CROSS BROWSER COMPATIBILITY ISSUES

________________

A Thesis

Presented to the

Faculty of

San Diego State University

________________

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

________________

by

Dhanya Ramdas

Fall 2015
SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Dhanya Ramdas:

Cross Browser Compatibility Issues

Alan Riggins, Chair
Department of Computer Science

Carl Eckberg
Department of Computer Science

J. Carmelo Interlando
Department of Mathematics and Statistics

August 12, 2015
Approval Date
ABSTRACT OF THE THESIS

Cross Browser Compatibility Issues
by
Dhanya Ramdas
Master of Science in Computer Science
San Diego State University, 2015

Websites today have evolved from static, content-oriented web pages to feature-rich, complex and dynamic applications over the web. Web applications are popular because they can run on variety of web browsers and platforms on the client side. A user has the option to use one or more browser to view or interact with the web application. This introduces problems for web application developers because HTML, CSS and JavaScript specifications are implemented differently by browser vendors on different web browsers. These differences lead to how a web application looks and behaves in different browsers and they affect the end-user capability to effectively use that web application. The inconsistencies seen between browsers are what we term as “cross-browser issues” which remains the focus for this thesis.

With rapidly evolving web technologies, new features in browsers, variations in browser behaviors and evolving web standards, maintaining cross browser compliance is a challenging task for web developers. They spend a considerable amount of their time maintaining compatibility with several popular browsers. To ensure that a web application will function in a similar manner across all browsers, a web developer needs to identify issues and provide solutions that are in compliance with standards while maintaining backward and forward compatibility with previous and future browser versions respectively.

In this paper, we identify relevant cross-browser issues, the underlying cause for many of these issues and different frameworks and tools that web developers are currently using to address these issues.
TABLE OF CONTENTS

ABSTRACT ............................................................................................................................. iv
LIST OF FIGURES ................................................................................................................. vi

CHAPTER

1 INTRODUCTION .........................................................................................................1

2 CROSS-BROWSER ISSUES ........................................................................................5
   2.1 HTML Issues .....................................................................................................6
   2.2 CSS Issues ..........................................................................................................8
   2.3 Document Object Model (DOM) .......................................................................17
   2.4 JavaScript .........................................................................................................22

3 UNDERLYING CAUSES OF CROSS-BROWSER ISSUES .........................27
   3.1 Rendering Engines ...........................................................................................27
   3.2 AJAX ...............................................................................................................29
   3.3 Lack of Uniform Support for Web Standards ..................................................32
   3.4 Lack of Validation ...........................................................................................34
   3.5 DOCTYPE .......................................................................................................37

4 CURRENT SOLUTIONS ............................................................................................39

5 CONCLUSION ............................................................................................................42

REFERENCES ..................................................................................................................43
LIST OF FIGURES

Page

Figure 1.1. The browser market share [1] of major browsers between the years 1994 and 2014 .............................................................. 3

Figure 2.1. Screenshot of the errors displayed by the W3C HTML validator tool (http://validator.w3.org/check) for the above HTML code. ........................................ 7

Figure 2.2. W3C and Internet Explorer CSS box model. .............................................................. 10

Figure 2.3. Screenshots of CNN website on Internet Explorer 6 on Windows XP Platform, Internet Explorer 8 on Windows 8 Platform, and Firefox 29.0 on Windows 8 Platform. ................................................................................................... 14

Figure 2.4. The DOM representation of the above HTML document. ........................................ 18

Figure 2.5. Screenshot of Firebug tool to view the DOM tree structure. .................................. 21

Figure 2.6. Screenshot of GroovyCandies site when viewed on IE 11 browser. ....................... 23

Figure 2.7. Screenshot Groovycandies site viewed on IE 8 browser ........................................ 24

Figure 2.8. IE developer tool displays the JavaScript runtime error. ....................................... 25

Figure 3.1. Reference architecture for Web Browser. ............................................................. 27

