CREU 2011 Final Report
Toward an Effective Data Model and User Session Dependency Model

Anna Pobletts, Camille Cobb, Lucy Simko
poblettsa12@mail.wlu.edu, cobbc12@mail.wlu.edu, simkol11@mail.wlu.edu

5/1/11

1 Introduction

Web applications are becoming increasingly common, and people are becoming more and more dependent on these applications to accomplish tasks such as managing money and buying goods. It it therefore imperative that web applications work properly and consistently, which means *they must be thoroughly tested*; however, testing web applications is difficult and expensive.

One approach to making the testing of web applications cheaper and easier is to automate the testing process. Although this approach is promising, current automated testing methods are not efficient or accurate enough. Another approach is user session based testing. Figure 1 shows how user session based testing records actual user accesses to older versions of the application and parses them into user sessions, which are then used as test cases. User session based testing is inexpensive and creates test cases that are representative of actual users.

We based our approach on work by Sant et al. [3], who have done work in user-representative automated test case generation. They proposed generating test cases using a model of user sessions that requires less space than the original user sessions. The

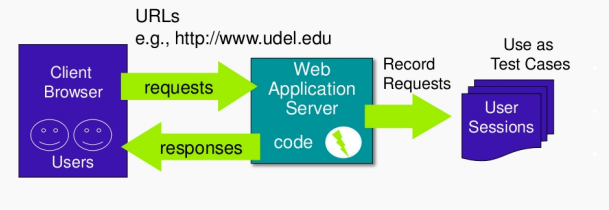

Figure 1: User Session Based Testing.

model has two parts: a *control model*, which we call the *navigation model*, that represents a user's navigation through a web application as a sequence of URL requests and a *data model* that represents the user's parameter values associated with these requests.

Our research focuses on maintaining the benefits of user session based testing while improving on the current limitations of an inadequate data model and a test suite that does not account for dependencies. Our goals are to create test cases that are 1) effective in terms of failure detection and code coverage 2)representative of users and 3) cost effective to generate. Our research group proposed modularizing control models and data models and explored control models in depth in previous work [4].

In section 3, we address the problem of determining the best data model for a parameter, which will

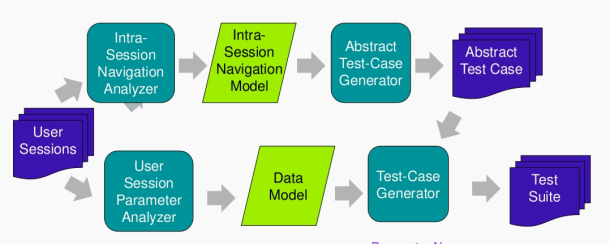

Figure 2: Test Case Generation Process.

not necessarily be the same for all parameters. In previous work, we have identified a number of factors that influence the data model. They are used in conjunction with the parameters to generate the data model. We focus on *improving the data model* by customizing the factors used for a specific parameter.

In section 4, we examine the generated test suites and how their order can affect the effectiveness of the test cases. Since dependencies between sessions can affect the validity of future session, we attempt to automatically estimate *user session dependencies* to arrange the test suite.

We present the motivation for examining these issues, our approach to solving them, and preliminary observations that suggests that these are promising areas of research and our approach is worth pursuing. The main contributions of this paper are:

- a sampling of factors that may affect how parameters appear in web applications.
- an approach to automatically identify and estimate test case dependencies in web applications

2 Background: Test Case Generation Process

Figure 2 shows an overview of the test-case generation process. The test-case generation process begins with logs of user interactions with a web application, then builds navigation and data models, which generate test cases for the web application. Broadly defined, a web application is a set of web pages and components that form a system in which user input (navigation and data input) affect the system's state. Users interact with a web application using a browser, making requests over a network using HTTP. When a user's browser transmits an HTTP request to a web application, the application produces an appropriate response, typically an HTML document that the browser displays. The response can be either static, in which case the content is the same for all users, or dynamic such that its content may depend on user input or application state.

