

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 as well as 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 reduce 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 based on URL attributes can be a considerably less expensive approach; however, the tradeoff may be that the clustering is not representative of dynamic behavior similarities. This paper describes our analysis of user session data to reveal any correlations between user sessions clustered on attributes of the user sessions themselves and the relative dynamic behavior of the program for those user sessions. The results of our analysis of user session clustering can be used to formulate test suite reduction techniques as well as 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 from around the world 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. Another use of user sessions is testing that involves creating test suites from user sessions gathered in the field and replaying selected sessions to test an evolving application, particularly during the beta testing and maintenance phases.

A *user session* is defined as a collection of user requests in the form of URL and name-value pairs and we view them as representative of the application’s use cases [8]. A specific 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 the test suites generated by testers in-house [5].

In previous work [12], we found that interesting usage patterns of a web application can be uncovered through concept analysis combined with common URL subsequence analysis. Analysis of the user sessions revealed that indeed there exists commonality in orderings of URLs between user sessions clustered by concept analysis and that these common subsequences cover a high percentage of the attributes of that cluster. In addition, the analysis suggested that clustering based on single URLs is reasonable for clustering similar use cases, and choosing one user session from a given cluster as the representative test case will not result in loss of the attributes covered or the use cases represented by other user sessions in the cluster. We also observed that test suite reduction based on clustering user sessions only by their shared URL attributes can be competitive with program coverage requirements-based techniques with respect to reduced test suite size, program coverage, and fault detection [11, 14].

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

1. We define the notions of program coverage and fault detection overlap among user sessions clustered on the basis of user session attributes, as well as overlap of common program coverage and fault detection among 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 a program using user sessions in terms of program coverage and fault detection.
3. Based on results of analyzing clusters with our program-based and fault detection-based metrics, we propose heuristics for test suite reduction.

Our results aid in evaluating the clustering of user sessions based on the sessions’ URL attributes, with respect to testing related concerns—program coverage and fault detection. Our results also show that clustering based on attributes provides similar clustering of program coverage and fault detection capabilities, and thus the expense of test suite reduction using mappings between user sessions and program coverage requirements can be eliminated, and on-the-fly test

suite reduction can be performed with respect to attributes of user sessions alone. In addition, our results can be used to formulate additional heuristics for test suite reduction as well as 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.

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 [9]. Researchers have also applied concept analysis to evaluating class hierarchies [13], debugging temporal specifications [1], redocumentation [10], and test coverage data [2].

To apply concept analysis to user sessions of a web application, we define the objects to represent the 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).

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 that will cover all of the URLs executed by the original test suite while representing the common URL subsequences of the different use cases that the original test suite represents. 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 that have 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* and on applying the heuristic, the reduced test suite is $\{us2, us6\}$.

Though using concept analysis to reduce the test suite may not result in the minimum test suite for a given criterion, we believe our reduction technique maintains the use case representation of the user sessions in the reduced test suite [12]. Details on the theory behind applying concept analysis and the heuristic for test suite reduction can be found in our previous papers [12, 11].

3. DYNAMIC BEHAVIOR OF CLUSTERS

Concept analysis is one form of clustering. Dickinson et al. [4] have utilized different cluster analysis techniques along with a failure pursuit sampling technique to uncover program failures. Our goal is to uncover any correlations between clustering user sessions based on common URLs and the relative dynamic behavior of those user sessions. The research questions that we target in this paper are

¹The \perp contains the URLs that all the user sessions request.

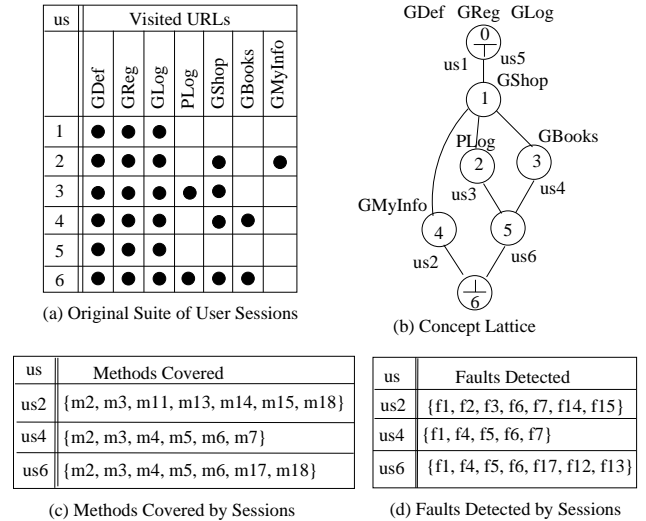


Figure 1: Example

Question 1. How does the clustering of user sessions within a concept node relate to the program code covered by the sessions of the node?

