Characterizing the JavaScript Errors that Occur in Production Web Applications

An Empirical Study

by

Frolin S. Ocariza, Jr.

B.A.Sc., The University of Toronto, 2010

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

The Faculty of Graduate Studies

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

April 2012

Abstract

Client-side JavaScript is being widely used in popular web applications to improve functionality, increase responsiveness, and decrease load times. However, it is challenging to build reliable applications using JavaScript. This work presents an empirical characterization of the error messages printed by JavaScript code in web applications, and attempts to understand their root causes.

We find that JavaScript errors occur in production web applications, and that the errors fall into a small number of categories. In addition, we find that certain types of web applications are more prone to JavaScript errors than others. We further find that both non-deterministic and deterministic errors occur in the applications, and that the speed of testing plays an important role in exposing errors. Finally, we study the correlations among the static and dynamic properties of the application and the frequency of errors in it in order to understand the root causes of the errors.

Preface

This thesis is an extension of an empirical study of JavaScript errors in production web applications conducted by myself in collaboration with Professor Karthik Pattabiraman and Benjamin Zorn. The results of this study were published as a conference paper on November 2011 in the International Symposium on Software Reliability Engineering (ISSRE) [22]. I was responsible for devising the experiments, creating test cases, running the experiments, evaluating and analyzing the results, and writing the manuscript. My collaborators were responsible for guiding me with the creation of the experimental methodology and the analysis of the results, as well as editing and writing portions of the manuscript.

F.S. Ocariza Jr, K. Pattabiraman, and B. Zorn. JavaScript Errors in the Wild: An Empirical Study. In Software Reliability Engineering (ISSRE), 2011 IEEE 22nd International Symposium on, pages 100-109. IEEE, 2011.

Table of Contents

Appendix

List of Tables

List of Figures

Acknowledgements

First of all, I would like to thank my advisor Karthik Pattabiraman for his unwavering support. The months I spent as a Master student were some of the most intellectually and professionally rewarding I have ever had, and much of it is attributable to Karthik constantly motivating me to think more critically and get my ideas across more effectively. His exemplary supervision allowed me to learn a lot and paved the way for me to begin my professional engineering career.

Many thanks also go to Ali and my colleagues in CSRG for their help and critiques. I particularly would like to thank my lab mates who always managed to make me laugh and made this entire experience enjoyable.

None of this would have been possible if it had not been for the unconditional love accorded to me by my family and friends. I would like to thank my dad Frolin, my mom Jeannette, my siblings Linnette and Jeno, my cousin Mikky, and all my relatives for always encouraging me to do my best. I would also like to thank my dearest friends Kylo, Paul, and Erwin, who constantly inspire me to keep going and are like my little brothers. Finally, I thank the members of CMPC for their prayers and support.

Last but not least, I would like to thank God for giving me the chance to pursue this wonderful profession, and for guiding me throughout the process.

Dedication

To my friends and family

Chapter 1

Introduction

Modern web applications, or Web 2.0 applications, retrieve information asynchronously without reloading the page or navigating to a new one. They are hence much more interactive than traditional web applications. This interactivity is accomplished through the use of JavaScript in the client, which allows for the creation, modification, and deletion of nodes in the application's Document Object Model (DOM). Today, as many as 97 of the top 100 most visited websites use client-side JavaScript[1], and each consists of thousands of lines of JavaScript code. Therefore, in this thesis, the words website and web application are used interchangeably.

JavaScript is weakly typed, and allows the creation and execution of new code at runtime. It is widely believed that these factors make it prone to programming errors. Further, web browsers are typically tolerant of errors in JavaScript code, although they differ in their handling of the errors. For example, web browsers do not stop executing a web application when it throws an exception; rather they continue to execute (other parts of the application) in response to user events and web browser notifications. This leads to subtle bugs that are difficult to find during testing [19].

[1] From www.alexa.com, April 2011.

With the above issues in mind, it is important to ask: *are JavaScript errors prevalent in today's web applications and if so, what are the most important steps that must be taken to improve the reliability of client-side JavaScript in such applications*? Providing an answer to this question is the overarching goal of this thesis. We approach the first part of this question by conducting an empirical study that not only shows the presence of JavaScript errors in web applications – in particular, JavaScript errors that lead to exceptions in the code – but also characterizes these errors to help further our understanding of them. We approach the second part of the question by studying the error characteristics obtained from our results and providing suggestions based on our results' implications on web application programmers, testers, and static analysis tool developers. By gaining this understanding and exposing common problems, we can help in carving out the precise steps that must be taken to ensure the reliable design of current as well as future web applications.

1.1 Objectives

The main goal of this work is to empirically study the errors in JavaScript-based web applications and to identify common error categories in these applications. We also seek to understand the sources of these errors, by studying their correlation with the application's static and dynamic characteristics, such as the number of calls to the *eval* construct. Another goal is to formulate design guidelines and principles to help developers and testers improve their web applications' reliability.

Although JavaScript was designed in 1995 by Brendan Eich and was part of the Netscape 2.0 browser, it became popular only in the last five years with the advent of applications such as Gmail and Google Docs. As a result, there have been few academic papers on JavaScript, and fewer still on empirical studies of the behaviour of JavaScript-based Web 2.0 applications. Recent work has studied the performance and runtime behaviour of JavaScript [27, 29], and the security and privacy of JavaScript-based websites [11, 33]. However, to our knowledge, there has been no study on characterizing *errors* encountered in JavaScript-based web applications.

An error in a JavaScript-based web application can have severe consequences, including loss of its functionality. For example, one of the errors we found in our study prevented the header and navigational items of ifeng.com from displaying in one of its pages. Instead, these items were replaced by a warning at the top of the page indicating that an error has occurred (see Figure 1.1). While this particular error may be caught by traditional testing techniques, there may be much more subtle errors that are not[2]. Therefore, it is important to study the reliability of JavaScript code in the wild, in order to understand the errors that occur in them. This is the goal of our study.

We base our study on error messages printed to the JavaScript console by web applications executing in the wild. Whenever the JavaScript code throws an exception, an error message is printed to the JavaScript console[3]. We use Firebug[4], an add-on to the Firefox web browser, to capture the

[2]Indeed, most errors do not result in such explicit warnings/alerts.
[3]This console is hidden from the user, but can be enabled on demand.
[4]http://getfirebug.com

Figure 1.1: A JavaScript error in the website ifeng.com causing the page header to disappear and be replaced by an error message

messages. Although the focus of this study is on JavaScript errors that lead to exceptions, note that other types of errors are possible. For instance, the JavaScript code may set the property of an element in the page – such as the element's colour or size – to an incorrect value because of a semantic error made by the programmer; this is still considered an error, even though an exception may not necessarily take place. Hence, the results of this study serve as a conservative estimate of web applications' reliability, as the errors considered in this study are only a subset of all the errors that are actually in the web applications.

Our evaluation set consists of fifty websites from the Alexa top 100 most visited websites, as well as a group of ten interactive web applications, both of which we interact with in a "normal" manner. To avoid confusion, the former set of fifty web applications will be referred to as "popular websites", while the latter set will be referred to as "interactive web applications". We analyze the error messages, categorize them and determine if they are non-deterministic, i.e., occur only in a subset of the executions. We also correlate the web application's characteristics with the types and frequencies of error messages to understand their relationship.

1.2 Pros and Cons

In this study, we assume that an error message corresponds to an actual error, i.e., a software defect. Although this assumption may not hold for every error message, we believe that an error message is an indication of a potential problem in the application. For instance, when an exception is thrown at a certain line in the JavaScript code, subsequent lines in the code may not get executed, leading to unexpected failures. Further, a benign error message may have a serious, unforeseen consequence after a code update. However, we do not consider the consequences of errors in this study.

Static analysis is an alternative technique to using error messages for identifying errors, and has been successfully applied to large code bases such as the Linux Kernel [5]. However, error messages have several advantages over static analysis. First, console messages represent errors in real settings after the web application has been released to the public, and hence these errors likely escaped traditional testing methods. Further, the messages capture erroneous interactions between the web application and the DOM, which static analysis tools are likely to miss, as they do not typically model the DOM. In the extreme case, some static analyzers such as JSLint[5] and Closure Compiler[6] only analyze the code's syntax, disregarding semantics, and hence exhibit false positives and false negatives. Finally, JavaScript is a challenging language to analyze statically, and hence many such analyzers confine themselves to a "sane" subset of the language [9, 10]. However, as recent studies have shown [29], many web applications do not confine

[5] www.jslint.com
[6] code.google.com/closure/compiler/

themselves to this subset.

1.3 Thesis Contributions

This thesis presents three main contributions. They are:

- We develop a systematic methodology to execute web applications in multiple testing modes, and categorize their error messages. The methodology has been implemented using both existing tools, as well as tools we have developed and made open-source[7].

- We run the tools on 50 of the top 100 most visited websites as well as ten interactive web applications to study the characteristics of their errors. An *interactive* web application is one in which most of the user events such as clicks and mouse movements trigger the execution of JavaScript code (in contrast, user events in less interactive web applications are dominated by URL navigation, and most of the JavaScript code is executed at load time). For the fifty popular websites, we further correlate the messages with the static and dynamic characteristics of these websites. We make all our experimental data freely available for reproducibility.

- We consider the implications of the findings for web application programmers, testers and tool developers. In a sense, this thesis is a "call to arms" for improving the reliability of JavaScript-based Web 2.0 applications.

[7] http://ece.ubc.ca/~frolino/projects/jser/

The main results from the study are as follows:

- **JavaScript errors occur in web applications**: Even production web applications that are mature and well-engineered exhibit errors (average of 4 distinct error messages per web application).

- **Errors fall into well-defined categories**: About 93% of the errors fall into one of four categories: Permission Denied (52%), Undefined Symbol (28%), Null Exception (9%), and Syntax Errors (4%).

- **Effect of testing mode**: The frequencies of errors depend on the speed of interaction with the web application (i.e., fast, medium or slow).

- **Presence of non-deterministic errors**: About 72% of the errors are non-deterministic (i.e., vary across executions).

- **Correlation with static and dynamic characteristics**: Error frequencies are positively correlated with some of the static and dynamic characteristics of the applications such as Alexa rank, the number of domains containing JavaScript, the number of function calls, the number of property deletions, and the number of object inheritance overridings, but not with others such as the size of the code or the number of dynamic *eval* calls.

