Leveraging User-Privilege Classification to Customize Usage-based Statistical Models of Web Applications

Sara Sprenkle and Camille Cobb
Dept of Computer Science
Washington and Lee University
Lexington, VA USA
Email: sprenkles@wlu.edu, cobbc12@mail.wlu.edu

Lori Pollock
Dept of Computer & Information Sciences
University of Delaware
Newark, DE USA
Email: pollock@cis.udel.edu

***Abstract*—Automatically creating test cases from statistical models of web application usage is an effective approach to generating test cases that represent actual usage. The models are typically generated from all collected user sessions. In this paper, we consider how grouping the user sessions—specifically by the user's privilege—creates different statistical models and the testing implications of those differences. We performed a study of user-privilege-specific navigation models and the resulting abstract test cases generated from over 19,000 user sessions to four deployed web applications. Our results suggest that grouping user sessions by the users' privileges results in smaller navigation models, which yield realistic test cases that represent users with that privilege well while also exploring navigations not seen in the input user sessions. In some cases, the user-privilege-specific models are significantly smaller, which allows the tester to either (a) generate relatively few test cases and still represent the user type well or (b) create test cases from a less abstract model—without exorbitant model space costs or the need for additional models to generate executable test cases. However, the benefits are not universal for all applications; thus, we present guidance to testers on metrics to determine whether creating user-privilege-specific test cases will be advantageous.**

I. Introduction

Web applications are ubiquitous and are growing in complexity. Generating test cases that test the application in the same way that various types of users access the application is important to effectively find faults. One approach to generating effective test cases that represent actual users is by generating test cases from usage-based statistical models constructed from user sessions collected from deployed versions of the web applications[1], [2], [3]. Briefly, a statistical navigation model is based on the frequencies of the users' usage patterns, as recorded in user sessions, which are cheap to obtain. Usage information is important to model because user behavior does not follow exactly a statically determined model and, when given several options, users tend not to choose between the options equally [3].

In existing test-case generation approaches that leverage usage-based statistical models, all user sessions are treated equally when generating the model; however, in many applications, there are groups of users who use the application in distinct ways. For example, in an application like Amazon.com, some users sign in to buy something, while others never sign in. Users who have signed in to the application often have different privileges than those who have not signed in. Because of these different access privileges and intentions for using the application, the patterns in how users navigate the application and the values the user enters will also be different. By classifying the user sessions into groups, testers can create models that are better tailored to these groups' behaviors. We refer to these tailored models as *user-privilege-specific models*. Beyond improving user representativeness, testers also gain control over what types of test cases they generate because they can produce more test cases for the types of users that they are most concerned with or who are likely to access parts of the application that the testers are focused on testing. For example, in applications where users have very few options without signing in (e.g., course management applications like Sakai and BlackBoard), a tester could generate fewer test cases for user sessions without a login.

We identified a spectrum of ways to categorize user sessions: the entire set of user sessions (the existing approach [4], [2]), user sessions grouped based on access privileges, user sessions grouped by user, and individual user sessions. We can use these categorized subsets of user sessions as the input to the usage-based test case generation process. There are tradeoffs between these approaches to categorization. Using each individual user session as the input results in test cases that are very representative of the original user sessions but requires space for each separate usage model, without aggregate knowledge of usage, which will limit the variety of test cases. Categorizing by individual user has similarly large space requirements without aggregate usage knowledge.

At the other end of the spectrum, using the entire set of user sessions may create an unrealistic model, where a user can go from one part of the application that requires authorized access privileges to a shared part to another part of the application that requires different privileges that are mutually exclusive from the first required privileges. For example, in a course management application, some users are instructors and others are students. Members of these groups have exclusive access to certain parts of the application; however, there may also be a significant shared part—for example, help pages and pages that do not require logging in. Furthermore, the likelihood of a particular parameter value for a given page could be significantly different for different types of users. The same kinds of differences could be seen between users as opposed to groups of users.

Categorizing by user privilege may balance the space and aggregate usage requirements without generating test cases that are unrealistic when taking into consideration user privileges. Thus, in this paper, we explore generating usage-based statistical models from user sessions classified by access privileges. We classify user sessions from four deployed applications, generate navigation models from the classified user session sets, generate abstract test cases from the models, and evaluate the practicality of testers using this approach. The main contributions of this paper are

- Identification of the challenges and open questions of creating test cases customized to user privileges,
- Results from an experimental study of over 19,000 user sessions to four deployed web applications that show that generating test cases from user-privilege-specific yields effective, realistic test cases that represent the specific users' usage patterns well while also exploring navigations not seen in the input user sessions. Furthermore, in some cases, the user-privilege-specific models are significantly smaller, which allows either (a) the creation of relatively few test cases to represent the specific user or (b) the creation of test cases from a navigation model that includes parameter values without large space costs or the cost of a separate, possibly less accurate data model, and
- Recommendations to testers about model overlap and common use metrics to apply to help determine if creating user-privilege-specific models will be beneficial.

II. Statistical Model-based Testing

A. Test Case Generation Process

We will use the test-case generation process shown in Figure 1. 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) affects 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.