Figure 3.2. Flow of the rendering engine ................................................................................ 28

Figure 3.3. Browser displays a message when JavaScript support is disabled .................... 31

Figure 3.4. Validation results of various sites ........................................................................ 35

Figure 3.5. Validation result of yahoo.com ........................................................................... 36
CHAPTER 1

INTRODUCTION

A web browser is a software application that runs on a client computer. With a browser, a user can request for documents that reside on a web server and on response from the server, render the document information in a viewable format for the user. These documents include web pages, images, videos and other multimedia files. Browsers use a protocol called Hyper Text Transfer Protocol (HTTP) to send the request for a document onto the web server.

Tim Berners-Lee invented the first web browser in 1990 and called it WorldWideWeb. Later, it was renamed to Nexus to avoid confusion with the World Wide Web (WWW). Then, the browser did not have a user friendly interface, but it allowed scientists from all over the world to view and share their documentation. The browser’s biggest limitation was that it ran on the NeXT operating system and it was not easy to port the browser on other operating systems. To overcome this, the browser’s software code was made public so that other developers could build their own browsers. By 1993, over half a dozen browsers were available; however, they were limited to the UNIX operating system, did not have a user-friendly interface and were restricted to the academic environment.

In 1993, Marc Andreessen and Eric Bina developed the first graphical web browser called Mosaic. Mosaic gained popularity because it ran on multiple operating systems, had a rich graphical user-interface, was easier to use and it was the first browser to support in-line images. Previous browsers did support the <img> tag but the images opened in a different window. It was also the first browser that could be downloaded freely. In 1994, Marc founded the company Mosaic Communications Corp but it was later renamed Netscape Communications and the browser was called Netscape Navigator. It became the most popular and widely used browser for surfing the internet in 1995.
The success of Netscape and the gaining popularity of the Internet led Microsoft to develop its own browser *Internet Explorer 1.0 (IE)* in August 1995. Three months later, Microsoft released a new version - *IE 2.0*. This led to intense competition between Netscape and Microsoft; both browser vendors started releasing new versions of browsers at a rapid pace over the years. This was the beginning of the browser wars. To gain market share, browser vendors developed proprietary HTML tags like `<blink>` and `<marquee>` that worked only in their respective browsers and encouraged web developers to use them. These proprietary tags did not meet the web standards specifications and were implemented differently. Netscape and Microsoft focused on introducing new features like *JavaScript* and *XMLHttpRequest* so they could leverage their competitor out of market. During this period, web developers had to build and maintain two different versions of the same web site for the two main browsers.

With the release of IE 4.0 in October 1997, Microsoft started dominating the browser market share. Microsoft gained advantage over Netscape because it bundled IE with the Windows operating system and it was not possible to buy a PC without the browser - IE became the browser of choice for everyone. With the release of IE 6 in 2001, Microsoft had won the browser wars. Even though IE had deviated from web standards, did not fully support CSS specifications and did not fix bugs (such as the IE box model that existed in IE 4 and 5 browsers), it became the dominant browser attaining nearly 90% of the market share. By the end of 2005 however, IE began losing its market share to a new browser called Firefox developed by the Mozilla Foundation. Web developers preferred using Firefox because of its strict adherence to web standards.

Figure 1.1 depicts the browser market share [1] of major browsers between the years 1994 and 2014.
Today, there are a number of web browsers out there but the most notable ones are **Google Chrome**, **Mozilla Firefox**, **Microsoft IE**, **Apple Safari** and **Opera**. These browsers run on diverse platforms and have similar user interfaces like an address bar to enter the URL, home, refresh, stop, back and forward buttons, tabbed browsing, bookmarking, content blocking etc. The 2 main areas where browsers differ from each other are -

**Rendering engines:** Every browser contains a rendering engine or a layout engine. A rendering engine processes the HTML document received from the web server and displays the formatted HTML document on the screen. There are 4 main rendering engines – **Trident** used by IE, **Gecko** by Firefox, **Webkit** by Opera and Safari and finally **Blink** by Chrome. Each of these engines handle the rendering of the web pages differently.

