XCompatibility Checker: a tool for detecting cross-browser incompatibilities

(alphabetically)
Arthur Marques, Mohammad Bajammal, Rodrigo Araújo
University of British Columbia
Vancouver, Canada
{msarthur, rodarauj}@cs.ubc.ca
{bajammal}@ece.ubc.ca

ABSTRACT

Web browsers are built by different organizations and writing software that runs smoothly on all existing browsers is a challenging task. Due to the pace that browsers implement or adopt certain web features, users' experience may be hindered by visual and functional incompatibilities due to unsupported or non-standard features in a given browser. In order to address the detection of cross-browser incompatibilities in early stages of web development, we propose *XCompatibility-Checker* a lightweight tool that automates the identification of features that are not supported by different browsers. The tool's evaluation was twofold (i) it was able to detect cross-browser incompatibilities in 38 open source web applications; as well as (ii) a user study and qualitative survey indicates that the tool improves developers' awareness and ability to detect cross-browser incompatibilities. Therefore, our proposed tool helps web developers improve the quality of their web applications.

Keywords

cross-browser compatibility; recommendation system; IDE tool

1. INTRODUCTION

Web applications are used in many applications and across various devices and platforms. Such platforms include high-resolution desktop computers, laptops and netbooks, tablets, mobile phones, televisions, among others. Fortunately, the differences between these devices are partially taken care of by web browsers, therefore enabling web application developers to more or less run the same application across many end-user devices.

However, web browsers are developed by different organizations with differing priorities and speeds of adopting web development features. While the World Wide Web Consortium (W3C) was formed to standardize web technologies, the adoption and implementation of standards is not uniform and consistent across browsers. As a result, not all web browsers adhere to the same standards, and different versions of various browser may or may not adopt features. Therefore, inconsistencies between different browsers, versions, and platforms may compromise user's experience [5].

In this project, we present the *XCompatibility Checker* tool. It aids developers in detecting defects related to web standards conformity. The tool exposes an API that inspects the source code of a web application and reports, non-standard compliant features in the code as well as an overall browser compatibility score for that application. Instead of deferring examining browser compatibility to a later stage after code development, the proposed tool shows compatibility defects in real-time during development and therefore allowing immediate repair of defects on the go. Furthermore, the proposed tool shows compatibility information for all major browsers on the spot, instead of requiring developers or testers to open the web application in multiple browsers and platforms and manually test them one at a time.

In order to evaluate our proposed tool, we used it to assess the overall cross-browser compatibility score of 38 open source systems. Our findings indicate that *XCompatibility-Checker* is able to identify compatibility issues on 60% of the evaluated projects. Additionally, a qualitative study verified that the tool does improve developers' awareness of cross-browser incompatibilities. Participants feedback state that the tool supports the early identification of incompatibilities, which could pass undetected without such tool support.

The major contributions of this paper are summarized as follows:

1. A command-line and IDE tool that can detect and report cross-browser incompatibilities;
2. An empirical evaluation of cross-browser incompatibilities on 38 open source systems;
3. An empirical evaluation on how developers awareness is affected by tools that detect cross-browser incompatibilities.

The remainder of the paper is structured as follows. Section 2 provides a motivating example of some of the issues related to cross-browser compatibility. Section 3 discusses related work and their limitations. Section 4 details our proposed approach and its implementation as a tool. Section 5 describes the methodology used to evaluate our tool, and Section 6 discusses evaluation's results. Section 7 discusses the threats to the validity of our experiment. Section 8 presents of final remarks and future work.

2. MOTIVATING EXAMPLE

In this section we present a motivating example to our work. Since Choudhary *et al* [1] [2] provide a really didactically explanation on possible cross-browser incompatibilities issues, we use their code snippets throughout our example.

Let us suppose that we have a web application which displays students' profiles in a tabular format. For each student, we are able to click on either their name or profile picture and get a detailed profile of that student, *e.g.* first and family names, student id, loans, and enrolled courses. In such web application, two distinct faculty workers (Alice and Bob) access the application on two different web browsers. The former uses Mozilla Firefox (FF), whereas the later uses Internet Explorer (IE).

At a first scenario, Alice wants to check whether a given student is enrolled in some courses and she clicks on his profile picture, hence the student profile is displayed. On the other hand, when Bob performs the same operation, the profile is not opened and he has to click on the student's name to open its profile. Listing 1 presents the code snippet that caused the previously described incompatibility. The reason for such incompatibility is that the `onclick` function is not supported on the `img` tag on IE, thus we observe the absence of a functionality on this browser.

Listing 1: An example of cross-browser incompatibilities in an HTML snippet.

```
<img src="student-id.png" onclick="
    openProfile(event)" />
```

At each user profile, the student's name is the header of that profile. Designers had decided that a small blue shadow would be present on the names header `h1`, as presented in Listing 2. In such scenario, Alice would notice the text shadow on FF whereas Bob would not notice it on IE. The reason for that incompatibility is that the `text-shadow` property is absent on IE. Notice that such incompatibility is simply an aesthetic one, though it is still an incompatibility between different browsers.

Listing 2: An example of cross-browser incompatibilities in a CSS snippet.

```
h1 { text-shadow: blue 2px 2px 2px;}
```

Finally, Listing 3 presents a code snippet which counts the number of enrolled courses for a given student. Once again, Alice would be able to see this feature on her FF browser, whereas Bob would not notice it, as the DOM property `childElementCount` is not supported on IE. In this scenario one of the web application features, *i.e.* counting the number of enrolled courses of a student, is not fully accomplished through different browsers.

Listing 3: An example of cross-browser incompatibilities in a JavaScript snippet.

```
var txt = $("items").childElementCount + "
    enrolled courses";
$("ncourses").innerHTML = txt;
```

For all the aforementioned examples, we can divide the cross-browser incompatibilities into two categories, that is functional/behavioral ones and visual/aesthetic ones. These categories merely divide incompatibilities into how severely they hinder the user experience. Nonetheless, unexpected behaviors in both categories would result in a new bug, which depending on its severity and complexity would impact the quality as well as the cost of that web application.

As a final remark, we would like to emphasize that all the aforementioned scenarios compare only two distinct browsers, *i.e.* IE and FF. However, incompatibilities may occur on any given browser. For instance, describing that some property has a `webkit` suffix/prefix will prevent both IE and FF of properly rendering any html component with such property. On the other hand, the Google Chrome browser would render such component without any issues.

