Improving Web Application Testing with User Session Data

Sebastian Elbaum[†], Srikanth Karre[†], Gregg Rothermel[‡],

[†]Department of Computer Science and Engineering
University of Nebraska - Lincoln
Lincoln, Nebraska
{elbaum,skarre}@cse.unl.edu

[‡]Department of Computer Science
Oregon State University
Corvallis, Oregon
grother@cs.orst.edu

Abstract

Web applications have become critical components of the global information infrastructure, and it is important that they be validated to ensure their reliability. Therefore, many techniques and tools for validating web applications have been created. Only a few of these techniques, however, have addressed problems of testing the functionality of web applications, and those that do have not fully considered the unique attributes of web applications. In this paper we explore the notion that user session data gathered as users operate web applications can be successfully employed in the testing of those applications, particularly as those applications evolve and experience different usage profiles. We report results of an experiment comparing new and existing test generation techniques for web applications, assessing both the adequacy of the generated tests and their ability to detect faults on a point-of-sale web application. Our results show that user session data can produce test suites as effective overall as those produced by existing white-box techniques, but at less expense. Moreover, the classes of faults detected differ somewhat across approaches, suggesting that the techniques may be complimentary.

1. Introduction

Web applications are among the fastest growing classes of software systems in use today. These applications are being used to support a wide range of important activities: business functions such as product sale and distribution, scientific activities such as information sharing and proposal review, and medical activities such as expert-system based diagnoses. Given the importance of such applications, faulty web applications can have far-ranging consequences on businesses, economies, scientific progress, and health. It is important that web applications be reliable, and to help with this, they should be validated.

To address this problem, many types of web application validation techniques have been proposed and many tools have been created. Most of these techniques and tools, however, focus on aspects of validation such as protocol conformance, load testing, broken link detection, and various static analyses, and do not directly address validation of functional requirements. Those tools that do focus on functional requirements primarily provide infrastructure to support capture-replay: recording tester input sequences for use in testing and regression testing.

Recently, a few more formal approaches for testing the functional requirements of web applications have been proposed [11, 17]. In essence, these are "white-box" approaches, building system models from inspection of code, identifying test requirements from those models, and requiring extensive human participation in the generation of test cases to fulfill those requirements. The approaches have shown promise in early empirical studies in terms of support for constructing "adequate" (by some criterion) test suites. However, the approaches also have drawbacks, in part due to differences between web applications and systems developed and operated under more traditional paradigms.

Among these differences, we consider three in particular. First, the usage of web applications can change rapidly. For example, a web site with a popular name that is caught by a search engine can suddenly find itself receiving hundreds of thousands of hits per day rather than just dozens [12]. In such cases, test suites designed with particular user profiles in mind may turn out to be inappropriate.

Second, web applications typically undergo maintenance at a faster rate than other software systems; this maintenance often consists of small incremental changes [9]. To accommodate such changes, testing approaches must be automatable and test suites must be adaptable.

Finally, web applications typically involve complex, multi-tiered, heterogeneous architectures including web servers, application servers, database servers, and clients

acting as interpreters. Testing approaches must be able to handle the various components in this architecture.

Although some recently proposed techniques for testing the functional requirements of web applications [11, 17] partially address this third difference, the first two differences have not yet been addressed. Thus, in this paper, we propose a testing approach that utilizes data captured in user sessions to create test cases. We describe two stand-alone implementations of this approach, and a hybrid implementation that combines the approach with techniques such as the existing white-box approaches just described. We report results of an empirical study comparing these implementations of our approach with two different implementations of an existing approach recently proposed by Ricca and Tonella [17]. Unlike previous studies of web application testing techniques, however, our study assesses the fault detection effectiveness of the approaches.

In the next section we briefly describe the characteristics of the class of web applications that we are considering, and then we review related work on testing these applications. Section 3 describes Ricca and Tonella's technique in greater detail, and then describes our new approach and its implementations. Section 4 presents the design and results of our empirical study of these techniques, and Section 5 summarizes and comments on future work.

2. Background and Related Work

2.1. Web applications

A web site can be differentiated from a web application based on the "ability of a user to affect the state of the business logic on the server" [4]. In other words, requests made of a web application go beyond navigational requests, including some form of data that needs further decomposition and analysis to be served.

Figure 1 shows how a simple web application operates. A user (client) sends a request through a web browser. The web server responds by delivering content to the client. This content generally takes the form of some markup language (e.g., HTML) that is later interpreted by the browser to render a web page at the user site. For example, if a request consists of just a URL (Uniform Resource Locator – a web site address), the server may just fetch a static web page.

Other requests are more complicated and require further infrastructure. For example, in an e-commerce site, a request might include not just a URL, but also data provided by the user. Users provide data primarily through forms consisting of input fields (textboxes, checkboxes, selection lists) rendered in a web page. This information is translated into a set of name-value pairs (input fields' names and their values) and becomes part of the request.

