Analyzing Clusters of Web Application User Sessions

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

Amie Souter
Computer Science
Drexel University
Philadelphia, PA 19104
souter@cs.drexel.edu

ABSTRACT

User sessions provide valuable insight into the dynamic behavior of web applications. They also play a key role in user-session-based testing, which gathers user sessions in the field and replays selected sessions to test an evolving application. To decrease the testing and analysis effort, testers reduce the set of collected user sessions by either clustering user sessions by their shared URL attributes or by program coverage requirements-based reduction techniques. Clustering URL attributes can be considerably less expensive; however, the tradeoff may be that clustering is not representative of dynamic behavior similarities. This paper describes our analysis of user session data to reveal correlations between sessions clustered on the sessions' attributes and the relative dynamic behavior of the program for those sessions. The results of our analysis also motivate other clustering and test suite reduction techniques. Our results can also be used to learn more about how clusters of web application use cases are related in terms of the underlying user session attributes, program coverage, and fault detection.

1. INTRODUCTION

Web application code and web site usage evolve as diverse users have different expectations for communication and application functionality. User sessions represent actual user interactions with the application and can be used to learn about the dynamic behavior of web applications. Particularly during the beta testing and maintenance phases, user sessions gathered in the field can be used to create test suites and replay selected sessions to test an evolving application.

A *user session* is a collection of user requests in the form of a base URL and name-value pairs (input field names and values). We view user sessions as representative of an application's use cases [7]. A user session begins when a request from a new IP address reaches the server and ends when the user leaves the web site or the session times out. Testing with user sessions reflects current usage that testers may not have anticipated during earlier development stages. User sessions from real users in the field have been shown to complement test suites generated by testers in-house [4].

In previous work [12], we found that interesting usage patterns of a web application can be uncovered through concept analysis and common URL subsequence analysis. Analysis of user sessions revealed commonality in URL orderings between user sessions clustered by concept analysis, and these common subsequences cover a high percentage of each cluster's attributes. The results suggested that clustering on single URLs is reasonable for clustering similar use cases and choosing a representative user session from a given cluster does not lose the use cases or attributes covered by the other user sessions. We also observed that test suite reduction based on clustering user sessions by their shared URL attributes is competitive with program coverage requirements-based techniques in terms of reduced test suite size, program coverage, and fault detection [11, 14].

This paper describes our analysis of user session data to reveal correlations between user sessions clustered by their attributes and the relative dynamic behavior of the program for those user sessions. We investigate the relation between user sessions within the same cluster and their program coverage and fault detection capabilities. In particular, this paper provides the following contributions:

1. We define the notions of common program coverage and fault detection among clustered user sessions. We also define the concepts of common program coverage and fault detection across user sessions in different clusters.

2. We perform two case studies to assess the correlations between clustering based on user session attributes and the relative dynamic behavior of executed user sessions in terms of program coverage and fault detection.

3. Based on the results of analyzing clusters with our program-based and fault detection-based metrics, we propose alternate techniques to clustering and test suite reduction.

Our results aid in evaluating clustering by user sessions' URL attributes, with respect to testing-related concerns: program coverage and fault detection. Our results also show that clustering on attributes provides similar clustering of program coverage and fault detection capabilities. Thus, testers can eliminate the expense of mapping user sessions to program coverage requirements to reduce test suites and perform on-the-fly test suite reduction with respect to user sessions' URL attributes. Our results can also be used to

Workshop on Dynamic Analysis (WODA 2005) 17 May 2005, St. Louis, MO, USA

formulate additional test-suite-reduction heuristics. Finally, our results give insight into the relationship between clusters of web application use cases in terms of underlying user session attributes, program coverage, and fault detection.

2. BACKGROUND: CONCEPT ANALYSIS

Concept analysis is a mathematical technique for clustering objects that have common discrete attributes [3]. Snelting first introduced the idea of concept analysis for use in software engineering tasks, specifically for configuration analysis [8]. Researchers have also applied concept analysis to evaluating class hierarchies [13], debugging temporal specifications [1], redocumentation [9], and test coverage data [2].

To apply concept analysis to user sessions of a web application, we define the objects to represent information uniquely identifying user sessions (i.e., test cases) and attributes to represent URLs. Figure 1(b) shows the *sparse* concept lattice for Figure 1(a)'s user sessions. For example in Figure 1(b), *node 3*'s objects are the sessions *us4, us6*, and the attribute set is *GDef, GReg, GLog, GShop, GBooks* (from the *full* representation of the lattice).

