window to "Start New Session" or to "Open Test Suite

(.xml)". If the users start a new session, this allows them to open a web log that they wish to parse into an XML formatted test suite. Alternatively, if they open an existing test suite, this allows them to open an XML formatted test suite, including one that they may have created during a previous session of using this tool.

**Step 2: Observe the statistics about the test suite** "Log Statistics" about the test suite are provided on the upper left corner of the screen.These include: (a) Total Sessions/test cases (b) Total URLs (c) Average number of parameters in the test cases that are within the test suite, (d) Average length of test cases in test suite in terms of GET/POST requests.

**Step 3: Prioritize the test suite** The user has the option to choose any of the four aforementioned criteria to prioritize their test suite. This will result in a modification of the ordering of test cases shown on the left side of the screen. The user can click directly on any of the test case names in that left pane in order to view the content of a test case.

**Step 4: Reduce the test suite** The user may check the box to reduce the test suite using the 2way combinatorial-based criteria. If the user unchecks this box, the test suite reverts to the original full test suite.

**Step 5: Review selected session statistics** Once a user has prioritized a test suite, the individual sessions (test cases) are listed in the "Sessions" frame. Users may select individual sessions to view the following statistics: (a) Session ID: this is the name of the test case, (b) Number of Parameters: this is the number of parameters that are set to values in the test case, (c) Length (Gets/Posts): this is the number of GET/POST requests that a user made during their session (test case), (d) 2-way Score: this is the number of previously uncovered 2-way interactions in the test case.

### *F. Example of execution time and memory usage of the example used in the demo*

To provide readers with insight into the efficiency of using CPUT, we present a brief example that uses a web application called Schoolmate. Schoolmate was used by Artzi et al. [10] in their work on finding bugs in dynamic web applications. Schoolmate is a PHP/MySQL web application that is designed for elementary, middle and high schools [11]. It contains 63 PHP files with 8,120 lines of code. We hosted the web application on an Apache web server. Prior to hosting the application, we installed our logger as a module in the Apache installation and started logging the user sessions.

Students from a software testing course accessed Schoolmate daily for 15 days. Each student received a user id and password to log in as a teacher, student, and parent. We also had two admin accounts that the two faculty members used to login to the system and explore the system's features. We collected 173 test cases. Table I shows the execution time and output space for creating, prioritizing, and reducing this test suite. From Table I, we see that the different components of CPUT take a few seconds to complete their operation.

## III. Conclusion and Future Work

CPUT is a novel tool that allows testers to create and manage user-session-based test suites. No tool currently exists that provides test prioritization of user-session-based test cases. CPUT combines a logger for Apache that logs all relevant data for web systems and allows the conversion of the log file into test cases that can be used to create, append, or overwrite a test suite. CPUT then allows a tester to select a test prioritization criterion and generates a test order that the tester can then replay against the web application. Users may also reduce a test suite by 2way combinatorial coverage. Through a small example, we show the working of CPUT and data on the execution time and memory costs of the tool. The tool will not only enable practitioners to quickly create, prioritize, and reduce test suites in practice, but it will also help researchers that want to build upon existing work in this area.

## Acknowledgment

CPUT is available upon request to the authors as our sponsor would like to track the types of users (industry or research) and the volume of users as one measure of impact. This work is supported by National Institute of Standards and Technology, Information and Technology Lab Award number 70NANB10H048. Any opinions, findings and conclusions expressed herein are the authors' and do not reflect those of the sponsors. We also thank undergraduate researcher, Devin Minson, for his contributions to CPUT.

## References

[1] S. Yoo and M. Harman, "Regression testing minimisation, selection and prioritisation : A survey," *Journal of Software Testing, Verification and Reliability*, p. to appear, 2011.

[2] S. Elbaum, S. Karre, and G. Rothermel, "Improving web application testing with user session data," in *the Intl. Conference on Software Engineering*. Portland, Oregon: IEEE Computer Society, May 2003, pp. 49–59.

[3] S. Sampath, S. Sprenkle, E. Gibson, L. Pollock, and A. S. Greenwald, "Applying concept analysis to user-session-based testing of web applications," *IEEE Transactions on Software Engineering*, vol. 33, no. 10, pp. 643–658, Oct. 2007.

[4] S. Sprenkle, E. Gibson, S. Sampath, and L. Pollock, "Automated replay and failure detection for web applications," in *The Intl. Conference of Automated Software Engineering*. Long Beach, CA, USA: ACM, Nov. 2005, pp. 253–262.

[5] S. Sprenkle, E. Gibson, S. Sampath, and L. Pollock, "A case study of automatically creating test suites from web application field data," in *the Workshop on Testing, Analyis and Verification of Web Services and Applications*. ACM Press, July 2006, pp. 1–9.

[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 Intl. Conference on Software Testing, Verification and Validation*. Lillehammer, Norway: IEEE Computer Society, Apr. 2008, pp. 141–150.

[7] R. Bryce, S. Sampath, and A. Memon, "Developing a single model and test prioritization strategies for event-driven software," *Transactions on Software Engineering*, vol. 37, no. 1, pp. 48–64, Jan. 2011.

[8] "January 2011 web server survey," http://news.netcraft.com/archives/2011/01/12/january-2011-web-server-survey-4.html, accessed on Apr. 5, 2011.

[9] S. Sampath, V. Mihaylov, A. Souter, and L. Pollock, "Composing a framework to automate testing of operational web-based software," in *the Intl. Conference on Software Maintenance*. IEEE Computer Society, Sep. 2004, pp. 104–113.

[10] S. Artzi, A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D. Ernst, "Finding bugs in dynamic web applications," in *ISSTA '08: Proceedings of the 2008 Intl. Symposium on Software testing and analysis*. Seattle, WA, USA: ACM, Jul. 2008, pp. 261–272.

[11] "Schoolmate," <http://sourceforge.net/projects/schoolmate>, 2011.