Although the web server receives this request, further elements are needed to process it. First, a group of scripts and

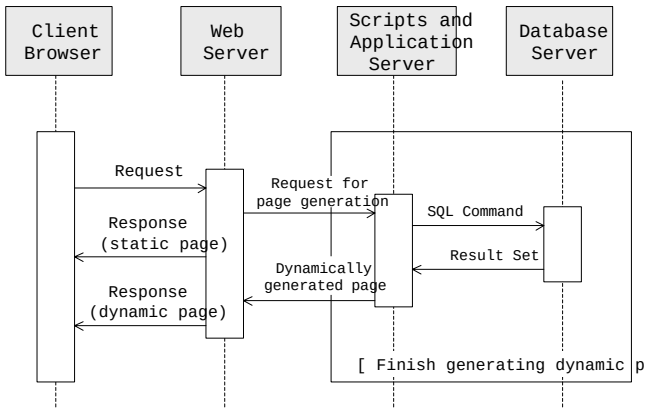

Figure 1. Sequence diagram of a web application.

perhaps an application server may parse the request, query a database server to retrieve information about the item, and then employ further formatting scripts to generate HTML code on the fly to address the user request. This newly generated page, generated at run time and depending on user's input, is called a dynamic web page.

In this context, the application server, database, and scripts collaborate to assemble a response that fits the request. Although in practice requests can be more complex, this example illustrates that there are multiple and varied technical components behind the web server. It is also important to note that scripts such as those just referred to are changed frequently [13], and the technologies supporting them change often, as evident in the frequent appearance of new standards for web protocols (e.g., XML, XSL, SOAP and CCS [10]).

2.2. Related work

The testing of web applications has been led by industry, whose techniques have been oriented primarily toward validation of non-functional requirements. This is evident in the number and variety of existing tools available for the relatively new web application domain. These tools range from markup text language validators and link checkers to various load testing and performance measurement tools.[1]

The variety and quantity of tools for testing functional requirements of web applications, on the other hand, is much more limited. The most common class of functional testing tools provide infrastructure to support the capture and replay of particular user scenarios [16, 18]. Testers execute possible user scenarios, and the tools record events and translate them into a series of scripts that can be replayed later for functional and regression testing. Other classes

[1] A comprehensive list of these tools is available at http://www.softwareqatest.com/qatweb1.html.

of functional testing tools generate test cases by combining some type of web site path exploration algorithm with tester provided inputs [13, 15]. A prototype framework integrating these various features is presented in [19].

Recently, two more formal techniques have been proposed to facilitate testing of functional requirements in web applications. Both techniques employ forms of model-based testing, but can be classified as "white-box" techniques, since they rely on information gathered from the web application code to generate the models on which they base their testing. Liu et al. [11] propose WebTestModel, which considers each web application component as an object and generates test cases based on data flow between those objects. Ricca and Tonella [17] propose a model based on the Unified Modeling Language (UML), to enable web application evolution analysis and test case generation. Both these techniques, in essence, extend traditional path-based test generation and data flow adequacy assessment to the web application domain; the second also builds on the existence of popular UML modelling capabilities.

It is worth noting that the effectiveness of these techniques has been evaluated only in terms of ability to achieve coverage adequacy. We find no reports to date of studies assessing fault detection capabilities of the techniques.

3. Web-Application Testing Techniques

In this section we provide further details about the particular techniques that we have investigated, including existing approaches and user-session based approaches. We first describe our implementations of Ricca and Tonella's approach. This approach is representative of the white-box techniques described in Section 2; it has also shown success in terms of coverage adequacy, and has been presented in detail sufficient to allow its implementation, given a few assumptions. Next, we introduce two new techniques for use in testing functional requirements of web applications. Our techniques are based on data captured in user sessions, and they exploit the availability of user inputs for automatic test case generation. Last, we present one possible integration of these two approaches. Table 1 summarizes the techniques.

3.1. Ricca and Tonella's approach

Conceptually, Ricca and Tonella's [17] approach creates a model in which nodes represent web objects (web pages, forms, frames), and edges represent relationships and interactions among the objects (include, submit, split, link).

For example, Figure 2 shows a model of a component of an application for on-line book purchasing, following the graphical representation used in [17]. The diagram starts at the BookDetail node. This node is dynamically generated in response to a request to browse a particular book.

Label	Description	Type
WB-1	Simplest Ricca and Tonella implementation [17]	White box
WB-2	WB-1 with boundary values	White box
US-1	Replay captured user sessions	User-session
US-2	Combine interactions from different user sessions	User-session
HYB	Insert user session values into WB-1	Hybrid

Table 1. Web application testing techniques.

When rendered by the browser, this page contains information about the book and also includes (through edges e1 and e4) two forms: one to add the book to the shopping cart and one to rate the book. Both forms collect user input. If the rating form is submitted (e5), a new BookDetail page is generated with the updated rating for the book. If a valid book quantity is submitted (e2), the shopping cart is updated and a corresponding dynamically generated page is sent to the browser. Otherwise, BookDetail is sent again (e3).

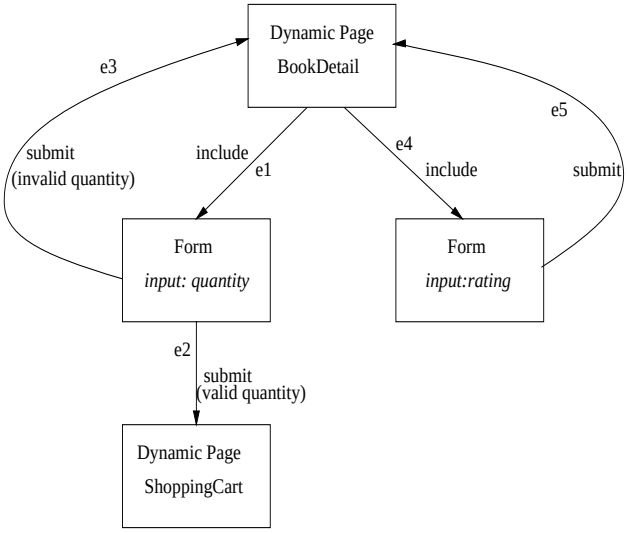

Figure 2. UML model of an e-commerce application.