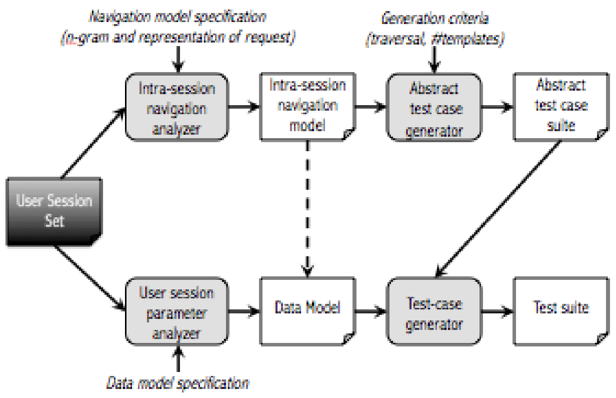

Navigation model specification
(n-gram and representation of request)
Generation criteria
(traversal, #templates)
Intra-session navigation analyzer
Intra-session navigation model
Abstract test case generator
Abstract test case suite
User Session Set
User session parameter analyzer
Data Model
Test-case generator
Test suite
Data model specification

Figure 1: The Test Case Generation Process

Before the test-case generation process shown in Figure 1 begins, the user requests are recorded, parsed, and segmented to create user sessions. Each *user session* is a sequence of user requests in the form of base requests and name-value pairs. The request recorder treats hidden parameters the same as regular parameters. 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, after a 45 minute gap between two requests from a user [5].

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 analyzer* uses the navigation model and abstract test case criteria to produce a set of abstract test cases. The abstract test cases are input into the *test case generator* with the data model to output a set of test cases—the test suite. The data model is constructed through an analysis of the user sessions. Various data models can be used to generate parameter values [2]. A tester can generate many test cases from one abstract test case by adding parameter values generated from different data models (or even the same model if the model is nondeterministic) to the test cases.

In this paper, we explore how categorizing user sessions by access privilege into separate sets of user sessions to use as input to the test-case generation process affects the navigation models and generated abstract test cases under various configurations.

B. Usage-based Statistical Models

In this paper, we base the usage-based models on the Sant et al. approach to generating test cases from a statistical model of user sessions [2]. The *navigation model* (called the "control model" in Sant et al. [2]) is an n-gram Markov model of the user sessions' requests, where n is one more than the number of previous requests used to predict the next request. A navigation model where n=1 (called 1-gram or *unigram*) models the probability of a user making a request, regardless of previous requests. For $n \geq 2$, the navigation model is a directed graph, where the states are the set of $(n-1)$-length request sequences in a set of user sessions. A state represents a particular request sequence and its outgoing edges are labeled with the conditional probabilities of the next request. The generated model may have several ”start” states, since a user's session may start from any page, e.g., a bookmarked page, a page found by a search engine, or from trying to continue a timed-out session.

A possible drawback of this approach is that the model is not complete unless users access all parts of the application. However, the goal of this kind of testing is not completeness but to focus on actual usage.

In previous work, two authors of this paper explored how tuning the navigation model's configuration—including how requests were represented and the amount of history used—affected the resulting navigation model and generated abstract test cases [3]. For example, consider how to represent the request `GET /bookstore/search?query=potter&type=paper` for paperback books with the search term “potter”. If the model represents requests by their **r**equest type, **r**esource, and parameter **n**ames and **v**alues of the data, which we refer to as RRNV, the request is unaltered in the model. If, instead, the model does not include the parameter values (RRN), the request is represented by `GET /bookstore/search?query&type`. Other requests with different parameter values will be represented by the same state in the model.

The authors concluded that, in general, testers should represent requests using RRN, which balances scalability and representation requirements, but requires the development of a data model to create executable test cases. However, a tester may want to consider including parameter values in the representation (RRNV) if using a larger n; using RRNV will require more space but not the development of a data model.

III. Customizing Test Cases by User Privilege: An Empirical Study

In this section, we will discuss the challenges and open questions in using user-privilege classifications to improve testing with usage-based statistical models. We also present an empirical study that evaluates the open questions posed.

A. Subjects

We studied four publicly deployed web applications on servers administered by our research group. The applications are written in Java using servlets and JSPs and consist of a backend data store, a Web server, and a client browser. Since our usage-based testing techniques are language-independent—requiring user sessions but not source code for testing, our techniques can be easily extended to other web technologies.

Subject	# of Classes	NCLOC
Book	11	5279
CPM	76	7430
Logic	106	10704
Logicv2	135	16491
DSpace	291	29430

Table I: Subject Application Characteristics

We created 8 subject user-session sets from user requests to the applications. The applications were of varying sizes, technologies, and representative of web application activities and usages: an e-commerce bookstore (Book) [6]; a course project manager (CPM) used as part of computer science courses at the University of Delaware; an online symbolic logic tutorial (Logic and a significantly revised version, Logicv2) used as part of philosophy courses at Washington and Lee University; and a customized digital library (DSpace) used by our research group to make our publications easily searchable and accessible [7]. Table I summarizes the applications' code characteristics.

Book was the only application for which an email was sent to local newsgroups asking for volunteer users. These user requests were also used by Sant et al. [2]. For the remaining applications, users accessed the applications naturally, i.e., they were not solicited for experiments. We collected accesses for each application over a long period of time: CPM: 5 academic semesters, Logic: 2 academic semesters, DSpace: 3.5 years.