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

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

Our hypotheses in regard to these questions are

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

Hypothesis 2. Each cluster represents a different set of use cases—as measured by its program coverage and fault detection capability. We expect the program coverage and fault detection to be different across the *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 that concept analysis generates 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, *overlap_{pc}*, for a node *n* with the set of user sessions $\{us_{n1}, us_{n2}, \dots, us_{nm}\}$ as

$$overlap_{pc}(n) = \left| \bigcap_{1 \leq i \leq m} coverage(us_{ni}) \right|$$

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

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

$$overlap_fd(n) = \left| \bigcap_{1 \leq i \leq m} faultdetection(us_{ni}) \right|$$

where $faultdetection(us_{ni})$ contains the faults detected by user session us_{ni} .

For example, the methods covered by us_4 and us_6 of *node 3* are shown in Figure 1(c). Both sessions cover methods m_2, m_3, m_4, m_5, m_6 . Thus, the program coverage overlap between user sessions us_4 and us_6 in *node 3* is five. The faults detected by all user sessions in *node 3*, as shown in Figure 1(d), are f_1, f_4, f_5, f_6 . Thus, the fault detection overlap for *node 3* is four.

3.2 Visualizing Next-to-bottom Clusters

The second analysis involves visualizing the method coverage and fault detection obtained by executing the sessions in the *next-to-bottom* nodes. This analysis indicates the diversity among the *next-to-bottom* nodes, of methods covered and the faults detected by the *next-to-bottom* nodes and thus aids in determining whether the *next-to-bottom* nodes reflect clusters of different sets of use cases. The visualization also motivates the need for quantifying the overlap across *next-to-bottom* nodes.

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, $p = \{us_{p1}, us_{p2}, \dots, us_{pm}\}$ and $q = \{us_{q1}, us_{q2}, \dots, us_{qn}\}$, we define program coverage overlap between concept nodes as

$$internode_overlap_pc(p, q) = \left| \left(\bigcap_{1 \leq i \leq m} coverage(us_{pi}) \right) \cap \left(\bigcap_{1 \leq j \leq n} coverage(us_{qj}) \right) \right|$$

Similarly, the fault detection overlap between concept nodes is defined as

$$internode_overlap_fd(p, q) = \left| \left(\bigcap_{1 \leq i \leq m} faultdetection(us_{pi}) \right) \cap \left(\bigcap_{1 \leq j \leq n} faultdetection(us_{qj}) \right) \right|$$

We compute $internode_overlap_pc(p, q)$ and $internode_overlap_fd(p, q)$ for all possible pairs of *next-to-bottom* nodes. Intranode program coverage and fault detection overlap of each *next-to-bottom* node identifies the representative program coverage and fault detection of the node, while the $internode_overlap_pc$ and $internode_overlap_fd$ identifies whether the nodes cover similar code regions and detect similar faults, respectively.

For the example in Figure 1, *node 4* and *node 5* are the *next-to-bottom* nodes. Figure 1(c) shows the methods covered by user sessions us_2 of *node 4* and us_6 of *node 5*. In the absence of multiple sessions in the *next-to-bottom* nodes, we consider the covered program code and the detected faults by the

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

single session in the node. In this example, the methods covered by sessions us_2 and us_6 are m_2 and m_3 , thus the $internode_overlap_pc$ is two. For the example in Figure 1 (d), the common faults are the intersection of nodes 4 and 5's detected faults. Both nodes detect the faults f_1, f_6 , thus the $internode_overlap_fd$ is two.