Given the original suite of user sessions, Lindig's `concepts` tool [10], written in C, creates a lattice that clusters sessions by their common requested URLs. We developed a heuristic based on the concept lattice for selecting a subset of user sessions to be maintained as the current test suite [11]. Our heuristic for user-session reduction, *test-all-exec-URLs*, seeks to identify the smallest set of user sessions covering all of the URLs executed by the original test suite. Our heuristic also captures the common URL subsequences of different use cases represented by the original test suite. The reduced 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 with the largest number of shared attributes, and the union of the attribute sets covers all URLs in the original test suite. Thus, test suite reduction through our heuristic exploits the concept lattice's hierarchical clustering properties. In Figure 1(b), the *next-to-bottom* nodes are *node 4* and *node 5*. Applying our heuristic, the reduced test suite is {*us2, us6*}.

Though using concept analysis to reduce a test suite may not yield the minimum test suite for a given criterion, we believe our reduction technique maintains the original suite's use case representation in the reduced test suite [12]. Our previous papers [12, 11] contain details on applying concept analysis and our heuristic for test suite reduction.

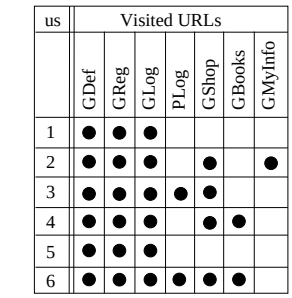
3. DYNAMIC BEHAVIOR OF CLUSTERS

Our goal is to uncover correlations between clustering user sessions based on common URLs and the relative dynamic behavior of those user sessions. The research questions we target in this paper are

1. How does clustering user sessions in a concept node relate to the program code covered by the node's sessions?

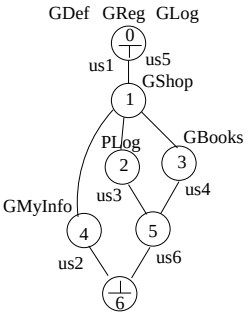
2. Do user sessions clustered together in a concept node detect similar faults in the code?

[1] $\perp$ contains URLs that all the user sessions request.



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

us	Methods Covered
us2	{m2, m3, m11, m13, m14, m15, m18}
us4	{m2, m3, m4, m5, m6, m7}
us6	{m2, m3, m4, m5, m6, m17, m18}

(c) Methods Covered by Sessions

us	Faults Detected
us2	{f1, f2, f3, f6, f7, f14, f15}
us4	{f1, f4, f5, f6, f7}
us6	{f1, f4, f5, f6, f17, f12, f13}

(d) Faults Detected by Sessions

Figure 1: Example

3. What overlap occurs in terms of program coverage and fault detection between *next-to-bottom* nodes? Are the covered program code and detected faults distinct between *next-to-bottom* nodes?

Our hypotheses in regard to these questions are

1. Concept analysis clusters together user sessions that are similar in terms of their attribute sets. We expect attribute set similarity to translate into similar program code coverage and similar fault detection by the node's sessions. As the attribute set size increases, we expect common program coverage and common faults detected by a node's sessions to increase.

2. Each cluster represents a different set of use cases—as measured by its program coverage and fault detection capability. By definition of clustering by concept analysis, each node in the lattice has some unique attribute. A quick analysis revealed that each *next-to-bottom* node with sessions containing non-static URLs represents unique dynamic program behavior characteristics. Furthermore, we expect program coverage and fault detection to be different across *next-to-bottom* nodes with little overlap between the nodes.

We use three different analyses to examine the relationship between clustered user sessions and their program coverage and fault detection.

3.1 Overlap Within a Cluster

The first analysis examines every cluster generated by concept analysis and determines if the sessions clustered into each node cover common program code and detect common faults, in addition to possessing common URLs. We define program coverage overlap, *common_cov*, for a node n with the set of user sessions $\{us_{n1}, us_{n2}, ..., us_{nm}\}$ as

$$common\_cov(n) = \left| \bigcap_{1 \le i \le m} coverage(us_{ni}) \right|$$

where $coverage(us_{ni})$ contains the program methods, branches, or statements covered by user session us_{ni}.

Similarly, the fault detection overlap, *common_fault*, for a node n with the set of user sessions $\{us_{n1}, us_{n2}, ..., us_{nm}\}$ is

$$common\_fault(n) = \Big| \bigcap_{1 \le i \le m} fault\_detection(us_{ni}) \Big|$$

where $fault\_detection(us_{ni})$ contains the faults detected by user session us_{ni}. We use the terms *common program coverage* and *program coverage overlap* interchangeably. Likewise, for fault detection, we use *common faults detected* and *fault detection overlap*.