We converted the user accesses into user sessions using Sprenkle et al.'s framework [5]. Before processing user accesses, we removed accesses from IP addresses that are known to be spiders, bots, or malicious to reduce the noise from non-users and better create models of human users' navigations. We partitioned the DSpace user sessions by the time periods in which they were collected to provide more sets of user session subjects to model and compare.

Table II shows the characteristics of the collected user sessions, in terms of the number of user sessions (totalling over 19,000 sessions), the number of user requests (totalling nearly 128K), and the percent of application code covered by the user sessions using Cobertura [8]. We report line coverage to show that the user sessions cover a large portion—but not all—of the application.

B. Threats to Validity

We performed our study on 8 sets of user sessions from four small to medium-sized publicly deployed applications. A study with additional sets of user sessions and additional user privilege types or usage patterns and on larger applications may be necessary to generalize the results. However, our four applications do represent common web applications, user privileges, and usage. Another possible threat is that

Subject	# User Sessions	# Requests	% Lines Cvd
Book	125	3564	61%
CPM	890	12352	78%
Logic	497	16,179	80%
Logicv2	374	16,052	78%
DSpace1	1087	12,277	74%
DSpace2	5012	14,110	46%
DSpace3	3853	15,126	45%
DSpace4	7687	38,155	49%
Total	19,525	127,827	–

Table II: Characteristics of User Session Sets

the user sessions were automatically parsed from the access log and automatically classified. Some user sessions may be incorrectly generated or classified, which may affect the results, but is representative of how the technique would be used in practice.

C. Classifying User Sessions By User Privilege

The first challenge is how to automatically classify user sessions as having some privileges. In this work, we consider a user session to have some privileges if the session contains a successful login request, i.e., the user requests to login using a username and password, the application authenticates the user, and the user now is authorized to access some subset of the application.

All but one of the applications we evaluated (not included in the subjects) has at least one resource used for logging in, which takes as parameters the user name and password. If a user session contains a request for that resource with a username and password, the user is considered a privileged user. Otherwise, the user session has no special privileges. In several applications we studied, privileged users had distinct roles or privileges. For those applications, we identified a key resource associated with each privilege type, which we could then use to classify users. We believe testers can use a similar process for most applications.

In some user sessions, users attempted to log into the application multiple times with different usernames. Multiple logins can occur for one of two main reasons: a user logs into the application using different user names (which may have different privileges) or a user attempts to login with an incorrect user name. Currently, if a user session contains multiple logins for the same privilege, we categorize the user session as that privilege. However, if the user session contains logins for different user privileges, we chose not to include the session in any of its privileges' sets of user sessions to maintain the generated models' focus on only one type of user.

We chose to classify users by their privilege. There may be other ways to classify the user sessions as well, such as by the user's intent [9], [10], [11]. We believe that, since privileges likely relate to intents, the results may be similar. Comparing the alternative classifications is an area of future work.

1) Methodology: For each set of user sessions, we grouped user sessions by the user's privilege. When possible, we classified the user sessions into the distinct privilege type, no login, or multiple logins. The classifying scripts—implemented in Python—executed on the order of a few minutes at most.

2) Results: We identified two distinct types of privileged users in CPM (Group and Grader) and three distinct types in Logic: Admin, Professor, and Student. In Book and DSpace, there was only one type of privileged user.

Table III shows the properties of the classified user session sets. The first column is the application. The second column is the classification of the user's privilege. The third and fourth columns represent the number of user sessions and requests, respectively, in the classified set of user sessions. The fifth and sixth columns represent the percentage of the original, complete set of user sessions or requests, respectively, in the classified set of user sessions. We do not show the amount of application code coverage of the classified sets of user sessions because the sessions depend on state created by the sessions that are not included in the set of sessions.

We were not able to classify all user sessions. In some user sessions, the user attempted to log in but did not make any subsequent requests that identified the user's type. There were 15 and 22 uncategorized user sessions in Logic and Logic2, respectively.

Our applications have different distributions of use by privileged and non-privileged users. The DSpace user sessions are dominated by non-privileged users, Book is balanced between users who log in and those who do not, while CPM and Logic have more privileged users—over 75% of user sessions and requests come from privileged users.

It is interesting to note that some types of users accessed the application more often (larger percentage of user sessions) but made fewer requests (smaller percentage of requests), while other types did the opposite. For example, in Logic, the No Login users make up 14% of the user sessions but less than 2% of the requests. On the other hand, DSpace1 privileged users make up less than 10% of the user sessions, but the requests make up over 48% of the requests.

A few types of users had relatively small numbers of user sessions. In the Logic user sesssion sets, the Admin user has only a total of 5 user sessions. The user sessions representing Admin users were often categorized as Mult Logins because Logic Admin users often logged in again as a Professor or Student to test what they changed in the system. By not including the Mult Logins in each type of user's set of user sessions, we lose the Admins' usage information, but we also will not create models that represent more than one type of user. Another approach may be to partition the user session into the distinct parts that represent each type of user and include those partitions in the appropriate privilege set.