To generate test requirements and cases, a path expression to match the graph is generated following a procedure similar to the one introduced by Beizer [2]. The path expression corresponding to the example from Figure 2 is $(e1e3)*+(e4e5)*$, where $*$ indicates zero or more occurrences of the immediately preceding edge(s) and $+$ indicates an alternative.

The path expression is used to generate quasi test cases in combination with heuristics to minimize the number of test cases required to satisfy a given criterion. For exam-

ple, from the path expression for Figure 2, we can derive the sample quasi test cases: $e1e3$, $e1e2$, and $e4e5$. A (human) tester then provides input values so that the tests can be executed. In this approach, then, a test case consists of a sequence of web pages to be visited, together with their corresponding name-value pairs.

We consider two implementations of this approach. Our first implementation, WB-1, attempts to match the methodology presented in [17]. Test cases are generated from the path expression following the suggested procedure, but we were forced to make some assumptions about the implementation where details were not available. Specifically: (1) we tested only independent paths; (2) for search fields that could generate many different pages, we tested using only one input value; and (3) we ignored circular links and textual differences in dynamically generated pages. Once the quasi test cases were generated from the path expression, we filled the forms so that they could be executed.

Our second implementation, WB-2, relaxes some of the assumptions established for WB-1 and incorporates a more elaborate approach for input value selection. In contrast to WB-1, WB-2 uses boundary values as inputs, and utilizes an "each condition/all conditions" strategy to combine them [3]. The test suite consists of test cases in which each variable is set to true once with all the other variables set to false, plus one test case in which all variables are true. The objective of using boundary values and the "each condition/all conditions" strategy was to add formalism to the process of inputting data into the forms, as recommended in one of the examples in [17]. In addition, this technique considers not only differences in links, forms, and frames, but also textual differences when evaluating dynamically generated pages.

3.2. User-session based techniques

One limiting factor in the use of white box web application testing techniques such as Ricca and Tonella's is the cost of finding inputs that exercise the system as desired. Selection of such inputs is slow and must be accomplished manually [17]. User-session based techniques can help with this problem by transparently collecting user interactions and transforming them into test cases. The techniques capture and store the clients' requests in the form of URLs and name-value pairs, and then apply strategies to these to generate test cases.

Because normal web application operation consists of receiving and processing requests, and because a web application runs in just one environment which the organization performing the testing controls, the collection of client request information can be accomplished easily. For example, with minimal configuration changes, the Apache web server can log all received requests [1]. Another slightly more powerful but less transparent alternative that can capture all name-value pairs consists of adding snippets of javascript to the delivered webpages so that all requests invoke a server-side logging script.

As a consequence, user-session based techniques do not require additional infrastructure to collect this data, limiting the impact on web application performance. This is equivalent to having a built-in instrumentation mechanism, an approach well suited to web applications. Another advantage of collecting just the requests is that at that higher abstraction level, some of the complexities introduced by heterogeneous web application architectures are hidden. This lessens the dependencies of user-session based techniques on changes in web application components.

Given the collected URL and name-value pairs, there are many ways in which test cases could be generated. The simplest approach is to sequentially replay individual user sessions. A second approach is to replay a mixture of interactions from several users. A third approach is to replay sessions in parallel so that requests are handled concurrently. A fourth approach is to mix regular user requests with requests that are likely to be problematic (e.g., navigating backward and forward while submitting a form).

A complicating factor for these approaches involves *web application state*. When a specific user request is made of a web application, the outcome of that request may depend on factors not completely captured in URL and name-value pairs alone; for example, an airline reservation request may function differently depending on the pool of available seats. Further, the ability to execute subsequent tests may depend on the system state achieved by preceding tests. The simplest approach of replaying user sessions in their entirety is not affected by application state, provided that initial system state is known and can be instantiated. The use of more complex approaches such as intermixed or parallel replay, however, might often be affected by state.

In such cases, one approach for using user-session data is to periodically take snapshots of the state values (or of a subset of those values) that potentially affect web application response. Associating such snapshots with specific requests, or sequences of requests, increases the likelihood of being able to reproduce portions of user sessions, at the cost of resources and infrastructure.

A second alternative is to ignore state when generating test cases. The resulting test cases may not precisely reproduce the user activity on which they are based, but they may still usefully distribute testing effort relative to one aspect of the users' operational profile (the aspect captured by the operation) in a manner not achieved by white-box testing. From this perspective, the process of using user session data to generate test cases is related to the notion of partitioning the input domain of an application under test in the hopes of being able to effectively sample from the resulting partitions [21]. In this context, the potential usefulness of

user-session based testing techniques, like the potential usefulness of white-box testing techniques, need not rest solely on being able to exactly reproduce a particular user session. Rather, that usefulness may reside in using user session data to provide effective partitioning heuristics, together with input data that can be transformed into test cases related to the resulting partitions.

The approaches that we have described for generating test data from user sessions and for addressing the problem of application state each have potential costs and benefits that must be explored. In this paper, we focus on two specific user-session based techniques — a technique that applies entire sessions, and a technique that replays a mixture of sessions — each without incorporating information on state. These techniques are relatively simple, and if they prove effective this would motivate further research on more complex techniques, and further exploration of the tradeoffs among techniques.

