Coverage Criteria for Testing Web Applications

Sreedevi Sampath, Emily Gibson, Sara Sprenkle, Lori Pollock
Department of Computer and Information Sciences
University of Delaware
Newark, DE 19716
{sampath, gibson, sprenkle, pollock}@cis.udel.edu

ABSTRACT

As web applications evolve and their usage increases, their complexity also increases, thus creating a great demand for techniques and tools to ensure well-tested, reliable applications. While program-based coverage and fault detection capability can be used to measure the quality of test suites, the dynamic characteristics of web applications motivate additional criteria to complement these traditional test adequacy criteria. This paper presents novel *dynamic* coverage criteria customized for web applications—criteria intended for testing at a page level. Based on a changing universe of test requirements, as indicated by evolving usage of the application, our criteria avoid the difficulties of building an accurate static model of a web application's structure. We define a class of dynamic coverage criteria, present the subsumption relation among them, and describe two case studies to demonstrate their usefulness. Among other possible uses, the proposed criteria can be used to compare the quality of test suites, to select test cases, and to examine how usage of a web application changes over time. The proposed criteria complement traditional program coverage and fault detection capability criteria.

1. INTRODUCTION

The increasing reliance on web applications for global business and consumer activities necessitates tools and methodology for systematic testing and validation. Both program-based and functional testing techniques, which use user sessions gathered in the field, have been developed for web applications [4, 22, 16, 15]. Tools for validating non-functional requirements (e.g., markup text validators, link checkers, load testers, and compatibility testers) have also been developed [23]. Similar to traditional testing methodology, the *quality* of test suites for web applications can be measured in terms of program-based coverage criteria and fault detection capability. However, web applications exhibit characteristics different from conventional software, motivating the development of additional criteria.

Broadly defined, a web-based software system consists of a set of web pages and components that interact to form a system (structured as in Figure 1) which executes using web server(s), network, HTTP, and a browser, and in which user input affects the state of the system. A web page can be either *static*, in which case the content is fixed, or *dynamic*, such that its contents may depend on user input. User input to a web application consists of both navigational requests and data provided often through forms, which eventually affect the state of the underlying code on the server. The input field names and their values, called *name-value pairs*, become part of the request parsed by scripts or the application server. A database is often queried in response to a request, resulting sometimes in dynamically generated HTML code (i.e., a new web page), which is displayed to the user.

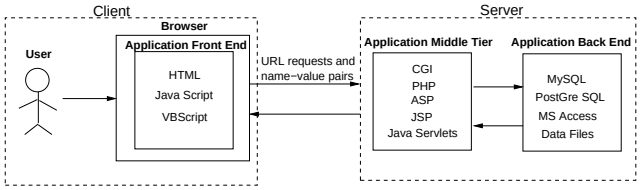


Figure 1: Structure of a Web Application

An existing tool for instrumenting code and computing program (e.g., method or statement) coverage can be utilized to judge the adequacy of test suites for web applications. The most commonly used program-based coverage criteria are based on constructing a flow-graph model of program structure. Two main groups of structural test adequacy criteria, *control-flow* and *data-flow* criteria, can be expressed in terms of the abstract model [26].

The difficulty in statically modeling and analyzing a web application limits this approach. The major challenges in statically modeling web applications are

1. **Control flow dependence on individual usage patterns and location.** In general, statically determining the application's control flow is difficult because the control flow is highly dependent on user input and sometimes in terms of trends in user behavior over time or user location. For example, `Amazon.com` displays web pages to users tailored to the users' interests, collected over time. `BBC.com` delivers different news content depending on the user's location. Not knowing which page an application is likely to display hinders statically modeling the control flow with accuracy and efficiency.

2. **Browser interactions.** Besides hyperlinks coded into a displayed page, browser interactions can alter an application's control flow. For example, from almost any page in

the application, the user can change the next viewed page by clicking the browser's back or forward buttons or by entering a URL to visit an application page directly. Considering all the possible browser interactions at every page in the application will lead to a very large, impractical representation of the application. A test suite based on a fairly accurate control-flow representation of a web application and adequate according to a criterion—even as commonly used in conventional software as all-branches—contains an impractical number of test cases. Ignoring browser interactions can lead to an inaccurate static model and subsequently poor testing and limited knowledge of the application's potential execution behavior. Since a user can enter a web application at any point by a browser action, representing and testing all potential control flow in a web application is similar to the problem of testing the infinite event sequences in GUIs [14].

3. **Numerous technologies, languages, and unknown components.** Web applications may have a number of characteristics, including integration of numerous technologies; modularization into reusable components that may be constructed by third parties; dynamically generated pages with dynamic content; and typically extend an application framework. The lack of information about reused components and libraries causes imprecision in the static analysis and modeling. In addition, the integration of different technologies and languages makes the analysis and modeling over components complex.

To enable practical analysis of a web application's structure in the presence of these challenges, the analysis typically ignores browser interactions, does not consider dynamic user location and behaviors, and models only parts of the application. Test criteria based on these static models can thus lead to inadequate test suites with respect to established criteria because the model lacks the program structures that should be satisfied to meet the criteria.

In this paper, we propose *a novel strategy for assessing quality of a web application's test suites.* We define the notion of a **dynamic test criterion**, which we informally define as **a criterion based on a changing universe of test requirements, as indicated by evolving usage of the application.** In contrast, we refer to the traditional notion of program-based test data adequacy criteria as *static test criteria* because they are based on models of a program's static structure. A static test criterion can be used as a stopping rule that determines when sufficient testing has been done and also as a measure of test quality by associating a degree of adequacy with each test suite. Among other possible uses, our proposed dynamic criteria can be used to provide more information about a test suite to the tester, compare the quality of test suites, to select test cases, and to examine how usage of a web application changes over time. In addition, the dynamic approach avoids the challenges of building an accurate static model of a web application's structure. However, dynamic criteria cannot be used to judge the adequacy of a single test suite with respect to how it satisfies a criteria; the universe of possible points to be covered is with respect to dynamic information about the application's execution—for some set of inputs, not with respect to static program structure—which is fixed with respect to usage. Thus, the proposed notion of dynamic criteria complements the use of static program coverage criteria.

The proposed coverage criteria are customized for web applications—criteria intended for testing at a page level. Test cases to satisfy the criteria are collected through logging user sessions. The proposed coverage criteria are motivated by (1) evidence that user-session-based testing is complementary to white-box testing techniques that were derived from testing conventional software operating under more traditional paradigms [4], (2) the difficulty of static modeling of web applications, and (3) observed usefulness of criteria based on observed user behavior. The key insight is to develop criteria based on coverage in terms of user session elements—URLs (web pages), names, and name-value pairs. The important contributions of this paper include
(a) Defining a class of *dynamic coverage criteria* for web applications to complement traditional *static coverage criteria*
(b) Presenting a subsumption hierarchy specifying the relation between individual proposed criteria
(c) A case study demonstrating the usefulness of user-session-based coverage information for longitudinal study of a web application's changing usage profiles
(d) A case study demonstrating the usefulness of dynamic user-session-based coverage criteria for test suite selection by correlating coverage in terms of user session elements and program code

Section 2 describes the state of the art in modeling web applications and the existing coverage criteria proposed specifically for web applications. In Section 3, we present an overview of dynamic coverage criteria for testing web applications and describe various applications of the criteria. In Sections 4 and 5, we formally define a set of dynamic coverage criteria based on user session elements, and prove the subsumption relation between them. In Section 6 we present two case studies that investigate the usefulness of these dynamic coverage criteria. We conclude and present future work in Section 7.