Before the test-case generation process shown in Figure 2 begins, the user accesses are parsed and segmented to create users sessions. Each *user session* is a sequence of user requests in the from of base requests and name-value pairs. We say a user session begins when a request from a new Internet Protocol(IP) address arrives at the server and ends when the user leaves the web site or the session times out. We consider a 45 minute gap between two requests from a user to be equivalent to a session timing out.

From a set of user sessions and a navigation model specification, the *intra-session navigation analyzer* constructs an intra-session navigation model. The *abstract test-case generator* uses the navigation model and template criteria to produce a set of test-case templates.

Meanwhile, the user sessions are also analyzed by the user session analyzer to create an intra-session data model. From here, the test case templates and the data model are used by the test case generator to output a set of test case—the test suite. The generator makes these test cases by assigning the values for parameters within each template. These values are determined by the data models.

In previous work, a navigation model was created

that looked at a URL's resources with ordered parameter names [4]. This was determined to be a useful model because it provided more coverage information than only the URL's resources by itself. The other main contributions of previous work were that after analyzing the test case *templates* (made from the navigation model), they proposed a practical way to use test case templates. This allow a tester to a) more easily tune parameters to make sure the resulting test suite can meet the URL-based guarantees with lower costs b) reduce the size of template suites(which reduce redundancy) c) apply multiple data models to a set of high URL+name coverage test case templates.

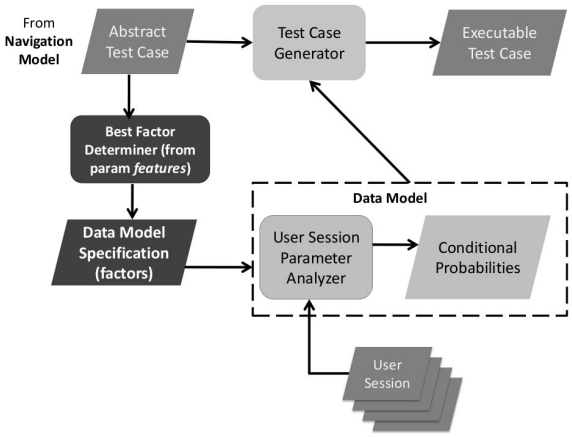

Figure 3: Framework for analyzing effects of factors.

3 Exploring Potential Data Models

3.1 Rationale

Our current approach of user session based testing involve automatically generating test suites in two steps: First, the tester generates a representation of the users' paths through the application as a sequence of abstract URLs containing the resource name and parameter names, and then she adds parameter values into the sequence using a data model. Previous work has found that there is a large set of *factors* that influence the parameter values generated by the data model. These factors include the parameter name, the current and previous resources, and other current and previous parameter names.

We have observed that not every factor in our set of potential factors has influence on every parameter value. Our insight is to categorize parameters and find correlation between the types of parameters and the factors that positively affect them. Our goal in doing this is to *educe the number of factors we need to consider when picking a parameter value in order to produce the most effective and condensed data model.*

3.2 Approach

Our approach is to add pre-processing step, which is the **Best Factor Determiner** in Figure 3. To determine the most relevant factors for any given parameter, we first assume every factor is relevent for every parameter. By analyzing the logged user sessions, we gather statistics on the use of the each parameter name-value combination with every individual factor. For each parameter value-factor pair, we determine whether the difference between the probabiliy of the value occuring given the current parameter and the factor and the probabilty of the value occuring given just the current parameter name is significantly high; our threshold was a 1% difference. If the difference is significant, we consider the factor important. Otherwise, we may discard the factor.