Our first technique, US-1, transforms each individual user session into a test case. Given m user sessions, $U_1, U_2, \ldots U_m$, with user session U_i consisting of n requests $r_1, r_2, \ldots r_n$, where each r_i consists of $url[name-value]*$, the test case corresponding to U_i is generated by formatting each of the requests, from r_1 to r_n, into an *http* request that can be sent to a web server. The resulting test suite contains m test cases, one for each user session. (For simplicity, we define a user session 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.)

Our second user-session based technique, US-2, generates new user sessions based on the pool of collected data, creating test cases that contain requests belonging to different users. US-2 is meant to expose error conditions caused by the use of sometimes conflicting data provided by different users. US-2 generates a test case as follows:

- randomly select unused session U_a from session pool;
- copy requests r_1 through r_i, where i is a random number greater than 1 but smaller than n, into the test case;
- randomly select session U_b, where $b \neq a$, and search for any r_j with the same URL as r_i, and if an equivalent request is not found, select another session U_b;
- add all the requests from U_a after r_j to the test case;
- mark U_a "used", and repeat the process until no more unused sessions are available in the pool.

In a sense, US-1 is analogous to a constrained version of a capture-replay tool (e.g, Rational Robot [16]) in which we capture just the URL and name-value pairs that occur throughout a session. In contrast to approaches that capture user events at the client site, however, which can become complicated as the number of users grow, our approach captures just the URL and name-value pairs that are the result of a sequence of the user's events, captured at the server site. This alleviates some of the privacy problems introduced by the more intensive instrumentation used by some capture-replay tools.

Both US-1 and US-2 also have several other potential advantages. First, by utilizing user requests as the base for generating test cases, the techniques are less dependent on the complex and fast changing technology underlying web applications, which is one of the major limitations of white box approaches designed to work with a subset of the available protocols. Second, the level of effort involved in capturing URL and name-value pairs is relatively small as these are already processed by web applications. This is not the case with white box approaches such as Ricca and Tonella's, which require a high degree of tester participation. Third, with these approaches, each user is a potential tester: this implies potential for an economy of scale in which additional users provide more inputs for use in test generation. The potential power of the techniques resides in the number and representativeness of the URL and name-value pairs collected, and the possibility of their use in generating a more powerful test suite (an advantage that must be balanced, however, against the cost of gathering the associated user-session data). Finally, both approaches, unlike traditional capture and replay approaches, automatically capture authentic user interactions for use in deriving test cases, as opposed to interactions created by testers.

We are not proposing, however, to rely solely on users to assess web application quality. Our approach is meant to be applied either in the beta testing phase to generate a baseline test suite based on interactions with friendly customers, or during subsequent maintenance to enhance a test suite that was originally generated by a more traditional method. Further, the approach can help testers monitor and improve test suite quality as the web application evolves, and as its usage proceeds beyond the bounds anticipated in earlier releases and earlier testing.

3.3. Integrating user-session and white box methods

The strengths of structured white-box web application testing techniques such as Ricca and Tonella's might be more successfully leveraged if they can make use of inputs collected during user sessions. This suggests a hybrid testing technique combining techniques presented in the preceding sections with white-box techniques. Such a technique could decrease the cost of input selection, but it remains to be seen whether a user session's value pairs will be effective at detecting faults.

We wished to investigate this question and thus we also implemented a hybrid technique. Conceptually, the integration process matches equivalent user-session sequences with the paths defined by Ricca and Tonella's quasi test cases (quasi because they do not provide values, just target URL paths). Given a quasi test case qt generated by an

WB implementation, qt is translated into a URL sequence $urlSeq_{qt}$. Then, the technique iterates through all collected user sessions to identify those that contain the entire $urlSeq_{qt}$. Once the relevant user sessions have been identified, the name-value pairs corresponding to the matching requests in each session are used to complete the $urlSeq_{qt}$, transforming qt into an executable test case. The process continues until all possible test cases have been generated by combining each WB quasi test case with all user sessions with an equivalent sequence.

4. Empirical Study

4.1. Research questions

RQ1. The first question we address is, how effective are the WB techniques at exposing faults in a web application? Evaluating the effectiveness of the WB techniques will help us assess them, and also provide a baseline against which to compare user-session based techniques.

RQ2. Our second research question concerns the cost-effectiveness of user-session based techniques. We conjecture that test suites generated through these techniques provide a useful combination of effectiveness and efficiency.

RQ3. Last, we want to know what relationship exists between the number of user sessions and the effectiveness of the test suites generated based on those sessions' interactions. Answering this question will help us explore the cost-effectiveness of user-session based test suite generation.

4.2. Variables and metrics

The testing techniques listed in Table 1 constitute our independent variables, and are represented by the labels WB-1, WB-2, US-1, US-2, and HYB.

The dependent variables we wish to capture are coverage and fault detection effectiveness. We use two coverage metrics: percentage of functions covered and percentage of blocks covered in the perl code that generated the dynamic web pages and accessed the databases. Percentage of faults detected constitutes our other measure of effectiveness.

4.3. Experimental setting

4.3.1 E-commerce site

The free and open source on-line bookstore available at gotocode.com constituted the skeleton of our e-commerce application. The e-bookstore functionalities are divided into two groups: customer activities and administration activities. This study concentrates on functionalities that are accessible to the customer, not the administration component. Figure 3 provides a screenshot of the application. Customers can search, browse, register, operate a shopping cart, and purchase books on-line through this site, which operates like other similar popular sites on the web.