**Implementation of web standards:** Browser vendors do not always implement the web standards in similar manner. To gain competitive edge over other browsers, they interpret and implement the standards differently.

These differences in browsers cause a website to look or behave differently in different browsers.

There are a vast number of complex and dynamic web applications available today. We use them for work and personal activities such as online shopping, banking, emails, social media, online reservations etc. Web applications evolved from websites, it is a
collection of web pages that runs in a web browser. The client side technologies for building a web page are

- **HTML** – provides the structure of the page
- **CSS** – provides the presentation of the page
- **JavaScript** - controls the behavior of the page

With rapid evolution of web technologies, web developers are building more complex interactive web applications. The web application is developed on one primary browser and developers expect the application to behave consistently across all popular browsers. The ability of a web application to function in a consistent manner across all available versions of multiple browsers and in every version of each browser that run on different platforms is called a “cross-browser” [2].

However, when an application is tested on other browsers the inconsistencies on how different browsers behave are highlighted. These inconsistencies are called “cross-browser issues”. Maintaining cross browser compatibility is a crucial and challenging task for web developers.
CHAPTER 2

CROSS-BROWSER ISSUES

In the early 90s, browsers like Netscape displayed only static HTML pages consisting of text and images. When Microsoft entered the browser market with IE, it started competing with Netscape to gain browser market share. Both vendors chose to ignore the evolving web standards and started adding proprietary HTML tags to their browsers. Web developers started to use these tags to improve the visual experience of a web page, but soon realized the web page did not look the same on Netscape and IE browsers. This was the beginning of cross browser compatibility issues.

With the advent of CSS and client side scripting language like JavaScript in the mid-90s, websites became dynamic and enhanced the user experience. Browser vendors deviated from the standards and implemented these technologies differently. They focused on enhancing these technologies by adding their own extension to the web standards and in doing so the behavior of the browsers differed. By early 2000 with client side technologies evolving at a rapid pace and with new browsers entering the market, the severity of cross browser issues increased.

Cross browser issues can be broadly categorized as

Layout issues: These issues occurs when a web application is rendered differently on different browsers. For example if an element like a button or image is visible on Chrome browser but not visible on IE browser. The core technologies used for page layout are

- HTML
- CSS

Functionality issues: These issues occur due to differences in the way the JavaScript is executed by different browsers. Such issues are difficult to identify because the browsers do not throw any errors when the element does not function correctly. For example a submit button may be displayed correctly but nothing gets submitted when a user clicks on the button. The user interaction in a web page is provided by
The following sections in this chapter explains how each technology contributes to cross browser issues.

### 2.1 HTML Issues

Hyper Text Markup Language (HTML) defines the structure of a web page by using tags and attributes around text, images or links in a web page. A browser parses the HTML code to display well formatted web page. Each browser has its own HTML parser and the rules for parsing the HTML code is specified by W3C.

HTML is not a programming language. With programming languages like C, C++ etc if there any errors, the program will not compile until the error is fixed. But with HTML if there are syntax errors in a web page the browser does not prevent the page from being displayed. Most browsers have error handling mechanisms HTML errors to fix the HTML errors or it will go ignore the error and do its best to render the page by guessing the author’s intention [3].

HTML errors occur when
- an HTML tag is not closed
- the HTML tag is improperly nested
- proprietary or undefined HTML elements are used

HTML specifications do not define how to handle HTML errors, so every browser has their own error handling mechanism.

In the example provided below, the errors are that the strong element is incorrectly nested across multiple block elements, and the anchor element is not closed [4].

```
<p><strong>This text should be bold</strong></p>
<p>Should this text be bold? How does the HTML look when rendered?</p>
<p><a href="#">This text should be a link</a></p>
```

Now this is a violation of HTML specifications, but the browser displays the text anyway and does not display any error message. The latest browsers fixes these errors as per
the HTML specifications and displays the page same across all browsers but with older browsers that do not follow the HTML specifications these errors can be interpreted different ways causing the page to be rendered differently.