Subject	Privilege	# User Sessions	# Requests	% of Original User Sessions	% of Original Requests
Book	Login	114	3365	91.20%	94.42%
	No Login	11	199	8.80%	5.58%
	Mult Logins	19	686	15.20%	19.25%
CPM	Group	507	5316	56.97%	43.04%
	Grader	217	4661	24.38%	37.73%
	No Login	152	780	17.08%	6.31%
	Mult Logins	101	3282	11.35%	26.57%
Logic	Student	284	12869	57.14%	79.54%
	Professor	78	1060	15.69%	6.55%
	No Login	70	222	14.08%	1.37%
	Admin	2	203	2.41%	1.25%
	Mult Logins	51	2543	10.26%	15.72%
Logicv2	Student	290	15213	77.54%	94.77%
	Professor	16	284	4.28%	1.77%
	No Login	20	113	5.35%	0.70%
	Admin	3	30	0.80%	0.19%
	Mult Logins	42	1530	11.23%	9.53%
DSpace1	No Login	981	6291	90.25%	51.24%
	Login	106	5986	9.75%	48.76%
	Mult Logins	2	77	0.18%	0.63%
DSpace2	No Login	4978	12595	99.32%	89.26%
	Login	34	1515	0.68%	10.74%
	Mult Logins	1	46	0.02%	0.33%
DSpace3	No Login	3807	13943	98.96%	92.18%
	Login	40	1183	1.04%	7.82%
	Mult Logins	1	9	0.03%	0.06%
DSpace4	No Login	7680	37937	99.91%	99.43%
	Login	7	218	0.09%	0.57%
	Mult Logins	1	15	0.01%	0.04%

Table III: Characteristics of Classified User Session Sets

D. *User-Privilege-Specific Navigation Models*

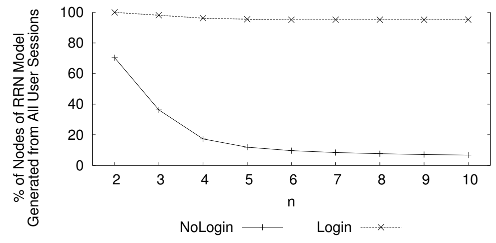
After we have classified the user sessions by their privileges, we use them as input to the test-case generation process in Figure 1. We will explore the characteristics of the resulting user-privilege-specific navigation models and how they are different from the navigation models created from the whole set of user sessions, which we will refer to as *general-usage* navigation models.

Since the sets of privileged users' user sessions are smaller and likely less diverse in terms of what parts of the application the users access and the sequences used to access the application, we expect the size of the user-specific navigation models to be smaller than the general-usage navigation models. The smaller the model, the more incentive to create the user-privilege specific model. Furthermore, we expect that users with the same privilege may enter or use more similar parameter values than what all users use. The reduced diversity in parameter values and the reduced navigation model size may warrant using RRNV to represent requests because the model growth is not prohibitive and testers do not have to develop a separate data model. We will explore generating navigation models using both RRN and RRNV as representations.

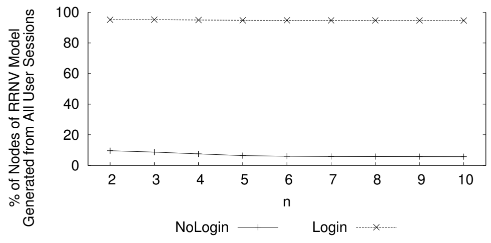
Testers also want to know how similar the various user-privilege-specific navigation models are. The amount of overlap between the user-privilege-specific navigation models indicates how much of the application is shared between the various types of users. Beyond how much of the application overlaps, we need to know how much time users spend in the shared parts. The combination of the amount of overlap and the amount of time spent in the overlap gives testers a better idea of whether they should separate user sessions into their privilege groups: if a lot of the application is shared and used often, there is less value in generating separate tests for the various types of users. If the common parts—regardless of the size—are used frequently, it is more likely that the models will yield test cases that traverse from user-privilege-specific parts of the application to the shared space to another, mutually exclusive user-privilege-specific part. While it is important to test these invalid navigations, testers may not want to test them exhaustively.

1) Methodology: We generate variations of navigation models by running the *intra-session navigation analyzer* (Figure 1) on each set of user sessions grouped by user privilege. The *intra-session navigation analyzer* is implemented in Python. We did not perform experiments using the Mult Logins sets of user sessions because we want to focus on models that represent only one type of user.

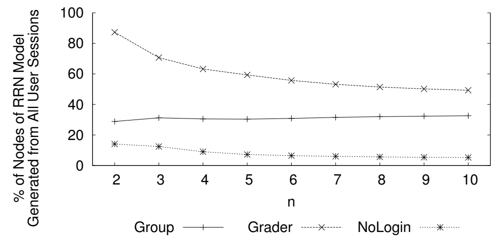
For each set of user-privilege-classified user sessions, we generated RRN and RRNV models using n from 2 to 10. We then measured the number of nodes and edges in each resulting user-privilege-specific model and compared these sizes to the size of the general-usage model to understand the amount of size reduction gained by using the user sessions classified by privilege as input.



