Web Application Fault Classification— An Exploratory Study

Yuepu Guo
Department of Information Systems
University of Maryland Baltimore County
Baltimore, Maryland 21250, USA
+1-410-455-8842
yuepu.guo@umbc.edu

Sreedevi Sampath
Department of Information Systems
University of Maryland Baltimore County
Baltimore, Maryland 21250, USA
+1-410-455-8845
sampath@umbc.edu

ABSTRACT
Controlled experiments in web application testing use seeded faults to evaluate the effectiveness of the testing technique. However, the classes of seeded faults are not always experimentally supported by real-world fault data. In this paper, we conduct an exploratory study on two large open source web systems to identify a fault classification that is representative of and supported by real world faults. Through our study we provide support to several categories of an existing web application fault classification, and identify one new fault category and six new sub-categories. Researchers and experimenters will find the proposed fault classification useful when evaluating techniques for testing web applications.

Categories and Subject Descriptors
D.2.0 [**Software Engineering**]: General

General Terms
Experimentation

Keywords
Web applications, fault classification, software testing

1. INTRODUCTION
New methods, models and techniques are developed for testing web applications because of their unique characteristics, such as (a) a large, diversified user group, (b) heterogeneous components, technologies, programming languages and operating environments, and (c) multiple entry points. One approach to evaluate the effectiveness of a new technique is to use fault-based software testing, where fault detection ability is used to evaluate the effectiveness of the testing methodology. Since a fault classification is a necessary foundation of fault detection empirical studies, a web application fault classification is required for studies on web applications.

While several studies have investigated testing techniques for web applications [1, 2], the fault classifications used in these studies for fault-based testing are not supported by empirical data from real world systems, and existing fault classifications for traditional systems [3] are not specifically tailored for web applications. The most widely used classification, the Orthogonal Defect Classification (ODC) presented by Chillarege et al. [3] where they analyzed defect data (the symptom and cause of defects) and used the causes of defects to determine the defect types is used to guide our study. However, the ODC is generic and thus cannot be applied directly to our study. Li and Tian's study [4] on web error logs to classify web application faults in terms of error attributes, such as response code from server logs and file type (HTML or JPEG) differs from our classification which is based on the physical location of a fault's occurrence. Marchetto et al. [5] proposed a web application fault taxonomy which they then refined using empirical data from several open source applications. Though we follow a different approach to the classification, some of our classes of faults are similar to theirs, thus providing further empirical support to the web application fault classification.

Sampath et al. [2] proposed a fault classification for web applications based on an initial classification proposed by Elbaum et al. [1]. In this paper, we build on the existing classification proposed by Sampath et al. [2] and provide experimental support to their classification by analyzing two large open source web systems and their bug repositories. In addition, we define additional fault categories that serve as an accurate categorization for a fault that does not appropriately fit into an existing category. Our fault classification is designed to inform researchers about the different types of faults that occur in large real world web applications. Based on an understanding of realistic faults, researchers conducting experiments in fault-based testing of web systems can seed faults of similar categories in their experimental applications.

The main contributions of this paper are: *(1) an exploratory study on two real world open source web systems to identify a fault classification scheme for web applications, (2) several new categories for web application faults that augment an existing fault classification scheme for web applications.*

2. WEB APPLICATION FAULT CLASSIFICATION METHODOLOGY
Methodology. Most taxonomies are developed as follows: (1) select a *dimension* that determines the perspective from which the faults will be categorized; (2) then specify the *baseline* on which

ESEM'08, October 9–10, 2008, Kaiserslautern, Germany.

the classification will be worked out; and (3) keep adding/adjusting the categories by analyzing data until the categories are exhaustive and detailed enough. We call this methodology the "induction" method because most of outcome comes from real data analysis. We use the induction methodology to define our fault classification scheme. We discuss the *dimension* and the *baseline* of our fault classification in the rest of Section 2 and our exploratory study in Section 3.

Classification dimension. Faults can be categorized in several dimensions, e.g., based on the cause of the fault or the lifecycle phase in which the fault was injected into the system. As mentioned in Section 1, running on different server platforms and web browsers is one of the unique characteristics which distinguish web applications from traditional software systems. This characteristic inspired us to choose *the physical location of a fault* as our *classification dimension*. A typical web application is multi-tiered, with a presentation tier (perceived in the client/browser), a logic tier (perceived in the application server) and a data tier (perceived in the database server). Based on the functionality of the three tiers, we begin with three broad fault categories—presentation, logic and data store faults.