Customer functionality is implemented through Perl scripts and modules to handle data and the dynamic generation of HTML pages, Mysql to manage database accesses, a database structure composed of seven tables tracking books, transactions, and other data objects, and Javascript and cookies to provide identification and personalization functionality. An Apache Web Server hosted the application.

We populated the database with information from 100 books (e.g., title, authors, short description, category, price, rating). We adapted the look of the site so that the registration procedure was expedited and logins minimized, which made the navigation process more similar to commercial sites. Last, to capture the information required by the user-session techniques, we modified the scripts generating dynamic web pages. As a result, the generated web pages included additional Javascript code to capture user events and to invoke a logging server-side script.

4.3.2 Fault seeding

We wished to evaluate the performance of web testing techniques with respect to the detection of faults. Such faults were not available with our subject application; thus, to obtain them, we followed a procedure similar to one defined and employed in previous studies of testing techniques [6, 8, 22]. We recruited two graduate students of computer science, each with at least two years of programming experience, and instructed them to insert faults that were as realistic as possible based on their experience. To direct their efforts we gave them a random generator that indicated the approximate location of where a fault was to be seeded, and gave them the following list of fault types to consider:[2]

- Scripting faults. This includes faults associated with variables, such as definitions, deletions, or changes in values, and faults associated with control flow, such as addition of new blocks, redefinitions of execution conditions, removal of blocks, changes in execution order, and addition or removal of function calls.
- Web page faults. This includes addition, deletion, or modification of name-value pairs. Such changes occur in one or more names or values.
- Database query faults. This type of fault consists of the modification of a query expression, which could affect the type of operation, the table to access, fields within the table, or the values of search keys or record values.

[2] These directives are adapted from the fault classification in [14].

Figure 3. **Screenshot of e-commerce site used in our experiment.**

The graduate students assigned to the task seeded a total of 50 faults. Nine faults were discarded: two were in unused sections of code and the rest had no impact on the application (e.g., the value of a variable was changed after its last usage).

4.3.3 Assignment for participants in study

Once the web application was set up, the challenge was to have users access the site and behave like "typical" users of this type of e-commerce site. Users navigate sites like ours to browse and perhaps purchase books if the material and price are appropriate for their needs and budget. We wished to provide the context and incentive for users to interact with our application under similar circumstances.

To achieve these goals, we instructed our study participants to follow a three step process. First, they completed an on-line form with simple demographic data. This step was required only for the first session of each user. Second, the description of four computer science courses was made available to the participants; the participants needed to select two of those courses. The final step required the participants to access the e-bookstore site to select the most appropriate book(s) for the courses they selected.

To select the most appropriate books, participants had to search and browse until they found those that they considered most appropriate. We provided no definition of appropriateness, so it meant different things to different users, which is supposed to lead to a variety of activities. However, we did provide an incentive so that users took the task completion seriously. The instructions indicated that on completion of the experiment, the five users who selected the most appropriate books for the courses would each receive a ten dollar gift certificate. Further directions specified that, if more than five users selected the most appropriate books, the amount spent would be evaluated, and ties would be broken by considering the time spent on the web site. Again, the objective was to recreate the conditions observed for similar web applications.

4.4. Execution and processing

The US-1, US-2, and HYB techniques required user participation to generate test suites. A list of candidate participants was assembled, containing primarily students from the Department of Computer Science and Engineering at UNL. An email stating participation requirements and incentives was sent to the students on the list. The e-commerce site was then made available to the users for two weeks. Data collected from the user sessions was logged to support subsequent test case generation, which occurred off-line. There were 99 user sessions logged. Fourteen percent of the sessions could not be used because, despite instructions, some users accessed the web site using browsers

Metric	WB-1		WB-2		US-1		US-2		HYB	
	abs	%	abs	%	abs	%	abs	%	abs	%
Test Suite Size	28	–	64	–	85	–	84	–	1089	–
Block Coverage	263	66	306	76	263	66	255	64	260	65
Function Coverage	65	97	66	99	65	97	64	96	65	97
Faults Detected	22	51	25	58	23	53	23	53	23	53

Table 2. Summary data on technique effectiveness.

other than those required. In total, 73 distinct users contributed to the 85 useful sessions. The users had an average age of 24 and 94 percent had on-line buying experience.

After the test suites corresponding to each technique were generated, we ran them on the fault-free web application, which served as an oracle. Then, to measure function and block coverage, the application was instrumented so that a counter would be incremented each time one of those entities was executed. The e-bookstore had a total of 67 functions and 400 blocks. Measuring coverage required a second execution of each test suite.

Last, we evaluated the test suites generated through each technique by activating the seeded faults individually, and determining which faults were revealed by which test cases. Since we had a total of 43 faults, this last step required an equal number of runs of each test suite.

4.5. Results

4.5.1 On technique effectiveness

The results of executing the test suites generated by each of the techniques are presented in Table 2. These results correspond to RQ1 and RQ2, as defined in Section 4.1, and are presented in absolute values and percentages. Overall, WB techniques used the fewest test cases, while the HYB approach employed the greatest number (1089). WB-2 provided the greatest fault detection power with 58% of the faults detected, and the greatest coverage with 76% and 99% of the blocks and functions covered, respectively. Both user-session based techniques performed similarly, not as effectively as WB-2, but slightly better than WB-1. US-2 did not discover more faults than US-1 and the HYB approach did not provide additional fault detection or coverage. This summary, however, fails to reveal some interesting facts about the differences between techniques.