2. STATE OF THE ART

2.1 Current Models and Coverage Criteria

Static models and test criteria for web applications have been proposed by Ricca and Tonella [18], Kung et al. [11], Di Lucca et al. [13] and Andrews et al. [1].

Ricca and Tonella [18] developed a high level UML-based representation for web applications. In [25], Tonella et al. extend the model to include server pages; however, separate entities of a server page are created for each possible outcome. Also for increased analysis accuracy, every possible outcome of each dynamic and server page is represented as a separate entity. Unrolling each outcome of the server and dynamic pages can quickly lead to an impractical model. To model the flow between dynamic pages, model construction requires advance knowledge of the input values and any state information maintained by the application. It appears that the control flow *within* server pages is not considered, and the browser interactions are not modeled. The absence of modeling these features will lead to inaccurate control and data flow analysis. To our knowledge, the cost effectiveness of the proposed models has not been thoroughly evaluated.

In [18], Ricca and Tonella described a set of (static) cov-

erage criteria derived from traditional program-based criteria, but expressed over the UML-based representation. The proposed criteria are *web page, hyperlink, definition-use, all-uses, and all-paths*. The definition-use and all-uses criteria are considered only for data flow in the HTML pages. The data and control flow within the server pages are not captured by the model and thus it is not clear how to generate test cases to satisfy the data flow criteria.

Liu et al. [11] and Kung et al. [10] proposed multiple models, each targeted at capturing a different tier in web applications; they also suggested that data flow analysis can be performed at multiple levels. Though the models capture the interaction between different components of a web application, it is not clear if the models have been implemented and experimentally evaluated. We believe that as the data flow analysis progresses from the lower (function) level to higher (application) levels, in the presence of multiple models to represent the control flow, the models can easily become impractical in size and complexity for a medium-sized dynamic web application.

Di Lucca et al. [13] developed a web application UML-based model and a set of tools for the evaluation and automation of testing web applications. They consider single pages of the application as components to be tested at the unit level. They developed functional testing techniques based on decision tables, which help in generating test cases. However, their approach to generating test input is not automated. Di Lucca et al. [12] developed a statechart model of browser interactions and suggested integrating it with existing approaches to testing web applications. They also suggested that coverage criteria for browser interactions are similar to existing criteria for object-oriented systems. The model does not handle cookies or client-side scripts, and they do not discuss the practical effectiveness.

Andrews et al. [1] proposed an approach to modeling web applications with finite state machines and use coverage criteria based on FSM test sequences. It is not clear if the model or testing strategy have been evaluated.

3. THE NOTION OF DYNAMIC CRITERIA

3.1 An Overview

Test data adequacy criteria can be classified by the source of the information used to specify the test requirements. The common classes are *specification-based*, where requirements are expressed in terms of identified features or requirements of the software, and *program-based*, which are based on the program under test to determine how well the program code has been exercised [26]. We call these *static* criteria as they do not change with different user profiles of the software. Because a model constructed and used for static coverage criteria is constructed from analyzing the application code, the model is fixed if the application code does not change. Thus, the universe of requirements that a test suite is expected to satisfy remains constant.

In contrast, our notion of *dynamic* coverage criteria proposed for web applications is based on the *usage* of the application. Random testing and statistical testing are examples of testing based on prospective usage of software [26]; however, explicit usage-based web application criteria have not been previously defined, and few, if any, defined for conventional software. Our proposed dynamic criteria defines pages as a web application's basic building blocks. Thus, the dynamic coverage criteria test control- and data-flow in a web application at a page level. As with all dynamic approaches, the universe of requirements that a test suite is expected to satisfy changes as the suite of user sessions defining the requirements changes.

Our approach to testing based on dynamic coverage criteria takes user sessions automatically logged by a web server and formulated into test cases without employing a static application model. A user session is a sequence of URLs and name-value pairs, and we view the user session as representative of a use case [8] of the application. URLs are a unique characteristic of web applications that relate to server pages and their program code in the middle-tier of the application.

3.2 Applications of Dynamic Criteria

This section describes a number of different uses of dynamic coverage criteria for web application testing.

3.2.1 Assessing Quality of a Test Suite

Current program coverage analysis tools take as input a program under test and a test suite and provide a report that describes the program (e.g., method, statement, branch) coverage associated with executing the program code on the given test suite. For web applications, such a coverage tool can be augmented to present the tester with additional information about how the test suite also provides coverage for various dynamic coverage criteria. For example, we have built a tool that analyzes user sessions. This tool could present the tester with information about the additional consecutive URL pairs that a test suite contains—beyond those that a static application model depicts. These additional URL pairs demonstrate to the tester how browser actions have caused certain actions in the test case execution. Thus, valuable dynamic information about test cases can be expressed in terms of dynamic coverage criteria that a tester can quantify and use during testing.

3.2.2 Additional Criteria for Test Suite Comparison

The dynamic coverage criteria can be used to compare web application test suites as an additional criteria. For example, if two test suites, **T1** and **T2**, are both statement-coverage adequate, the two suites can be further distinguished by how they satisfy our dynamic coverage criteria. If **T1** satisfies more of our dynamic coverage criteria than **T2**, then **T1** is a better test suite for testing the application because it satisfies both the static, statement-coverage criterion and our dynamic coverage criteria, i.e., **T1** represents application usage better than **T2**.

3.2.3 Longitudinal Analysis for Focused Testing

Since the dynamic coverage criteria are based on application usage, they can be exploited as a measurement for longitudinal usage studies. El-Ramly et al. [3] study usage patterns in web applications and use pattern mining as a dynamic recommendation and adaptive mechanism. Their motivation was to provide users who follow frequently occurring navigation paths with recommendations at run-time. They have also applied pattern mining techniques to recover the

application's functional requirements with changes in usage scenarios. We believe a study of how test suites collected from different time periods satisfy dynamic coverage criteria can provide the tester with insight about changes in application usage over time. In particular, differences in criteria points satisfied by a test suite in one time period as compared to a test suite of another time period can indicate how usage changed from one period to another. Depending on the usage the tester wants to test, the tester can use the criteria as a metric to choose test suites.

3.2.4 Test Case Selection

As standalone criteria. Similar to static coverage criteria that are used as an explicit specification for test case selection [26], dynamic coverage criteria can guide test case selection. For test case selection, the original test suite dictates the universe for the dynamic coverage criteria. The quality of reduced test suites are then judged by how well they satisfy the established dynamic coverage criteria relative to the original test suite.

In our previous work [20], we presented a test case selection technique based on clustering by concept analysis following a simple dynamic coverage criterion. We have also performed studies [24] that showed how to use a dynamic coverage criterion of single URLs to perform test case selection by four different techniques. We compared three requirements-based techniques, Random, Greedy and Harrold et al.'s technique [7], against our test case selection technique based on clustering by concept analysis.

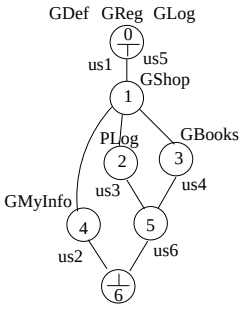
Concept analysis is a mathematical technique for clustering objects that have common discrete attributes [2]. To apply concept analysis to user sessions of a web application, we define the objects ,O, to represent the information uniquely identifying user sessions (i.e., test cases) and attributes, A, to represent the URLs according to certain criteria. Concept analysis builds a lattice where each node of the lattice is a tuple (O_i, A_i) such that all the objects in $O_i \subseteq O$ share all and only the attributes in $A_i \subseteq A$ and vice versa. The edges of the lattice denote the partial ordering between the concept nodes. Figure 2(b) shows the sparse concept lattice for Figure 2(a)'s user sessions, when the criteria is single URLs. For example in Figure 2(b), *node 3*'s objects are the sessions *us4, us6*, and the attribute set is *GDef, GReg, GLog, GShop, GBooks* (from the full lattice representation).