Figure 2.1 is a screenshot of the errors displayed by the W3C HTML validator tool (http://validator.w3.org/check) for the above HTML code.

Figure 2.1. Screenshot of the errors displayed by the W3C HTML validator tool (http://validator.w3.org/check) for the above HTML code.
Not all browsers fix the same HTML errors, one browser might fix the error and display the web page while another browser may not do anything causing the elements to be left or rendered incorrectly. Because browsers do not display the errors, instead they do their best to fix the invalid HTML document, web developers may not be aware of the HTML errors on the document unless they validate their web page.

After a certain point browsers should not tolerate such errors, this will force the web developers to test their code on various browsers before publishing it.

### 2.2 CSS Issues

In the early 90s browsers displayed simple text-based static web pages. The only styling applied were headings and paragraphs. Mosaic was the first browser to display inline images and this was achieved by adding `<img>` tags in HTML. Mosaic became the most popular browser due to this feature. Users started demanding more features to control the styling of web pages. To meet the demand of users and to gain competitive advantage over other vendors, browser vendors created proprietary HTML tags that controlled the style of the page. Netscape added tags like `<font>`, `<blink>`, `<frame>`, `<background>` and Microsoft added tags like `<iframe>`, `<marquee>`. This led to web pages looking different in the Netscape Navigator and IE browsers and this marked the beginning of browser wars.

With the introduction of the `<table>` tag in HTML 3.2 specifications, web developers were now able to arrange elements in multi-column layout. In 1997 W3C recommended that tables could be used for presenting data in tabular format and layout of a web page. Developers quickly realized table-based layout provided structure to the design of web page and they could position the elements – text, images, links and tables in rows and columns. With a table layout all columns will stretch to the same height to fit the content. It was the easiest way to divide a web page into subsections.

With the increase in demand for more design options web developers created WYSIWYG (What You See Is What You Get) applications such as Adobe Dreamweaver, Microsoft Publisher etc. It made web designing easy for users by generating the HTML code for the web page. These applications focused on design of the page and how it will be displayed on a web browser. However the beautifully designed web pages didn’t look the same on various browsers. On some browsers the images were blurred or the chosen font was
ignored and a default font was displayed or on text only browsers the web page was rendered differently. WYSIWYG applications failed because it used table-based layout to create complex page layouts. As the designs got complex the underlying tables got complex and complicated. Due to this browsers took longer to render pages that used table layout because browsers had to go through the entire table to calculate the positioning of elements on the page.

With the release of Macromedia Flash in 1996 the simple static web pages changed to dynamic and interactive web pages. Flash is a multimedia and web development software, it became popular because the software did not use complex tables for page layout and it enabled web designers to add animations and integrate multi-media content like videos on the web page. Major disadvantage of Flash was, if the plugin was not installed on the browser then the user could not view the animations or the video on the webpage.

Web designers began to realize by having better control over HTML and keeping web design simple, the web pages looked similar on various platforms and browsers. Because of HTML’s limitation to style web pages, the need for style sheet language arose.

Style sheet was not a new concept that was introduced in 2000s, it was available in browsers from 1990s but there were no syntax or standards for style sheets then. Due to presentational limitations on HTML and browser vendors not making any improvements on their default style sheets, various proposals for a style sheet language were made to W3C. The intent of this language was to define the positioning, layout, color and fonts of HTML elements in a web page and to separate the presentation from the content. Out of these proposals W3C recommended Cascading Style Sheets (CSS). One of the major advantage of CSS was the ability to group all styles into a separate CSS style sheet file which could be referenced by any web page.