First, although the overall results for fault detection are similar, differences between the approaches are more evident when individual faults are analyzed. A detailed comparison of the most powerful white box and user session based techniques is presented in Table 3. The first row of the table lists the blocks covered, functions covered, and faults detected by WB-2 and US-1. The second and third rows present the same information but focusing on blocks,

Technique Combination	Blocks		Functions		Faults	
	abs	%	abs	%	abs	%
$(WB2 \cap US1)$	255	64	65	97	20	47
$(WB2 - US1)$	51	13	1	1	5	12
$(US1 - WB2)$	8	2	0	0	3	7
$(WB2 \cup US1)$	314	79	66	99	28	65

Table 3. Detailed comparison of WB-2 and US-1.

functions, or faults uniquely associated with one technique and not the other. The fourth row shows the result of executing the tests generated by both techniques.

It is evident from the table that the blocks covered by WB-2 and US-1 are not the same: 8 blocks were covered only by US-1, while 51 others were covered only by WB-2. A similar observation can be made about functions covered. Also, 3 of the faults found by US-1 were not found by WB-2, and 5 faults were found only by WB-2. Similar differences occurred with the other techniques. This supports the hypothesis that the distinct approaches we considered are able to find different types of faults. Further, in spite of the unimpressive performance of HYB as we implemented it, the last row row of Table 3 shows that when the test cases of both techniques (representing both approaches) are combined, the resulting coverage and fault detection capabilities are better than for any individual technique.

Second, there are some type of faults that user-session based approaches could rarely capture. Those faults have to do with particular name-value pairs that could not be generated using the forms available through the delivered web pages. For example, when doing a book evaluation in the application, five levels (from 1 to 5 stars) are available through the web site. The only way to construct a request with evaluation values outside that range is to generate the request outside the rendered page. This scenario is rarely originated by regular users, but it could have appeared if the site were detected by search robots that perform that type of request. Again, additional exposure time could have helped accomplish this, but it is more likely that this type of request would be generated by design instead of by user activity. In addition, although we could not identify a certain type of faults that WB techniques are likely to miss, we observed

that strategies such as that followed by WB-2 helped lessen the impact of tester input choices on the effectiveness of WB techniques.

4.5.2 On user-sessions versus effectiveness

A detailed analysis of our data revealed that certain techniques could have performed better given some adjustments. WB-2 could have discovered one more fault if the "each condition/all condition" strategy was replaced by a more powerful strategy such as "all variants" [3]. Still, such an approach would require significant additional human participation, and lead to scalability problems.

US-1, on the other hand, could have detected five additional faults, increasing its fault detection effectiveness to 68% (better than the other techniques) if additional exposure was provided through more user sessions. Two of these additional faults were not discovered by WB-2, and required a special combination of input values to be exposed. For example, one fault could have been exposed if a user had attempted to access the shopping cart prior to completing the registration procedure. The three other faults that could have been discovered by US-1, and that were captured by WB-2, required erroneous inputs that did not appear in the collected user sessions. For example, the registration procedure required a password and a confirmation for that password. A fault was exposed when these inputs did not match. WB-2 caught that fault but US-1 did not since no user session exhibited that behavior.

In this study, the effectiveness of user-session based techniques improved as the number of collected sessions increased. Figure 4 shows the relationship between the number of user sessions collected for and employed by US-1, and its effectiveness under the various metrics considered. The x-axis presents the individual sessions from 1 to 85 (corresponding one to one with the test cases generated by US-1), the bars represent the US-1 test case value for the corresponding y-variable, and the line represents the cumulative count of unique faults exposed or blocks and functions covered, as additional sessions exercised the application.

The figure shows that test cases generated from user sessions varied in terms of coverage and fault exposure, and that the cumulative increases in the dependent variables were slower over time. Within the ranges observed, a positive trend in the cumulative counts suggests that user-session based techniques might continue to improve as additional sessions (and potentially additional user behaviors) are collected; however, this trend might also be tempered by the costs of collecting and analyzing additional data, and as the number of detectable faults remaining in the system is reduced. The costs of experimentation constrained our ability to collect further sessions to investigage this; further studies on commercial sites with thousands of sessions per day would provide the opportunity to further analyze such trends.

Using larger numbers of sessions, however, introduces at least one additional challenge in addition to the cost of collecting and managing sessions: the oracle problem [20]. That is, what is the expected output in response to a request, and how do we perform an effective comparison of results to expected results? In a regression testing context, where a web application evolves and produces new versions, this problem can be partially addressed by the procedure we employed in our experiment, where the outputs of one version constitute a baseline against which posterior versions' outputs are compared. This is similar to the solutions provided by capture-replay tools, but at a larger scale.

Still, this scenario presumes that the correctness of results of executing test cases on a baseline can be determined, that initial states can be collected and reused in test execution, and that the execution of thousands of generated test cases can be made affordable. One approach that might help with error detection involves techniques for automatically detecting anomalous behaviors among large sets of executions [5]; these techniques might allow large numbers of generated test cases to be retained usefully.