For example, the methods covered by *us4* and *us6* of *node 3* are shown in Figure 1(c). Both sessions cover methods *m2, m3, m4, m5, m6*. Thus, the program coverage overlap for *node 3* is five. The faults detected by all user sessions in *node 3*, as shown in Figure 1(d), are *f1, f4, f5, f6*. Thus, the fault detection overlap for *node 3* is four.

3.2 Visualizing Next-to-bottom Clusters

The second analysis involves visualizing method coverage and fault detection obtained by executing the sessions in the *next-to-bottom* nodes. The visualization motivates the need for quantifying overlap across *next-to-bottom* nodes. By illustrating the diversity among *next-to-bottom* nodes in terms of their method coverage and fault detection, this analysis aids in determining whether these nodes represent different sets of use cases.

3.3 Overlap Across Clusters

The third analysis examines the dynamic behavior similarities across *next-to-bottom* nodes. We examine whether the clusters of sessions in *next-to-bottom* nodes represent distinct use cases and thus cover a different set of methods in the program code and detect different faults.

For a pair of concept nodes, $p = \{us_{p1}, us_{p2}, ..., us_{pm}\}$ and $q = \{us_{q1}, us_{q2}, ..., us_{qn}\}$, we define common program coverage between concept nodes as

$$internode\_common\_cov(p,q) = \Big| \Big(\cap_{1 \le i \le m} coverage(us_{pi}) \Big) \cap \Big(\cap_{1 \le j \le n} coverage(us_{qj}) \Big) \Big|$$

Similarly, the common faults detected between concept nodes is defined as

$$internode\_common\_fault(p,q) = \Big| \Big(\cap_{1 \le i \le m} fault\_detection(us_{pi}) \Big) \cap \Big(\cap_{1 \le j \le n} fault\_detection(us_{qj}) \Big) \Big|$$

We compute *internode_common_cov(p,q)* and *internode_common_fault(p,q)* for all possible pairs of *next-to-bottom* nodes. Common intranode program coverage and fault detection for each *next-to-bottom* node identifies the representative program coverage and fault detection of a node. In contrast, *internode_common_cov* and *internode_common_fault* identify whether nodes cover similar code regions and detect similar faults, respectively.

In Figure 1, *node 4* and *node 5* are the *next-to-bottom* nodes. Figure 1(c) shows the methods covered by user sessions *us2* of *node 4* and *us6* of *node 5*. When a *next-to-bottom* node contains a single session, we consider the covered program code and the faults detected by that session. In this example, the methods covered by sessions *us2* and *us6* are *m2* and *m3*; thus, the *internode_common_cov* is two. For the example in Figure 1 (d), the common faults are the intersection of faults detected by nodes *4* and *5*. Both nodes detect faults *f1, f6*; thus, the *internode_common_fault* is two.

Metrics	Bookstore	Scheduler
Classes	11	75
Methods	385	172
NCLOC	7791	9298
Seeded Faults	40	86
Number of User Sessions	125	251
Total URLs Requested	3640	3260
Largest User Session	160 URLs	155 URLs
Average User Session	29 URLs	13 URLs

Table 1: Objects of Analysis

4. EXPERIMENTAL STUDY

The *independent variables* in our study are the user sessions and attributes of each concept node and the subject web applications. The *dependent variables* are program coverage and the faults detected.

We used the framework from [14] to measure each session's program coverage and detected faults. Based on the collected data, we compute the common program coverage and fault detection for sessions within each concept node and across *next-to-bottom* nodes. We describe our methodology's application-specific details in the following case studies.

4.1 Case Study 1: Bookstore

Table 1 shows the characteristics of our first subject program: an open-source, e-commerce Bookstore [5]. Bookstore allows users to register, login, browse for books, search for books by keyword, rate books, add books to a shopping cart, modify personal information, and logout. Since our interest is in user sessions, we did not include the administration code in our experiments. Bookstore uses a HTML interface generated by JSPs as the front-end and MySQL database backend. To collect 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 collected 125 user sessions and removed image requests and requests to access any administration-related pages. Table 1 also presents characteristics of the user sessions.

For the fault detection experiments, graduate and undergraduate students familiar with JSP/Java servlets/HTML manually seeded realistic faults in Bookstore. Because Bookstore lacks real-time dynamic content, such as time-dependent content, we use the simple oracle from our previous experiments [11, 14] for the fault detection experiments.

4.1.1 Data and Analysis

Overlap Within Clusters. Figures 2 and 3 show the values of *common_cov* and *common_fault* for all clusters (nodes) in the lattice for Bookstore. The clusters are ordered in ascending order along the x-axis by their attribute set sizes (number of common URLs). The y-axes show the *common_cov* and *common_fault* values, respectively, for each node. The lattice constructed for Bookstore contained 70 concept nodes with the largest attribute set size of 10.