In a second step, we attempt to classify the factors used for parameter values with certain qualities. We classify based on *features*, which are characteristics of parameter values inherent to their use in and by the application and the user. There are three main types of qualities that features describe: (1) dependency

relationships, (2) content value, and (3) source, destination, or intended/possible use of the value. A list of features and possible values (in brackets) appears below:

- Source of value [user, database, navigation]
- Written to database [yes, no]
- Verified [not checked, checked against database, checked against rule]
- Required [yes, no]
- Relation to user [independent, entered now, entered previously]
- Value type [string, number]
- HTML type [text area, file, selection box, hard-coded by applicaton, hidden, etc.]
- Number of unique values [number of values that only occur once]
- Average length of value [average number of characters in value]
- Number of distinct values [Number of different values]
- Depends on history [not all previous requests, user history]
- Depends on other parameters in the current request [yes, no]

For a given factor and a feature, we determine whether the factor is significant for most ($> 50\%$) of an application's parameters with that feature. For example, in Figure 3, we look at the effect of the factor resource name on parameters whose values come from the user, parameters whose values come from the application, and parameters whose values come from the database. Note that when a parameter only occurs with one factor (i.e., one resource), we discard the parameter from our statistics, as the parameter must always occur with that resource. Currently, parameters must be manually categorized, which is a limitation and an avenue of future work. However, when automatic parameter classification according to our feature set is possible, the scalability of our solution will allow an effective data model to be generated very quickly.

3.3 Preliminary Results

In a preliminary analysis of the factor resource name and the feature source, we found that not all parameters required the factor. (See Figure 4 for our findings.) We performed our analysis over five web applications: bookstore [2], CPM, DSpace [1], Logic, and Masplas. The largest difference made by the factor resource name varies wildly between applications, with bookstore at 66.67% and Masplas at 1.46%. All applications show a significant number of parameters occurring with only one resource, and these have been discounted from all of the following analyses.

Note the set of columns that represent the parameter-factor differences over the threshold. In most applications, the factor resource mattered at least 60% of the time if the parameter value was entered by the user or came from the database. However, parameter values determined by the application were only significantly tied to the resource in only one of the applications (bookstore).

3.4 Future Work

In the future, we will examine relationships between all features and all factors. We will also strive to automatically classify parameters, as that is by far the most tedious and therefore error-prone part of the

				Below Threshold source				At or Above Threshold source			
Application	Largest diff	Mean diff	Occurring with one resource	overall	user	application	database	overall	user	application	database
Bookstore	66.67%	6.69%	13 (39.39%)	2 (10%)	0	0	2 (40%)	18 (90%)	**11 (100%)**	**4 (100%)**	**3 (60%)**
CPM	14.72%	2.17%	13 (28.89%)	18 (56.25%)	15 (60%)	2 (66.67%)	1 (25%)	14 (43.75%)	10 (40%)	1 (33.33%)	**3 (75%)**
Dspace	71.5%	4.16%	724 (73.5%)	155 (59.39%)	13 (16.05%)	115 (87.97%)	27 (55.1%)	106 (40.61%)	**68 (83.95)%**	16 (12.21%)	22 (44.9%)
Logic	13.9%	3.73%	60 (82.19%)	3 (23.07%)	2 (28.57%)	1 (50%)	0	10 (76.92%)	**5 (71.43%)**	**1 (50%)**	**4 (100%)**
Masplas	1.46%	1.39%	36 (90%)	0	0	0	0	4 (100%)	**3 (100%)**	0	**1 (100%)**

Figure 4: Sample data for resource name.

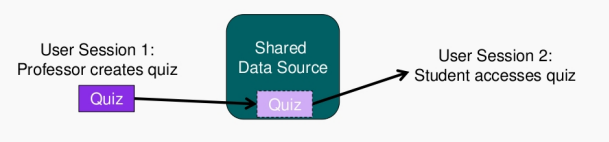

Figure 5: Example of a dependency between user sessions.

process.

Further work will also be done with conditional probabilities between parameter values. We believe that if we know one parameter value, we may be able to better predict what other parameter values in the URL will be. We hope that by looking into this idea, we can improve the effectiveness of the test cases.