In December 1996 W3C released CSS Level 1 specifications. It included properties for controlling fonts, text alignment, list formatting, spacing, margins, border, padding and positioning for most elements. It adopted a definition width in relation to content, border, margin and padding similar to that for a table cell. This has since become known as the W3C box model [5].
Microsoft IE 3, released on August 1996 was the first browser to support CSS. Even before CSS was published by W3C, Microsoft implemented CSS in its browser. Most of the features in CSS were implemented as per the standards except the box model.

As per CSS1 specification definition of box model is shown below where element width includes only the content. While IE’s implementation of the box model had element width including padding and border therefore expanding the size of the block and misconstruing the layout of web pages.

Figure 2.2 depicts the difference in how the width is interpreted between the W3C and Internet Explorer box models.

Microsoft’s misinterpretation of the box model was one of the major CSS bugs in IE until IE version 6 was released.

Netscape Navigator 4.0 was the next browser that supported CSS 1 but it did not fully test the CSS features it implemented. Instead it used JavaScript to manipulate the CSS rules
to dynamically change the style properties. This resulted in CSS bugs and limitations that prevented the implementations of CSS1.

Within few months after Netscape’s release, Microsoft released IE version 4. It was integrated with Microsoft Windows operation system and replaced its old layout engine with a new one called Trident to offer better support for CSS1. But it did not fix all the CSS bugs like the box model, inheritance within tables etc that existed in the previous version.

The only early browser with best implementation of CSS1 specifications was Opera 3.5, released in 1998. The browser vendor tested the CSS implementation before it was released. Compared to IE and Netscape, Opera was much faster, supported W3C standards and had a lot of unique features like page zoom, mouse gestures etc, it did not gain popularity. One of the main reason for this was Opera did not advertise itself and its market share was less than 2% to influence web developers.

To gain dominance in the browser market, the browser vendors battled by continuing to add new features like JavaScript to its browsers and deviated from web standards. The inconsistencies and limitations of CSS support in the 2 major browsers were highlighted when web developers used these new features. And to make sure that their websites rendered correctly in the various browsers, web developers had to implement CSS according to the wishes of these browsers. Thus, most websites used CSS in ways that didn’t quite match the specifications [6].

Microsoft dominated the browser market with its release of IE 5.5 in 1999. CSS 1 support in IE 5.5 had improved but it still fell short of fully implementing the web specifications. In this release Microsoft introduced browser specific extensions. The purpose of the extension was to test the CSS features before they are available as standard feature. These extensions should follow the specific format as shown below

```
'-' + vendor specific identifier + '-' + meaningful name
'_' + vendor specific identifier + '-' + meaningful name
```

Microsoft’s filter property is an example of browser specific extension. Instead of following the above format for the filter property, Microsoft used the following
IE5 browser for Macintosh was the only Microsoft browser that achieved nearly 100% support for CSS1 while none of the other Microsoft browsers for other platforms had this level of support.

To overcome the issues in CSS1 and to provide better control of page layout, W3C released CSS level 2 (CSS2) specifications in 1998. CSS2 is built on CSS1 and it contains more features to manipulate HTML tags. It added new features like absolute, relative, and fixed positioning of elements, z-index, table layout properties and concept of media types and so on. But some of the new features were not easy to implement by the browsers, so the browser vendors did not support it or like before they deviated from the W3C specifications.

IE 5 was the first browser to partially implement CSS2 specifications. Due to the improved CSS support on Microsoft browsers, IE started dominating browser market while Netscape was losing its influence on web design due to its numerous CSS bugs. With the release of IE 6 in 2001, Microsoft established its dominance and CSS gained popularity.

IE 6 fully implemented CSS2 specifications but some of the major CSS bugs like the box model issue, that existed in previous browsers still persisted. As most users used IE 6, web developers created web pages that was compatible with IE specifications rather than following W3C standards. From 2001 till 2005 there was no browser to compete with Microsoft, so it did nothing to fix the existing CSS bugs and improve CSS2 support.