However, in addition to the physical location of a fault, we also need to consider specific characteristics of web applications when deriving the fault classification. For example, an extra blank row in a table and a dead link displayed in web browser are both examples of presentation faults. However, the dead link fault can be observed only if the link is clicked by the user, whereas, the fault causing an extra blank row can be observed immediately after the page is displayed in a web browser. The two faults described above are different in terms of *when and where a fault is observed.* Therefore, sub categories are required to further distinguish between faults—we call the classification that we use as a source for the subcategories as our *baseline*.

Baseline. Since the classification developed by Sampath et al.[2] satisfies our goal with respect to dimension (location-dependence) and domain (web application), we selected their classification as the baseline of this study. They defined the fault categories as follows

- Data store faults: faults in the application code that manipulates data in any kind of data store.
- Logic faults: faults in the application code that implements business logic and control flow
- Form faults: faults in the application code that controls, modifies and displays name-value pairs in forms
- Appearance faults: faults in the application code that controls the way in which a web page is displayed
- Link faults: faults in the application code that changes the page pointed to by an URL

3. EXPLORATORY STUDY

Subject Selection. Considering the availability of defect data, we followed a two-stage screening process and selected web applications from sourceforge.net, the world's largest open source software development web site.

In the first stage, we selected applications based on the following criteria: a web-based Java application (since large (greater than 10K LOC) J2EE Java applications may contain different types of faults, and since we were familiar with Java, if source code analysis became necessary), with a bug tracking system, a source code repository and an activity index (a measure of how active a project is) equal to or greater than 70% (since systems with high activity indices are likely to have current and valid source code and fault data). Nine applications were short-listed. These applications were then screened according to number of developers, lines of code (LOC) and total number of bugs—all indicators of the size of the application. All criteria have equal weight. We also considered the number of open bugs (a reported bug that has not been fixed yet) and gave preference to applications with large closed bug counts because we found that developer comments can be used in addition to (sometimes vague) bug descriptions to correctly understand the bug. Also, once a bug is closed, conjecture regarding the fault is not necessary, hence avoiding any misunderstanding regarding the fault. Our criteria were designed to maximize the probability that the applications contain several kinds of faults, thus allowing for a generic fault classification.

After the two stage screening, we selected *Roller Weblogger* (No. of developers: 5, Activity Index: 79.1, Source LOC: 32848, Open bugs: 3, Total bugs: 104, Domain: Communications) and *qaManager* (No. of developers: 9, Activity Index: 99.9, Source LOC: 49030, Open bugs: 12, Total bugs: 161, Domain: Business) as our subjects of study. Roller Weblogger is a Java-based, full-featured, multi-user web logging system; qaManager, powered by OpenXava, is a platform-independent web-based application for managing quality assurance projects. The application handles functions such as project tracking, resource management, test cycle management, online library, and alerts.

Fault Analysis Procedure. In our study, we were interested in *what the faults are, where and how the faults are manifested and the physical location of the faults.* To gather this information, we first accessed the bug reporting system and then the code repository if more details about the fault were necessary. Then, we created a summary description of the faults and a tentative fault categorization. Totally, we examined 265 bugs in the two subject web applications. We use the following rules to classify each identified fault:

- Check the existing fault categories to see if the fault can be classified in one of them.
- If none of the existing categories are applicable to the fault, define a new category. For example, we found a fault which was caused because of a compatibility issue among IE 6 users. Since none of the existing categories were appropriate for this type of fault, we proposed a new category called **compatibility faults**.
- During the classification process, assess the degree of the detail of the category. If a category is too general, break it down further into a set of sub categories. For example, we found the **logic faults** category to be a broad category; therefore we defined sub categories in terms of the unique characteristics of web applications. Our fault categories are based on a web application's need to (a) allow for web pages to be displayed correctly when users manipulate the application through their *browser,* (b) maintain *session* information to remember the user across multiple HTTP requests, (c) allow for

proper *paging* when displaying documents that spread across multiple pages, (d) *parse* user entered data correctly at the *server side*, (e) *encode/decode* data so that data storage and transmission on the web is facilitated, (f) maintain *locale* information so that the page is displayed correctly to users across the world**.**