- **Effect of interactivity**: JavaScript errors in interactive web applications have characteristics similar to the errors found in popular websites. However, the presence of non-deterministic errors and permis-

sion denied errors is less pronounced due to fewer advertisements and the way interactive web applications are designed.

1.4 Thesis Organization

This chapter serves to establish the motivation and the overarching goal of the thesis. In Chapter 2, background information on web application reliability – particularly with respect to the use of client-side JavaScript – is provided, as well as related work. Chapter 3 describes in detail the experimental methodology used to collect JavaScript errors from the two evaluation sets (production websites and interactive web applications), and provides an overview of the tools used to perform this evaluation. Chapter 4 gives a detailed discussion of the results acquired from this study, and Chapter 5 discusses the implications these results have on web application programmers, testers, and tool developers. Finally, Chapter 6 concludes and presents future research directions.

Chapter 2

Background on Web Application Reliability and Previous Work

This chapter provides background information on the use of JavaScript in Web 2.0 applications. It highlights some of the inherent characteristics of the JavaScript language that could have direct implications on the reliability of the web application. In addition, we present prior work that has been conducted on the reliability, security, and performance of web applications.

2.1 JavaScript Background

JavaScript has gained prominence as the de-facto client-side programming language of the Web. In many respects, JavaScript is similar to languages such as C and Java. However, it differs from them in important ways. For example, JavaScript is dynamically typed, and allows code to be created and executed at runtime (e.g., through the *eval* construct). Therefore, it is believed to be prone to programming errors [19].

A web application[8] consists of three main components in the client side. First, there is the HTML code, which forms the basic building block of its webpages. Second, there are cascading style sheets (CSS), which are used to control the layout of elements in a webpage. Finally, there is the JavaScript code, which is either embedded in the webpages, or is imported as separate files. Unlike CSS and HTML, JavaScript is used for the web application's core functionality, and not only for display of elements. As a result, errors in JavaScript can have substantial impact, and may even be exploited by attackers [33].

Typical web applications are structured as a set of event handlers that are triggered by specific actions on the webpage, or by the loading of the page. For example, an 'on click' event handler will be executed whenever a certain element in the webpage is clicked, if the developer has specified a handler for the element. In addition, handlers may be triggered by the expiration of timers and the receipt of asynchronous messages from the server. Because of this structure, a JavaScript-based web application may continue execution even if one of the handlers throws an exception, as the other handlers continue to be fired (however, the code in the handler following the line that throws the exception does not execute). As a result, the application may throw multiple exceptions in a single execution.

JavaScript code may be loaded in the web browser either by statically including it in the web page, or by dynamically creating it at runtime (e.g., through eval). In both cases, the code must be parsed before it is executed, and errors in this process would manifest as *syntax errors*.

[8] We use the term web application to mean Web 2.0 application henceforth.

JavaScript is weakly typed, which means that there is no constraint on the types of objects that a JavaScript variable can refer to. Therefore, before invoking a method on an object or accessing its field, programmers need to ensure that the object has a member function or field by that name. Otherwise the code will throw an *Undefined Symbol* exception.

JavaScript code typically interacts with the elements of the webpage through a data structure called the Document Object Model (DOM). The DOM is an internal representation of the webpage within the web browser and is a tree-like structure. The JavaScript code often makes certain assumptions about the DOM, which, if violated, can lead to its failure. For example, an event handler may assume that the DOM contains a certain element and attempt to access the element. A *Null Exception* is thrown if the element is not present.

Finally, web browsers enforce the Same-Origin Policy (SOP), which ensures that JavaScript code from one domain cannot access methods or properties from another domain. Violations of the SOP result in a *Permission Denied* exception.

2.2 Related Work

In this section, we present prior work related to the performance, security, and reliability of web applications, both on the server-side and the client-side.

Server-Side Reliability: There have been several studies analyzing the causes of errors that occur at the server-side of web applications and

their ensuing reliability [8, 25, 30]. Other studies have examined the end-to-end availability of internet applications [12, 23]. Our study differs in that we focus on errors in JavaScript code, which executes on the client (i.e., web browsers). Further, server-side applications are written in languages such as C or Java, and hence have different failure modes compared to JavaScript applications.

Client-Side Reliability: Dynamic analysis techniques have been proposed to detect client-side errors in web applications. Examples are user behaviour analysis [14], robustness testing [24], invariant-based testing [17], and web fault taxonomy creation [16]. Static analysis techniques have also been used to find errors in web applications [9, 10, 34]. Record and replay tools such as Mugshot [18] and WaRR [1] aid in the reproduction of client-side errors. However, unlike our work, these papers do not conduct an empirical study of JavaScript errors in web applications.

Performance: Recent work has studied the performance and parallelism of JavaScript programs. For instance, Richards et al. [29] conduct an empirical study of dynamic JavaScript behaviour based on collected traces; similar work was done by Ratanaworabhan et al. [27] with their JSMeter tool. Fortuna et al. [7] perform a limit study on the parallelism available in JavaScript code. However, none of these papers investigate the reliability of web applications.

Security: Empirical studies on the security [9, 31, 33] and privacy [11] of web applications focus on the Alexa top websites and popular widgets. These papers also differ from our study in that they do not study web applications' errors, which may or may not lead to security vulnerabilities.

Empirical Studies of Software Errors: Finally, empirical studies of errors in open-source applications [4, 15], internet services, smart phones [6] and routers [32] have provided valuable insights regarding the nature of software errors. These studies illustrate the value of performing empirical studies on reliability in the real world. However, these studies, do not consider JavaScript errors, which are very different from errors in traditional applications.

To the best of our knowledge, ours is the first study to expose the reliability issues with client-side JavaScript in web applications, and the first to characterize client-side JavaScript errors. In addition, we accomplish this by performing an empirical study of the JavaScript errors that occur in both popular websites and interactive web applications.

Chapter 3

Experimental Methodology

In this chapter, we list the research questions in our experiments. We then describe the web applications used in our evaluation. Finally, we explain how we generate test suites, capture JavaScript errors in the web applications, and study their characteristics.

3.1 Research Questions

In conducting our experiments, we seek to answer the following questions:

Question 1: Are JavaScript errors prevalent in web applications, and if so, do these errors share common characteristics across web applications?

Question 2: Does the speed of interaction affect the frequency of JavaScript errors? An interaction refers to clicks, *mouseouts*, *mouseovers* and other events triggered by the user when visiting a web application. The speed of interaction refers to how quickly a user performs these interactions.

Question 3: Do non-deterministic JavaScript errors occur in web applications? An error is considered *non-deterministic* if its frequency differs from one execution to another.

Question 4: Are there any correlations between a web application's static and dynamic characteristics and the number of errors in that web

application?

Question 5: Are there inter-category correlations among the different error categories in web applications?

Question 6: Is the number of errors in a web application affected by the frameworks used in its construction?

Question 7: Are certain web application types more prone to error than other types?

Question 8: Are there differences in the characteristics of errors that occur in interactive web applications compared to errors that appear in the popular websites?

3.2 Web Applications

For our evaluation of the popular websites, we chose fifty web applications from the Alexa Top 100 (see Table 3.1) (as of January 7, 2011). Further, we ensured that the chosen web applications formed a representative set with sufficient variety. For example, the Alexa Top 100 has many country-specific Google based websites; since these websites have similar characteristics, only the main Google website was chosen. We also excluded adult websites and sites containing no JavaScript code from the study. The chosen websites often contain several kilobytes of JavaScript code (minimum of 506 bytes, maximum of 1.56 megabytes, and average of 315 kilobytes), and some span multiple domains (up to 18, average is 6). Table 3.1 provides more details on the websites' characteristics.

The ten interactive web applications were chosen from the JavaScript-

based web applications listed in the Open Directory Project[9]. These applications were chosen based on two criteria: (1) popularity, which was measured based on Alexa rank, and (2) size of JavaScript code, measured in kilobytes. The listed web applications were given scores based on these criteria, and the applications with the highest score were chosen. With this scoring scheme, more popular web applications were preferred over less popular web applications, and applications with a mix of JavaScript code sizes were chosen. To ensure that there is sufficient variety in the web applications we have chosen, we first came up with five interactive web application categories, and chose two web applications from each category. Table 3.2 shows the static characteristics and categories of these web applications.

3.3 Overview of Experiment - Popular Websites

In this section, we describe the steps in our experiment involving the fifty popular web applications from the Alexa Top 100. The tools used in the experiment are described in Section 3.5.

Our experiment consists of the following steps.

Step 1: Create *test cases* for each web application. A test case represents an "interaction" with a web application, which may consist of one or more events, depending on the context of use of the web application. For example, opening a webpage involves only a single click, so a test case emulating this interaction would consist only of one event (i.e., clicking the link). In contrast, using a search engine would consist of two events, namely

[9] http://www.dmoz.org/Computers/Internet/On_the_Web/Web_Applications/

Table 3.1: Website Static Characteristics. This table lists the fifty websites taken from the Alexa Top 100. Note that the extensions for the web applications are *.com* unless specified otherwise.

Web Application	Alexa Rank	Bytes of JavaScript Code	Total Number of Domains	Number of Domains with JavaScript
Google	1	164089	1	1
YouTube	3	420894	2	1
Yahoo	4	504503	4	3
Baidu	6	12759	1	1
QQ	9	210324	7	6
MSN	11	122143	7	5
Amazon	13	225149	3	2
Sina.com.cn	16	512392	18	17
WordPress	19	151959	8	7
Ebay	20	263615	3	2
LinkedIn	22	289599	6	5
Bing	23	28678	1	1
Microsoft	24	276465	9	9
Yandex.ru	25	221566	3	2
163	28	438689	12	11
mail.ru	30	201063	3	2
PayPal	31	258071	2	1
FC2	32	91775	6	5
Flickr	36	8736	3	1
IMDb	37	380061	7	6
Apple	38	416295	2	1
BBC	43	557137	11	11
Sohu	44	224148	12	12
Go	45	83512	6	6
Soso	46	40439	2	1
Youku	50	298149	6	5
AOL	51	301306	6	5
CNN	54	892169	11	11
MediaFire	59	485692	3	2
ESPN	61	628953	9	8
MySpace	62	720027	8	6
MegaUpload	63	139857	3	2
Mozilla	64	138855	2	1
4shared	66	233052	5	4
Adobe	67	591191	4	3
About	68	147027	2	2
LiveJournal	74	343701	7	6
Tumblr	75	247224	4	3
GoDaddy	77	317264	4	2
CNET	78	987612	13	12
YieldManager	82	164512	1	1
Sogou	83	8436	1	1
Zedo	84	96504	4	4
Ifeng	85	101255	11	10
ThePirateBay.org	86	506	2	1
ImageShack.us	88	425050	10	10
Livedoor	91	143131	3	3
Weather	94	1637291	8	8
NYTimes	95	762306	12	11
Netflix	97	208821	2	2

Table 3.2: Static Characteristics of Interactive Web Applications. Note that the extensions for the web applications are *.com* unless specified otherwise.