We developed a heuristic based on the concept lattice for selecting a subset of user sessions to be maintained as the current test suite [20]. Our heuristic for test case selection seeks to identify the smallest set of user sessions that satisfies a criterion C_a, as determined by the original test suite, while representing the original test suite's use cases. The selected test suite contains a user session from the bottom node, $\perp$[1], and a user session from each concept node that is one level up the lattice from $\perp$ (also called *next-to-bottom* nodes). These nodes contain objects that have the largest number of shared attributes, and the union of the attribute sets covers C_a. Thus, test suite selection through our heuristic exploits the concept lattice's hierarchical clustering properties. In

[1]The $\perp$ contains the sessions that are adequate to a certain criteria.

us	Visited URLs						
	GDef	GReg	GLog	PLog	GShop	GBooks	GMyInfo
1	●	●	●				
2	●	●	●		●		●
3	●	●	●	●	●		
4	●	●	●		●	●	
5	●	●	●				
6	●	●	●	●	●	●	

(a) Original Suite of User Sessions



(b) Concept Lattice

Figure 2: Example

Figure 2(b), the *next-to-bottom* nodes are *node 4* and *node 5* and on applying the heuristic, the selected test suite for single URLs criteria is {*us2, us6*}.

Though using concept analysis to select the test suite may not result in the minimum test suite for a given criterion, we believe our selection technique maintains the original suite's use case representation in the selected test suite [20]. Our previous papers [20] contain details on the theory behind applying concept analysis and the test-suite-selection heuristic.

Hybrid approaches. Sant et al. [22] generated an application model from user sessions and performed statistical analysis on the model to generate new test cases. Elbaum et al. [4] proposed a hybrid approach to test case generation, where they augment the generated model by applying Ricca et al's. [18] approach with user session data. We propose a hybrid approach that selects a test suite to satisfy our dynamic criteria. Then, a simple, high-level application model can be constructed that generates new test cases to satisfy additional static criteria (from the model), which were not satisfied by our dynamic criteria. Using a hybrid approach that generates test cases from a simple, high-level model of the web application with the test suite selected from user sessions according to our dynamic coverage criteria would reap the benefits of both approaches.

4. DEFINING COVERAGE CRITERIA

In this section we define novel dynamic coverage criteria for testing web applications. Test cases for web applications can be generated from user sessions or other static model-based techniques. In this paper we consider test cases as generated from user sessions and we will use the terms test cases and user sessions interchangeably. The coverage criteria we define can be used on test cases generated from static or dynamic context.

A **URL** u consists of a base address and possible name-value pairs (input field names and their values) as per the RFC definition ⟨**protocol**⟩**://**⟨**host**⟩ [**:**⟨**port**⟩] [⟨**path**⟩ [**?** ⟨**query**⟩]] [17]. A **test case** t, or user session, is an ordered sequence of URLs $\langle u'_1, u'_2, ..., u'_n \rangle$. We define t_{set} as the set of unique URLs occurring in t. We say that a test case t of length n **contains** a subsequence $s_x = \langle u_i, u_{i+1}, ..., u_{i+x} \rangle$ if the subsequence s_x occurs in t where $1 \leq i \leq n - x$. We define two functions for a URL u such that $strip_{n,v}(u)$ removes any name-value pairs of u, leaving the base URL,

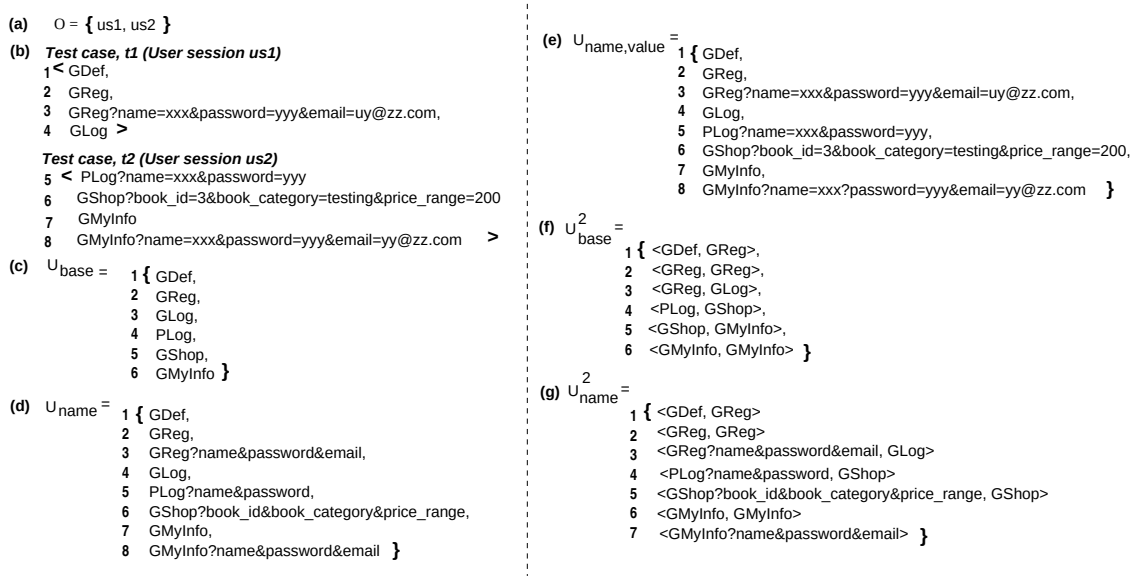


Figure 3: Example User Session for Different Criteria

and $strip_v(u)$ removes the value entries for u, if present, leaving the base URL and names intact. For a web application W, we define the **original suite** O as the set of test cases t collected during W's usage over a specified time. For our purposes, O represents the set of user sessions for application W. Each such user session is a sequence of URLs *requested* by a user. A **test suite T** is a set of test cases $\{t_1, t_2, ..., t_n\}$, where $t_i \in O$, for the application W.

Figure 3(a) shows O for a web application W to contain the set of user sessions $\{us1, us2\}$. Thus for W, the test suite $\mathbf{T}$ is the set of test cases $\{us1, us2\}$. Figure 3(b) shows the sequence of requested URLs in test cases *us1* and *us2* [2].

$\mathbf{U_{name,value}} = \{u_1, u_2, ..., u_n\}$ is the set of URLs such that for every test case t in O, if URL $u \in t_{set}$, then $u \in \mathbf{U_{name,value}}$. Note that a URL u' of $\mathbf{U_{name,value}}$ may not have name-value pairs if u' occurred without name-value pairs in some test case of O. Figure 3(e) illustrates $\mathbf{U_{name,value}}$. $\mathbf{U_{name}} = \{u_1, u_2, ..., u_m\}$ is the set of URLs obtained by applying $strip_v(u)$ to every URL u in $\mathbf{U_{name,value}}$. $\mathbf{U_{base}} = \{u_1, u_2, ..., u_p\}$ is the set of URLs obtained by applying $strip_{n,v}(u)$ to every URL u in $\mathbf{U_{name,value}}$. Figure 3(c) and 3(d) show the sets $\mathbf{U_{base}}$ and $\mathbf{U_{name}}$ for the example test suite.