A second approach involves using existing techniques to reduce test suite size while maintaining coverage by removing redundant test cases from the suite. We briefly explored this second approach, adapting the test suite reduction technique of Harrold et al. [7] and applying it to the test cases generated with US-1. When applied at a functional level, test suite reduction could have reduced test suite size by 98%, with a loss of only three faults (20 were detected). Test suite reduction on block level information reduced the test suite size by 93%, detecting one fewer fault than the complete US-1 original test suite. These results suggest that combining reduction techniques with user-session techniques could be helpful for handling the large number of requests found in commercial e-commerce sites. As defined, however, test suite reduction techniques function only on complete data sets. New techniques will have to be developed to incrementally handle the collection and processing of data as it arrives.

4.6. Threats to validity

This study, like any other, has some limitations. In this section we identify the primary limitations that could have influenced the results and we explain how we tried to control unwanted sources of variation. Some of these limitations are unavoidable consequences of the decision to use controlled experimentation; however, the advantage of control is more certain knowledge of causality.

First, we needed to set up infrastructure to reproduce a web application that resembled as closely as possible those

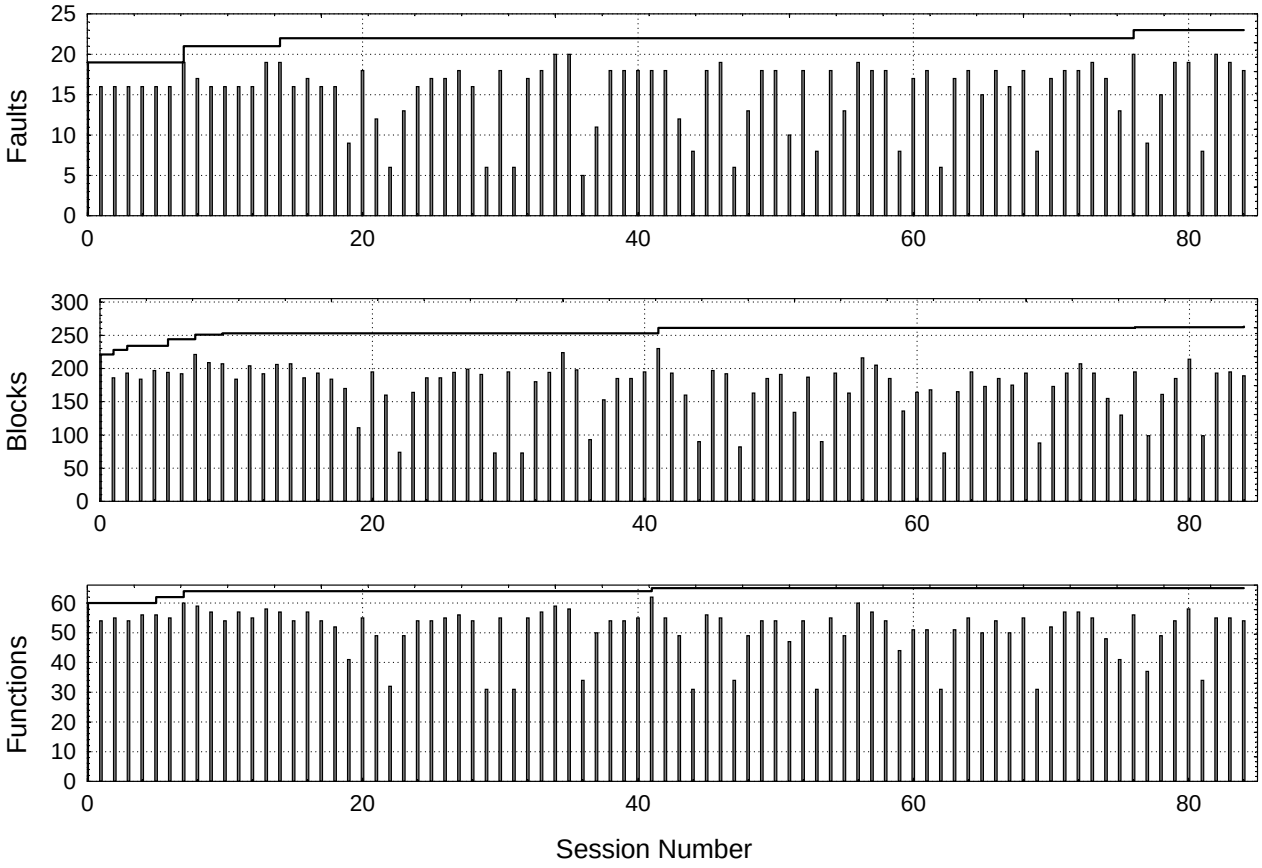

Figure 4. **Relationship between the number of user sessions collected and the effectiveness of US-1.**

found in the real world. To control for potential threats caused by the lack of representativeness of the web application, we carefully adapted an existing e-commerce application, populating its database, and configuring the site shipping and personalization attributes. In spite of this effort, the resulting web site included just a subset of the technologies used in the field, so our results can be claimed to generalize only to web applications using similar technologies. Additional web applications must be studied to overcome these threats to external validity.

Second, we needed to seed faults in the application so that we could evaluate the testing techniques. Although naturally occurring faults are preferred, obtaining a web application with a large number of known faults was not feasible. As a consequence, we opted for a fault seeding process similar to those commonly used in previous research to measure testing techniques' fault-detection effectiveness. The risk of seeding faults that are not representative of the faults found in web applications is still a threat to external validity. For example, we did not simulate the existence of faults on the client side (e.g., javascript faults).