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 the 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 program coverage overlap and fault detection overlap for the sessions within each concept node and across the *next-to-bottom* nodes. We describe application-specific details of our methodology 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 [6]. The 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. The bookstore uses JSP for its front-end and a 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 bookstore users. We collected 125 user sessions, all of which were used in these experiments. We removed image requests and requests that attempted to access any administration-related pages. Table 1 also presents the 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 the bookstore. Because the bookstore application 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 $overlap_pc$ and $overlap_fd$ for all clusters (i.e., concept nodes in the lattice of user sessions) for the bookstore application. The clusters are ordered in ascending order

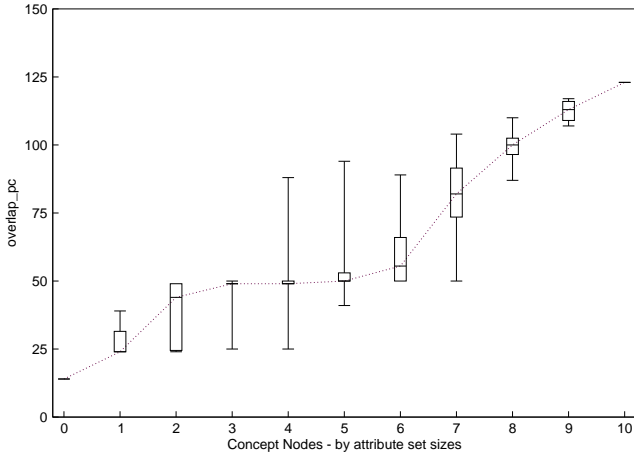


Figure 2: Bookstore: *overlap_pc* Relation

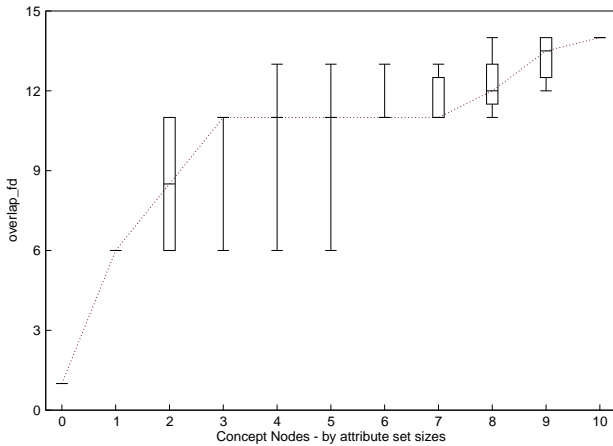


Figure 3: Bookstore: *overlap_fd* Relation

along the x-axis by their attribute set sizes. The y-axes show the *overlap_pc* and *overlap_fd* values, respectively, for each concept node. The lattice constructed for the bookstore application based on their common URL sets contained 70 concept nodes with the largest attribute set size of 10.

The increase in the median (as shown by the dotted line in Figures 2 and 3) in program coverage and fault detection overlap indicates that as the attribute set size increases, both the program coverage and fault detection overlap within the nodes increase (Question 1, Question 2). The *next-to-bottom* nodes appear in attribute set sizes 9 and 10 in Figures 2 and 3. From the figures it is evident that the *next-to-bottom* nodes possess the maximum program coverage overlap and fault detection overlap within the concept node relative to all the other nodes in the lattice. We believe that the variance in fault detection overlap (Figure 3) among clusters of the same attribute set size is due to the presence of atleast one session with low fault detection that decreases the intranode fault detection overlap.

Visualizing Program Coverage and Fault Detection

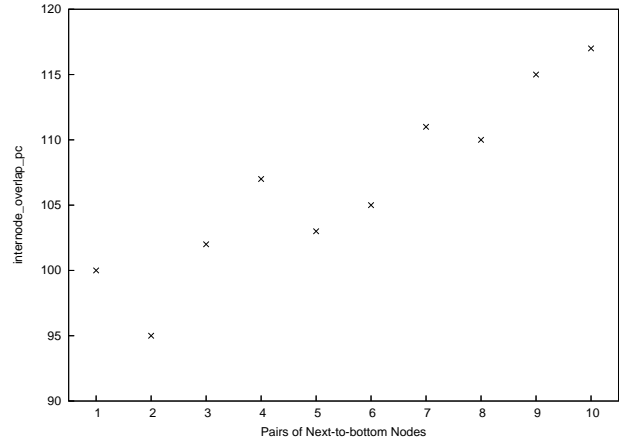


Figure 4: Bookstore: *internode_overlap_pc* Relation

of Next-to-bottom Clusters. We also studied the *next-to-bottom* nodes' program coverage and fault detection for the bookstore application (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]). We also note that each *next-to-bottom* node covers almost the same program code. In the bookstore, since the object set of \perp was not empty, all the sessions that belong to \perp appear in the *next-to-bottom* nodes. The *next-to-bottom* nodes thus differ only by a few sessions in most cases. The sessions in the *next-to-bottom* nodes contain all the URLs of the application and hence, cover most of the program code. Since the universe of methods to be covered is limited and the possible user interactions in the bookstore are few, the *next-to-bottom* nodes appear to cover similar regions of the program code.