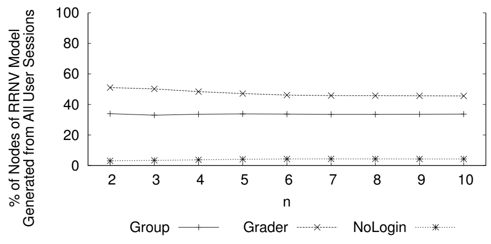
(a) Book - RRN



(b) Book - RRNV



(c) CPM - RRN



(d) CPM - RRNV

Figure 2: Percentage of General-usage Models in User-privilege-specific Models (RRN, RRNV for Book and CPM)

To measure the amount of application overlap, we count the number of common nodes from the 2-gram RRN navigation models generated for each pair of user types. We only consider overlap for the 2-gram RRN model because the overlap will decrease as n increases and when including parameter values. We also calculate the percentage of requests that the entire set of user sessions make to these common 2-gram RRN requests.

2) Results: **Size of Models.** Figures 2 and 3 show the size of the models generated from the sets of user sessions grouped by privilege type relative to the size of the general-usage model with the same configuration. The x-axis represents the value of n used to generate the model. The y-axis represents the percentage of nodes from the general-usage model generated that are in the model generated from the user sessions grouped by type. We show the results for both the RRN and RRNV models. Due to space constraints, we only show the results for some of the user session sets, and we do not show the results for edges, but the results and trends for edges were similar to the results for nodes.

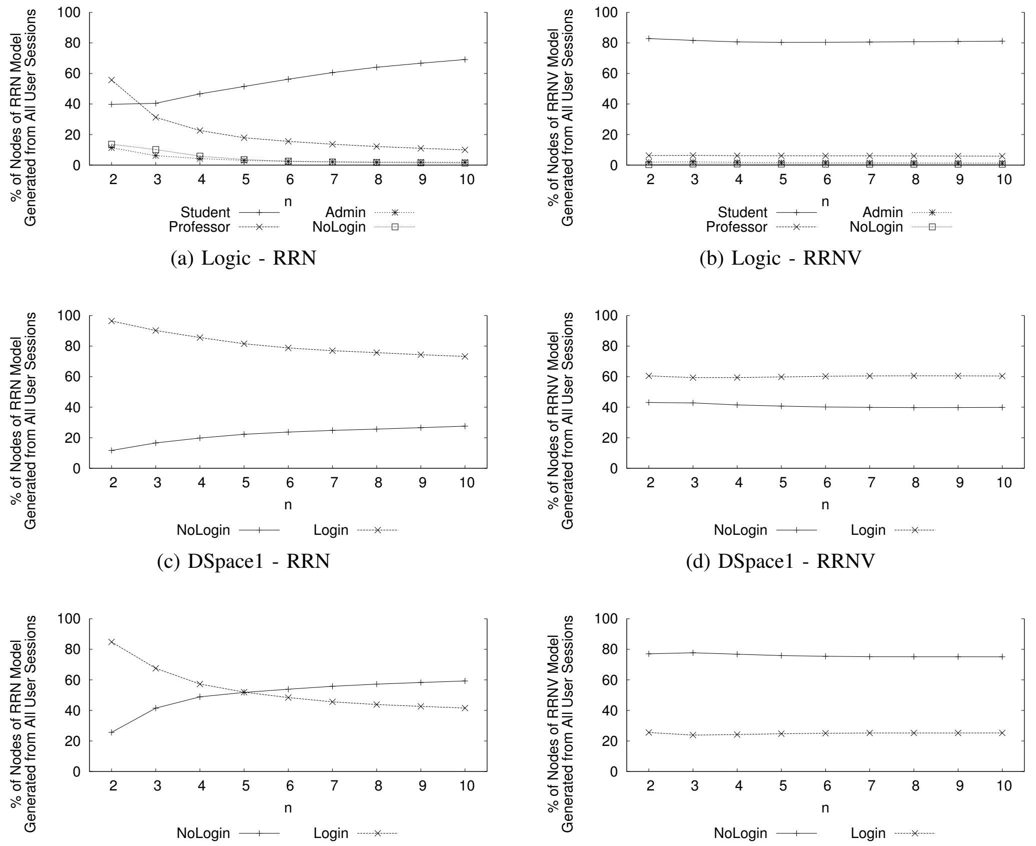
In general, a large size percentage means that the users of a specific privilege class access a large portion of the application that all users access, while a smaller percentage means that the users access a smaller portion of the application, leading to more space savings and a more specialized model. If the reduction is small, there is less incentive to create the user-privilege-specific models. The user-privilege-specific models generated for Book-No Login; CPM-Group, Grader; Logic and Logic2-Professor, Admin, NoLogin; DSpace1-No Login; DSpace3-Login; and DSpace4-Login were in general less than half the size of the general-usage models.

For most of the user session sets, privileged users access more of the application's resources than non-privileged users. Logic's Admin users and the privileged users in some of DSpace's sets of user sessions are exceptions. The numbering of the DSpace user session sets reflect the order in which they were collected. Initially, DSpace privileged users did more work to set up the application and add publications to the repository. As time went on, the privileged users did less work.

We do not see any common trends in terms of relative size of the RRN models as n increases. If the relative size percentage increases, that implies that the user-privilege-specific model is more diverse in terms of sequences as compared to the general-usage model. In other words, this privileged user is responsible for the usage sequences seen in the general-usage model.

In general, for all types of users, the size of the RRNV model is a similar reduced size to the RRN model, especially as n increases. The shape of the relative size curves for the RRNV models are nearly flat as n increases, which is reasonable to expect: the user group's parameter values add a fixed amount of nodes relative to general usage.