During this time W3C committee realized that some parts of CSS2 was difficult to implement and some of the features were not implemented at all. So they revised CSS2 specifications to CSS2.1 to fix the errors, remove poorly supported or not fully interoperable features and added the already-implemented browser extensions to the specification [7]. CSS2.1 was released on 2004.

By end of 2004 a new browser vendor, Mozilla entered the browser market. Mozilla’s browser Firefox strictly adhered to W3C standards and fully supported CSS2 and CSS2.1. Opera and Mozilla improved their browsers with every release by adding unimplemented parts of CSS2.1 and fixing the CSS bugs.
Web developers realized with varying level of browser interpretations of W3C specifications and rendering problems, it is difficult to achieve consistent appearance across browsers and various platforms. Take a look at the 3 screenshots of CNN.com website, this is an example of how a website can look different on two different browsers – IE and Firefox and two different versions of the same IE browser.

Figure 2.3 depicts screenshot of CNN website on 3 different browsers. These screenshots are downloaded from www.browsershots.org.com (http://browsershots.org/http://www.cnn.com/).

In Figure 2.3, the design on IE 6 falls apart and the column on the right falls below the left column while on IE8 the last box on the bottom right is not properly aligned and falls below. This problem was caused because all browsers have different default settings when it comes to padding, borders and margins around and between elements so the website is not displayed the same on all browsers. To add to this problem the varying levels of support for CSS specifications by each browser vendor affects the design and layout of the website.

Because of the varying levels of CSS support by each browser vendor there is no W3C specifications to supply different CSS rules to different browsers. So web developers came up with hacks to address CSS bugs in each browser. A hack is a slight modification to the CSS code to work around a specific CSS bug on a particular browser or group of browsers, so that alternative styling may be applied to them. This was supposed to be a temporary, inelegant solution to fix a problem instead CSS hacks gained popularity and developers started using them without understanding the actual problem. One of the most popular CSS hack is the Box Model hack.

In recent years browser vendors are focusing on correctly interpreting CSS specifications and fixing existing bugs with every new release. They want to get rid of CSS hacks but it is not easy to do it because of the existing legacy websites. To achieve backward compatibility browser vendors are forced to support these hacks.

Most of the browser compatibility issues are due to CSS. Despite the fact the CSS syntax looks simple and easy to learn, it is complex and challenging to understand the rules and cascading property of CSS. Developers will continue to use hacks and filters if they don’t understand how CSS works.
Figure 2.3. Screenshots of CNN website on Internet Explorer 6 on Windows XP Platform, Internet Explorer 8 on Windows 8 Platform, and Firefox 29.0 on Windows 8 Platform.
2.3 DOCUMENT OBJECT MODEL (DOM)

DOM is an application programming interface (API) for representing HTML
document. DOM allows programming language to access the underlying structure of HTML
documents. It is a cross-platform and language independent API. The functionalities of the
DOM remains the same with any programming language. Web developers cannot access
DOM directly, they need to map it to a programming language in order to manipulate the
HTML document.

DOM is used for dynamically accessing, traversing and updating the content,
structure and style of HTML documents. The nodes of every document are organized in a
tree structure, called the DOM tree, with topmost node named "Document object". The nodes
represent the various types of content in a document. Following is an example of a DOM
representation of a simple HTML document.