In the fault detection study, the original suite of 125 sessions for the bookstore application detected 36 faults. The *next-to-bottom* nodes detected 34 faults. Our analysis reveals that the two missed faults were each detected by only one session in the original suite of sessions that did not appear in the *next-to-bottom* nodes. One of the faults was detected only by the first session that accessed the faulty page because of the server's JSP class instantiation. The other fault was detected by a session that manifested a use case absent in any other user session. The lack of the two sessions in the *next-to-bottom* nodes resulted in a failure to detect the faults. In addition, the visualization of the fault detection of *next-to-bottom* clusters indicates the simple nature of the application 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 the *next-to-bottom* nodes. We ordered the pairs of nodes (p, q) on the x-axis in ascending order of $\text{minimum}(\text{overlap_pc}(p), \text{overlap_pc}(q))$ in Figure 4 and $\text{minimum}(\text{overlap_fd}(p), \text{overlap_fd}(q))$ in Figure 5. The y-axes represent *internode_overlap_pc* and *internode_overlap_fd* relations, respectively.

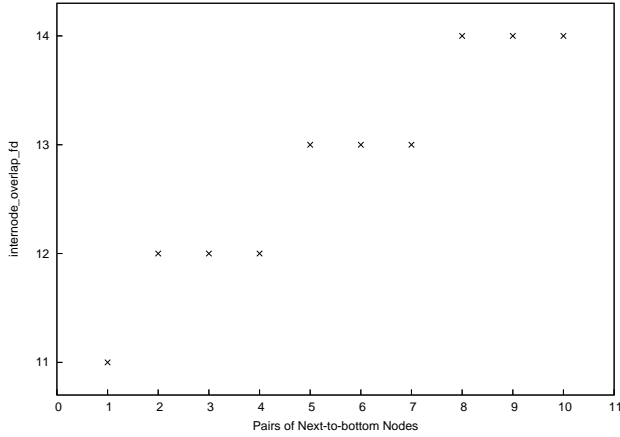


Figure 5: Bookstore: *internode_overlap_fd* Relation

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

4.2 Case Study 2: Scheduler

Table 1 shows the characteristics of our second subject program, a course project manager (CPM), also called the scheduler, developed and first deployed at Duke University in 2001².

In the scheduler, 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. The scheduler 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 sessions at the University of Delaware. The URLs in the user sessions mapped to the application’s 75 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 in the scheduler, we use a combination of the simple oracle from our previous experiments and a newly designed conservative oracle for the fault detection studies.

4.2.1 Data and Analysis

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

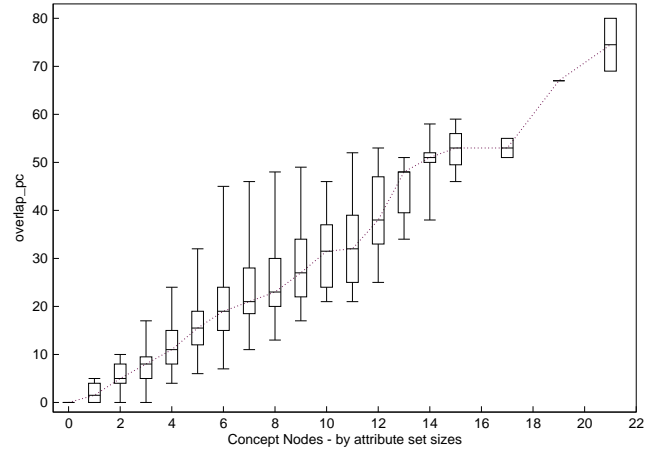


Figure 6: Scheduler: *overlap_pc* Relation

Overlap Within Clusters. Figures 6 and 7 show the values of the *overlap_pc* and *overlap_fd* relations for all clusters (i.e., concept nodes in the lattice of user sessions) for the scheduler application. The axes are the same units as the graphs for bookstore. The scheduler’s lattice contained 745 nodes with (URL) attribute set sizes ranging from 0 to 22 (for nodes with multiple sessions).