Web Application	Alexa Rank	KB of JavaScript Code	Category
aceproject	33104	476	Project Management
smartsheet	9485	62	Project Management
photobucket	154	264	Image Sharing
kodakgallery	7115	256	Image Sharing
dropbox	234	157	Storage
ziddu	784	173	Storage
dailymotion	103	142	Video Sharing
blip.tv	3742	122	Video Sharing
sfgames (jsgames.sourceforge.net)	158	12	Games
jssgames (javascriptsource.com/games)	3339	284	Games

typing the keyword and clicking the search button. Fifteen test cases are created for each web application using the Selenium tool (see Section 3.5); this group of fifteen test cases makes up one *test suite*. We created the test cases based on normal interactions with the web application i.e., no attempts were made to break the web applications to cause them to produce errors. On average, each test case consisted of 2.66 events and each test suite visited 29.46 webpages in our experiment.

Step 2: Replay the test suites corresponding to each web application multiple times. Each test suite is replayed in three *testing modes* — fast, medium, and slow — representing the speed of interaction (i.e., the speed at which events in the test suite are replayed in sequence), to answer Question 2. Note that the testing modes are consistent across all the web applications in the study. In slow mode, there is a delay of 1000 ms between each event (i.e., a delay of 1000 ms beyond the delay already present be-

tween each event due to processing); in medium mode, 500 ms; and in fast mode, 0 ms (or rather, negligible delay). To determine if JavaScript errors are non-deterministic (Question 3), each test suite is replayed three times in each of the three testing modes. Thus, each test suite is executed a total of nine times in our experiment. We use the Selenium tool to replay the test suites (Section 3.5). JavaScript errors that occur during a test suite's replay are typically displayed on a console. For each run of the test suite, the error messages are redirected to a file (called an *error file*).

Step 3: Parse the error files to collect error statistics.We have written a parser (see Section 3.5) to parse the error files and count the number of *distinct errors*. Figure 3.1 shows the three attributes of a message. Two error messages are considered distinct if any one of their three attributes are different, namely (1) their text descriptions, (2) the JavaScript files that triggered the errors, or (3) the lines of code that triggered the errors. Note that it may be possible for identical error messages to represent different errors due to JavaScript minification in web pages; thus, the number of errors we report is a conservative estimate of the actual number of errors. For each distinct error, the parser counts the *actual* number of times the error occurred in each test suite run (because an error may occur multiple times in a run).

For each distinct error, the parser determines if the error is non-deterministic by comparing its frequencies in each run. An error is considered non-deterministic *in a given testing mode* if its frequency differs across the three runs of that testing mode (because this indicates that the error was triggered in some executions but not in others). We *count* an error as non-

deterministic if it is non-deterministic in any of the three testing modes. Finally, the parser classifies each distinct error message into five mutually exclusive categories and counts the number of errors in each category. The error categories were determined based on an initial pilot study of five applications.

3.4 Overview of Experiment - Interactive Web Applications

The experimental methodology used to test the ten interactive web applications was similar to the methodology described in Section 3.3. The main difference is that the events were not recorded using the Selenium IDE, but were set up using the Selenium WebDriver API in Java (see Section 3.5). This does not affect the way the test cases are executed, as the Selenium IDE itself uses the WebDriver API to replay its test cases.

In addition, in this experiment, we try to attain as much coverage of the interactive web applications as possible, as we wish in the future to perform a more in-depth analysis of the JavaScript errors that occur in these applications. Hence, the number of test cases differs for each web application, depending on the number of possible user events. The number of test cases created for each web application is shown in Table 3.3.

Table 3.3: Number of Test Cases for Interactive Web Applications.

Web Application	Test Cases
aceproject	21
smartsheet	17
photobucket	26
kodakgallery	15
dropbox	9
ziddu	4
dailymotion	23
blip.tv	7
sfgames	10
jssgames	20

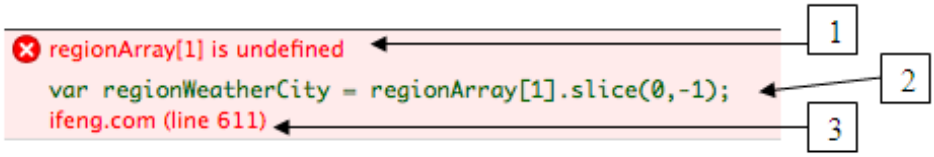

Figure 3.1: A screenshot of a JavaScript error message as shown in the Firebug console. The message consists of (1) the text description, (2) the line where the error occurred, and (3) the JavaScript file containing the erroneous line. We assume that two error messages containing identical values for each attribute map to the same error.

3.5 Tools and Datasets

For this experiment, the Firefox v. 3.6.13 web browser is used under the Mac OS/X Snow Leopard (10.6.6) platform. The machine used for the experiments was a 2.66 GHz Intel Core 2 Duo, with 4 GB of RAM.

The Selenium[10] IDE (v. 1.0.10) is used to create test cases and group the test cases together into test suites. Selenium is an extension to the Firefox web browser that captures and records user interaction with a webpage and converts these interactions into events for later replay. Examples of events

[10] http://seleniumhq.org/

are clicks, *mouseouts*, *mouseovers*, and dropdown selections.

To create the test suites, we interact with each web site in reasonable ways to exercise its behaviour. The Selenium IDE's recorder runs in the background and records this interaction to create the test case. In some cases, Selenium commands need to be entered manually due to limitations of the Selenium IDE. For example, the Selenium recorder currently does not support the recording of *mouseout* and *mouseover* events; therefore, commands for these events are added manually to the test case. Fifteen test cases for a given web application together constitute the test suite for the web application.

For the ten interactive web applications, the test cases were created by writing a Java program that uses the Selenium WebDriver API[11] (v. 2.9). This API provides means to open and interact with web applications using a specified browser (in this case, Firefox). The Selenium IDE uses this API to run its test cases; we determined after running the test on the fifty popular websites that WebDriver provides a more automated way to conduct the experiment.

Once a test suite is created, Selenium can replay it at a speed that can be set by the user. The replay speed is adjusted using a slider that ranges from "Slow" to "Fast". In our experiments, the testing modes correspond to three replay speeds — fast, medium, and slow. Selenium replays the fifteen test cases in a test suite at the chosen speed, for each application. Each test suite is run three times in each testing mode. The testing modes are identical across all the web applications.

[11] http://seleniumhq.org/projects/webdriver/

The Firebug 1.6.1 debugger is used to capture JavaScript errors during replay. Although Firebug can capture other errors such as those in CSS and XML, we modify its settings to capture only JavaScript errors, which are this study's focus.

A Firebug extension called ConsoleExport[12] is used to export the error messages to an error file. The error files are parsed to collect error statistics, as described in Section 3.3.

To help us answer Question 4, we collect each web application's static characteristics using two Firefox extensions: Web Developer[13] and Phoenix[14]. We use Web Developer to determine the JavaScript code size in the web application, and we use Phoenix to count the number of domains and the number of domains with JavaScript. The static characteristics are based on the initial loading of each website's homepage.

For the dynamic characteristics data, we use the traces collected by Richards et al. [29]. We downloaded the traces from the authors' website[15]. However, for our dynamic analysis, we only considered the web applications studied by Richards et al.; only 29 of the 50 applications overlap between the studies.

The dynamic characteristics considered in our study from Richards et al. are (1) number of function calls, (2) number of calls to *eval*, (3) properties deleted, and (4) object inheritance over-riding. The first two are self-explanatory. Properties deleted refers to the number of object fields,

[12]http://www.softwareishard.com/blog/consoleexport/
[13]http://chrispederick.com/work/web-developer/
[14]https://addons.mozilla.org/en-us/firefox/addon/phoenix/
[15]http://sss.cs.purdue.edu/projects/dynJavaScript/

object methods, or DOM elements that are deleted dynamically. Object inheritance overriding refers to the number of times a method belonging to a parent object is overriden by a child object. In other words, this metric measures the amount of polymorphism present in the application.

Finally, for Question 6, the frameworks were determined using the Library Detector[16] plugin available for Firefox.

[16]https://addons.mozilla.org/en-US/firefox/addon/library-detector/

Chapter 4

Results

The sections in this chapter parallel the research questions in Section 3.1. For each result, (1) we state our observation (*Observation*), (2) refer to the data from which we made this observation (*Data*), and (3) explain how we arrived at this observation and its possible causes (*Explanation*). We compiled our results data in a spreadsheet available online[17].

The results presented in Section 4.1 through Section 4.7 pertain to the experiment on the fifty popular websites taken from Alexa. In Section 4.8, we present results from the experiment on the interactive web applications.

4.1 Distribution of Error Categories

Table 4.1 presents the total number of distinct errors encountered across all nine runs of the popular websites' test suites; the appendix contains an expanded view of this table, showing the number of distinct errors for each of the popular websites. We make the following observations based on the table.

[17] http://ece.ubc.ca/~frolino/projects/jser/

Table 4.1: Popular Websites Error Data (Totals). The web applications included in the totals pertain to the fifty web applications from Alexa. The error frequency columns refer to the total number of distinct errors across all nine runs (slow-mode-only data in parentheses). The appendix contains an expanded view of the table showing the number of distinct errors that occurred in each of the fifty popular websites.

Total Errors in each category					Total	Non
Permission Denied	Null Exception	Undefined Symbol	Syntax Errors	Misc--ellaneous	JavaScript Errors	-Deterministic Errors
101 (81)	18 (15)	55 (46)	8 (8)	12 (12)	194 (162)	139 (100)

Observation 1: JavaScript errors occur in production web applications, with an average of around 4 distinct error messages per web application.