We use box and whisker plots in Figures 2 and 3 because multiple nodes can have the same *number* of attributes. Hence we can have multiple values for *common_cov* and *common_fault* in each attribute set size. The increase in the me-

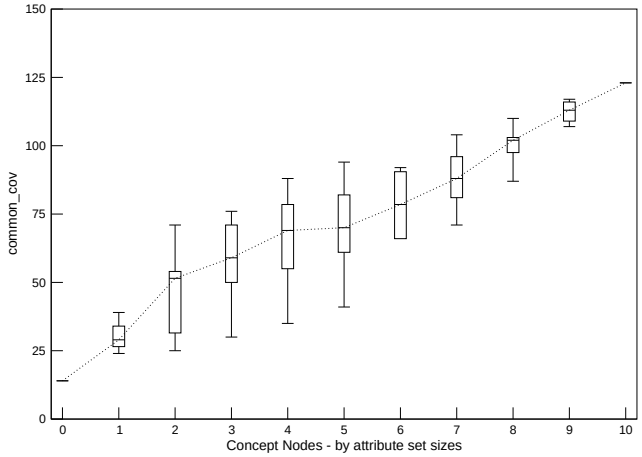


Figure 2: Bookstore: *common_cov* **Relation**

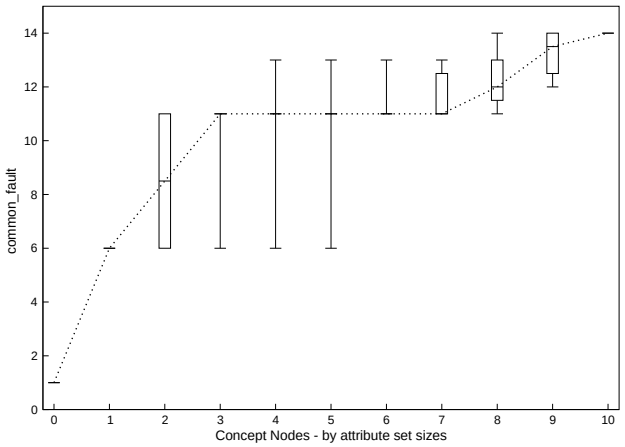


Figure 3: Bookstore: *common_fault* **Relation**

dian common program coverage and fault detection (shown by the dotted lines in Figures 2 and 3) indicates that as the attribute set size increases (move down lattice), both the common intranode program coverage and fault detection increase (Questions 1, 2). The *next-to-bottom* nodes appear in attribute set sizes 9 and 10. From the figures, we note that *next-to-bottom* nodes possess the maximum common program coverage and fault detection relative to all other nodes in the lattice. We believe the variance in fault detection overlap among clusters of the same attribute set size (Figure 3) is caused by at least one session with low fault detection that decreases the common faults detected within a node.

Visualizing Program Coverage and Fault Detection of Next-to-bottom Clusters. We also studied the *next-to-bottom* nodes' program coverage and fault detection for Bookstore; we do not present the graphs here due to space restrictions. As expected, the *next-to-bottom* nodes' total program coverage is equivalent to the original suite (from our previous results [11]). Unexpectedly, each *next-to-bottom* node covers almost the same program code. We believe this

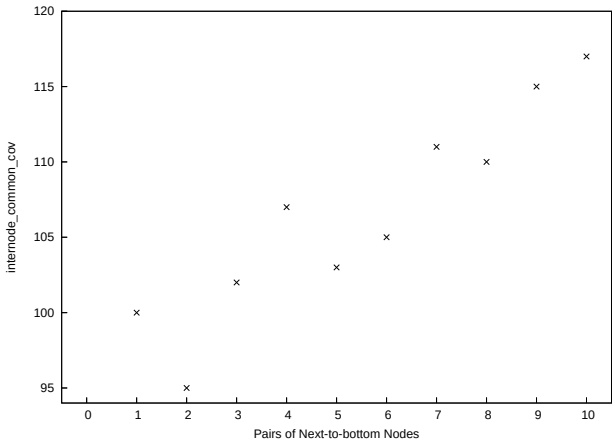


Figure 4: Bookstore: *internode_common_cov* **Relation**

is due to the simple nature of Bookstore. Since the object set of $\perp$ was not empty, all the sessions that belong to $\perp$ appear in the *next-to-bottom* nodes. In most cases, the *next-to-bottom* nodes differ only by a few sessions. The sessions in $\perp$, and hence the *next-to-bottom* nodes, contain all the URLs of the application and therefore cover most of the program code. Since the universe of methods to be covered is small and the possible user interactions in Bookstore are few, the *next-to-bottom* nodes appear to cover similar regions of program code.