The increase in the median (as shown by the dotted line in Figures 6 and 7) in program coverage and fault detection overlap indicates that as the attribute set size increases, both the program coverage and fault detection overlap within the nodes increase (Question 1, Question 2). We do not show the *next-to-bottom* nodes in the figures because each *next-to-bottom* node contained only one session (we do not compute the program coverage and fault detection overlap when the node contains a single session). Because computing program coverage and fault detection requires the name-value pairs while our current application of concept analysis does not, we see low program coverage and fault detection overlap in Figures 6 and 7 for some of the higher attribute set sizes. While two sessions may request the same URLs, the name-value pairs are critical in determining each session’s control flow. Thus, although sessions are clustered together because of common URLs, their program coverage and fault detection may vary. The variance in program coverage and fault detection overlap for certain attribute set sizes is due to the presence of at least one user session with low program coverage or fault detection reducing the overall program coverage and fault detection overlap.

Visualizing Program Coverage and Fault Detection of Next-to-bottom Clusters. Figures 8 and 9 present the results for the program coverage procured and faults detected on executing the *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 the bookstore application, 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), implying that the sessions in the *next-to-bottom* nodes emulate different

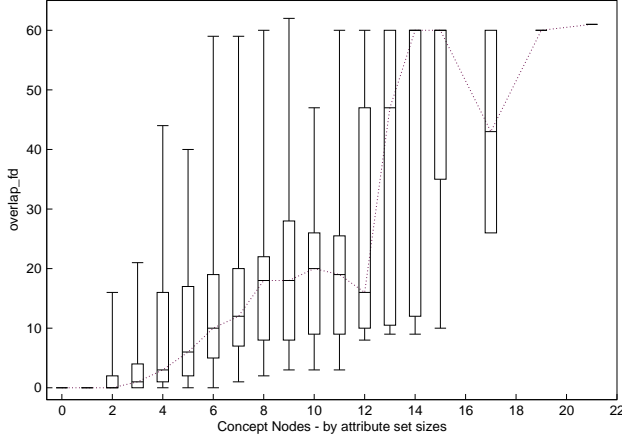


Figure 7: Scheduler: *overlap_fd* Relation

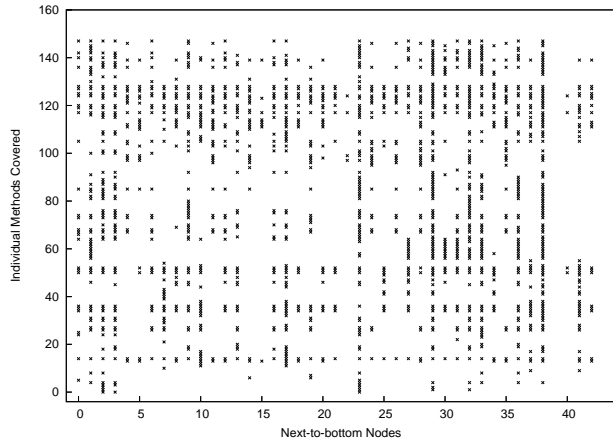


Figure 8: Scheduler: Visualizing Program Coverage of Next-to-bottom Clusters

user interactions, and thus different use cases. In Figure 8, node 39 covers no program code because its session contains only static HTML pages; however, the node is in the *next-to-bottom* nodes because it represents a unique use case. The scheduler’s inherent complexity of possible unique user interactions is evident from these charts. The scheduler’s original test suite of 251 session detects 79 faults. The user sessions in the *next-to-bottom* nodes, as shown in Figure 9, also detected 79 faults.

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

Because of the scheduler’s large size, complexity, and diverse user interactions, there are 43 nodes at the *next-to-bottom* level in the lattice. 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 internode overlap in program coverage and fault detection