3. RELATED WORK

Previous studies have surveyed the demand for tools that address different aspects of web testing [6] [5]. In such surveys, developers discussed requirements for better tool support. Analyzing and detecting browser compatibility was considered one of the most important issues to be addressed.

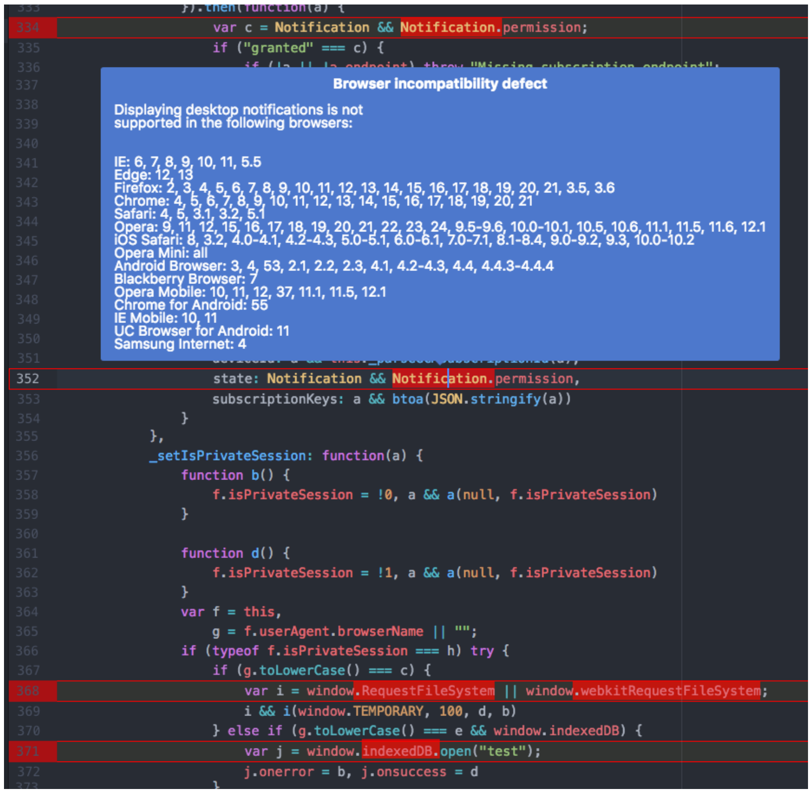
Automated approaches reduce the burden of manually inspecting web pages in order to find cross browser incompatibilities. For instance, Mesbah et. al. [4] proposed an automated cross-browser compatibility testing tool that captures the behavior of a web page in different specified browsers, and captures the behavior as a finite state machine diagrams. These diagrams are then compared in a pairwise manner and inconsistencies may be identified. Other approaches such as *CrossCheck* [1], or *X-PERT* [2] use crawlers on web applications, generating their DOM in different web browsers and then, comparing them in order to identify cross-browser incompatibilities.

However, the pair-wise comparison of DOM elements in different browsers is inherently limited and specific to the compared browsers and the compared versions only. In other words, the comparison is between pairs of browsers as opposed to comparing against ratified web standards. Therefore, an issue might not be detected if it appears to be similar in both of the examined browsers, while in reality it might show in other browsers which have not been included in the pair-wise comparison. Additionally, relying on the comparison of DOM elements through web page crawlers can be a time consuming task.

On the other side of the spectrum, manual approaches to identify cross-browser incompatibilities would typically involve consulting reference sites such as *quirksmode*[1] or *caniuse*[2]. In this approach, a developer would manually read through these websites in order to assess if a given feature is supported by a given set of browsers [2]. This approach would typically be very time consuming as it requires manual reading and inspection of a feature. Furthermore, this practice is highly dependent on the developer's existing knowledge whether or not a feature has possible incompatibilities and requires further checking. Therefore, the manual approach would often be unreliable and error prone, as well as time consuming.

Regarding the aforementioned approaches, they adhere to

[1]http://quirksmode.org
[2]http://canisue.com



```
        }).then(function(a) {
            var c = Notification && Notification.permission;
            if ("granted" === c) {
                if (!a || !a.endpoint) throw "Missing subscription endpoint";
```

Browser incompatibility defect

Displaying desktop notifications is not
supported in the following browsers:

IE: 6, 7, 8, 9, 10, 11, 5.5
Edge: 12, 13
Firefox: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 3.5, 3.6
Chrome: 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21
Safari: 4, 5, 3.1, 3.2, 5.1
Opera: 9, 11, 12, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 9.5-9.6, 10.0-10.1, 10.5, 10.6, 11.1, 11.5, 11.6, 12.1
iOS Safari: 8, 3.2, 4.0-4.1, 4.2-4.3, 5.0-5.1, 6.0-6.1, 7.0-7.1, 8.1-8.4, 9.0-9.2, 9.3, 10.0-10.2
Opera Mini: all
Android Browser: 3, 4, 53, 2.1, 2.2, 2.3, 4.1, 4.2-4.3, 4.4, 4.4.3-4.4.4
Blackberry Browser: 7
Opera Mobile: 10, 11, 12, 37, 11.1, 11.5, 12.1
Chrome for Android: 55
IE Mobile: 10, 11
UC Browser for Android: 11
Samsung Internet: 4

```
                state: Notification && Notification.permission,
                subscriptionKeys: a && btoa(JSON.stringify(a))
            }
        },
        _setIsPrivateSession: function(a) {
            function b() {
                f.isPrivateSession = !0, a && a(null, f.isPrivateSession)
            }

            function d() {
                f.isPrivateSession = !1, a && a(null, f.isPrivateSession)
            }
            var f = this,
                g = f.userAgent.browserName || "";
            if (typeof f.isPrivateSession === h) try {
                if (g.toLowerCase() === c) {
                    var i = window.RequestFileSystem || window.webkitRequestFileSystem;
                    i && i(window.TEMPORARY, 100, d, b)
                } else if (g.toLowerCase() === e && window.indexedDB) {
                    var j = window.indexedDB.open("test");
                    j.onerror = b, j.onsuccess = d
                }
```

Figure 1: Screenshot of the proposed IDE tool. A tooltip is shown whenever the user clicks on one of the highlighted red keywords, each of which indicates a cross-browser incompatibility. These results can also be retrieved via command-line interface.

the web testing framework proposed by Rode *et al* [7], which states that the overall quality of web applications should consider features, quality aspects, and time as their core factors. As these factors may compete, *e.g.* time constraints may affect quality, or complete features' coverage may not be achievable in a thigh schedule, we propose a lightweight approach to identify cross-browser incompatibilities by focusing on the time factor, *i.e.* the static analysis of frontend source code files can preemptively identify cross-browser incompatibilities even before a feature is fully functional.