In the fault detection study, the original 125 sessions detected 36 faults; the *next-to-bottom* nodes detected 34 faults. Further analysis revealed that only one session in the original suite detected the two missed faults, and these two sessions did not appear in any *next-to-bottom* node. Because of the server's JSP class instantiation, only the first session that accessed the fault page detected one of the missed faults. We attribute the loss of the second fault to not maintaining use case representation at a finer granularity, e.g., URLs with name-value pairs. Since the two sessions were not in the *next-to-bottom* nodes, the two faults were not detected. In addition, visualization of the fault detection of *next-to-bottom* clusters indicates the simple nature of Bookstore because all the *next-to-bottom* nodes detect the same faults.

Overlap Across Clusters. The results for comparing program coverage overlap and fault detection overlap across *next-to-bottom* nodes are presented in Figures 4 and 5. The x-axis denotes all possible pairs of *next-to-bottom* nodes. We ordered the pairs of nodes *(p,q)* on the x-axis in ascending order of $minimum(common\_cov(p), common\_cov(q))$ in Figure 4 and $minimum(common\_fault(p), common\_fault(q))$ in Figure 5 (in case of ties, pairs are ordered randomly). The y-axes represent *internode_common_cov* and *internode_common_fault* relations, respectively.

Consistent with our expectations, the overlap between *next-to-bottom* nodes is smaller than the program coverage and fault detection overlap within the nodes (with attribute set sizes 9 and 10 in Figures 2 and 3). Earlier we noted that Bookstore is simple with a small universe of methods in the code, small number of seeded faults, and few possible unique user interactions. Bookstore's simple nature results in all the *next-to-bottom* nodes covering similar methods and detecting similar faults and consequently increasing the overlap

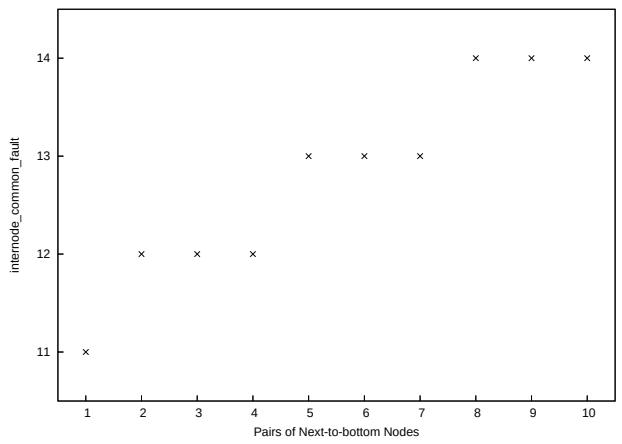


Figure 5: Bookstore: *internode_common_fault* **Relation**

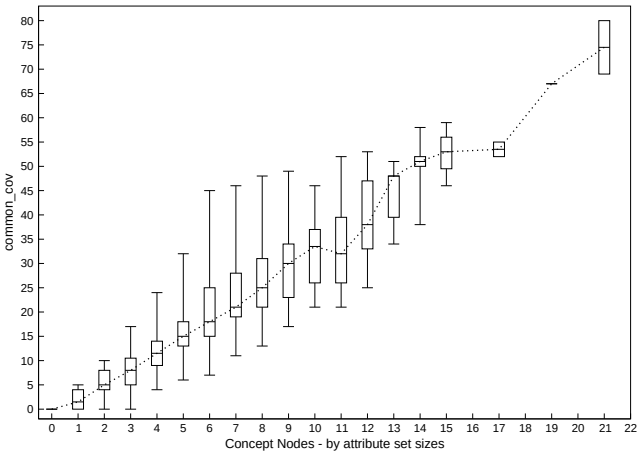


Figure 6: CPM: *common_cov* **Relation**

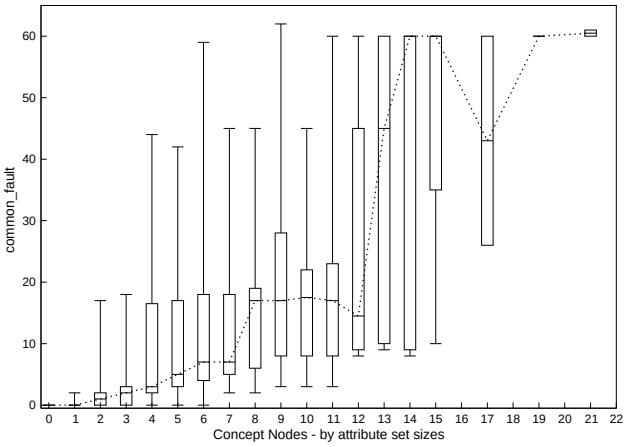


Figure 7: CPM: *common_fault* **Relation**

between the pairs. To gain more understanding on the relation between actual and expected values of common program coverage for the node pair (p, q), we define $universe\_cov$ as $\left|\left(\left(\cap_{1\leq i\leq m} coverage(us_{pi})\right) \cup \left(\cap_{1\leq i\leq n} coverage(us_{qi})\right)\right)\right|$ and expected common program coverage as