Overlap in Models. Tables IV through VI show (1) the relative overlap of the 2-gram RRN user-privilege-specific models, calculated as the number of nodes in common divided by the number of nodes in the 2-gram RRN general-usage navigation and (2) the percentage of requests in the entire set of user sessions to the common 2-gram RRN nodes. For example, in Table V, the amount of application overlap between the models for the Group and Grader users in terms of the 2-gram RRN nodes is 6.25% and



(a) Logic - RRN

(b) Logic - RRNV

(c) DSpace1 - RRN

(d) DSpace1 - RRNV

(e) DSpace2 - RRN

(f) DSpace2 - RRNV

Figure 3: Percentage of General-usage Models in User-privilege-specific Models (RRN, RRNV for Logic, DSpace1, and DSpace2)

Subject	% Common; % Requests
Book	32.76%; 94.53%
DSpace1	4.05%; 75.59%
DSpace2	5.16%; 79.66%
DSpace3	5.60%; 57.65%
DSpace4	3.36%; 62.92%

Table IV: Book and DSpace: User-privilege-specific Models Overlap as % of General-Usage Model; User-privilege-specific Overlap as % of Requests in User Sessions

		Privilege	
	Group	**Grader**	**No Login**
Group	-	6.25%; 60.25%	4.17%; 37.34%
Grader	6.25%; 60.25%	-	4.86%; 37.53%
No Login	4.17%; 47.34%	4.86%; 37.53%	-

Table V: CPM: User-privilege-specific Models Overlap as % of General-Usage Model; User-privilege-specific Overlap as % of Requests in User Sessions

those common 2-gram RRN nodes make up 60.25% of the requests in the complete set of user sessions.

Again, we see very different trends across the applications. For Book, the various users access almost one-third of the same RRN nodes, and those nodes were commonly accessed. The shared nodes included search and browse pages, registration pages, and failed login attempts, which look the same as successful login requests. CPM's users

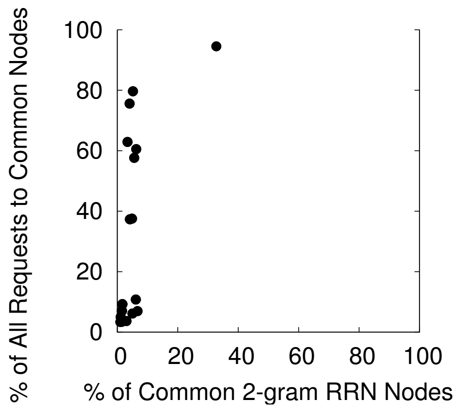


Figure 4: Comparing Model Overlap to Overlap Use

Logic	Privilege			
	Students	Professors	Admins	No Login
Students	-	6.11%; 10.79%	1.67%; 9.27%	6.67%; 6.95%
Professors	6.11%; 10.79%	-	1.67%; 9.27%	5.00%; 6.16%
Admins	1.67%; 9.27%	1.67%; 9.27%	-	1.11%; 4.99%
No Login	6.67%; 6.95%	5.00%; 6.16%	1.11%; 4.99%	-

(a) Logic

Logic2	Privilege			
	Students	Professors	Admins	No Login
Students	-	1.52%; 7.03%	1.52%; 7.03%	3.03%; 3.68%
Professors	1.52%; 7.03%	-	1.52%; 7.03%	1.52%; 3.45%
Admins	1.52%; 7.03%	1.52%; 7.03%	-	1.01%; 3.37%
No Login	3.03%; 3.68%	1.52%; 3.45%	1.01%; 3.37%	-

(b) Logic2

Table VI: Logic and Logic2: User-privilege-specific Models Overlap as % of General-Usage Model; User-privilege-specific Overlap as % of Requests in User Sessions

share less than 10% of the application, but those shared pages are commonly accessed. Some of the shared pages are documentation pages. Interestingly, some of the shared requests with No Login users are caused by users making a request to a privileged resource after their session has logged out, and they do not log back in. The Logic applications have the fewest shared application resources across users, and the shared resources are not commonly used. In DSpace, there is not much overlap in the application, but the overlap is commonly accessed—for example, for searching and browsing functionality.

Figure 4 relates the two metrics graphically. The x-axis represents the amount of common 2-gram RRN nodes, while the y-axis represents the relative amount of common usage. Testers can use this graph to determine if the various user types are different enough to warrant making separate models. If the points are focused in the upper-right quadrant, the application has a large amount of shared application code that is used frequently, meaning that creating separate models for the various users will not greatly improve testing effectiveness. On the other hand, if there is any common application code, the general-usage model may yield test cases that go from one privileged user's application space to shared space to a different privileged user's application space, which may not be feasible.

In Figure 4, the point closest to the upper middle represents Book, the points representing Logic are clustered in the bottom-left quadrant, DSpace is clustered in the top left, while CPM is in the left center. Based on this graph, there is less incentive to create user-privilege-specific models for Book, but all models are different enough to warrant creating test cases from the models.

Testing Implications: In many cases, user-privilege-specific models are significantly smaller than models generated from the complete set of user sessions. If a user-privilege-specific model has a sufficiently small amount of application overlap with the models of other user-privilege-specific models, there is more potential value in generating test cases from the user-privilege-specific models.