4. PROPOSED TOOL

Considering the necessity of the early identification of cross-browser incompatibilities, in this section we describe our proposed tool, the *XCompatibility Checker*, a tool that flags the parts of a front-end page that may cause incompatibilities.

In a nutshell, the tool is to be used in two possible ways: as a command-line checking tool (for example, as part of continuous integration), and as an IDE tool. The approach considers web browser features that are the cause of common and well known compatibility issues and assign a cross-browser compatibility (XBC) score to them, *i.e.* a 100% compatibility score means that the feature has no compatibility issues and runs in all existing browsers. By cumulatively identifying keywords that match a feature catalog, the approach is able to compute a project's overall XBC score.

4.1 Tool's overview

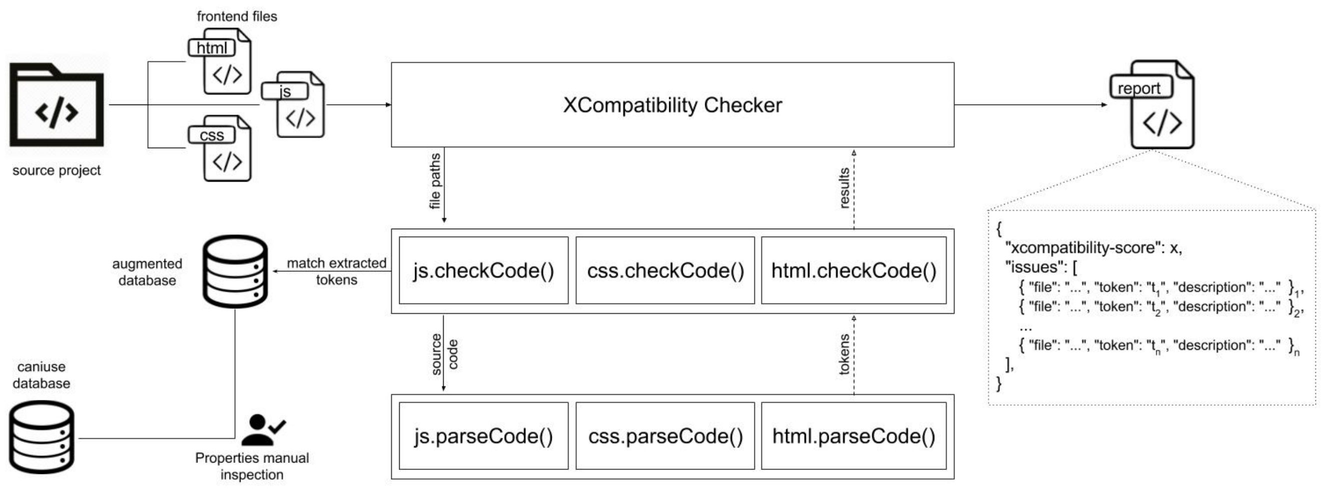
XCompatibility Checker is a tool that can be used in two ways: 1. to compute a compatibility score for project, and 2. to provide real-time feedback to developers while coding. Figure 2 presents the tool's overview. As an input, the tool receives a set of frontend files and a dataset which map keywords to features with cross-browser incompatibilities. According to file extensions, the tool parses each file, extracts its tokens and matches them against the feature-keywords dataset. Therefore, identifying keywords in the files input and computing a XBC score as an output.

The feature-keywords dataset is based on *caniuse*, an online service that provides insightful feedback for web application features that have cross-browser incompatibilities issues. On *caniuse*, one may query a feature and check the cross-browser compatibility of this single feature. Although, the service has some limitations, which are detailed as follows.

1. In order to identify features, the service relies on web developers knowledge and his/her active querying of keywords. Therefore, it is not possible to check every single keyword used in a project in a non cost/time consuming way;

2. There is no real-time feedback about the cross-browser compatibility of a feature. Therefore, a incompatible feature may be used in the project without taking into account its impact on the overall project's compatibility.

XCompatibility Checker targets these two restrictions by providing an API that *(i)* computes the project's overall



```
{
  "xcompatibility-score": x,
  "issues": [
    { "file": "...", "token": "t1", "description": "..." }1,
    { "file": "...", "token": "t2", "description": "..." }2,
    ...
    { "file": "...", "token": "tn", "description": "..." }n
  ],
}
```

Figure 2: Tool's overview

XBC score; and *(ii)* provides a comprehensive list of all keywords that caused incompatibilities on the project. This API is leveraged by a command line tool that can present the overall XBC score of a project as well as its list of incompatibilities and an IDE plugin (as shown in Figure 1) that can identify and highlight incompatibilities in a single file, pinpointing browsers that are (in)compatible with the feature related to that keyword.

4.2 Keyword-Features Matching

Our tool relies on a dataset that contains statistics about the XBC score for individual browser features. In order to build this dataset, we relied on the GitHub browsers compatibility dataset maintained by the *caniuse* community[3]. However, this database does not provide a precise way to match keywords to a feature. Therefore, our augmented dataset uses the compatibility statistics or the *caniuse* dataset, from which we derive our XBC score, and improves it by adding keywords that match browser features.

As an example, let us consider the drag and drop feature of web browsers. This feature is not supported in mobile cellphones, and is only available on desktop web browsers, such as Chrome or Firefox. *Caniuse* dataset provides the list of browsers and versions that are compatible with this feature, as well as a 44% score, indicating the percentage of browsers that run this feature. However, it does not provide which keywords are related to this feature. For instance, keywords such as `draggable`, `dragover`, `dragstart`, `dragend`, or `dragleave` may be used to model events related to the drag and drop feature. Therefore, our augmented dataset contains these keywords as a matching criteria.

With the augmented dataset at hand, HTML, CSS and Javascript parsers (presented in the lower most box of Figure 2) are used to extract tokens from the source code of the project and then, the tokens are matched against our set of mapped features and keywords. This functionality is exposed through a single method in the XCompatibility Checker's API and is exploited both by the command line tool and the IDE plugin in order to identify the keywords that are related to cross-browser incompatibilities.