Note that the sets $\mathbf{U_{base}}$, $\mathbf{U_{name}}$, and $\mathbf{U_{name,value}}$, *only* contain those URLs found in O. For example, if O contains URLs `http://mysite.com?name`$_1$`=hello` and `http://mysite.com?name`$_2$`=goodbye`, $\mathbf{U_{name,value}}$ *does not* contain `http://mysite.com?name`$_1$`=hello&name`$_2$`=goodbye`.

We define $\mathbf{U^2_{name,value}}$ as the set of all possible size 2 subsequences of URLs $\langle u_i, strip_{n,v}(u_{i+1})\rangle$, where the subsequence $\langle u_i, u_{i+1}\rangle$ occurs in a test case of O. In addition, for any test case t=$\langle u_1\rangle \in O$ where $|t| < 2$, $\mathbf{U^2_{name,value}}$ also contains the sequence $\langle u_1\rangle$. $\mathbf{U^2_{name,value}}$ also contains $\langle u_{i+1}\rangle$ to ensure the names and values of the last URL are maintained. We strip off all names and values of the last URL in a sequence because we are interested in variable names and values responsible for altering control flow from one URL to another and at the same time contain the user session elements represented in the universe.

Similarly, $\mathbf{U^k_{name,value}}$ is the set of (1) all possible subsequences of size k, $\langle u_1, u_2, ..., strip_{n,v}(u_k)\rangle$, where $1 \le k \le p$, p is the length of the longest test case in O, and the subsequence $\langle u_1, u_2, ..., u_k\rangle$ occurs in a test case of O; (2) all $\langle u_k\rangle$, with names and values intact; (3) for any test case t=$\langle u_1, u_2, ..., u_i\rangle \in O$ where $|t| < k$, $\mathbf{U^k_{name,value}}$ contains the sequence $\langle u_1, u_2, ..., strip_{n,v}(u_i)\rangle$ and the subsequence $\langle u_i\rangle$.

Similarly, we can define $\mathbf{U^k_{name}} = \langle strip_v(u_1), strip_v(u_2), ..., strip_{n,v}(u_k)\rangle \cup \langle strip_v(u_k)\rangle \cup$ for any test case $t \in O$ such that $|t| < k$, the sequence $\langle strip_v(u_1), strip_v(u_2), ..., strip_{n,v}(u_i)\rangle \cup \langle u_i\rangle$; and $\mathbf{U^k_{base}} = \langle strip_{n,v}(u_1), strip_{n,v}(u_2), ..., strip_{n,v}(u_k)\rangle \cup$ for any test case $t \in O$ where $|t| < k$, the sequence $\langle strip_{n,v}(u_1), strip_{n,v}(u_2), ..., strip_{n,v}(u_i)\rangle$. Figures 3(f) and (g) show the sets $\mathbf{U^2_{base}}$ and $\mathbf{U^2_{name}}$, respectively.

Our criteria are defined for a triple (W, O, $\mathbf{T}$), where the original suite O represents the universe of test cases containing all requested URLs for application W, and test suite $\mathbf{T} \subseteq O$. We presently define six basic coverage criteria for web applications: `single_URLs`, `URL_seq2`, `URL_seqk`, `URL_names`, `URL_seq2_names`, `URL_seqk_names`, `URL_names_values`, `URL_seq2_names_values`, and `URL_seqk_names_values`.

4.1 Single_URLs

The `single_URLs` coverage criterion requires all URLs in $\mathbf{U_{base}}$ to be covered at least once by the test suite. URLs are a unique attribute of web applications and we believe URLs to be similar to basic blocks in traditional programs.

[2]Line numbers shown in the figure are for ease of reference.

We assert that a relation exists between a test suite that covers all the URLs in a web application and its program coverage and fault detection capabilities.

Definition of Satisfaction: A triple (W, O, $\mathbf{T}$) satisfies `single_URLs` iff for all URLs $u \in \mathbf{U_{base}}$, $\exists$ a test case $t \in \mathbf{T}$ that contains a URL u' such that $strip_{n,v}(u')$=u.

For example in Figure 3, a test suite satisfying `single_URLs` would contain all the elements of $\mathbf{U_{base}}$ (Figure 3(c)).

4.2 URL_seq2

The second coverage criterion we propose seeks to capture control flow between URLs during application usage by including URL subsequences of size 2.

Definition of Satisfaction: A triple (W, O, $\mathbf{T}$) satisfies `URL_seq2` iff for all sequences $\langle u_1, u_2 \rangle \in \mathbf{U^2_{base}}$, $\exists$ a test case $t \in \mathbf{T}$ containing a subsequence $\langle u'_1, u'_2 \rangle$ such that $\langle strip_{n,v}(u'_1), strip_{n,v}(u'_2) \rangle = \langle u_1, u_2 \rangle$.

For example in Figure 3, we would expect a test suite satisfying `URL_seq2` to contain all the elements of $\mathbf{U^2_{base}}$ (Figure 3(f)). Similarly, we can define the `URL_seqk` criterion:

Definition of Satisfaction: A triple (W, O, $\mathbf{T}$) satisfies `URL_seqk` iff for all sequences $\langle u_1, u_2, ..., u_k \rangle \in \mathbf{U^k_{base}}$, $\exists$ a test case $t \in \mathbf{T}$ containing a subsequence $\langle u'_1, u'_2, ..., u'_k \rangle$ such that $\langle strip_{n,v}(u'_1), strip_{n,v}(u'_2), ..., strip_{n,v}(u'_k) \rangle = \langle u_1, u_2, ..., u_k \rangle$.

A test suite satisfying sequence-based criteria intuitively tries to capture control flow occurring during actual application usage. As the length of the sequence increases, more control flow between URLs is captured.

4.3 URL_names

The `URL_names` coverage criterion incorporates data names (variable names) associated with each URL.

Definition of Satisfaction: A triple (W, O, $\mathbf{T}$) satisfies `URL_names` iff for all URLs $u \in \mathbf{U_{name}}$, $\exists$ a test case $t \in \mathbf{T}$ containing a URL u' such that $strip_v(u')$=u.

For example in Figure 3, the test suite satisfying `URL_names` would contain all the elements of $\mathbf{U_{name}}$ (Figure 3(d)).

4.4 URL_seq2_names

Next we define `URL_seq2_names`, which seeks to capture names of variables that alter URL control flow in the web application. A test suite satisfying sequence-based criteria with names intuitively tries to capture control flow dependent on names associated with a URL during actual application usage.

Definition of Satisfaction: A triple (W, O, $\mathbf{T}$) satisfies `URL_seq2_names` iff for all sequences, $\langle u_1, u_2 \rangle \in \mathbf{U^2_{names}}$, $\exists$ a test case $t \in \mathbf{T}$ containing a subsequence $\langle u'_1, u'_2 \rangle$, where $\langle strip_v(u'_1), strip_{n,v}(u'_2) \rangle = \langle u_1, u_2 \rangle$.

Note we strip names and values from the last URL (u_2) because we are interested in variable names responsible for altering control flow from u_1 to u_2. For example in Figure 3, a test suite satisfying `URL_seq2_names` would contain all the elements of $\mathbf{U^2_{name}}$ (Figure 3(g)). Similarly, we can define the `URL_seqk_names` criterion:

Definition of Satisfaction: A triple (W, O, $\mathbf{T}$) satisfies `URL_seqk_names` iff for all sequences, $\langle u_1, u_2, ..., u_k \rangle \in \mathbf{U^k_{names}}$, $\exists$ a test case $t \in \mathbf{T}$ containing a subsequence $\langle u'_1, u'_2, ..., u'_k \rangle$, where $\langle strip_v(u'_1), strip_v(u'_2), ..., strip_{n,v}(u'_k) \rangle = \langle u_1, u_2, ..., u_k \rangle$.