Data: JavaScript Errors column in Table 4.1

Explanation: Table 4.1 (along with the expanded table in the appendix) shows that one or more JavaScript errors occurred in 49 of the 50 web applications in our experiment (Google was the only exception, perhaps due to its simplicity). The maximum distinct error count was 16 (CNET). On average, 3.88 distinct errors occurred in each application, with a standard deviation of 3.02.

Observation 2: Errors predominantly fall into four distinct categories, which are described below.

Data: Permission Denied, Null Exception, Undefined Symbol, and Syntax Error columns in Table 4.1

Explanation: The error messages were found to belong to the following categories: As explained in Section 3.3, we determined these based on the results of a pilot study.

Permission Denied - These errors occur when JavaScript code from one domain attempts to access an object or variable belonging to a different domain, thereby violating the same-origin policy (SOP). In our study, these errors are often caused by advertisements from domains attempting to access the Location.toString method in the domain of the web application. For example, the error message "Permission denied for http://view.atdmt.com to call method Location.toString on http://www.imdb.com." appeared in the IMDb application. In this case, the view.atdmt.com is the domain used to serve advertisements in the IMDb application, and is attempting to call

a function in the IMDb domain, which violates the SOP.

Null Exception - These errors occur when a null value is used to access properties or methods. In our study, this error often arises due to missing or mistyped DOM elements. For example, the error message "C is null" was encountered in the Yahoo application. Subsequent analysis revealed that the error was caused by a typographical error in the value of the "id" attribute of a *div* element in the DOM. The incorrect id caused the getElementById method to return a null value, which, in this case, was assigned to the variable "C". The variable "C" was later used to update the class name of the div element, causing a null exception to be thrown.

Undefined Symbol - These errors occur when the JavaScript code (1) calls a function that has not been defined, (2) refers to a method or property that does not belong to a particular object, or (3) uses a variable that has either not been declared or assigned a value. An example of this error occurs in Amazon, where the error message "gbEnableTwisterJS is not defined" occurred. We found that the variable "gbEnableTwisterJS" was used as a condition for an *if* statement, but that the variable had not been defined in the code. Further investigation revealed that prior versions of the code did in fact include the statement "gbEnableTwisterJS = 0", defining the gbEnableTwisterJS variable. This finding suggests that gbEnableTwisterJS *was* initially defined in the code, but was later removed. However, not all references to gbEnableTwisterJS were removed from the code, thus leading to the error.

Syntax Errors – These errors occur due to syntactic violations in JavaScript code. Examples include missing end brackets, missing semi-colons, and un-

terminated string literals. An example of a syntax error is "missing ; before statement" in mail.ru. Syntax errors can occur either in static JavaScript code or in dynamic code created at runtime. However, all eight syntax errors found in our study came from static code, with six occurring in the main application, and two occurring in advertisements.

Miscellaneous Errors – Errors that occur in only a single web application and do not fall under the above categories are categorized as "Miscellaneous" errors. For example, an "uncaught exception" error occurred in the LinkedIn application, but did not occur in other applications, and is therefore classified under the "Miscellaneous" category.

Distribution of errors: Figure 4.1, which is based on Table 4.1 data, shows the distribution of error categories across all applications. Based on the data from Table 4.1, permission denied errors make up 52.1% of all errors; null exception errors make up 9.3%; undefined symbol errors make up 28.4%; and syntax errors make up 4.1%. Together these encompass 93% of the errors. The remaining errors are in the Miscellaneous category. As mentioned, permission denied errors are mostly caused by advertisements. We find that advertisements are present in over 30 of the 50 web applications, and hence the dominance of this category.

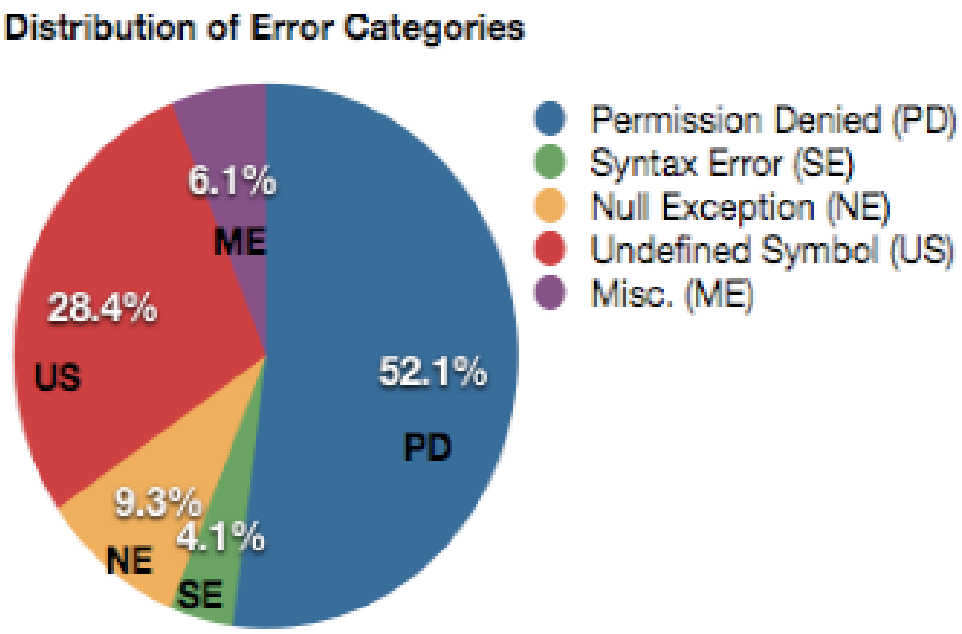

Figure 4.1: Percent distribution of each error category

Table 4.2: Actual number of occurrences of errors in CNN across different runs (long error messages have been shortened to save space).

	Fast Mode				Medium Mode				Slow Mode			
Error Message	Run 1	Run 2	Run 3	Average	Run 1	Run 2	Run 3	Average	Run 1	Run 2	Run 3	Average
Permission Denied for view.atdmt.com to call method Location.toString on marquee.blogs.cnn.com	4	4	4	4.00	1	3	3	2.33	2	2	3	2.33
Permission Denied for view.atdmt.com to call method Location.toString on www.cnn.com	20	17	20	19.00	22	22	16	20.00	25	20	16	20.33
Permission Denied for ad.doubleclick.net to call method Location.toString on www.cnn.com	8	16	13	12.33	3	6	4	4.33	7	12	11	10.00
targetWindow.cnnad_showAd is not a function	0	2	5	2.33	0	0	0	0.00	0	0	0	0.00
window.parent. CSI-Manager is undefined	0	0	0	0.00	0	0	0	0.00	1	1	0	0.67

4.2 Effect of Testing Mode

In the previous section, we studied the number of distinct error messages. In this section, we analyze the actual number of occurrences of the error messages in order to understand the effect of testing mode. We focus on one application — CNN.com — to illustrate the trends we observe across all web applications. Table 4.2 shows the occurrences of a subset of the error message that occurred in CNN for each testing mode. Similar trends hold for the other applications studied.

Observation 3: The occurrence of an error message depends on testing mode (i.e., the speed of interaction).

Data: Table 4.2 (Average columns)

Explanation: Looking at the "Average" columns in Table 4.2, it becomes apparent that some error messages occur only in one mode. For example, the error "targetWindow.cnnad_showAd is not a function" occurs in fast mode, but does not occur in the other two modes. Similarly, the error "window.parent.CSIManager is undefined" occurs in slow mode, but not in other modes.

Further, the tables show that some errors are more frequent in one mode compared to others. For example, the error message "Permission Denied for ad.doubleclick.net to call method Location.toString on www.cnn.com" occurs an average of 12.33 times and 10.00 times in fast and slow mode, respectively, but only occurs an average of 4.33 times in medium mode.

4.3 Occurrence of Non-deterministic Errors

In this section, we study the occurrence of non-deterministic errors in web applications. Recall that a non-deterministic error is one whose frequency varies across multiple executions of the web application in the *same* testing mode. In other words, a non-deterministic error occurs different number of times in each execution of the application.

Observation 4: Non-deterministic errors occur in many web applications, and are exposed by all three testing modes.

Data: Table 4.1, Table 4.2

Explanation: Table 4.2 shows the actual number of occurrences of several errors in different runs of the CNN application. From this data, it can be seen that for a given testing mode, the number of actual occurrences of some errors vary across different executions. These are classified as non-deterministic errors. For example, the error "Permission Denied for view.atdmt.com to call method Location.toString on www.cnn.com" in CNN (second row) in slow mode, occurs 25 times in the first run, 20 times in the second run, and 16 times in the third run.

Non-deterministic errors are caused by different factors in each of the modes. Non-deterministic errors in the fast and medium modes are typically caused by transitioning among pages (i.e., navigating to a new webpage) in the middle of accessing a member of the "parent" or "window" objects. During the transition, the value of the "parent" and/or "window" object changes because the values of these objects are dependent on the page being visited. As a result, if the transition happens *while* a member of these objects

is being accessed by JavaScript code in the previous page, the objects will be undefined during the transition, leading to the error. Such errors occur in the event handler of the old page only if the transition happens *during* execution of the specific line of code that uses "parent " or "window", and are hence non-deterministic in nature. These page transition errors may lead to undesirable consequences; for example, if a document is in the midst of being saved, transitioning to a new page will cause the save to abort in the event of an error.

In contrast, most of the non-deterministic errors in slow mode are caused by advertisements, mainly due to permission denied errors. In some runs, the advertisement appears in a given page, but in other runs, a different advertisement may appear, explaining the non-deterministic behaviour. This behaviour is not as prominent in fast and medium modes because in these modes, the test suites are transitioning between pages so quickly that the erroneous JavaScript code is not triggered, and hence they do not throw exceptions.

Summary: When all distinct errors across all web applications are considered, fast mode exposes a total of 82 non-deterministic errors, medium mode exposes 90, and slow mode exposes 100 (not shown in the table). Thus, counter-intuitively, slow mode actually exposes the maximum number of non-deterministic errors across the three modes. This is also reflected in Table 4.1, in which the numbers of distinct errors exposed by slow mode are shown within parentheses. As shown in the table, slow mode exposes a total of 162 distinct errors, which corresponds to about 83% of the total errors.

Table 4.3: Spearman coefficients between error categories and static web application characteristics. Correlations at the 0.05 level are marked with *, while those at the 0.01 level are marked with **.