[3]https://git.io/vyhqJ

4.3 Computing the cross-browser compatibility score

In order to compute the cross-browser compatibility score of a project, our API elects the least compatible HTML, CSS or Javascript feature extracted from the set of matched keywords as the global compatibility of the analyzed project. The rationale behind this is that a complete user experience will exercise all components of a web page and thus, the lowest compatibility score will define the percentage of browsers that do not present behavioral or visual inconsistencies to the user due to cross-browser incompatibilities.

For instance, let us suppose that we are analyzing a project that makes use of a CSS feature called `text-decoration`, which has an overall compatibility of 19%. The same project also makes use of `video` feature, which is supported on 94% of the browsers In such scenario, one may say that the cross-browser compatibility of the project is 19%. The least compatible feature (`text-decoration`) has 19% of compatibility and thus, this reflects the total reachability of the browsers that do not present inconsistencies on the evaluated project.

Similar to token extraction, this functionality is exposed through a single method in the XCompatibility Checker's API (illustrated by the middle box detailed in Figure 2) and is exploited by our command line tool to report project's overall XBC score.

5. METHODOLOGY

We are interested on finding to what extent cross-browser incompatibilities occur throughout different frontend web applications as well as what are the most common causes of cross-browser incompatibilities. Additionally, we are also interested on investigating the effects of identifying cross-browser incompatibilities in an integrated development environment. Can a mechanism that give warns or alerts about possible incompatibilities improve developers awareness on cross-browser incompatibility?

In order to answer the aforementioned questions, in this section, we detail our methodology and how we evaluated our proposed approach. To this end, Section 5.1 formally defines our research questions. Section 5.2 describes our data collection, whereas Section 5.3 describes our selection criteria for participants of a qualitative evaluation. Section 5.4 describes the design and execution of experiments that could assess our research questions.

5.1 Research Questions

We define and detail our research questions as follows.

RQ1: How often cross-browser incompatibilities occur?

By investigating open source systems, we plan to verify how often cross-browser incompatibilities occur in practice. Providing data describing the frequency of incompatibilities can be of valuable interest to researches and practitioners. Researchers can both use these findings as a reference on future work as well as discuss that this is indeed a recurring problem with different issues that need to be addressed. Additionally, a high number of incompatibilities can signal a red flag to the industry and the open source community, thus supporting the necessity to adhere and follow standards.

RQ2: What are the most common cross-browser incompatibilities?

As one identifies different cross-browser incompatibilities, it is interesting to verify what are the most common incompatibilities. By identifying recurring incompatibilities, researchers and practitioners can focus on designing tools and techniques that can mitigate/address those specific incompatibilities or defining new standardized features that overcome these incompatibilities. For instance, a review tool could leverage the automatic detection of those common incompatibilities and highlight them, hence code owners could be aware of their introduction on a pull request.

RQ3: Does a recommendation system improve software developers awareness on cross-browser incompatibilities?

Finally, we hypothesize that if there exists a high number of cross-browser incompatibilities, the reason behind incompatibilities is just that developers are not aware of them. Developers usually focus on the task at hand and, as a consequence, the definition of done usually omit that a feature should run in all browsers. If such definition is not considered, cross-browser incompatibilities may only arise later in the project's life-cycle through reported bugs. By providing a mechanism that could detect the introduction of cross-browser incompatibilities on the fly, developers could reason about alternative solutions and thus, the number of bugs due to cross-browser incompatibilities could be minimized.

As a final remark, it is important to emphasize that *RQ1* and *RQ2* foster the necessity of a tool that can address *RQ3*. All our research questions can be answered by our proposed tool, but *RQ1* and *RQ2*'s output may give more insight into how common cross-browser incompatibilities are and thus, the necessity of an IDE plugin that can leverage such tool, *i.e. RQ3*.

5.2 Data Collection

In order to answer *RQ1* and *RQ2*, we gathered data from a plethora of open source systems that had client side source code. With such corpus, we could assess the aforementioned research questions. While gathering the open source systems, our objective was to have a small yet representative set of applications with different frontend characteristics. The data extraction process was executed as follows.

Table 1: Survey questions regarding the IDE tool. Questions are based on the technology acceptance model questionnaire [3]

questions on perceived usefulness
Q1: Using the defect highlighting tool enables me to accomplish tasks more quickly
Q2: Using the defect highlighting tool increases my productivity
Q3: Using the defect highlighting tool makes it easier to do my job
Q4: I would find the defect highlighting tool useful in my job
questions on perceived ease of use
Q5: Learning to operate the defect highlighting tool is easy for me
Q6: My interaction with the defect highlighting tool is clear and understandable
Q7: I find it easy to get the defect highlighting tool to do what I want it to do

We selected a random sample of applications from a curated list of web applications[4] available on GitHub[5]. The selected list contains web applications in different languages, such as python and javascript, as well as different frameworks, *e.g.* Django, ReactJS, or Rails. The list is frequently updated by different contributors and provides an easy sample of projects with different sizes and contributors.

It is important to emphasize that selecting only projects with some threshold on its statistics could have biased our evaluation. High standard projects with consolidated communities and a long life span usually adhere to high standards and well defined practices, which could not represent the overall characteristics of open source projects. On the other hand, by selecting projects from a random sample our aim was to have projects in different spectrums of the open source community. Therefore, our selected projects range from small web applications with few contributors to well established ones, such as Redmine[6] a project management tool used by different companies to track their software development tasks.

We started with a sample of 48 projects and our minimum selection criteria defined that the projects had to be active in the last 3 months from the time that the data collection process was carried over. Since the list was curated by GitHub users, we considered the risk of a simple API been listed as a web application, thus we manually inspected the selected sample in order to verify if they indeed had frontend source code. From 48 projects, 41 (85%) were still active and 38(79%) had frontend source code. Therefore, these 38 projects consist the corpus or object of study for questions *RQ1* and *RQ2*.

Table 2 describes some of the characteristics of the selected projects. It presents the project's name and a brief

[4]https://git.io/vySRf
[5]https://github.com/
[6]http://www.redmine.org/

description of the project. It also contains the number of Javascript files, HTML files, and css files, respectively. Finally, the `files` column represents the sum of all the aforementioned file types, and `LOC` the total number of lines of code in all these files. The last two columns represent the number of detected incompatibilities and the overall cross-browser compatibility score (XBC) of each project, which will be further discussed in Section 6.