4.5 URL_names_values

The `URL_names_values` coverage criterion captures all the information traveling on a URL to determine the coverage of a test suite.

Definition of Satisfaction: A triple (W, O, $\mathbf{T}$) satisfies `URL_names_values` iff for all URLs, $u \in \mathbf{U_{name,value}}$, $\exists$ a test case $t \in \mathbf{T}$ that contains a URL u' such that $u' = u$.

For example in Figure 3, a test suite satisfying `URL_names_values` would contain all elements of $\mathbf{U_{name,value}}$ (Figure 3(e)). We believe a test set satisfying `URL_names_values` captures names and values of variables responsible for the dynamic behavior of a web application.

4.6 URL_seq2_names_values

The `URL_seq2_names_values` criterion requires test sets adequate in terms of URLs, names, and values.

Definition of Satisfaction: A triple (W, O, $\mathbf{T}$) satisfies `URL_seq2_names_values` criteria iff for all sequences of URLs, $\langle u_1, u_2 \rangle \in \mathbf{U^2_{name,value}}$, $\exists$ a test case $t \in \mathbf{T}$ that contains the sequence $\langle u'_1, u'_2 \rangle$ such that $\langle u'_1, strip_v(u'_2) \rangle = \langle u_1, u_2 \rangle$.

We believe a test set satisfying `URL_seq2_names_values` will strive to capture control flow as well as the names and values of variables responsible for changing URL control flow. Similar to our other criteria associated with sequences, we can define the `URL_seqk_names_values` criterion:

Definition of Satisfaction: A triple (W, O, $\mathbf{T}$) satisfies `URL_seqk_names_values` iff for all sequences of URLs, $\langle u_1, u_2, ..., u_k \rangle \in \mathbf{U^k_{name,value}}$ $\exists$ a test case $t \in \mathbf{T}$ that contains the sequence $\langle u'_1, u'_2, ..., u'_k \rangle$ such that $\langle u'_1, u'_2, ..., strip_v(u'_k) \rangle = \langle u_1, u_2, ..., u_k \rangle$.

The coverage criteria we define exhibit a *strict* subsumption relationship. In the next section, we present proofs in support of the subsumption relations.

5. SUBSUMPTION RELATION

A test coverage criterion C_a subsumes coverage criterion C_b ($C_a \longrightarrow C_b$) iff every triple (W, O, $\mathbf{T}$) that satisfies C_a also satisfies C_b; where W is the web application, O represents the universe of requested URLs for W, and test suite $\mathbf{T} \subseteq O$. Figure 4 presents the subsumption hierarchy of the coverage criteria we presented in Section 4. The edges are labeled with the theorem number that proves the relation.

Lemma 1: If test case $t \in \mathbf{T}$ contains a sequence $s_k = \langle u_1, u_2, ..., u_k \rangle \in \mathbf{U^k_{name,value}}$, then t also contains $\langle strip_v(u_1), strip_v(u_2), ..., strip_{n,v}(u_k) \rangle \in \mathbf{U^k_{name}}$ and $\langle strip_{n,v}(u_1),$

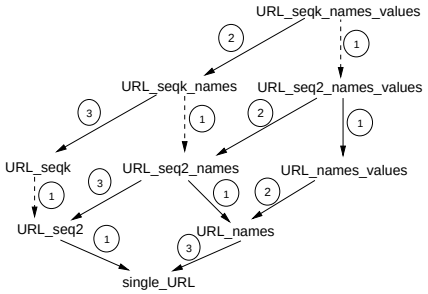


Figure 4: Subsumption Relation Between Criteria

$strip_{n,v}(u_2)$, ..., $strip_{n,v}(u_k)\rangle \in \mathbf{U^k_{base}}$, where $1 \leq k \leq p$ and p is the longest test case in O. The proof is straightforward and not shown here due to space constraints.

Theorem 1:
`URL_seqk_names_values` $\longrightarrow$ `URL_seq(k-1)_names_values`
Proof: By definition, a test suite, $\mathbf{T}_k$ satisfying `URL_seqk_names_values` contains all test cases of length $< k$ and all subsequences of length k for test cases of length $\geq k$. A test suite, $\mathbf{T}_{k-1}$ satisfying `URL_seq(k-1)_names_values` by definition contains all test cases of length $< k-1$ and all subsequences of length $k-1$ for test cases of length $\geq k-1$. Let s_{k-1}=$\langle u_1,\ u_2,\ ...,\ u_{k-1}\rangle \in \mathbf{U^{k-1}_{name,value}}$. If $|s_{k-1}| < k-1$, then s_{k-1} is in a test case of length $< k-1 < k$ and by definition of satisfying `URL_seqk_names_values`, $\mathbf{T}_k$ contains s_{k-1}. Otherwise, s_{k-1} must be part of some subsequence of length k, $\langle u_x,\ u_1,\ u_2,\ ...,\ u_{k-1}\rangle$ or $\langle u_1,\ u_2,\ ...,\ u_{k-1},\ u_x\rangle$ in some test case of length $\geq k$. Since $\mathbf{T}_k$ contains all subsequences of length k for test cases of length $\geq k$, by definition `URL_seqk_names_values` $\longrightarrow$ `URL_seq(k-1)_names_values`.

To establish the *strict* subsumption relation, we now show that `URL_seq(k-1)_names_values` does not subsume `URL_seqk_names_values`. Consider the suite O=$\{t_1, t_2, t_3\}$ where $t_1 = \langle u_1, u_2, u_3\rangle$, $t_2 = \langle u_2, u_1, u_2\rangle$, and $t_3 = \langle u_1, u_2, u_1\rangle$. Let $k = 3$ and $\mathbf{T}$=$\{t_1, t_2\}$. Then $\mathbf{U^2_{name,value}} = \{\langle u_1, u_2\rangle, \langle u_2, u_3\rangle, \langle u_2, u_1\rangle\}$ and $\mathbf{U^3_{name,value}} = \{\langle u_1, u_2, u_3\rangle, \langle u_2, u_1, u_2\rangle, \langle u_1, u_2, u_1\rangle\}$. Therefore $\mathbf{T}$ satisfies `URL_seq2_names_values` but not `URL_seq3_names_values` because the subsequence $\langle u_1, u_2, u_1\rangle$ does not appear in any test case in $\mathbf{T}$.

Similar proofs can be derived for the subsumption relationships `URL_seqk_names` $\longrightarrow$ `URL_seq(k-1)_names` $\longrightarrow$ ... $\longrightarrow$ `URL_seq2_names`, `URL_seq2_names_values` $\longrightarrow$ `URL_names_values`, `URL_seqk` $\longrightarrow$ `URL_seq(k-1)` $\longrightarrow$ ... $\longrightarrow$ `URL_seq2`, `URL_seq2_names` $\longrightarrow$ `URL_names`, and `URL_seq2` $\longrightarrow$ `single_URLs`.