$$\left(|\left(\cap_{1\leq i\leq m} coverage(us_{pi})\right)| \,/universe\_cov\right) * \left(|\left(\cap_{1\leq i\leq n} coverage(us_{qi})\right)| \,/universe\_cov\right)$$

The actual common program coverage is calculated as the ratio between $internode\_common\_cov(p, q)$ and $universe\_cov$. Similarly, we computed expected and actual values for internode common faults detected. For the Bookstore, the actual value was always less than or equal to expected for internode common program coverage and common faults detected.

4.2 Case Study 2: Scheduler

Table 1 shows characteristics of our second subject program, Course Project Manager (CPM). CPM was developed and first deployed at Duke University in 2001[2]. 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 data store.

We collected 251 user sessions from instructors, teaching assistants, and students using CPM during the 2004 summer and fall semesters 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. Graduate students with JSP/Java servlets/HTML experience manually seeded realistic faults in CPM for the fault detection experiments. To handle real-time dynamic content for the fault detection studies, we use a newly designed conservative oracle.

[2]We thank Jeffrey Chase, Richard Kisley, and Sara Sprenkle for their development efforts.

4.2.1 Data and Analysis

Overlap Within Clusters. Figures 6 and 7 show the values of the *common_cov* and *common_fault* relations for all clusters (nodes) in the lattice for CPM. The axes are the same units as the graphs for Bookstore. CPM's lattice contained 745 nodes with (URL) attribute set sizes ranging from 0 to 22 (for nodes containing multiple sessions).

The general increase in the median (the dotted line in Figures 6 and 7) program coverage and fault detection overlap indicates that as the attribute set size increases, both intra-node program coverage and fault detection overlap increase (Questions 1, 2). Because each *next-to-bottom* node contained only one session, we do not show the *next-to-bottom* nodes in the figures for CPM, since the overlap is 100%.

We see low program coverage and fault detection overlap in Figures 6 and 7 for some of the higher attribute set sizes because our clustering does not include name-value pairs. Thus, while two sessions may request the same URLs, the name-value pairs may alter each session's control flow. The variance in program coverage overlap and fault detection overlap for attribute set sizes is caused by at least one user session with low program coverage or fault detection that reduces the overall overlap for a node.