5.3 Participant Selection

In order to answer *RQ3* it is necessary to evaluate how software developers perceive browser incompatibilities while they execute daily frontend software development tasks. To this end, we required software developers with at least some experience developing web applications. Due to time constraints, we were unable to recruit software developers from industry and, as a consequence, we relied on recruiting graduate students from the departments of computer science and electrical engineering from the University of British Columbia.

Out of 10 invitations, 4 graduate students accepted and participated on our experiment. Regarding recruited participants, it is important to emphasize that graduate students may have a ground knowledge on software development practices and most of them had previous experience designing and developing complex systems, though they might not be so familiarized with web development practices and cross-browser incompatibilities. We believe that such characteristics reflect an open source community, in which different contributors have different levels of expertise. Therefore, our small sample of participants might reflect this scenario and thus, we considered them as the source of information to assess *RQ3*.

5.4 Research Method

In this section, we describe the design of empirical experiments that could assess our hypotheses and research questions. We break up the design according to our research questions and the corpus of study, *i.e.* open source projects, or human participants.

5.4.1 Design and evaluation of cross-browser incompatibilities in open source systems

In order do evaluate *RQ1* and *RQ2* it is necessary to measure a cross-browser incompatibility score in each one of the open source systems of our dataset. At each project, we gather the overall cross-browser compatibility score of that project as well as a list with the identified cross-browser incompatibilities of that project.

We compute the XBC score of each project through our command line tool. For each project, the tool's input is a list that contains pointers to the project's GitHub source code location as well as a list of files and folders that contain the location of the project's frontend files. The list of files and folder was gathered according to the manual inspection described on Section 5.2 and the inspection was assisted by a regular expression filter, which looked for the three types of files that are supported by our approach, *i.e.* `js`, `css`, and `html` files.

As we gathered the cross-browser compatibility score and incompatibilities list of each project, we analyzed the data and plotted the number of found issues through a histogram, presented in Figure 3. In a similar way, we counted the frequency of each incompatibility feature throughout the projects and plotted the top 10 incompatibility feature, as presented in Figure 4. Therefore, deriving our discussion and conclusions according to the extracted list of issues, overall XBC score and keywords frequency.

5.4.2 Design and evaluation of software developers awareness on cross-browser incompatibilities

In order to evaluate *RQ3* it is necessary to assess software developers' awareness on cross-browser incompatibility issues. We hypothesize that developers do not consider cross-browser incompatibilities while executing their daily basis activities. Therefore, this section describes an empirical experiment that can verify our refute this hypothesis.

The overall objective of the experiment is to check whether participants consider cross-browser incompatibilities while executing frontend tasks. To this extend, we designed small code review tasks that could be executed within the time frame of the experiment. Each experiment section spams from 20 up to 30 minutes and requires that participants review a random code snippet out of 4 real web pages in two different environments, which will be detailed in the following paragraphs.

Table 3 provides a small description of each one of the evaluated applications. On the other hand, Listing 4 presents a code snippet of one of the reviewed applications, in which the styles `cursor` and `translate3d` are examples of incompatibilities. For the sake of simplicity, we considered that the reviewed code was a new feature with just added code, rather than a comparison of previously existing code and added/changed one. At each section, we randomly present the tasks to the participants one at a time and then, we presented a random sample of a code snippet of that application. With those artifacts at hand, we asked the participants to point out if there are any possible defects or issues, as in a normal code review process. We also record the time taken to complete each one of the reviews. By the end of the sections, a simple debrief explained the experiment purpose to the participants and expected outcomes. Finally, a follow-up survey gathered demographical data about the participants as well as asked them if they have any statements regarding experiment's tasks.

Listing 4: Sample reviewed code

```
<div style="transform:translate3d(0, 0, 0)
    ;">
  <ul class="slider-list" style="transform
      :translate3d(0px, 0px, 0);cursor:
      inherit;...">
    <li><div>
      <a href="...">
        <img title="Introducing FREE 2-Day
            Shipping." />
      </a>
    </div></li>
   ...
  </ul>
</div>
```

Regarding the reviewed code snippets, it is important to emphasize that some of the presented snippets did not have any issues and were fully functional in any browser, whereas other ones have cross-browser incompatibilities, but were still runnable in some browsers. Such scenario might reflect frontend development practices, in which a feature is developed without taking into consideration the feature behavior

Table 2: Projects overview and evaluated cross-browser compatibility scores

Project	Description	# files				#LOC	# issues	XBC (%)
		js	html	css	total			
Trello tribute	A clone of Trello with React and Phoenix	40	0	0	40	2339	0	100
Cerebro	One-input productivity booster	142	23	2	167	6770	0	100
Wekan	The open-source Trello-like kanban	82	0	0	82	7101	0	100
Spina	A beautiful CMS for Rails Developers	13	0	5	18	10665	0	100
SoundRedux	SoundCloud Client Isomorphic Quiz Wall for itsquiz.com	91	0	2	93	5261	0	100
ReactionCommerce	A JavaScript Ecommerce App	829	6	176	1011	56141	0	100
CoderMania	An E-Learning Platform	5	1	4	10	401	0	100
Kinematic	Visual Docker Container Management on Mac and Windows	76	0	1	77	6768	0	100
Favesound-redux	The SoundCloud Client in React + Redux made with passion!	122	0	1	123	4267	0	100
In-Browser Playground	A playground for in-browser interpreters	29	0	1	30	1090	0	100
Itsquiz-wall	Isomorphic Quiz Wall for itsquiz.com	37	2	0	39	1538	0	100
Hours	Time registration that doesn't suck	3	0	3	6	267	0	100
Calypso	The new JavaScript- and API-powered WordPress.com	1868	1	0	1869	161743	13	97
Discourse	A Platform for Community Discussion	1695	3	140	1838	113810	19	94
React-Powered	Hacker News Client	42	1	1	44	2620	14	19
Huginn	Agent system that perform tasks online for you	20	3	5	28	14574	3	19
OpenSourceBilling	Beautiful Simple Billing Software	1868	1	0	1869	161743	3	19
Fulcrum	Agile Project Management Tool	53	3	4	60	8091	3	19
Eventx	Event Management System without Hassle	32	9	5	46	2230	82	19
Loomio	Make decisions together	3	2	5	10	509	5	19
Codango	Social Network for Coders	11	4	43	58	60519	22	19
Mezzanine	CMS framework for Django	104	29	87	220	24489	73	19
microapps-donation	A single page application that allows people to donate money	11	1	1	13	5328	29	19
OpenProject	Project Management System	277	3	106	386	19228	8	19
Coderwall	Professional network for Software	137	5	4	146	57477	8	19
Django-leonardo	CMS for everyone, easy to deploy and scale, robust modular system with many packages	51	11	202	264	19571	32	19
Huginn	Agent system that perform tasks online for you	20	3	5	28	14574	3	19
OpenSourceBilling	Beautiful Simple Billing Software	1868	1	0	1869	161743	3	19
Wagtail	A Django content management system focused on flexibility and user experience	100	6	279	385	18804	16	19
RedMine	Project Management Application	101	13	28	142	10257	30	19
Rocket.Chat	A web chat platform	965	11	182	1158	69262	6	19
Sharetribe	A Marketplace Platform	196	45	2	243	31611	28	19
Shoop	E-commerce Platform	119	5	8	132	12977	21	19
Django-CMS	Easy to use and developer friendly CMS	173	9	163	345	33229	33	19
Spectacle	A React library for Deck/Slide Presentations	93	6	1	100	4705	6	19
Helpy	Mobile First Helpdesk Application	110	114	22	246	55144	417	19
Spree	Ecommerce Solution	94	17	29	140	18493	3	19
MERNMAP	An interactive map for MERN(MongoDB, ExpressJS, ReactJS, NodeJS) Stack Developers	55	3	1	59	17525	75	19
Django-fiber	A simple, user-friendly CMS for all your Django projects	444	65	15	524	49140	858	0
Perseus	Perseus is Khan Academy's new exercise question editor and renderer	109	8	2	119	360438	259	0