Theorem 2: `URL_seqk_names_values` $\longrightarrow$ `URL_seqk_names`
Proof: Suppose this does not hold. Then $\exists$ a test suite $\mathbf{T} \subseteq O$ for W that satisfies `URL_seqk_names_values` but not `URL_seqk_names`. Thus there exists at least one size k subsequence of URLs, s_k=$\langle u_1, u_2,\ ..., u_k\rangle \in \mathbf{U^k_{name}}$ and no test case in $\mathbf{T}$ contains the subsequence r_k=$\langle u'_1,\ u'_2,\ ...,\ u'_k\rangle \in \mathbf{U^k_{names,values}}$ such that $\langle strip_v(u'_1),\ strip_v(u'_2),\ ...,\ strip_{n,v}(u'_k)\rangle$=$s_k$. Since $\mathbf{T}$ satisfies `URL_seqk_names_values`, let t be the test case $\in \mathbf{T}$ that contains r_k. By Lemma 1, t also contains the subsequence $\langle strip_v(u'_1),\ strip_v(u'_2),\ ...,\ strip_{n,v}(u'_k)\rangle$=$s_k$. Therefore $\mathbf{T}$ satisfies `URL_seqk_names`, a contradiction.

App	# US	Tot URLs	Largest US (# URLs)	Avg US (# URLs)
Book	125	3640	160	29
CPM1	261	3881	152	15
CPM2	130	2005	152	15
CPM3	131	1876	97	14
CPM4	66	1469	152	22
CPM5	65	539	34	8
CPM6	65	729	95	11
CPM7	65	1144	97	17

Table 1: User Session Characteristics

To show the *strict* subsumption relation, we now show that `URL_seqk_names` does not subsume `URL_seqk_names_values`. Consider the suite O=$\{t_1, t_2\}$ where $t_1 = \langle u_1,\ u_1?\text{n}_1{=}\text{v}_1,\ u_2\rangle$ [3] and $t_2 = \langle u_1,\ u_1?\text{n}_1{=}\text{v}_2,\ u_2\rangle$. Let $k = 3$ and $\mathbf{T}$=$\{t_1\}$. $\mathbf{U^3_{name}}$=$\{\langle u_1,\ u_1?\text{n}_1,\ u_2\rangle\}$ and $\mathbf{U^3_{name,value}}$=$\{\langle u_1,\ u_1?\text{n}_1{=}\text{v}_1,\ u_2\rangle, \langle u_1,\ u_1?\text{n}_1{=}\text{v}_2,\ u_2\rangle\}$. Then $\mathbf{T}$ satisfies `URL_seq3_names` but not `URL_seq3_names_values` because no test case in $\mathbf{T}$ contains the subsequence $\langle u_1,\ u_1?\text{n}_1{=}\text{v}_2,\ u_2\rangle$.

Similar proofs can be derived for `URL_seq2_names_values` $\longrightarrow$ `URL_seq2_names` and `URL_names_values` $\longrightarrow$ `URL_names`.

Theorem 3: `URL_seqk_names` $\longrightarrow$ `URL_seqk`
Proof: Suppose this does not hold. Then $\exists$ a test suite $\mathbf{T} \subseteq O$ for W that satisfies `URL_seqk_names` but not `URL_seqk`. Thus there exists at least one size k subsequence of URLs, s_k=$\langle u_1,\ u_2, ..., u_k\rangle \in \mathbf{U^k_{base}}$ and no test case in $\mathbf{T}$ contains the subsequence r_k=$\langle u'_1, u'_2, ..., u'_k\rangle \in \mathbf{U^k_{names,values}}$ such that $\langle strip_{n,v}(u'_1), strip_{n,v}(u'_2), ..., strip_{n,v}(u'_k)\rangle$=$s_k$. Since $\mathbf{T}$ satisfies `URL_seqk_names`, let t be the test case $\in \mathbf{T}$ that contains r_k. By Lemma 1, t also contains the subsequence $\langle strip_{n,v}(u'_1),\ strip_{n,v}(u'_2),\ ...,\ strip_{n,v}(u'_k)\rangle$=$s_k$. Therefore $\mathbf{T}$ satisfies `URL_seqk`, a contradiction.

To show the *strict* subsumption relation, we now show that `URL_seqk` does not subsume `URL_seqk_names`. Consider the suite O=$\{t_1, t_2\}$ where $t_1 = \langle u_1,\ u_1?\text{n}_1{=}\text{v}_1,\ u_2\rangle$ and $t_2 = \langle u_1, u_1?\text{n}_2{=}\text{v}_2,\ u_2\rangle$. Let $k = 3$ and $\mathbf{T}$=$\{t_1\}$. $\mathbf{U^3_{base}}$ = $\{\langle u_1,\ u_1?\text{n}_1,\ u_2\rangle, \langle u_1, u_1?\text{n}_2,\ u_2\rangle\}$ and $\mathbf{U^3_{name}}$ = $\{\langle u_1,\ u_1?\text{n}_1,\ u_2\rangle, \langle u_1, u_1?\text{n}_2,\ u_2\rangle\}$. Then $\mathbf{T}$ satisfies `URL_seq3` but not `URL_seq3_names` because no test case in $\mathbf{T}$ contains the subsequence $\langle u_1,\ u_1?\text{n}_2,\ u_2\rangle$.

Similar proofs can be derived for `URL_seq2_names` $\longrightarrow$ `URL_seq2` and `URL_names` $\longrightarrow$ `single_URLs`.

6. CASE STUDIES

6.1 Subject Web Applications

For our empirical case studies, we used two subject programs: an open-source, e-commerce bookstore (Book) [6] (no. Classes: 11, no. Methods: 385, NCLOC: 7791, no. Seeded Faults: 40) and a course project manager (CPM) (no. Classes: 75, no. Methods: 172, NCLOC: 9300, no. Faults: 86), developed and first deployed at Duke University in 2001.

The bookstore allows users to register, login, browse for books, search for books by keyword, rate books, add books

[3]Name-values pairs follow '?' in the URL.

to a shopping cart, modify personal information, and logout. Since our interest was in testing user functionality, we did not include the administration code in our experiments. The bookstore uses JSP for its front-end and a MySQL database backend. To collect 125 user sessions for bookstore, we sent email to local newsgroups and posted advertisements in the university's classifieds web page asking for volunteer users. We removed image requests and requests that attempt to access any administration-related pages. URLs in the user sessions thus mapped directly to the 11 classes/JSP files of bookstore. Table 1 presents the characteristics of the collected user sessions.

In CPM, course instructors login and create *grader* accounts for teaching assistants. Instructors and teaching assistants set up *group* accounts for students, assign grades, and create schedules for demonstration time slots for students. CPM also sends emails to notify users about account creation, grade postings, and changes to reserved time slots. Users interact with an HTML application interface generated by Java servlets and JSPs. CPM manages its state in a file-based datastore. We collected 261 user sessions from instructors, teaching assistants, and students using CPM during the 2004-05 academic year at the University of Delaware. The URLs in the user sessions mapped to the application's 60 servlet classes and to its HTML and JSP pages.

For the fault detection experiments, graduate and undergraduate students familiar with JSP/Java servlets/HTML manually seeded realistic faults in bookstore and CPM. In general, the seeded faults inserted errors into the application's control flow, the generated UI web pages, and/or the datastore interactions.

6.2 Case Study 1: Longitudinal Analysis

Research Questions.
Question 1. How can we compare application usage under different test suites using our dynamic coverage criteria?
Question 2. How can this comparison be used to help a tester select between two test suites?

Variables and Measurements. The *independent variables* in our study are the test suites and the coverage criteria. The *dependent variables* are the percent frequency of URLs in the test suites being compared.

Experimental Setup. The framework for collecting user sessions is similar to what we used in previous papers [19, 20]. We separated CPM's test suites according to the semester in which they were recorded. We compared two test suites from the summer (58 user sessions) and the fall semesters (145 user sessions) with respect to two of our criteria `URL_seq2` and `URL_names_values`.