	Correlations			
Error Category	Alexa Rank	JavaScript Size (Bytes)	Domains	Domains with JavaScript
Permission Denied	0.222	0.166	0.465**	0.450**
Null Exception	0.213	0.401**	0.334*	0.312*
Undefined Symbol	0.374**	0.246	0.152	0.200
Syntax Error	0.339*	0.305*	0.420**	0.435**
All Errors	0.375**	0.273	0.397**	0.396**

Observation 5: Non-deterministic errors are more prominent than deterministic errors in web applications, and constitute 72% of the total distinct errors.

Data: JavaScript Errors, Table 4.1

Explanation: From Table 4.1, the total number of distinct errors found across all web applications is 194. Of these 194 errors, 139 are non-deterministic in one or more of the three testing modes.

4.4 Correlation with Static and Dynamic Characteristics

In this section, we study the correlation of JavaScript errors with the static and dynamic characteristics of the web applications. We use the Spearman rank correlation coefficient because it is non-parametric and hence does not require the data to be normally distributed [13]. Table 4.3 shows the Spearman coefficients between the error categories and static characteristics of the web application. Table 4.4 shows the Spearman coefficients of the JavaScript error categories with the application's dynamic characteristics. The higher

Table 4.4: Spearman coefficients between error categories and dynamic web application characteristics. Correlations at the 0.05 level are marked with *, while those at the 0.01 level are marked with **.

	Correlations			
Error Category	Function Calls	Eval Calls	Properties Deleted	Inheritance Overriding
Permission Denied	0.159	0.126	0.056	-0.070
Null Exception	0.426*	0.195	0.448*	0.269
Undefined Symbol	0.074	0.200	0.033	0.490**
Total	0.308	0.257	0.128	0.186

the magnitude of the coefficient, the higher the correlation (a positive correlation means one value increases as the other increases, while a negative correlation means the opposite).

As mentioned in Section 3.5, dynamic characteristics data were available for only 29 of the 50 applications. Of these 29 web applications, only two incurred syntax errors. As a result, we do not report the values for this category. Further, we study the correlations with four dynamic characteristics in their study.

Note: As always, it is important to remember that correlation does not imply causation. However, correlations can still provide explanations as to the *possible* causes of the JavaScript errors, which can be verified through additional investigation. Although we have performed a detailed study on the potential causes of null exception errors through manual analysis (see Observation 10), we have not done so for other error categories. We leave a detailed investigation of causation to future work.

Significance: We now report the significant trends in the correlation coefficients. For most observations, we report the correlations for which $p < 0.05$ (i.e., significant at the 0.05 level). We call such correlations *signifi-*

cant.

Observation 6: There is a significant correlation between the total number of JavaScript errors and the number of domains with and without JavaScript code.

Data: Table 4.3

Explanation: Table 4.3 indicates that JavaScript errors have a 0.397 correlation with the total number of domains, and a 0.396 correlation with the number of domains with JavaScript, suggesting that applications using more domains have more errors. Further, permission denied errors have a 0.465 correlation with the total number of domains, and a 0.450 correlation with the number of domains with JavaScript. A possible reason for this behavior is that permission denied errors occur when JavaScript code from one domain tries to access resources from another domain. Thus, the more domains there are (with or without JavaScript), the higher the chances of different domains trying to access resources from one another.

Observation 7: There is no significant correlation between the total number of distinct JavaScript errors and the JavaScript code size (i.e., number of bytes of JavaScript).

Data: Table 4.3

Explanation: The Spearman rank correlation coefficient between the total number of distinct JavaScript errors and the JavaScript code size (in bytes) is 0.273, which is not significant at the 0.05 level. Thus, smaller code sizes will not necessarily lead to fewer errors. We use the code size instead of the number of lines of JavaScript code, because many web applications minify JavaScript by packing it in a single line, thus making the number of

lines an unreliable metric for correlation.

Observation 8: There is a significant correlation between the total number of distinct JavaScript errors and the Alexa rank of the web application.

Data: Table 4.3

Explanation: The Spearman rank correlation coefficient between the total number of distinct JavaScript errors and the Alexa rank is 0.375, suggesting that less popular applications may have higher number of errors (and vice versa). This result may stem from the fact that more popular web applications are likely to have undergone a more rigorous development and testing process than less popular web applications, since they are used by more people and therefore have a larger user base.

Observation 9: There is a significant correlation between the total number of distinct null exception errors and the number of functions called dynamically by the web application.

Data: Table 4.4

Explanation: As shown in Table 4.4, the Spearman rank correlation coefficient between the total number of distinct null exception errors per web application and the number of functions called at runtime is 0.426. This significant correlation can be explained by the fact that null exception errors are often caused by failed accesses to the DOM of the web application (e.g., undefined DOM element ids causing a variable to be null after a call to getElementById). This is also supported by the next observation. Because DOM manipulation is one of the most common usages of JavaScript [2], an increase in the number of functions called at runtime would increase the

number of DOM accesses, thereby increasing the likelihood of null exception errors.

Observation 10: There is significant correlation between the number of null exception errors and the average number of element/property deletions in the JavaScript code.

Data: Table 4.4

Explanation: The correlation coefficient between the number of null exception errors and the average number of property and element deletions is 0.448, which is significant at the 0.05 level. We believe this behaviour is also due to the relationship between null exception errors and DOM accesses (fourteen of the eighteen null exception errors in our study were due to DOM accesses, based on our manual analysis of the code). Specifically, if a DOM element is deleted and the JavaScript code tries to subsequently access that element, the resulting value will be null, thus resulting in a null exception.

Observation 11: There is significant correlation between the number of undefined symbol errors and the average number of object inheritance overridings in the code.

Data: Table 4.4

Explanation: The Spearman coefficient between the number of undefined symbol errors and the average number of object inheritance overridings is 0.490, which is significant at the 0.05 level. If we assume that object inheritance overridings are representative of the amount of polymorphism in the application, then this number reflects the fact that programmers are more likely to be confused about the identity of objects when the code has a lot of polymorphism, and hence are more likely to make mistakes in accessing

member functions or fields. For example, if an object B inherits from object A, and object B has a method called b() which object A does not have, then, a call to A.b() would lead to an undefined symbol error.

Observation 12: There is no significant correlation between the total number of distinct JavaScript errors and the number of calls to the *eval* construct.

Data: Table 4.4

Explanation: The correlation coefficient between the total number of distinct JavaScript errors and the number of *eval* calls is 0.257. Prior studies [28, 33] have suggested that calls to *eval* can compromise the reliability and security of the web application. However, we do not find evidence for the claim that they can compromise the reliability of the application, perhaps because *eval* is used primarily for JSON and other idiomatic purposes that are less prone to errors [29].

4.5 Inter-Category Correlations

In this section, we present the inter-category correlations we found in our study, namely those among different categories of errors identified in Section 4.1.

Observation 13: The correlations of each non-miscellaneous error category (permission denied, null exception, and undefined symbol) with syntax errors are significant.

Data: Table 4.1

Explanation: After calculating the Spearman rank correlation coef-

ficients, the correlations between total syntax errors and total permission denied, null exception, and undefined symbol errors are 0.378, 0.635, and 0.409, respectively, all of which are significant. This result suggests that syntax errors often lead to errors belonging to other categories (except miscellaneous). The expanded version of Table 4.1 in the appendix supports this observation – all web applications with a syntax error had one or more errors in the other categories as well.

Observation 14: There is a significant correlation between the number of non-deterministic null exception errors and the number of non-deterministic undefined symbol errors.

Data: Table 4.1

Explanation: We found the Spearman rank correlation coefficient between non-deterministic null exception errors and non-deterministic undefined symbol errors to be 0.560, suggesting that there is a significant correlation between these two error categories when it comes to non-deterministic errors. This behaviour requires further investigation.

4.6 JavaScript Framework

Table 4.5 shows the classification of applications by framework used in its construction. Applications using multiple frameworks are classified as "Mixed", while those using no frameworks are classified as "None". Frameworks encompassing fewer than three websites are not shown as they may not be significant. As can be seen from the table, the majority of web applications in the study were constructed using one or more frameworks, with

Table 4.5: Average number of distinct errors for each framework

JavaScript Framework	Average Errors	Number of sites
jQuery	4.04	26
Yahoo UI	3.67	6
Prototype	3.00	3
Mixed	5.25	4
None	2.10	10

jQuery being the most popular.

Observation 15: Web applications using multiple JavaScript frameworks have a higher number of JavaScript errors compared to those using only a single framework.

Data: Table 4.5

Explanation: Web applications using multiple JavaScript frameworks had an average of 5.25 errors — higher than the average for applications using only a single framework. It has been suggested that using multiple frameworks can increase the loading time of web applications, as it forces the client to download more JavaScript code [26]. Our result suggests that multiple frameworks also make the application more error-prone, perhaps due to inconsistencies between the frameworks, and the difficulty of maintaining the code.

Observation 16: Web applications using no frameworks have a lower number of JavaScript errors compared to those using at least one framework.

Data: Table 4.5

Explanation: The average number of errors for web applications using no frameworks is 2.10, which is lower than the average for web applications using at least one framework. The reason for this behaviour may be that frameworks abstract away details of the JavaScript code, making it more

Table 4.6: Average number of distinct errors for each web application type (popular websites)

Website type	Average Errors	Average Non-deterministic errors	Number of sites
Search Engine	2.17	1.50	6
Media Download	4.25	3.50	8
News	5.35	4.00	17
Blogs	2.00	2.00	3
Shopping	4.67	3.33	3
Social Networking	3.50	3.00	2
Business	2.64	1.09	11

difficult for programmers to map back errors to source code. It is also possible that the library functions used in the frameworks are themselves responsible for the errors, in which case the corresponding error message would point to the file containing the framework.

4.7 Web Application Type

We categorized the fifty websites in our study based on the category of content they serve. The categories are roughly based on the the categorizations provided in CyberPatrol[18]. Table 4.6 shows the number of websites in each category, the average number of errors in each category, and the average number of non-deterministic errors.

Observation 17: For the fifty popular websites, news websites have the highest average number of errors and non-deterministic errors across all application types.

Data: Table 4.6

Explanation: On average, news websites have 5.35 errors, which is higher than any other web application type. This is also true for non-

[18] www.cyberpatrol.com/research/sitereview.asp

deterministic errors. Although this behaviour requires further investigation, we believe that the prominence of errors in news websites has to do with the overall structure of these kinds of websites. News websites tend to be very dynamic in structure, containing advertisements and flashing content, and allows many interactive features such as polls. In other words, news websites look "busier" compared to other website types and hence are more likely to exhibit errors [3].