4 Toward a User Session Dependency Model

4.1 Rationale

A limitation of the current test case generation approach is that the order of user sessions within a test suite can negatively impact the test suite's ability to expose faults. That is, there are dependencies among some user sessions that determine the validity of later user sessions. For example, on an online tutorial Web application, professors must create problems before students can solve them, i.e., the problems must be in the shared data store to be accessed, as shown in Figure 5. The intra-session navigation model and the data model only consider intra-session dependencies and flow, but the problems occur with inter-session dependencies.

We propose creating a user session dependency model that allows us to order the user sessions within a test suite. Since Web application code is highly decoupled, we cannot use static analysis to determine these dependencies. One approach is to base the model on the user access log: each session is dependent upon all previous sessions. However, this model may overstate the dependencies, and a finer-grained model is more appropriate.

4.2 Approach

Rather than considering user sessions, we look at the dependence between specific requests within and among user sessions. Our insight is that action words like submit, create, view in the resource name may indicate dependencies. For example, create, register and upload imply writes to the shared data store. View, login, and download imply reads from the shared data store. We also believe we can use patterns in how these known dependencies appear in user sessions to suggest unknown dependencies.

The first step of the process for determining these dependencies is to manually examine the code,

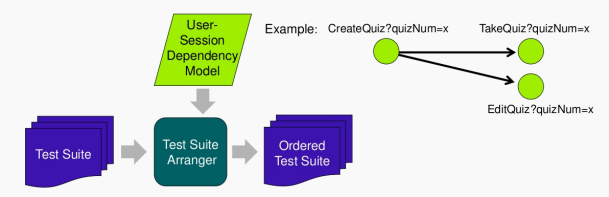

Figure 6: The test suite arranger and user session dependency model.

URLs, and user sessions for a set of dependencies to train on. Since we have had extensive experience with our test applications and helped develop them, this is not a difficult step. Then, we look for action words in the resource name of known dependencies that imply reads or writes. We have found that most of the manually found dependencies can be revealed using clues from these action words. Next, we mine user sessions to find patterns in how they appear in the URLs of our test cases. Implementation and evaluation of a user session dependency model and test suite arranger are part of our future work.

4.3 Observations and Future Work

We have found that action words reveal many of the dependencies between URLs in our applications, and did not reveal any false dependencies in our applications. We believe that this will also be the case for other applications.

Although we have found some patterns for these urls in our collected user session data, we will do more extensive data mining for patterns in our future work. We are also considering mining source code to reveal patterns in how these URLs occur in source code. Given these patterns, we hope to reveal unknown dependencies in our applications. This will allow us to create a dependency model that automatically arranges the final test cases within a test suite so that dependencies are satisfied, as shown in Figure 6. We believe that this will create better test cases. We will evaluate the revealed dependencies for accuracy, and we will evaluate the final generated test suites in terms of code coverage and fault exposure.

5 Conclusion

In this paper, we present some limitations of an approach to automatically testing Web applications and introduce some ideas for improving upon it. We present a preliminary approach to categorizing parameters and data that shows that there is correlation between certain factors, the type of parameter, and the effects on the parameter value. We also draw attention to the problem of an incorrectly ordered test suite—that dependencies between URLs can negatively affect fault exposure. We present a method of automatically estimating these dependencies and using them to arrange the test suite. In future work, we will fine-tune these approaches, incorporate them into the test case generation process, and evaluate their effectiveness.

References

[1] DSpace Federation. `http://www.dspace.org/`, 2010.

[2] Open source web applications with source code. `http://www.gotocode.com`, 2003.

[3] Jessica Sant, Amie Souter, and Lloyd Greenwald. An exploration of statistical models of automated test case generation. In *International Workshop on Dynamic Analysis*, May 2005.

[4] Sara Sprenkle, Lori Pollock, and Lucy Simko. A study of usage-based navigation models and generated abstract test cases for web applications. In *International Conference on Soft-*

ware Testing, Verification and Validation (ICST).
IEEE, March 2011.