Visualizing Program Coverage and Fault Detection

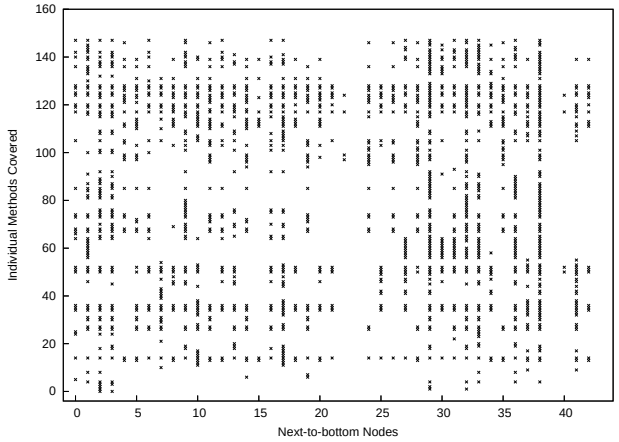


Figure 8: CPM: Visualizing Program Coverage of Next-to-bottom Clusters

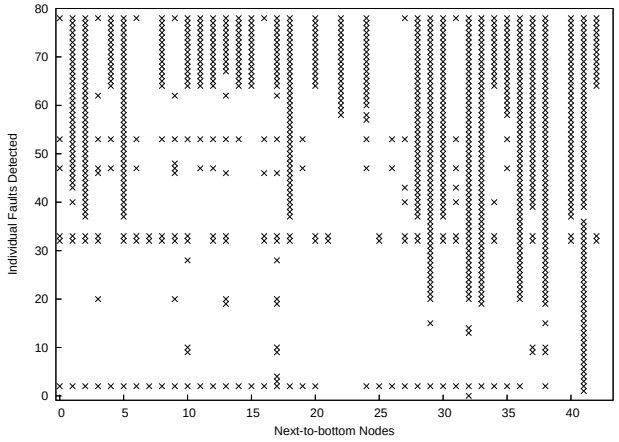


Figure 9: CPM: Visualizing Fault Detection of Next-to-bottom Clusters

of Next-to-bottom Clusters. Figures 8 and 9 present the program coverage and faults detected on executing *next-to-bottom* nodes. In Figures 8 and 9 the x-axis represents the *next-to-bottom* nodes and the y-axis denotes the application's methods or the faults seeded in the application, respectively, labeled by a random numbering. In contrast to Bookstore, the figures show that each *next-to-bottom* node covers a different set of methods and detects a different set of faults, as seen by the different y-values for each node. This implies that the sessions in *next-to-bottom* nodes emulate different user interactions, and thus different use cases. In Figure 8, nodes 23 and 39 cover no program code because the corresponding sessions contain only static HTML pages; however, the sessions are in the *next-to-bottom* nodes because they represent unique use cases. CPM's inherent complexity of possible unique user interactions is evident from these charts. The user sessions in the *next-to-bottom* nodes, as shown in Figure 9, detected the same 79 faults as the original suite.

Overlap Across Clusters. Figures 10 and 11 present our comparison of common program coverage and fault detection, respectively, between pairs of *next-to-bottom* nodes. The axes are similar to the graphs for Bookstore.

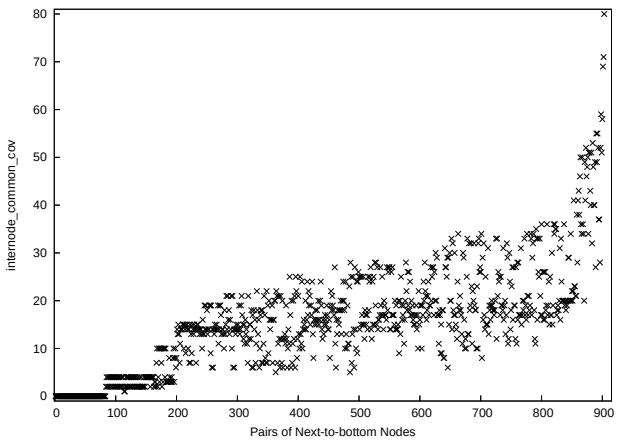


Figure 10: CPM: *internode_common_cov* Relation

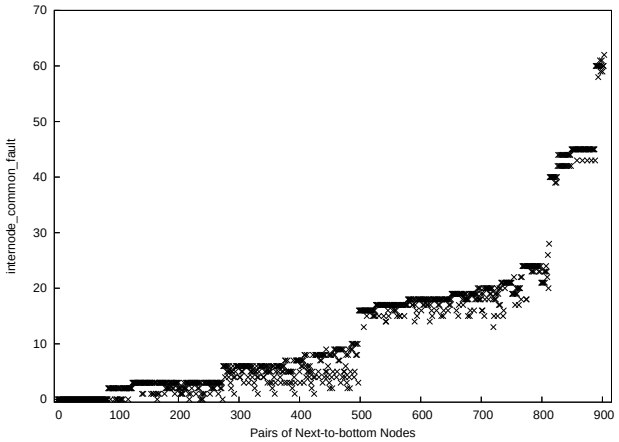


Figure 11: CPM: *internode_common_fault* Relation

Because of CPM's large size, complexity, and diverse user interactions, 43 nodes are in the lattice's *next-to-bottom* level. Figures 8 and 9 illustrate that the *next-to-bottom* nodes cover very few of the same methods and detect few of the same faults, respectively. The low common internode program coverage and detected faults for the *next-to-bottom* nodes in Figures 10 and 11 (majority have less than 30 common methods and faults, respectively) is due to the diversity in the individual *next-to-bottom* nodes' method coverage and fault detection. When common program coverage and detected faults are high across *next-to-bottom* nodes in Figures 10 and 11, the compared nodes also exhibit high intranode overlap. We computed the expected and actual values for common program coverage and common faults detected for CPM as described in Section 4.1. The actual value was always less than or equal to expected. Figures 10 and 11 also support our hypothesis that *next-to-bottom* nodes represent different use cases because the nodes cover different methods and detect different faults (Question 3).

4.3 Analysis Summary

Hypothesis 1. We described two case studies: a small open source e-commerce Bookstore application and a more complex course manager application. Both Bookstore's and

CPM's program coverage and fault detection results (Figures 2, 3, 6, and 7) support our first hypothesis: increasing attribute set size translates to increasing overlap in program coverage and fault detection within a cluster. We note the variance in program coverage and fault detection overlap for each attribute set size. Overlap depends on all sessions in each node; one session with low coverage/fault detection will reduce the node's common coverage/fault detection.