Third, our user-session based techniques employ user interactions, which implies that we required collection of data from users interacting with the e-commerce site we had set up. User navigation and buying patterns were not our focus, but we wished to reproduce the activities users might perform in this type of site. The instructions for participants included a task and an incentive to make the experience more realistic for a set of potential customers, but are still just an approximation of reality and constitute a threat to external validity because of representativeness, and a threat to internal validity because they could have constituted a nuisance variable that affected the results. Similarly, the input selection that drive the WB techniques is influenced by the testers's ability and intuition for input selection. We diminished the impact of this threat by providing a common strategy for all testers to select their input values.

5. Conclusion

We have presented and quantified a new approach for testing web applications. This new approach differs from

existing approaches in that it leverages captured user behavior to generate test cases, leading to a reduction in the amount of required tester intervention. Further, the results of a controlled experiment indicate that this approach's effectiveness is comparable and likely complementary to more formal white box testing approaches.

These results suggest several directions for future work. First, the combination of traditional testing techniques and user-session data would seem to possess a potential that we have not been able to fully exploit. New ways must be found to successfully integrate these approaches. In addition, more complex techniques that consider other factors affecting the approach must be explored; such factors include web application states and concurrent user requests.

Second, the analysis of user-session techniques suggests that using a large number of captured used sessions involves tradeoffs. Additional sessions may provide additional fault detection power, but a larger number of sessions also implies more test preparation and execution time. Techniques for filtering sessions will need to be investigated.

Third, the applicability of the user-session test generation approach will certainly be affected by the efficiency of the data collection process. Further studies are needed to determine under what types of loads this approach is cost-effective. Furthermore, given the observed asymptotic improvement in fault detection, studies will need to consider whether this approach should be applied to all or just a subset of the user-sessions.

Finally, we believe that there are many applications for user-session data that have not been fully explored in the domain of web applications. For example, user-session data could be used to assess the appropriateness of an existing test suite in the face of shifting operational profiles. Through such approaches, we hope to be able to harness the power of user profile data to improve the reliability of this important class of software applications.

Acknowledgments

This work was supported in part by the NSF Information Technology Research program under Awards CCR-0080898 and CCR-0080900 to University of Nebraska, Lincoln and Oregon State University. The e-commerce application was made available by gotocode.com. Madeline Hardojo assisted in the fault seeding process. We especially thank the users who participated in the study, and we thank the reviewers for their helpful comments.

References

[1] Apache-Organization. Apache http server version 2.0 documentation. http://httpd.apache.org/docs-2.0/.

[2] B. Beizer. *Softw. Testing Techniques*. Van Nostrand Reinhold, New York, NY, 1990.

[3] R. Binder. *Testing Object-Oriented Systems*. Addison Wesley, Reading, MA, 2000.

[4] J. Conallen. *Building Web Applications with UML*. Addison-Wesley Publishing Company, Reading, MA, 2000.

[5] W. Dickinson, D. Leon, and A. Podgurski. Finding failures by cluster analysis of execution profiles. In *Proceedings of the International Conference on Software Engineering*, pages 339 – 348, May 2001.

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

[7] M. Harrold, R. Gupta, and M. Soffa. A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology*, 2(3):270–285, July 1993.

[8] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proceedings of the International Conference on Software Engineering*, pages 191–200, May 1994.

[9] E. Kirda, M. Jazayeri, C. Kerer, and M. Schranz. Experiences in Engineering Flexible Web Services. *IEEE Multi-Media*, 8(1):58–65, Jan. 2001.

[10] T. Lee. World wide web consortium. http://www.w3.org/.

[11] C. Liu, D. Kung, P. Hsia, and C. Hsu. Structural testing of web applications. In *Proceedings of the 11th IEEE International Symposium on Software Reliability Engineering*, pages 84–96, Oct. 2000.

[12] S. Manley and M. Seltzer. Web facts and fantasy. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems*, Monterey, CA, 1997.

[13] B. Michael, F. Juliana, and G. Patrice. Veriweb:automatically testing dynamic web sites. In *Proceedings of 11th International WWW Conference*, Honulolu, May 2002.

[14] A. Nikora and J. Munson. Software evolution and the fault process. In *Proceedings of the Twenty Third Annual Software Engineering Workshop, NASA/Goddard Space Flight Center*, 1998.

[15] Parasoft. WebKing. http://www.parasoft.com/jsp/products.

[16] Rational-Corporation. Rational testing robot. http://www.rational.com/products/robot/.

[17] F. Ricca and P. Tonella. Analysis and testing of web applications. In *Proceedings of the International Conference on Software Engineering*, pages 25–34, May 2001.

[18] I. Software Research. eValid. http://www.soft.com/eValid/.

[19] J. Tzay, J. Huang, F. Wang, and W. Chu. Constructing an Object-Oriented Architecture for Web Application Testing. *Journal of Information Science and Engineering* , 18(1):59–84, Jan. 2002.

[20] E. J. Weyuker. On testing non-testable programs. *The Computing Journal*, 15(4):465–470, 1982.

[21] E. J. Weyuker and B. Jeng. Analyzing partition testing strategies. *IEEE Transactions on Software Engineering*, 17(7):703–711, July 1991.

[22] W. Wong, J. Horgan, S. London, and A. Mathur. Effect of test set minimization on fault detection effectiveness. In *Proceedings of the 17th International Conference on Software Engineering*, pages 41–50, Apr. 1995.