Table 3: Participants tasks for the user study.

Task	Web page	Description
A	Amazon	Review a piece of the code related to the on-line store
B	Yahoo	Review the code related to the page's news feed
C	MSN	Review the code related to the page's news feed
D	Walmart	Review a piece of the code related to the on-line store

throughout different browsers. As a consequence, a later bug would have to be registered asking to address the feature in a specific browser/version.

The experiment was executed in two environments, presented to the participants in a randomly selected order. In the first environment, participants are given a html web page and its source code as well as a standard version of the Atom IDE[7]. On the other hand, the second environment consider the same artifacts with an augmented version of the Atom IDE. In this augmented version, we run our cross-browser incompatibility approach and highlight possible keywords that might give hints to the developers on the incompatibility issues. Considering these two scenarios, we gathered the number of issues on each review and the time taken on each one of them.

6. RESULTS AND DISCUSSION

In this section, we present the results and discussion of our proposed approach according to our defined research questions, *i.e.* *RQ1* (Section 6.1), *RQ2* (Section 6.2), and *RQ3* (Section 6.3).

[7]https://atom.io/

6.1 What is the overall measured cross-browser compatibility?

In order to measure the overall cross-browser compatibility score, we computed the compatibility score of our *corpus* of open source systems. Table 2 presents the computed scores per project on descending order.

By defining a threshold of 90% for projects as an acceptable cross-browser compatibility score, 14 projects (36%) have an acceptable XBC score. On the other hand, 24 projects (64%) had compatibility issues, with a median XBC score of 19%, meaning that in their evaluated versions such projects would only run in this percentage of browsers. Table 2 divides the upper half of the evaluated projects as cross-browser compatible projects and the lower half, as incompatible ones.

Regarding the projects without compatibility issues, it is worth to note that in general they are fairly small projects with less than 10,000 lines of code. Nonetheless *Discourse*, *Calypso*, *ReactionCommerce*, and *Spina* are complex systems and they still have a 100% XBC score. By inspecting GitHub issues on these projects, we were able to identify active discussion regarding cross-browser incompatibilities. For instance, GitHub issue #370 of the ReactionCommerce project discusses how default styling may interfere with cross-browser compatibility: "*the components do need to have some default styling and structure, so I'm not sure there's anyway around including some BS3 styles when needed for compatibility*". A similar query on the *Calypso* project indicates that it has 1,501 issues discussing compat-

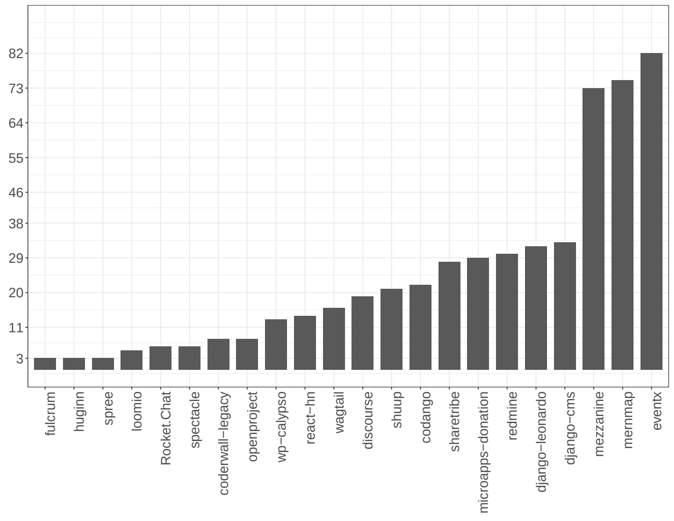


Figure 3: Number of issues per project

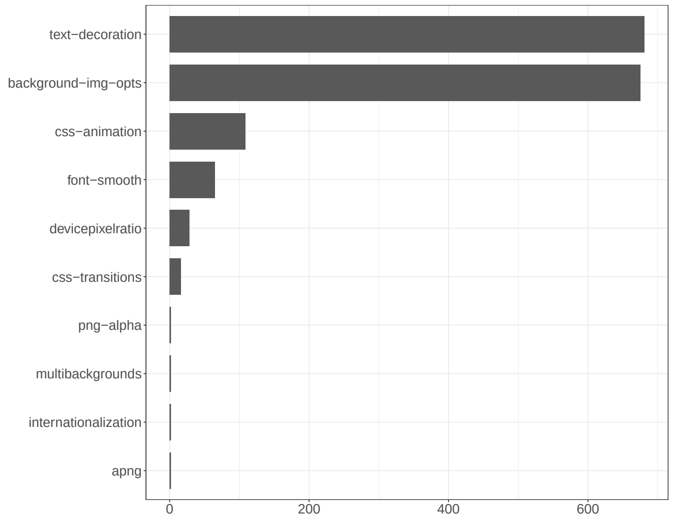


Figure 4: 10 most frequent properties that cause cross-browser incompatibilities

ibility. Disconsidering middleware compatibility discussions, manual inspection of these issues point out to the fact that some developers actively discuss browser compatibility, *e.g.* "*in this PR we are resetting styles to support the compatibility also for the dialog*".