Proposed Web Application Fault Classification. Based on our study on the two open source web applications, we augment the existing fault classification proposed by Sampath et al. [7] with the following categories of faults:

- **Logic faults:** faults in the application code that implements business logic and control flow
  - **Browser interaction faults:** faults in the application code that controls the web browser, such as code that disables the "back" button on the browser, or code that is affected by user-defined browser settings, such as disabled cookies.
  - **Session faults:** faults in the application code that deals with maintaining state of application or other session-based operations such as using sessions to save and data entered into a form and display the data after the sessions has been validated.
  - **Paging faults:** faults in the application code that deals with paging when displaying large amounts of data on the screen
  - **Server-side parsing faults:** faults in the application code that deals with server-side parsing of HTML, XML and JavaScript tags
  - **Encoding/decoding faults:** faults in the application code that encodes or decodes characters for transmission, storage and display
  - **Locale faults:** faults that exist in application code that sets or gets locale-specific information, such as date format or language
  - **Other:** other logic faults that do not belong to any of the above sub categories
- **Compatibility faults:** faults in application code that ensures that the web application complies with different browsers, versions of browsers and other client environments.

Table 1 shows the number of faults found in each category for the two subject applications.

4. DISCUSSION

In this paper, we present an exploratory study of two open source web applications to identify fault classifications for web applications. Our study provides support to Sampath et al.'s fault classification[2], which was initially based primarily on their experience with developing, deploying and using web-based systems, and augments their classification with one new fault category and several new sub-categories.

The main purpose of our study is to facilitate experimentation in web application testing, especially fault-based testing. Our classification aims to associate faults with the uniqueness of web applications. Fault data from the two web applications helped us identify the elements which distinguish web applications from traditional software systems. Maintaining *sessions* and *paging* are unique for web applications due to the statelessness of HTTP. Cookies and client cache are related to web *browser interaction*. Diversified user group requires solutions to handle different languages, date convention and other *locale* setting. Thus, our fault categories embody the uniqueness of web applications. We still keep "other" as a sub category for logic faults that are common for web applications and traditional software systems. We envision web application testing researchers will use our proposed fault classification during the evaluation of their testing techniques, especially within the domain of fault-based testing.

In our study we did not find evidence of a fault that is present in Javascript or AJAX—technologies a web application typically uses to exhibit dynamic behavior in response to user input, but it is conceivable to find such faults in real world web systems. In the future, we plan to conduct a larger empirical study with more web applications implemented with technologies other than Java, so we can conduct statistical analysis on the fault data and generalize our results.

Table 1: Frequency of Faults

Fault Category	Number of Faults	
	Roller Weblogger	**qaManager**
Data Store	12	19
Appearance	13	24
Link	13	9
Form	1	4
Compatibility	3	6
Logic	62	99
• Browser interaction	4	4
• Paging	3	3
• Session	5	2
• Server-side parsing	6	0
• Encoding/decoding	4	2
• Locale	5	3
• Other	35	85
Total Faults	**104**	**161**

5. REFERENCES

[1] S. Elbaum, G. Rothermel, S. Karre, and Marc Fisher II, "Leveraging User-Session Data to Support Web Application Testing", IEEE Trans. on Software Engineering, vol. 31, pp. 1-16, 2005.

[2] S. Sampath, S. Sprenkle, E. Gibson, L. Pollock, and A. S. Greenwald, "Applying Concept Analysis to User-Session-Based Testing of Web Applications", IEEE Trans. On Software Engineering, vol. 33, pp. 643-657, 2007.

[3] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, "Orthogonal Defect Classification-A Concept for In-Process Measurements", IEEE Trans. on Software Engineering, vol. 18, pp. 943-956, 1992.

[4] M. Li, J. Tian, "Web error classification and analysis for reliability improvement", Journal of Systems and Software vol. 80, pp. 795-804, 2007.

[5] A. Marchetto, F. Ricca, P. Tonella, Empirical Validation of a Web Fault Taxonomy and its usage for Fault Seeding, In the IEEE Int. Symposium on Web Site Evolution, 2007.