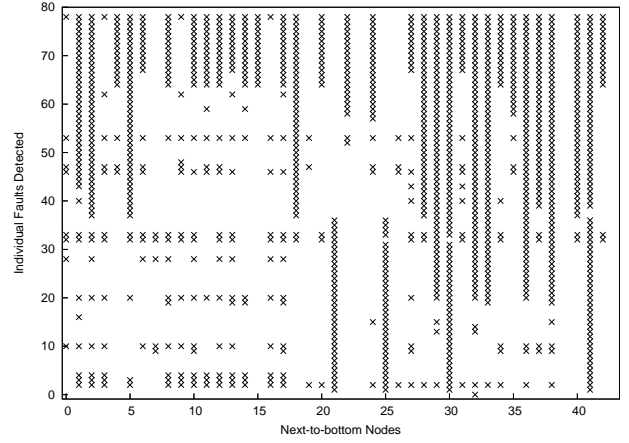


Figure 9: Scheduler: Visualizing Fault Detection of Next-to-bottom Clusters

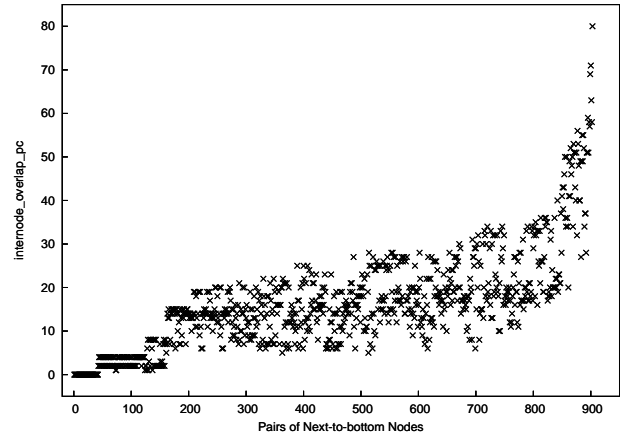


Figure 10: Scheduler: *internode_overlap_pc* Relation

for the *next-to-bottom* nodes in Figures 10 and 11 (majority overlap under 30 methods and faults respectively) is due to the diversity in the individual *next-to-bottom* nodes’ method coverage and fault detection. When there is high program coverage and fault detection overlap across *next-to-bottom* nodes in Figure 10 and 11, the compared nodes also have high intranode overlap in program coverage and fault detection. Figures 10 and 11 also support our hypothesis that *next-to-bottom* nodes represent different use cases as measured by the different methods that the nodes cover and the different faults the nodes detect (Question 3).

4.3 Analysis Summary

In this section, we described two case studies: a small open source e-commerce bookstore application and a more complex scheduler application. Both bookstore’s and scheduler’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 re-

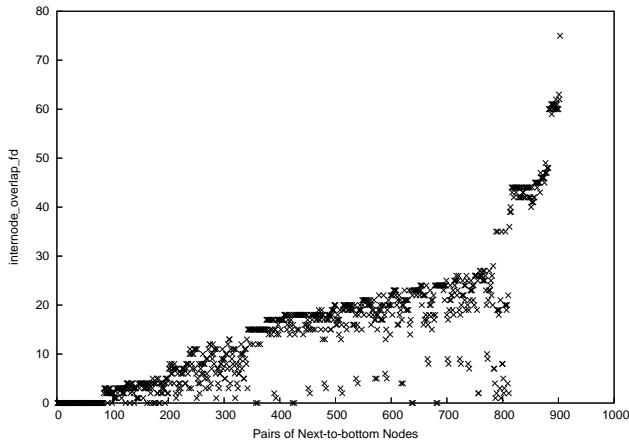


Figure 11: Scheduler: *internode_overlap_fd* Relation

duce the node’s common coverage/fault detection.

For most pairs of scheduler’s *next-to-bottom* nodes, the internode program coverage and fault detection are much lower compared to 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.

Although user sessions within the same concept node (i.e., cluster) share common URLs, our results show that the clustered sessions may not have high program coverage and fault detection overlap because the program execution also depends on the name-value pairs. 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.

Figures 8 and 9 also motivate new heuristics for test suite reduction. The selection heuristic proposed by Harrold et al. [7] (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. A promising approach would be a heuristic that is a hybrid approach which maintains use cases and fault detection as well as minimal test suite size.

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. However, even though two sessions have common URLs, they may cover different regions of code because of different values. We believe that a use case study that considers 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 program coverage overlap and fault detection overlap in clusters of user sessions 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 the relations between 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.

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] W. Dickinson, D. Leon, and A. Podgurski. Pursuing failure: the distribution of program failures in a profile space. In *ESEC/SIGSOFT FSE*, 2001.
- [5] S. Elbaum, S. Karre, and G. Rothermel. Improving web application testing with user session data. In *Int Conf on Soft Eng*, 2003.
- [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, July 1993.
- [8] 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.
- [9] M. Krone and G. Snelting. On the inference of configuration structures from source code. In *Int Conf on Soft Eng*, 1994.
- [10] T. Kuipers and L. Moonen. Types and concept analysis for legacy systems. In *Int Workshop on Prog Compr*, 2000.
- [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.