4.8 Interactive Web Applications

In this section, we examine the errors that occur in interactive web applications and compare the results to those gathered from the prior experiment on the 50 popular websites. A web application is considered interactive if the majority of user events – e.g., clicks, *mousedowns*, etc. – trigger JavaScript code to execute. The 50 popular websites studied earlier, in contrast, primarily triggered page transitions from one URL to another via anchor tags, and the execution of JavaScript code was limited mostly to page loads.

The ten interactive web applications have also been categorized according to their classifications in the Open Directory Project. Table 3.2 shows the categories to which each web application belongs, and Table 4.7 shows the number of errors found in each category.

Note that in this section, we focus on the observed differences between errors in interactive web applications and popular websites. We do not conduct correlational studies on the interactive web applications because the number of interactive web applications considered is relatively small

compared to the popular websites.

Observation 18: Non-deterministic errors still comprise the majority of errors in interactive web applications, but are not as prominent as the non-deterministic errors found in the popular websites

Data: JavaScript Errors, Table 4.8

Explanation: For the interactive web applications, 56% of the errors are non-deterministic, thereby forming the majority of the errors in the tested web applications. In contrast, non-deterministic errors comprise 72% of all the errors found in the fifty popular websites. There are two explanations for this result. First, the interactive web applications tested contained few advertisements; as mentioned in Section 4.3, advertisements play a big role in introducing non-deterministic errors – particularly permission denied errors – in slow mode. Second, page transitions, which caused many of the non-deterministic errors in fast and medium modes for the fifty popular websites, are uncommon in the interactive web applications. This is because for the interactive web applications, most interactions rely on the JavaScript code to alter the appearance of the page – for instance, to show dialogue boxes or change the style of certain elements – instead of transitioning to another page.

Observation 19: For the interactive web applications, game applications have the highest number of errors and non-deterministic errors compared to other interactive web application types, and project management applications have the fewest.

Data: Table 4.7

Explanation: According to Table 4.7, a total of 35 errors (12 of which

Table 4.7: Number of distinct errors for each web application type (interactive web applications). Each type is represented by two web applications.

Interactive Web Application Type	Number of Errors	Number of Non-deterministic errors	Avg. Number of Errors Per Test Case
Project Management	2	0	0.053
Image Sharing	9	8	0.220
Storage	8	7	0.615
Video Sharing	9	6	0.300
Games	35	12	1.167

are non-deterministic) appeared in the game applications studied; this number is higher than the total number of errors that appeared in any of the other interactive web application types. In contrast, the web application type with the fewest errors was project management applications, with only 2 errors (none of which are non-deterministic). When the results are normalized based on the total number of test cases for each web application type, these results still hold, with an average of 1.167 errors appearing per test case in game applications, and an average of 0.053 errors appearing per test case in project management applications. Note that the prominence of JavaScript errors in game applications may have resulted primarily from the inordinately large number of errors found in jssgames, which displayed a total of 32 errors (the other 3 game application errors come from sfgames).

On the other hand, project management applications displayed very few errors. One possible explanation is that project management applications generally involve tasks that are monotonous and well-defined (e.g., create a new file, open a new file, add new data to the file, save the file, etc.). In contrast, the video sharing and image sharing applications studied contained many distinct features (e.g., for image sharing applications – editing photos,

creating a photo album, etc.), thereby complicating their development.

Observation 20: The JavaScript errors in interactive web applications comprise mostly of null exception, undefined symbol, and syntax errors, with very few permission denied errors.

Data: JavaScript Errors, Table 4.8

Explanation: Table 4.8 shows that null exception, undefined symbol, and syntax errors are more prevalent than permission denied errors in interactive web applications. This starkly contrasts the observation with the popular websites, where most of the JavaScript errors found were permission denied errors.

Null exception, undefined symbol, and syntax errors can be viewed as *functionality-based* errors in the sense that these exceptions generally occur as a result of errors within the core implementation of the JavaScript code. In contrast, permission denied errors are mostly caused by third-party advertisements as explained in Section 4.1, and are hence not generally caused by the core implementation of the application's JavaScript code.

This distinction is important because it provides potential explanations for the prevalence of functionality-based errors over permission denied errors. First, few permission denied errors are observed because the interactive web applications contained very few advertisements compared to the popular websites. Second, functionality-based errors are abundant in interactive web applications compared to the popular websites because the core functionality of the interactive web application is spread out over many user events, and is therefore encompassing of the entire application. In contrast, the popular websites' core JavaScript functionality is limited mostly to load

time processing (e.g., to render certain parts of the page), effectively providing fewer opportunities for functionality-based errors.

In some cases, the effects of functionality-based errors can be particularly difficult to predict. For instance, the error found in smartsheet originated from its "Accounts Management" module, but also subsequently caused a failure in its "Help" module. As we discuss in Chapter 5, this brings to light the importance of extensive testing in interactive web applications.

Table 4.8: Interactive Web Application Error Data. The error frequency columns refer to the total number of distinct errors across all nine runs (slow-mode-only data in parentheses).

Web Application	Errors in each category					Total JavaScript Errors	Non -Deterministic Errors
	Permission Denied	Null Exception	Undefined Symbol	Syntax Errors	Misc- -ellaneous		
aceproject	0 (0)	0 (0)	0 (0)	0 (0)	1 (1)	1 (1)	0 (0)
smartsheet	0 (0)	1 (1)	0 (0)	0 (0)	0 (0)	1 (1)	0 (0)
photobucket	0 (0)	0 (0)	1 (1)	0 (0)	2 (0)	3 (1)	2 (0)
kodakgallery	0 (0)	2 (2)	1 (1)	3 (2)	0 (0)	6 (5)	6 (5)
dropbox	0 (0)	1 (1)	0 (0)	0 (0)	0 (0)	1 (1)	0 (0)
ziddu	2 (0)	1 (1)	1 (1)	3 (0)	0 (0)	7 (2)	7 (1)
dailymotion	1 (1)	2 (1)	0 (0)	3 (2)	1 (0)	7 (4)	6 (1)

Continued on next page

Table 4.8 – continued from previous page

Web Application	Errors in each category					Total JavaScript Errors	Non -Deterministic Errors
	Permission Denied	Null Exception	Undefined Symbol	Syntax Errors	Misc- -ellaneous		
bliptv	0 (0)	0 (0)	0 (0)	2 (2)	0 (0)	2 (2)	2 (2)
sfgames	0 (0)	0 (0)	3 (3)	0 (0)	0 (0)	3 (3)	0 (0)
jssgames	0 (0)	0 (0)	29 (29)	3 (3)	0 (0)	32 (32)	12 (9)
Total	3 (1)	7 (6)	35 (35)	14 (9)	4 (1)	63 (52)	35 (18)

Chapter 5

Implications

Table 5.1 presents a summary of the answers to the research questions posed in Chapter 3, based on our gathered results. A surprising result of this study is the relatively high frequency of errors in highly popular, production web applications, especially because our test cases constitute normal interactions with them. This may be because many of the errors occur due to the interaction of the JavaScript code with the webpage's DOM, and because they are non-deterministic. Hence, the errors are difficult to find during development and testing using current practices (e.g., unit testing).

While it may be argued that errors do not really matter as users continue to use web applications extensively, our experiments unearthed many instances of errors impairing a web application's key functionality. Further, we have shown that there is a correlation between the Alexa rank of a web application and the average number of errors in it (Observation 8). Finally, the rapid pace at which applications are being migrated to the web, and the fact that some of the errors we uncover are fairly straightforward to fix (e.g., syntax errors) suggests that not enough effort is being expended in making web applications reliable, perhaps due to the lack of established tools and practices in this domain. Therefore, there needs to be a concerted effort on

Table 5.1: Summary of Research Question Answers. The research questions are listed in Section 3.1.

Research Question	Answer
Q1: Are JavaScript errors prevalent in web applications, and if so, do these errors share common characteristics across web applications?	An average of 4 JavaScript errors occur in each web application, and these errors fall into well-defined categories.
Q2: Does the speed of interaction affect the frequency of JavaScript errors?	Errors vary by speed of interaction (i.e., testing mode). Some errors appear in one testing mode, but not in others.
Q3: Do non-deterministic JavaScript errors occur in web applications?	The majority of JavaScript errors (over 70%) are non-deterministic.
Q4: Are there any correlations between a web application's static and dynamic characteristics and the number of errors in that web application?	JavaScript errors are correlated with characteristics such as the number of domains (with or without JavaScript) and Alexa rank, but not with others such as the number of *eval* calls and the code size.
Q5: Are there inter-category correlations among the different error categories in web applications?	Syntax errors have significant correlations with the other three error categories. Non-deterministic null exception and undefined symbol errors are correlated.
Q6: Is the number of errors in a web application affected by the frameworks used in its construction?	Web applications using multiple frameworks have a high number of JavaScript errors.
Q7: Are certain web application types more prone to error than other types?	News websites have the highest average number of errors among all website types.
Q8: Are there differences in the characteristics of errors that occur in interactive web applications compared to errors that appear in the popular websites?	Interactive web applications display error trends similar to popular websites, but non-deterministic errors are not as prominent and functionality-based errors are abundant.

the part of programmers, testers and tool developers (of static and dynamic analysis tools) to improve the reliability of JavaScript web applications.

5.1 Implications for Programmers

For programmers, our observations could act as guidelines that help them write more reliable JavaScript code. For instance, Observation 11, which states that object inheritance overriding correlates with undefined symbol errors, suggests that inheritance in JavaScript is best avoided unless absolutely essential. Methods such as namespacing and reuse of methods across objects have been suggested as alternatives to inheritance in JavaScript [20]. In addition, Observation 6 suggests that using fewer domains may result in fewer errors. Finally, Observation 15 suggests that mixing of JavaScript frameworks should be avoided.

An interesting implication of our results is that often errors arise from code over which the programmer may not have direct control. For example, we find that many errors are caused by advertisements, and from unwanted interactions with the DOM (which may or may not be under the programmer's control). These observations lead us to posit that web applications must be tolerant of errors that arise in other components in order to be reliable. Further, the application should have consistency checks and other mechanisms to ensure that errors are caught early and not allowed to propagate.

5.2 Implications for Testers

One of the most significant implications of our results is that testing should be done at multiple speeds (i.e., testing modes). Each testing mode exposes different kinds of errors (Observation 3); thus, testing in only one mode would catch only a subset of the errors.