Hypothesis 2. For most pairs of CPM's *next-to-bottom* nodes, the internode program coverage and fault detection are much lower than each node's covered methods and faults detected. The low internode overlap supports our second hypothesis that each cluster represents a different set of use cases. The Bookstore's results did not support the second hypothesis because of the application's simplicity in both code and user interactions.

Towards A New Clustering Technique. Although user sessions within the same concept node share common URLs, our results show that sometimes clustered sessions may not have high program coverage or fault detection overlap because the program execution also depends on the name-value pairs of each URL. The observed low overlap for nodes with many common attributes motivates a more precise clustering approach based on user sessions' URLs and name-value pairs. Such a clustering is likely to generate smaller clusters with higher program coverage and fault detection overlap. However, the tradeoff for the more precise clustering is the cost of constructing and maintaining a lattice that incorporates name-value pairs.

Towards A New Heuristic for Test Suite Reduction. Figures 8 and 9 also motivate new heuristics for test suite reduction. A promising heuristic would be a hybrid approach which maintains use cases and fault detection as well as minimal test suite size. The selection heuristic proposed by Harrold et al. [6] (`HGS`) selects test cases that satisfy a requirement, e.g., method coverage. While `HGS` approximates the minimal reduced suite size, the reduced suite loses some of the original suite's use cases and therefore the suite's ability to detect faults. The reduction technique based on clustering user sessions by URL attributes captures use cases but does not reduce the suite as much as `HGS`.

4.4 Threats to Validity

While the bookstore application is similar in appearance and navigation to a real e-commerce application, users could not complete a monetary transaction. The reduced functionality may have simplified the pool of sessions collected, thus presenting an internal threat to validity. Our approach simplifies relating URLs to program coverage by ignoring the name-value pairs. Even though two sessions have common URLs, they may cover different regions of code because of different values. We believe a use case study considering sessions with URLs and name-value pairs abates this conclusion validity threat. The in-house nature of the subject applications and the data collection are threats to the external validity of our experiments.

5. CONCLUSIONS AND FUTURE WORK

Our previous work on common subsequence analysis [12] provided interesting results regarding dynamic usage patterns of web applications. By performing the user session analysis presented in this paper, we gain a better understanding of the dynamic behavior of clusters of user sessions from web applications. Our case studies demonstrate the trends in common program coverage and fault detection in user-session clusters created by concept analysis using single URLs as attributes. These results suggest new heuristics for test case reduction as well as an increased understanding of the relations between user sessions with common URL attributes and their program code coverage and fault detection capabilities. Future work includes exploring new clustering heuristics and examining the use cases associated with individual concept nodes.

Acknowledgements. We thank Madhusri Nayak at the University of Delaware for her assistance with one of the analyses.

6. REFERENCES

[1] G. Ammons, D. Mandelin, and R. Bodik. Debugging temporal specifications with concept analysis. In *ACM SIGPLAN Conf on Prog Lang Design and Implem*, 2003.

[2] T. Ball. The concept of dynamic analysis. In *ESEC / SIGSOFT FSE*, 1999.

[3] G. Birkhoff. *Lattice Theory*, volume 5. American Mathematical Soc. Colloquium Publications, 1940.

[4] S. Elbaum, S. Karre, and G. Rothermel. Improving web application testing with user session data. In *Int Conf on Soft Eng*, 2003.

[5] Open source web applications with source code. <http://www.gotocode.com>, 2003.

[6] 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, July 1993.

[7] I. Jacobson. The use-case construct in object-oriented software engineering. In J. M. Carroll, editor, *Scenario-based Design: Envisioning Work and Techn in Sys Dev*, 1995.

[8] M. Krone and G. Snelting. On the inference of configuration structures from source code. In *Int Conf on Soft Eng*, 1994.

[9] T. Kuipers and L. Moonen. Types and concept analysis for legacy systems. In *Int Workshop on Prog Compr*, 2000.

[10] C. Lindig. ftp.ips.cs.tu-bs.de:pub/local/softech/misc, 2002.

[11] S. Sampath, V. Mihaylov, A. Souter, and L. Pollock. A scalable approach to user-session based testing of web applications through concept analysis. In *Autom Soft Eng Conf*, Sept 2004.

[12] S. Sampath, A. Souter, and L. Pollock. Towards defining and exploiting similarities in web application use cases through user session analysis. In *Int Work on Dyn Anal*, May 2004.

[13] G. Snelting and F. Tip. Reengineering class hierarchies using concept analysis. In *SIGSOFT FSE*, 1998.

[14] 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, University of Delaware, 2005.