Data and Analysis. Figure 5 shows the usage of the application as depicted by two test suites that satisfy `URL_seq2` and `URL_names_values` criteria. For each test suite, we compute the ratio between the number of times a URL appears in each test suite and the total number of URL accesses in the suite and present the results in percentages. In both figures, the x-axis represents the percent frequency of URLs in the test suite from summer 2004, and the y-axis represents the percent frequency from fall 2004. Each point in the graph is a URL according to the application's respective criteria. We divide the figure into quadrants: URLs in Quadrants I and III indicate that application usage remained similar over time while URLs in Quadrants II and IV mean application usage changed.

Figure 5(a) suggests that the users across two semesters followed fairly similar paths in terms of size 2 sequences of URLs because only a few URLs are in Quadrants II and IV. Since most of the frequency values for URLs are low in Figure 5(b), it suggests that the user community (as identified by their name-value pairs) changed over the semesters, except for a few outliers. Using these graphs, testers can visualize the change in an application's operational profile and decide how to focus the testing effort.

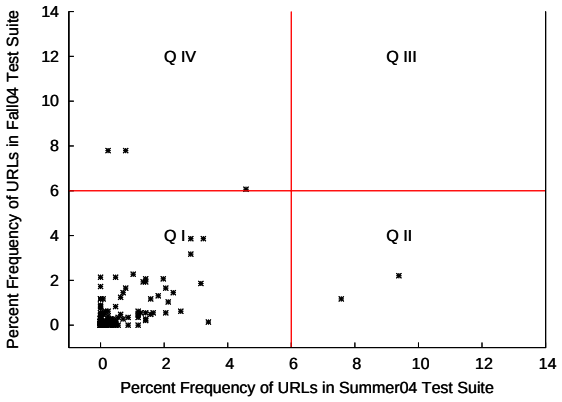
6.3 Case Study 2: Test Case Selection

Using our concept analysis based approach to test case selection, we examined the following questions with respect to `single_URLs`, `URL_seq2`, `URL_names`, `URL_names_values`, and `URL_seq2_names` criteria.
Research Questions.
Question 1. How does program coverage and fault detection effectiveness of a test suite that satisfies our dynamic coverage criteria vary?
Question 2. How do the time and space costs compare for reducing a test suite that satisfies the coverage criteria?

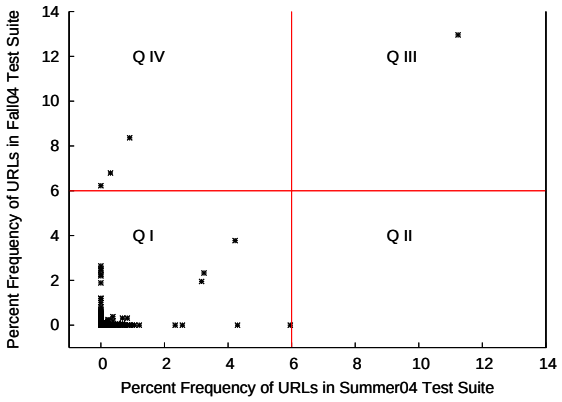
Variables and Measurements. The *independent variables* in our study are the user sessions, coverage criteria, test case selection heuristic, and subject web applications. The *dependent variables* are program coverage, fault detection effectiveness, test suite size, and cost of test selection.

Experimental Setup. The experimental setup for concept analysis-based reduction, program coverage, and fault detection studies are similar to the setup described in our previous papers [20, 19]. We collected 261 sessions for CPM over three academic semesters. In this study, we used the original bookstore and CPM sessions and divided the original 261 CPM sessions into six test suites to analyze the variation in effectiveness according to the different dynamic coverage criteria. Table 1 lists the test suites' characteristics.

Data and Analysis.
Effectiveness of Test Suite Selection. Figure 6 presents the percent reduction from applying the different criteria on the original suite. In our results we abbreviate the criteria `single_URLs` as **s1**, `URL_seq2` as **s2**, `URL_names` as **n**, `URL_names_values` as **n-v** and `URL_seq2_names` as **s2-n**. Figure 6 shows that as the complexity of the criteria increases, the percent reduction decreases. Tables 2 and 3 present the statement coverage and fault detection effectiveness of the reduced test suites. We present only statement coverage in Table 2 because method coverage remained constant across the different criteria. The loss in statements shown in Table 2 indicates the potential to cover more statements with increased criteria complexity until coverage equals the original suite's. In Table 3, we present fault detection results for the two test suites that showed the most variance across criteria. A summary of the fault detection results of the test suites not shown in Table 3 is as follows: the original suite



(a) URL_seq2



(b)URL_names_values

Figure 5: Comparing Test Suites to Understand Application Usage.

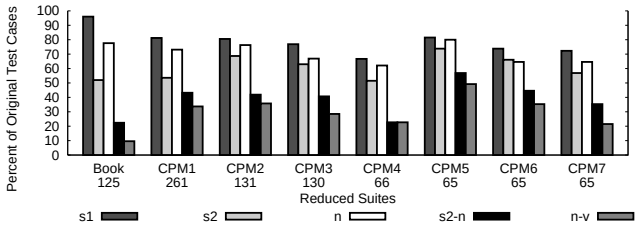


Figure 6: Reduction in Test Suite Size

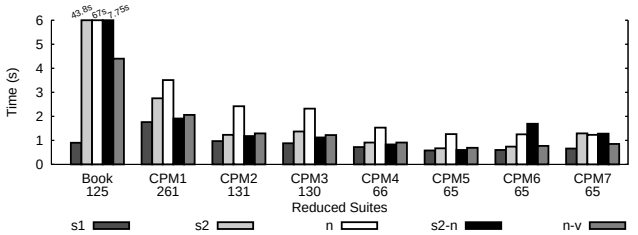


Figure 7: Time Costs

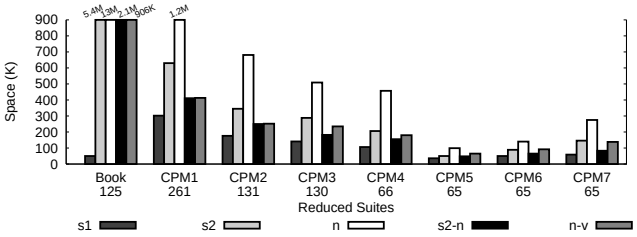


Figure 8: Space Costs

	% Total Code	# of Statements Lost				
TS	**Orig**	**s1**	**s2**	**n**	**s2-n**	**n-v**
Book	59.26	65	3	46	0	0
CPM1	69.28	113	42	58	35	0
CPM2	62.95	77	37	30	0	0
CPM3	65.01	91	62	42	14	0
CPM4	60.97	30	4	24	0	0
CPM5	41.09	21	7	14	0	0
CPM6	45.63	15	8	12	0	0
CPM7	58.36	60	56	23	22	0

Table 2: Statement Coverage Loss

for **CPM1** detects 79 of the 86 seeded faults, **CPM2** detects 79, **CPM3** detects 78, **CPM4** detects 79, **CPM6** detects 46, and **CPM7** detects 78 faults. When the dynamic coverage criteria are applied to these test suites, each reduced suite detected the same number of faults as the original. We speculate that the trends in fault detection capabilities are not conspicuous from our results because of how we seeded the faults. **Book** and **CPM5** show an increase in fault detection with increased criteria complexity, suggesting that user sessions selected for more complex criteria covered code that were populated with more faults.