On the other side of the spectrum, 24 projects had incompatibility issues. Figure 3 presents the number of issues per project. The graph excludes three projects (*Helpy*, *Django-fiber*, and *Perseus*) that are outliers with a high number of issues ($\#issues > 100$) as well as projects with a 100% XBC score ($\#issues = 0$). Regarding the remaining projects, the median value of the number of issues per project is 7, though they highly compromise the compatibility of the projects and, as a consequence their XBC score has a median value of 19%.

As a final remark, we repeated the process of querying the compatibility keyword on a random sample of the evaluated projects. We checked discussions on *Redmine*, *Huginn*, and *Mezzanine*. From our manual inspection of the Redmine project, we believe that incompatibilities are resolved on-the-fly, *i.e.* eventual functional/behavior incompatibilities are identified through registered defects. As an example, *e.g.* Redmine's defect #7954 discusses incompatibility issues on IE 9: "*IE 9 can not select issues, does not display context menu*". By our manual inspection, we conclude that the majority of the compatibility issues that matched our query discuss middleware compatibility rather than cross-browser compatibility. The other two randomly selected projects, Huginn and Mezzanine, follow the same pattern.

According to the aforementioned discussion we summarize our findings on *RQ1* as follows.

RQ1: Roughly, 60% of the evaluated projects have cross-browser incompatibilities. Most of the evaluated projects would run in only 20% of current existing web browsers.

6.2 What are the most frequent cross-browser incompatibility issues?

In order to measure the most frequent cross-browser incompatibility issues, we counted the frequency of the properties that cause incompatibilities throughout the evaluated projects. Figure 4 presents the properties' frequency.

The most frequent issue is `text-decoration`[8]. Text decoration styling refer to the method of defining the type, style and color of lines. Text decoration is not supported in most of the existing browsers (*e.g.* IE, Edge, or Android Browser), though it can be enabled in some browsers (Chrome and Opera). The overall XBC score of the property is 18.49% and as a consequence, the reason why most of the evaluated projects had an overall score of 19%. It is important to note that, despite being a visual/aesthetic incompatibility, this property might affect users' experience and diminish the overall readability of a web page. For instance, text decoration could be used to highlight price values or promotion codes and if not properly presented, user experience and a company revenue could be affected.

Roughly as frequent as the `text-decoration` property, the `background-img-options`[9] property is a CSS3 feature that is used in background images. Its XBC scoreis 97.83% and it is supported in almost all browsers (with partial support on Opera Mini). The same applies to most of the other top 10 evaluated properties, *e.g.* `css-animation`, or `css-transitions`. As all of them are well supported properties, there are few concerns regarding cross-browser incompatibilities within those features.

On the other hand, `font-smooth` controls the application of anti-aliasing when fonts are rendered and its overall XBC score is 37.26%. The property is only supported on Firefox, Chrome, Safari, and Opera and it is currently marked as a non-standard feature and the Mozilla Developer Network flags it as a largely incompatible, with different and unexpected behaviors. Similar to text decoration, this feature might affect user experience and it might compromise the overall quality of a web page. The `font-smooth` XBC score is still higher than the `text-decoration` and due to how we compute the XBC score, the lowest one prevails. Nonetheless, both properties should be taken into account.

[8]https://www.w3schools.com/cssref/pr_text_text-decoration.asp
[9]https://www.w3schools.com/css/css_background.asp

Table 4: Results of user study for the proposed IDE tool

participant #	task	proposed IDE tool on/off	task duration (minutes : seconds)	# of reported incompatibilities
1	A	off	6:50	8
1	B	off	4:44	8
1	C	on	7:27	28
1	D	on	5:41	12
2	D	off	10:11	2
2	C	on	7:41	7
2	B	off	6:24	1
2	A	on	7:19	10
3	C	off	12:56	5
3	D	off	3:47	0
3	A	on	14:32	22
3	B	on	6:13	14
4	B	on	6:48	13
4	A	off	7:33	4
4	D	on	5:46	34
4	C	off	5:32	2

According to the XBC score of the top 10 properties, it would be advisable that the research community addresses `text-decoration` and `font-smooth` properties. Since these are the two properties with the lowest XBC score, providing new properties or ways to deprecate them would highly improve projects' overall XBC score. For instance, by fixing `text-decoration` issues on the Redmine project, the project's XBC score would increase from 19% to 92%.

RQ2: The most frequent cross-browser incompatibilities are `text-decoration` *and* `background-img-option`*. It is worth to note that* `font-smooth` *and also* `text-decoration` *are the properties with the lowest identified XBC scores.*

6.3 Does a recommendation system improve awareness of cross-browser incompatibilities?

In order to assess if a recommendation system can be useful in increasing developers' awareness of browser incompatibilities, we performed a user study as described in section 5.4, in addition to a post-experiment survey using the technology acceptance model questionnaire [3].

Table 4 shows the results of the user study for the proposed IDE tool. Based on the results of this table, we observe that having the IDE tool ON for any of the tasks results in more cross-browser incompatibilities being reported by participants. For example, participants reported between 4 - 8 incompatibilities for task A when the IDE tool was off, but reported 10 - 22 incompatibilities for the same task when the IDE tool was on. The same trend can be observed for the other tasks: 1 - 8 (tool off) vs. 13 - 14 (tool on) for task B, 2 - 5 (tool off) vs. 7 - 22 (tool on) for task C, and 0 - 2 (tool off) vs. 12 - 34 (tool on) for task D. These results show that developers become aware of more incompatibilities when the proposed IDE tool is being utilized.

Figure 5 shows the results of the post-experiment survey we asked the participants to fill. The majority of responses (about 90%) had a favorable opinion (defined as all responses more favorable than neutral) on the usefulness and ease of use of the IDE tool. These results indicate that participants generally found the IDE tool to be useful in detecting cross-browser incompatibilities and that it was easy for them to report these incompatibilities using the tool.

6.3.1 Participants' comments

In addition to the user study and the technology acceptance questionnaire previously described, we also surveyed participants on open-ended questions and asked them to provide feedback regarding how useful the IDE tool was in their opinion. We include some of the participants' comments in the following paragraphs.

Some participants had a preference for visual examination in their analysis:

"I mostly detected any incompatibilities visually."