We have also shown in our results that non-deterministic errors are prominent in web applications (Observations 4 and 5). Thus, it is important to develop testing schemes that specifically attempt to catch these kinds of errors. In our results, for example, we found that several non-deterministic errors were caused by advertisements, particularly in slow mode (see Observation 4). This observation calls for the need for more extensive integration testing, in which the JavaScript code is tested after the advertisements have been integrated. This may even be offered as a service by advertisement serving applications such as Google AdSense.

Integration testing can be particularly useful for interactive web applications as well. In our study, we found that interactive web applications often contain many distinct features, and are therefore often given a modular design. Without extensive integration testing, the danger is that an error in a certain module may lead to errors or failures in another seemingly unrelated module. For example, the null exception found in smartsheet originated from the module that takes care of showing the Accounts Management dialogue box. As it turns out, this error subsequently causes the "Help" module (which shows the "Help" dialogue) to be rendered unusable. In the worst case, features which the programmer may perceive to be non-critical may

contain errors that could break other, more critical features. This brings to light the need for more thorough integration testing in interactive web applications, especially since JavaScript errors in such applications are dominated by functionality-based errors (Observation 20) that could affect particular modules in the application.

5.3 Implications for Tool Developers

The dependence of the appearance of errors on testing mode (Observation 3) means that more emphasis should be placed on the speed of interaction when testing JavaScript code. Such tests could be simplified if JavaScript testing tools are developed to automatically perform tests in different testing speeds. Further, dynamic tools should execute the web application in realistic settings, with advertisements and other third-party code.

Our observations also suggest the need for advanced static analysis tools. Simple syntactic checks no longer suffice, as the execution of JavaScript code depends not only on the semantic correctness of individual event handlers, but also on the order in which events are triggered and the current state of the DOM. For example, Observation 10 suggests that the number of element or property deletions correlates with the number of null exception errors. Static analyzers of JavaScript code should therefore consider the DOM in their analysis.

Indeed, based on our observation from this study that JavaScript errors typically arise from improperly set up DOM accesses or updates in the JavaScript code, we have developed a tool called AutoFLox [21]. This tool

performs automatic fault localization of DOM-related JavaScript errors, and uses a form of backward slicing to pinpoint precisely which DOM access in the code is causing an exception to be thrown. It uses both dynamic analysis and static analysis to achieve its goal.

5.4 Threats to Validity

An internal threat to the validity of our results is that the number of web applications considered in our study is limited. In addition, we restricted our study of the popular websites to the 100 most visited websites according to Alexa, and the study of interactive web applications had a preference for choosing web applications based on popularity. It is possible that some of our observations may not hold for websites that are not as popular.

Fewer interactive web applications are considered in our study (10) compared to the number of popular websites (50). There is a possibility that trends in the interactive web applications may alter if more interactive web applications are considered; however, we do not expect any significant changes in the trends because of the general design characteristics of such applications. For instance, it is unlikely that permission denied errors will suddenly dominate the distribution of errors because third-party advertisements are not commonly used in interactive web applications.

An external threat to validity is that we performed our study on only one browser (Firefox). We believe the Firefox browser is a fitting choice to carry out our empirical study because of its popular usage. However, future work may have to consider the behaviour of web applications in other browsers.

Our results represent a snapshot of web applications at a specific point in time during which the experiments were performed, and may hence change over time. This is also an internal threat to validity of the results.

In our study, we assume that all the JavaScript error messages are actual bugs. JavaScript bug reports may be used to confirm the nature of these error messages; unfortunately, such bug reports are often not available even for popular web applications such as the ones we studied. As such, we consider this a potential construct threat to our results' validity.

The study by Richards et al. [29] is based on Safari, not Firefox, and they use different test suites for the web applications in their study. Further, their study precedes ours by at least two years; as a result, some of their data may be outdated.

Finally, it is possible that our results may be an underestimation of the number of JavaScript errors present in the web applications. One reason is that our test suites for the popular websites have limited coverage, containing only 15 test cases consisting of approximately 3 events each. In addition, JavaScript minification is used in some web applications, causing some distinct error messages to be labeled as being the same. Nonetheless, the fact that our results still show a considerably large number of errors despite being conservative goes to show that web applications truly are prone to these JavaScript errors.

Chapter 6

Conclusions and Future Work

We have performed an empirical study of JavaScript error messages in both popular websites from the Alexa top 100 and interactive web applications. Our results show that JavaScript errors in the popular websites: (1) occur in these web applications, (2) fall into well-defined categories, (3) depend on the speed of testing, (4) are often non-deterministic in nature, and (5) exhibit correlations with static and dynamic characteristics of the application. In addition, our results show that while JavaScript errors in interactive web applications follow similar trends, they differ from the popular websites in terms of the increased prominence of functionality-based errors, and the smaller percentage of non-deterministic errors.

By characterizing the nature of JavaScript errors, we were able to expose the prevalence of such errors in today's web applications. Consequently, we have taken an important step in helping improve the reliability of Web 2.0 applications by providing meaningful suggestions based on the error characteristics we have observed. As we have shown, these observations have direct implications on web application programmers, testers, and tool developers,

all of whom play important roles in the development and maintenance of web applications.

6.1 Future Work

Future work will focus on expanding the range of web applications considered in the study. Instead of considering only fifty popular websites, we plan to perform a similar experiment on the Alexa Top 1000. The results from such a study will be compared with the results of this thesis to see if our observations hold for a wider range of web applications.

The study presented in this thesis can be considered a breadth study that aims to characterize JavaScript errors based on aggregated results collected over many web applications. To complement this study, we also plan to conduct a depth study in which we attempt to identify the root causes of the errors discovered, thereby giving us a fuller understanding of the observations presented.

Finally, as mentioned in Chapter 1, the overarching goal of this work is to identify key characteristics of JavaScript errors that happen in today's web applications to pinpoint the most important actions that could help improve the reliability of such applications. Hence, it is also our goal to create schemes or tools – developed based on the results from this thesis – that would simplify the process of programming and testing client-side JavaScript for reliability. One such project currently in development is AutoFLox [21], which is an automatic fault localizer for client-side JavaScript.

Bibliography

[1] S. Andrica and G. Candea. WaRR: High Fidelity Web Application Recording and Replaying. In *IEEE Intl. Conference on Dependable Systems and Networks*, 2011.

[2] H. Bidgoli. *The Internet Encyclopedia*. John Wiley & Sons Inc, 2004.

[3] M. Butkiewicz, H.V. Madhyastha, and V. Sekar. Understanding website complexity: Measurements, metrics, and implications. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 313–328. ACM, 2011.

[4] Subhachandra Chandra and Peter M. Chen. Whither generic recovery from application faults? a fault study using open-source software. In *Intl. Conference on Dependable Systems and Networks (formerly FTCS-30 and DCCA-8)*, DSN '00, pages 97–106, 2000.

[5] Andy Chou, Junfeng Yang, Benjamin Chelf, Seth Hallem, and Dawson Engler. An empirical study of operating systems errors. In *ACM Symposium on Operating Systems Principles*, SOSP '01, pages 73–88, 2001.

[6] M. Cinque, D. Cotroneo, Z. Kalbarczyk, and RK Iyer. How Do Mo-

bile Phones Fail? A Failure Data Analysis of Symbian OS Smart Phones. In *Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP Intl. Conference on*, pages 585–594, 2007.

[7] E. Fortuna, O. Anderson, L. Ceze, and S. Eggers. A limit study of JavaScript parallelism. In *IEEE Intl. Symposium on Workload Characterization (IISWC)*, pages 1–10, 2010.

[8] Katerina Goseva-Popstojanova, Sunil Mazimdar, and Ajay Deep Singh. Empirical study of session-based workload and reliability for web servers. In *15th Intl. Symposium on Software Reliability Engineering*, pages 403–414, 2004.

[9] Salvatore Guarnieri and Benjamin Livshits. Gatekeeper: mostly static enforcement of security and reliability policies for JavaScript code. In *Conference on USENIX Security Symposium*, SSYM'09, pages 151–168, 2009.

[10] Arjun Guha, Shriram Krishnamurthi, and Trevor Jim. Using static analysis for AJAX intrusion detection. In *Intl. Conference on World Wide Web*, pages 561–570, 2009.

[11] Dongseok Jang, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. An empirical study of privacy-violating information flows in JavaScript web applications. In *ACM Conference on Computer and Communications Security*, pages 270–283, 2010.

[12] M. Kalyanakrishnan, R.K. Iyer, and J.U. Patel. Reliability of inter-

net hosts: a case study from the end user's perspective. *Computer Networks*, 31(1-2):47–57, 1999.

[13] S.H. Kan. *Metrics and models in software quality engineering*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2002.

[14] W. Li, M.J. Harrold, and C. Gorg. Detecting user-visible failures in AJAX web applications by analyzing users' interaction behaviors. In *IEEE/ACM Conference on Automated Software Engineering*, pages 155–158, 2010.

[15] Zhenmin Li, Lin Tan, Xuanhui Wang, Shan Lu, Yuanyuan Zhou, and Chengxiang Zhai. Have things changed now?: an empirical study of bug characteristics in modern open source software. In *1st workshop on Architectural and system support for improving software dependability*, ASID '06, pages 25–33, 2006.

[16] A. Marchetto, F. Ricca, and P. Tonella. Empirical Validation of a Web Fault Taxonomy and its usage for Fault Seeding. In *IEEE Intl. Workshop on Web Site Evolution-Volume 00*, pages 31–38, 2007.

[17] A. Mesbah and A. van Deursen. Invariant-based automatic testing of AJAX user interfaces. In *Intl. Conference on Software Engineering*, pages 210–220, 2009.

[18] J. Mickens, J. Elson, and J. Howell. Mugshot: deterministic capture and replay for JavaScript applications. In *7th USENIX Conference on Networked Systems Design and Implementation*, pages 11–11, 2010.

[19] T. Mikkonen and A. Taivalsaari. Using JavaScript as a Real Programming Language. *Sun Microsystems Laboratories Technical Report*, 168, 2007.

[20] Robert Nyman. JavaScript namespacing—an alternative to JavaScript inheritance, October 2008.

[21] F.S. Ocariza Jr, K. Pattabiraman, and A. Mesbah. Autoflox: An automatic fault-localizer for client-side javascript. In *Proceedings of the 5th IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2012.