E. Generating Abstract Test Cases

Finally, we need to consider the characteristics of the abstract test cases generated from the user-privilege-specific models. Can we generate a smaller number of specialized test cases to achieve similar usage patterns to the user-privilege-grouped user sessions? Since the set of user sessions used as input to create the navigation model is smaller and likely less diverse, does the testing process still create abstract test cases that are different from any in the original user sessions?

Another interesting question is whether the resulting subsets of user-privilege grouped user sessions will be large enough to warrant generating navigation models and abstract test cases from them. The answer depends on several factors, including the number of requests in the user sessions and the diversity of usage patterns within the user sessions. Determining the number of different paths through the navigation model may be impossible or at least impractical to determine without generating the abstract test cases at the same time.

1) Methodology: For each user-privilege-specific RRN navigation model, the *abstract test case generator*—implemented in Python—generated a suite containing 500 unique abstract test cases. Because of the generator's weighted random walk, the 500 abstract test cases approximately represent the 500 most-probable paths through the model. For each abstract test case suite, we measured the abstract test cases' representativeness of the various users as the number of the user-privilege-specific user sessions' RRN sequences of various lengths that are represented by the abstract test case suite, divided by the total number of RRN sequences in the user-privilege-specific user sessions. We do not measure sequence coverage with respect to the complete set of user sessions because a tester would likely use this technique to generate user-privilege-specific abstract

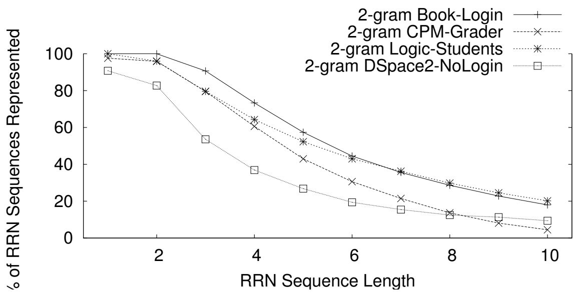


Figure 5: Representative RRN Sequence Coverage Results

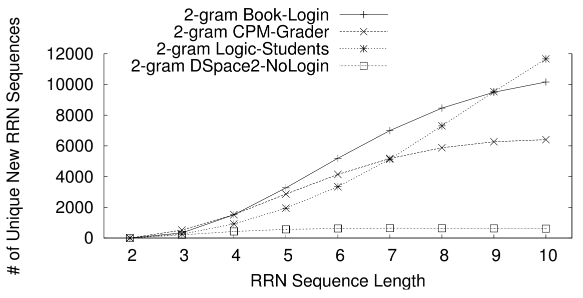


Figure 6: Representative New, Unique RRN Sequence Coverage Results

test cases and augment the test suite with test cases generated from the complete set of user sessions to achieve sequence coverage representative of the whole set. To measure the ability of the abstract test cases to test sequences not in the complete set of user sessions, we count the number of new, unique RRN sequences of various lengths that the abstract test cases explore.

2) Results: **Models with Fewer than 500 Distinct Paths.** The abstract test-case generator was not able to generate 500 unique test cases for the following models: CPM-No Login, $n \geq 9$; Logic-No Login, $n \geq 5$; Logic2-Admins, $n \geq 3$; and Dspace4-Admin, $n = 10$. For all but CPM-No Login when $n = 9$, fewer than 100 abstract test cases were generated; for CPM-No Login, $n = 9$, less than 150 test cases were generated. Interestingly, these models are not necessarily the smallest models in terms of nodes and edges. For example, even with only 2 user sessions for Logic-Admin, there is sufficient diversity in the model's paths to generate 500 unique abstract test cases, even for $n = 10$.
Testing Implications: Testers may be able to generate relatively few test cases for some types of users and still achieve high sequence coverage for that user type. These user groups are also likely good contenders for using RRNV models to generate test cases without exorbitant space costs or the need for a data model.

Representation of Various Users. In general, the representation results followed the same trends found for the general-usage models in Sprenkle et al. [3]: as n increases, the greater the representation in terms of sequence coverage. Figure 5 shows some representative sequence coverage results for 2-gram models, which have the smallest RRN sequence coverage. The x-axis represents the length of the RRN sequence. The y-axis represents the percentage of the length-x RRN sequences in the user-privilege-specific user sessions that are represented in the abstract test cases. The lines show the data for the test cases generated from the various n-gram models. Since the user-privilege-specific models are smaller, the abstract test case suites' RRN sequence coverage of the user-privilege-specific user sessions tended to be very high. All models produced abstract test suites that covered more than 95% of the RRN requests except for DSpace1-No Login for $n = 2$; DSpace2-No Login for $2 \leq n \leq 5$, and DSpace3 and DSpace 4-No Login for all n. In general, the sequence coverage results for the user-privilege-specific abstract test cases are greater than those for the abstract test cases generated from the general-usage models [3], which is not surprising.

Testing New Navigation Sequences. In general, the new sequence results followed the same trends found for the general-usage models in Sprenkle et al. [3]: as n increases, the fewer new sequences not found in the user sessions. Thus, the models most likely to produce new sequences are the 2-gram models. All the 2-gram models produced abstract test suites with length-3 RRN sequences not seen in the entire set of user sessions. Figure 6 presents representative new unique sequence results for 2-gram models. The x-axis represents the length of the RRN sequence. The y-axis represents the number of length-x RRN sequences that are represented in the abstract test cases but not in the complete set of user sessions. The lines indicate the test cases generated from the various n-gram models.
Testing Implications: Testers can generate test cases tailored to specific types of users that have high sequence coverage of what those users do, while still testing sequences that were not in the entire set of user sessions.