"The code highlight is very nice, but its not immediate to know where it is in the visual page".

These participants assumed that all highlighted code would correspond to a certain visual element in the page, which is not accurate since highlighted incompatibilities also include background Javascript processes or other non-visual (*i.e.* functional) attributes.

Another comment mentions the difficulty of manually looking up compatibilities:

"[manually] looking up the compatibility of every feature is tedious, and unlikely to to happen for many developers."

which is indeed one of the main issues this paper is trying to solve by providing an automated recommendation system. As one comment mentions:

"it was very easy [to use] because it directs your attention towards finding potential inconsistencies"

RQ3: A recommendation system does seem to improve awareness of cross-browser incompatibilities. Participants became aware of more incompatibilities when using the tool versus not using the tool. In addition, the majority of survey responses had a favorable opinion on the usefulness and ease of use of the tool.

7. THREATS TO VALIDITY

Regarding our proposed approach and designed/developed tool, it is important to highlight threats to the validity of our results. We base our threats to validity discussion according to Wohlin guidelines on experimentation in software engineering [8].

The threat to our *conclusions* is twofold: *(i)* due to the usage of parsers and how keywords are extracted, our XBC

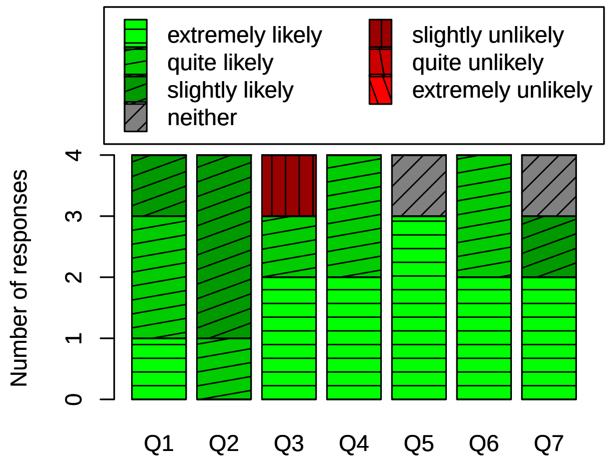


Figure 5: Post-experiment survey of participants.

score represents a lower bound on the compatibility of the evaluated projects, *i.e.* we are not able to extract mixed files such as embedded Javascript into HTML files, or inline css styling on some of the evaluated frameworks, thus there might be more incompatibilities than the detected ones; *(ii)* due to the relatively small number of participants, we do not have statistical power to derive conclusions from our experiment with human participants, hence we rely on the qualitative discussion of the tool's usage and also on the technology acceptance model survey [3].

Internal threats were mitigated by the randomization of our experiment, *i.e.* the IDE plugin was turned on and off in a random manner and the order of executing tasks was also randomized. Therefore, history and maturation threats were diminished. Also, participants' selection imposes a threat to our results since we used graduate students rather than web developers. Nonetheless, our recruiting criteria was designed in order to mitigate such threat.

In order to mitigate *external* threats, we have considered a corpus of 38 open source systems and evaluated our tool running it in this dataset. The evaluated systems differ on size, complexity, life-spas, and number of contributors. Therefore, they constitute a representative sample of different web applications. However, we emphasize that we did not evaluate if and how web designers contribute to such systems, hence there is a risk involving the usage of only applications that contain core and non-rich web browser features that may be widely cross-browser compatible. Future investigation of rich frontend applications will address this issue.

8. CONCLUSION

Web browsers are built by different organizations and writing software that runs smoothly on all existing browsers is a challenging task. The importance of reaching a good user base, which is distributed across different browsers, is high. Even though, cross-browser incompatibilities may hinder user experience and affect the quality and revenue of web pages. In this work we proposed a lightweight tool that could address this issue by bringing awareness to developers regarding cross-browser incompatibilities. By automatizing the process of identifying properties that are not supported by existing browsers, developers can improve the overall reachability of their web pages, *i.e.* they can deliver a better user experience with less browser-incompatibility issues.

The tool's evaluation was twofold: *(i)* we used it to assess the cross-browser incompatibilities of a corpus of open source projects; and *(ii)* we integrated it into an IDE in order to assess how it can assist developers on the task of detecting cross-browser incompatibilities issues. Our findings indicate that 60% of the evaluated projects have cross-browser incompatibilities, and most of the evaluated projects correctly run on only 20% of current existing browsers. The most frequent cross-browser incompatibilities' properties have an overall XBC score of 90%, *i.e.* they can run in this percentage of browsers. However, `font-smooth` and `text-decoration` properties do not adhere to this pattern and these two properties are the cause of the 20% computed compatibility for the evaluated projects.

As future work, a number of tool improvements can be done. For instance, the tool could be run on a larger dataset to provide more insightful data, more GitHub's issues could be mined to find discussions about cross-browser compatibilities and this could be linked to the actual results from the tool. Finally, more keywords and tokens could be added to our datasets in order to improve the tool's accuracy and precisely reflect the current state of web development.

9. REFERENCES

[1] S. R. Choudhary, M. R. Prasad, and A. Orso. CrossCheck: Combining crawling and differencing to better detect cross-browser incompatibilities in web applications. *Proceedings - IEEE 5th International Conference on Software Testing, Verification and Validation, ICST 2012*, pages 171–180, 2012.

[2] S. R. Choudhary, M. R. Prasad, and A. Orso. X-PERT: Accurate identification of cross-browser issues in web applications. *Proceedings - International Conference on Software Engineering*, pages 702–711, 2013.

[3] F. D. Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Q.*, 13(3):319–340, Sept. 1989.

[4] A. Mesbah and M. R. Prasad. Automated cross-browser compatibility testing. *Proceeding of the 33rd international conference on Software engineering - ICSE '11*, page 561, 2011.

[5] R. Ramler, E. Weippl, M. Winterer, W. Schwinger, and J. Altmann. A quality-driven approach to web testing. *Ibero-american Conference on Web Engineering, ICWE 2002*, 1:81–95, 2002.

[6] F. Ricca and P. Tonella. Web testing: A roadmap for the empirical research. *Proceedings - Seventh IEEE International Symposium on Web Site Evolution, WSE 2005*, 2005:63–70, 2005.

[7] J. Rode, M. B. Rosson, and M. Perez-Quinones. The challenges of web engineering and requirements for better tool support. *Methods*, 2002.

[8] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering: An Introduction.* Kluwer Academic Publishers, Norwell, MA, USA, 2000.