[22] F.S. Ocariza Jr, K. Pattabiraman, and B. Zorn. Javascript errors in the wild: An empirical study. In *Software Reliability Engineering (ISSRE), 2011 IEEE 22nd International Symposium on*, pages 100–109. IEEE, 2011.

[23] Venkata N. Padmanabhan, Sriram Ramabhadran, Sharad Agarwal, and Jitendra Padhye. A study of end-to-end web access failures. In *2006 ACM CoNEXT conference*, CoNEXT '06, pages 15:1–15:13, 2006.

[24] K. Pattabiraman and B. Zorn. DoDOM: Leveraging DOM Invariants for Web 2.0 Application Robustness Testing. In *IEEE Intl. Symposium on Software Reliability Engineering (ISSRE)*, pages 191–200, 2010.

[25] S. Pertet and P. Narasimhan. Causes of failure in web applications. *Parallel Data Laboratory, Carnegie Mellon University, CMU-PDL-05-109*, 2005.

[26] Pingdom. JavaScript framework usage among top websites, Jun. 2008.

[27] P. Ratanaworabhan, B. Livshits, D. Simmons, and B. Zorn. JSMeter: Measuring JavaScript behavior in the wild. *Usenix Conference on Web Application Development (WebApps)*, 2010.

[28] Gregor Richards, Christian Hammer, Brian Burg, and Jan Vitek. The eval that men do: A large-scale study of the use of eval in javascript applications. In *European Conference on Object-Oriented Programming (ECOOP)*, 2011.

[29] Gregor Richards, Sylvain Lebresne, Brian Burg, and Jan Vitek. An analysis of the dynamic behavior of JavaScript programs. In *ACM Conference on Programming Language Design and Implementation*, PLDI '10, pages 1–12, 2010.

[30] Jeff Tian, Sunita Rudraraju, and Zhao Li. Evaluating web software reliability based on workload and failure data extracted from server logs. *IEEE Trans. Softw. Eng.*, 30:754–769, 2004.

[31] J. Weinberger, P. Saxena, D. Akhawe, M. Finifter, R. Shin, and D. Song. An Empirical Analysis of XSS Sanitization in Web Application Frameworks. Technical Report EECS-2011-11, UC Berkeley, 2011.

[32] Z. Yin, M. Caesar, and Y. Zhou. Towards understanding bugs in open source router software. *ACM SIGCOMM Computer Communication Review*, 40(3):34–40, 2010.

[33] Chuan Yue and Haining Wang. Characterizing insecure JavaScript

practices on the web. In *Intl. Conference on World Wide Web (WWW)*, pages 961–970, 2009.

[34] Yunhui Zheng, Tao Bao, and Xiangyu Zhang. Statically locating web application bugs caused by asynchronous calls. In *Intl. Conference on the World-Wide Web (WWW)*, pages 805–814, 2011.

Appendix A: Error Data for Popular Websites

The table in the following page shows the number of distinct errors that occurred in each of the fifty popular websites we studied. This table is an expanded view of Table 4.1.

Table A.1: Popular Websites Error Data. The web applications pertain to the fifty web applications from Alexa. The error frequency columns refer to the total number of distinct errors across all nine runs (slow-mode-only data in parentheses).

Web Application	Errors in each category					Total JavaScript Errors	Non -Deterministic Errors
	Permission Denied	Null Exception	Undefined Symbol	Syntax Errors	Misc- -ellaneous		
Google	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
YouTube	2 (2)	2 (2)	0 (0)	0 (0)	0 (0)	4 (4)	4 (4)
Yahoo	1 (0)	1 (1)	1 (1)	0 (0)	1 (1)	4 (3)	3 (2)
Baidu	1 (1)	0 (0)	0 (0)	0 (0)	0 (0)	1 (1)	0 (0)
QQ	0 (0)	1 (1)	0 (0)	0 (0)	0 (0)	1 (1)	1 (1)
MSN	4 (3)	0 (0)	1 (1)	0 (0)	0 (0)	5 (4)	4 (3)

Continued on next page

Table A.1 – continued from previous page

Web Application	Errors in each category					Total JavaScript Errors	Non -Deterministic Errors
	Permission Denied	Null Exception	Undefined Symbol	Syntax Errors	Misc- -ellaneous		
Amazon	0 (0)	1 (1)	1 (1)	0 (0)	0 (0)	2 (2)	0 (0)
Sina.com.cn	4 (4)	0 (0)	2 (2)	0 (0)	0 (0)	6 (6)	5 (5)
WordPress	0 (0)	0 (0)	1 (1)	0 (0)	0 (0)	1 (1)	1 (1)
Ebay	1 (1)	0 (0)	0 (0)	0 (0)	0 (0)	1 (1)	0 (0)
LinkedIn	0 (0)	0 (0)	0 (0)	0 (0)	2 (2)	2 (2)	2 (2)
Bing	3 (3)	0 (0)	0 (0)	0 (0)	0 (0)	3 (3)	2 (2)
Microsoft	1 (1)	0 (0)	0 (0)	2 (2)	0 (0)	3 (3)	1 (1)
Yandex.ru	0 (0)	0 (0)	1 (1)	0 (0)	0 (0)	1 (1)	0 (0)
163	2 (2)	0 (0)	1 (1)	0 (0)	1 (1)	4 (4)	2 (2)
mail.ru	0 (0)	1 (0)	0 (0)	1 (1)	0 (0)	2 (1)	1 (0)
PayPal	0 (0)	0 (0)	2 (2)	1 (1)	0 (0)	3 (3)	0 (0)

Continued on next page

Table A.1 – continued from previous page

Web Application	Errors in each category					Total JavaScript Errors	Non -Deterministic Errors
	Permission Denied	Null Exception	Undefined Symbol	Syntax Errors	Misc- -ellaneous		
FC2	2 (1)	0 (0)	0 (0)	0 (0)	0 (0)	2 (1)	2 (1)
Flickr	7 (4)	0 (0)	0 (0)	0 (0)	0 (0)	7 (4)	7 (3)
IMDb	4 (2)	0 (0)	0 (0)	0 (0)	0 (0)	4 (2)	4 (2)
Apple	0 (0)	2 (0)	1 (1)	0 (0)	1 (1)	4 (2)	3 (1)
BBC	0 (0)	0 (0)	1 (1)	0 (0)	0 (0)	1 (1)	0 (0)
Sohu	2 (2)	0 (0)	1 (1)	1 (1)	0 (0)	4 (4)	3 (3)
Go	4 (3)	0 (0)	0 (0)	0 (0)	0 (0)	4 (3)	4 (3)
Soso	1 (1)	0 (0)	0 (0)	0 (0)	3 (3)	4 (4)	0 (0)
Youku	0 (0)	0 (0)	1 (1)	0 (0)	0 (0)	1 (1)	1 (1)
AOL	1 (1)	1 (1)	1 (1)	0 (0)	0 (0)	3 (3)	2 (2)
CNN	4 (4)	0 (0)	5 (3)	0 (0)	0 (0)	9 (7)	9 (7)

Continued on next page

Table A.1 – continued from previous page

Web Application	Errors in each category					Total JavaScript Errors	Non -Deterministic Errors
	Permission Denied	Null Exception	Undefined Symbol	Syntax Errors	Misc- -ellaneous		
MediaFire	0 (0)	0 (0)	1 (1)	0 (0)	0 (0)	1 (1)	0 (0)
ESPN	3 (2)	0 (0)	2 (1)	0 (0)	0 (0)	5 (3)	5 (3)
MySpace	4 (3)	0 (0)	1 (1)	0 (0)	0 (0)	5 (4)	4 (3)
MegaUpload	6 (6)	0 (0)	0 (0)	0 (0)	0 (0)	6 (6)	6 (6)
Mozilla	0 (0)	0 (0)	1 (1)	0 (0)	0 (0)	1 (1)	0 (0)
4shared	2 (2)	0 (0)	2 (2)	0 (0)	1 (1)	5 (5)	2 (2)
Adobe	0 (0)	0 (0)	3 (3)	0 (0)	2 (2)	5 (5)	0 (0)
About	3 (1)	0 (0)	2 (2)	1 (1)	0 (0)	6 (4)	5 (3)
LiveJournal	4 (3)	0 (0)	0 (0)	0 (0)	0 (0)	4 (3)	4 (3)
Tumblr	0 (0)	1 (1)	0 (0)	0 (0)	0 (0)	1 (1)	1 (1)
GoDaddy	0 (0)	0 (0)	1 (1)	0 (0)	0 (0)	1 (1)	0 (0)

Continued on next page

Table A.1 – continued from previous page

Web Application	Errors in each category					Total JavaScript Errors	Non -Deterministic Errors
	Permission Denied	Null Exception	Undefined Symbol	Syntax Errors	Misc- -ellaneous		
CNET	12 (8)	3 (3)	0 (0)	1 (1)	0 (0)	16 (12)	11 (7)
YieldManager	0 (0)	0 (0)	1 (1)	0 (0)	0 (0)	1 (1)	0 (0)
Sogou	0 (0)	0 (0)	3 (3)	0 (0)	0 (0)	3 (3)	0 (0)
Zedo	1 (1)	0 (0)	1 (1)	0 (0)	0 (0)	2 (2)	0 (0)
Ifeng	2 (2)	3 (3)	3 (3)	1 (1)	0 (0)	9 (9)	8 (4)
ThePirateBay.org	2 (2)	0 (0)	0 (0)	0 (0)	0 (0)	2 (2)	2 (2)
ImageShack.us	6 (5)	1 (1)	1 (1)	0 (0)	0 (0)	8 (7)	6 (5)
Livedoor	2 (1)	0 (0)	2 (2)	0 (0)	0 (0)	4 (3)	4 (2)
Weather	4 (4)	0 (0)	1 (1)	0 (0)	0 (0)	5 (5)	4 (4)
NYTimes	6 (6)	1 (1)	0 (0)	0 (0)	0 (0)	7 (7)	6 (6)
Netflix	0 (0)	0 (0)	10 (4)	0 (0)	1 (1)	11 (5)	10 (3)

Continued on next page

Table A.1 – continued from previous page

Web Application	Errors in each category					Total JavaScript Errors	Non -Deterministic Errors
	Permission Denied	Null Exception	Undefined Symbol	Syntax Errors	Misc- -ellaneous		
Total	101 (81)	18 (15)	55 (46)	8 (8)	12 (12)	194 (162)	139 (100)