IV. Related Work

Several researchers have proposed representing web applications via navigation models. Wang et al. essentially spider the application from a defined start page and use a combinatorial approach to input values into the application's forms to generate the navigation model [12]. Their model does not leverage usage information. Tonella and Ricca's navigation is similarly generated by spidering the application from a start page and inputting values from equivalence classes into the forms [1]. They augmented their navigation model with usage information, adding usage-based probabilities to the edges. Neither Wang et al.'s nor Tonella and Ricca's navigation model generation is completely automated. Both require the values to be input into forms to be known beforehand. Another limitation of both approaches is that neither seems to explicitly handle navigation that may depend on different application state (e.g., if a search fails to find any matches because of the contents of the database). Neither approach considers partitioning the model based on the type of user.

Elbaum et al. [4] proposed leveraging user sessions to test web applications. Instead of using the entire set of user sessions as one test suite as in their approach, we could classify user sessions by user type to create multiple, user-type-specific test suites. The resulting test suites would have the same characteristics as a reduced test suite in terms of issues in replaying the test suite effectively; discussion of the issues and a proposed solution was discussed in work by two of the authors of this paper [13].

Brooks et al. [14] present a usage-based navigation model that was developed for GUI testing. Classifying users sessions by their privileges may not be directly applicable to this work. However, classifying users by their intent or skill level may be.

An active area of research is how to classify a web application user's intent, typically for use in improving web searches [10], [11] and/or the user's experience [9]. These classification techniques may be useful in classifying user sessions to create intent-specific navigation models.

V. Summary and Future Work

In this paper, we presented the case for classifying user sessions by the user's privilege and generating test cases customized to the type of user. Our results from an empirical study of four deployed web applications show that the approach is promising: *testers can generate user-privilege-specific test cases that represent usage while also exploring new, likely navigations*. For some user types, the resulting models are small enough that a tester can generate a few test cases and still produce high coverage or even use a model with less abstraction of requests, which would not require another model to generate executable test cases and without the high space costs usually associated with less abstract models. Since not all applications benefit from the approach, we provide guidance to testers about when to apply the technique using common application and usage metrics.

Future work includes (1) developing approaches to help testers select the number of abstract test cases to generate from each user-privilege-specific model and the general-usage navigation model to create an effective, comprehensive test suite, (2) creating test cases based on other groupings of user sessions—for example, intentions of the user, by cookies, or other ways, and (3) applying the approach to other usage-based test case generation approaches (e.g., [4]).

References

[1] P. Tonella and F. Ricca, "Statistical testing of web applications," *Journal of Software Maintenance and Evolution*, vol. 16, no. 1-2, pp. 103–127, 2004.

[2] J. Sant, A. Souter, and L. Greenwald, "An exploration of statistical models of automated test case generation," in *International Workshop on Dynamic Analysis (WODA)*, 2005.

[3] S. Sprenkle, L. Pollock, and L. Simko, "A study of usage-based navigation models and generated abstract test cases for web applications," in *International Conference on Software Testing, Verification and Validation (ICST)*, 2011.

[4] S. Elbaum, G. Rothermel, S. Karre, and M. Fisher II, "Leveraging user session data to support web application testing," *IEEE Trans. on Software Engineering*, vol. 31, no. 3, 2005.

[5] S. Sprenkle, E. Gibson, S. Sampath, and L. Pollock, "A case study of automatically creating test suites from web application field data," in *Workshop on Testing, Analysis, and Verification of Web Services and Applications*, 2006.

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

[7] "DSpace Federation," http://www.dspace.org/, 2012.

[8] "Cobertura," http://cobertura.sourceforge.net/, 2012.

[9] H. K. Dai, L. Zhao, Z. Nie, J.-R. Wen, L. Wang, and Y. Li, "Detecting online commercial intention (oci)," in *International Conference on World Wide Web (WWW)*, 2006.

[10] B. J. Jansen, D. L. Booth, and A. Spink, "Determining the informational, navigational, and transactional intent of web queries," *Information Processing and Management*, vol. 44, no. 3, 2008.

[11] H. Cao, D. H. Hu, D. Shen, D. Jiang, J.-T. Sun, E. Chen, and Q. Yang, "Context-aware query classification," in *SIGIR conference on Research and Development in Information Retrieval*, 2009.

[12] W. Wang, Y. Lei, S. Sampath, R. Kacker, R. Kuhn, and J. Lawrence, "A combinatorial approach to building navigation graphs for dynamic web applications," in *International Conference on Software Maintenance (ICSM)*, 2009.

[13] S. Sprenkle, E. Gibson, S. Sampath, and L. Pollock, "Automated replay and fault detection for web applications," in *International Conference on Automated Software Engineering (ASE)*, November 2005.

[14] P. A. Brooks and A. M. Memon, "Automated GUI testing guided by usage profiles," in *International Conference on Automated Software Engineering (ASE)*, 2007.