Time and Space Costs in Selecting Test Suites. Figures 7 and 8 present the time for concept analysis to select test cases and the space requirements of the concept lattice. As the complexity of the coverage criteria increases, the time and space to select the test suite also increases. In most cases the time and space requirements follow the trend, **s1** $\leq$ **s2** $\leq$ **n**, and **s2-n** $\leq$ **n-v**. It is interesting to note that the time and space requirements for **Book** with `URL_names` criteria are greater than `URL_names_values` criteria. We believe the original test suite has many more use cases when it satisfies `URL_names` criterion than `URL_names_values` criterion. As a result, clustering by concept analysis creates a bigger lattice to represent the varied use cases.

`single_URLs`'s large percent reduction coupled with its high program coverage and fault detection effectiveness and low cost suggest that `single_URLs` criterion may be an effective criterion for user-session-based testing of web applications.

6.4 Analysis Summary

Comparing To Understand Application Usage. We believe that by comparing test suites, a tester can study the usage profile of an application. The tester can examine different test suites that satisfy dynamic coverage criteria and

TS	Orig	s1	s2	n	s2-n	n-v
Book	36	33	35	34	36	36
CPM5	32	27	27	27	32	32

Table 3: Number of Faults Detected

channel the testing effort. Test suites can also be compared to determine if, in addition to being adequate according to a program-coverage-based metric, the suites also satisfy our coverage criteria. We believe that a test suite adequate according to a program-based criterion and satisfying most of our coverage criteria is a good candidate for testing a web application.

Test Case Selection. The results presented in Figures 6, 7, 8 and Tables 2, 3 relate a test suite selected to satisfy a dynamic coverage criterion to other test criteria such as program coverage, fault detection, and cost effectiveness. We observe that as the complexity of the criteria increases, the reduction achieved decreases. Though the loss in statement coverage is relatively small across criteria, we believe that faults may exist in these statements that a complex criteria such as `URL_seq2_names` is more likely to detect than `single_URLs`. We make similar observations regarding the costs of reducing the test suite (Figures 7 and 8): as the criteria complexity increases, the time and space requirements increase. Our results show that a tradeoff exists between the reduced test suite size and the effectiveness of the test suite. Although the variance in effectiveness was not prominent for our subject applications, we believe distinct trends can be observed for larger, more dynamic web applications.

6.5 Threats to Validity

The main construct threat to validity of dynamic coverage criteria is that the universe of requirements is constantly changing, and the test suites are not guaranteed to cover all the requirements that may exist in a static model of the application. In our experiments, internal threats to validity arise from the lack of a large number of user sessions that constitute the requirements universe. Our applications may not be complex enough to show large differences in program coverage and fault detection when comparing the dynamic coverage criteria. Though we tried to model the seeded faults as closely as possible to naturally occurring faults, some of the seeded faults may not be accurate representations of natural faults, leading to an external threat to validity. The obvious conclusion threat to validity is that we conducted our experiments two applications and it might not be fair to generalize our results to all web applications.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we defined the notion of dynamic coverage criteria for testing web applications at a page level. We presented various uses of the criteria and evaluated their applicability in two case studies—we studied the change in operational profile of the application and test case selection using the criteria and concept analysis. Our results indicate that dynamic coverage criteria are relevant to web applications, due to dynamic nature of the applications, and that with an increase in complexity of the criteria, more statement coverage is achieved. The tradeoff however is the test suite size and the time and space requirements to generate the test suite. In the future we plan to apply our criteria to other web applications and investigate additional uses.

Acknowledgments. We thank Andrew Danner from Duke University for his helpful suggestions on Sections 4 and 5.

8. REFERENCES

[1] A. Andrews, J. Offutt, and R. Alexander. Incremental concept formation algorithms based on Galois (concept) lattices. *Soft and Sys Modeling (to appear)*, 2004.
[2] G. Birkhoff. *Lattice Theory*, volume 5. American Mathematical Soc. Colloq Pubs, 1940.
[3] M. El-Ramly and E. Stroulia. Analysis of web-usage behavior for focused web sites: A case study. *Journal of Soft Maint and Evol: Research and Pract*, 2004.
[4] S. Elbaum, S. Karre, and G. Rothermel. Improving web application testing with user session data. In *Int Conf on Soft Eng*, 2003.
[5] P. G. Frankl and E. J. Weyuker. An applicable family of data flow testing criteria. *IEEE Trans on Soft Eng*, 14(10):1483–1498, Oct 1998.
[6] Open source web applications with source code. ⟨http://www.gotocode.com⟩, 2003.
[7] M. J. Harrold, R. Gupta, and M. L. Soffa. A methodology for controlling the size of a test suite. *ACM Trans on Soft Eng and Meth*, 2(3):270–285, Jul 1993.
[8] I. Jacobson. The use-case construct in object-oriented software engineering. In J. M. Carroll, editor, *Scenario-based Design: Envisioning Work and Technology in System Development*, 1995.
[9] G. M. Kapfhammer and M. L. Soffa. A family of test adequacy criteria for database-driven applications. In *ESEC/FSE-11*, pp. 98–107. ACM Press, 2003.
[10] D. C. Kung, C.-H. Liu, and P. Hsia. An object-oriented web test model for testing web applications. In *Asia-Pacific Conf on Qual Soft*, pp. 111–120, 2000.
[11] C.-H. Liu, D. C. Kung, and P. Hsia. Object-based data flow testing of web applications. In *Asia-Pacific Conf on Qual Soft*, 2000.
[12] G. A. D. Lucca and M. D. Penta. Considering browser interaction in web application testing. In *Int Work on Web Site Evol*, Sept 2003.
[13] G. D. Lucca, A. Fasolino, F. Faralli, and U. D. Carlini. Testing web applications. In *Int Conf on Soft Maint*, 2002.
[14] A. M. Memon, M. L. Soffa, and M. E. Pollack. Coverage criteria for GUI testing. In *ESEC/FSE-9*, pp. 256–267. ACM Press, 2001.
[15] Parasoft WebKing. ⟨http://www.parsoft.com⟩, 2004.
[16] Rational Robot. ⟨http://www-306.ibm.com/software/awdtools/tester/robot⟩, 2003.
[17] RFC 1738: Uniform resource locators (url). ⟨ftp://ftp.isi.edu/in-notes/rfc1738.txt⟩, 1994.
[18] F. Ricca and P. Tonella. Analysis and testing of web applications. In *Int Conf on Soft Eng*, 2001.
[19] S. Sampath, V. Mihaylov, A. Souter, & L. Pollock. Composing a framework to automate testing of operational web-based software. In *Int Conf on Soft Maint*, Sep 2004.
[20] S. Sampath, V. Mihaylov, A. Souter, & L. Pollock. A scalable approach to user-session based testing of web applications through concept analysis. In *Auto Soft Eng Conf*, Sep 2004.
[21] S. Sampath, S. Sprenkle, E. Gibson, L. Pollock, and A. Souter. Analyzing clusters of web application user sessions. In *Int Work on Dyn Anal*, May 2005.
[22] J. Sant, A. Souter, and L. Greenwald. An exploration of statistical models of automated test case generation. In *Int Work on Dyn Anal*, May 2005.
[23] Web site test tools and site management tools. ⟨http://www.softwareqatest.com/qatweb1.html⟩, 2003.
[24] S. Sprenkle, S. Sampath, E. Gibson, L. Pollock, and A. Souter. An empirical comparison of test suite reduction techniques for user-session-based testing of web applications. Tech Report 2005-09, U of Delaware, 2005.
[25] P. Tonella, F. Ricca, E. Pianta, and C. Girardi. Evaluation methods for web application clustering. In *Int Work on Web Site Evol*, Sep 2003.
[26] H. Zhu, P. A. V. Hall, and J. H. R. May. Software unit test coverage and adequacy. 29(4), Dec 1997.