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <h1>HTML Document</h1>
    <p>This is a simple HTML Document</p>
  </body>
</html>
```

Figure 2.4 depicts the DOM representation of the above HTML document.
The history of DOM is tightly coupled with beginning of JavaScript language [8] but it is not tied to JavaScript. DOM was developed in 1996 by Netscape Communications for detecting user generated events and modifying HTML elements. It was known as DOM Level 0 or Legacy DOM and was limited to accessing forms, links and images in HTML document. Around the same time Microsoft started using DOM Level 0 in IE 3 but it did not give access to images in HTML documents. Even though both browser vendors used same DOM level 0, the implementation of the DOM varied. Netscape 4 and Microsoft IE 4 and 5 accessed the elements in HTML document differently. Netscape browser used `document.layer` object to access the elements while IE browsers used `document.all` object. There was no standardization for DOM Level 0. So web developers had to check if the browser supported the required features. This was the beginning of support detection.

There are 3 standardized DOM levels. W3C published DOM Level 1 specifications in 1998. The specifications define the core interfaces such as Document, Node, Attr, Element and Text interfaces that can represent any structured document. It also defines various HTML specific interfaces. DOM Level 2 was released in 2000. The specifications extended the functionality of DOM Level 1 core and defined the functionality for events, style, views,
traversal and range. DOM level 3 specifications released in 2004 extended the core and events functionality. It focused on handling keyboard events.

By 2001 most browsers fully supported the DOM Level 1 standardizations and partially implemented Level 2 specifications. Despite supporting DOM Level 1 IE 6 continued to use its proprietary document.all object while Netscape 6 discontinued its proprietary document.layer object to support standardization.

The layout engine in a browser parses the HTML document to DOM tree. The construction of the DOM tree depends on the quality of HTML document. So if the HTML document is invalid, each browser will apply its error handling mechanisms to fix the HTML markup to construct a DOM tree. So the construction of DOM tree varies in each browser.

Let’s look at an example of how DOM tree varies between different browsers when encountered with invalid HTML markup.

```html
<!DOCTYPE HTML>
<head>
  <div><h1>DOM Testing</h1></div>  --- Error 1
</head>
<body>
  <p>Hello World!!!</p>  --- Error 2
  <p><strong>This text should be bold</strong></p>
  <p>Should this text be bold? How does the HTML look?</p>  -- Error 3
  <p><a href=""></strong>This text should be a link</p></a>
  <p>Should this text be a link? How does the HTML look when rendered?</p>  
  <p id="someId"></p>  --- Error 4
</body>
</html>
```

Using the web development tool available in browsers, let’s look how the browser parses the incorrect HTML markup into a DOM tree. The above HTML example was tested on the latest versions of Firefox, IE and Chrome browsers and it was observed that all the errors in the HTML document were handled in same manner by the browsers. The same cannot be said if we had used Netscape 4 or IE 6.
Figure 2.5 depicts the screenshot shows how the Gecko rendering engine in Firefox version 31 parses the invalid markup to form the DOM tree. The developer tool used to view the DOM structure is Firebug.

Different layout engines in browsers implement the DOM standards differently. Due to this the DOM tree generated varies from browser to browser. An example to illustrate the point is the handling of whitespace in the text content of an HTML document. Firefox includes whitespace in DOM whereas IE8 and earlier versions did not consider whitespaces in DOM. As a result DOM tree generated in both browsers were different. Microsoft corrected this issue on IE9.
DOM Testing

Hello World!!!

This text should be bold

Should this text be bold? How does the HTML look when rendered?

This text should be a link

Should this text be a link? How does the HTML look when rendered?

Some content

Figure 2.5. Screenshot of Firebug tool to view the DOM tree structure.
2.4 JavaScript

JavaScript is a loosely typed scripting language developed by Brendan Eich in 1995. It is used for client-side web development to add interactivity, improve functionality in web sites and to communicate with the web servers.

The idea behind JavaScript’s development was to allow web developers to add animations to static HTML pages and make web forms interactive. This is achieved by embedding the JavaScript code in a web page or imported as a separate file. After the browser loads the web page, JavaScript code traverses and manipulates the HTML document through the DOM.

JavaScript 1.0 was first used in Netscape 2.0 browser. The first version of JavaScript could manipulate only web forms, anchors and images in the HTML document. With the release of Netscape 4.0 browser JavaScript could now interact with the CSS rules to change the style of a web page dynamically based on user input.

Microsoft implemented JavaScript in IE 3.0 in 1996 but they couldn’t get a license from Sun so they called their implementation JScript. In the same year Netscape passed the JavaScript to European Computer Manufacturers Association (ECMA) for standardization. Because of trademark issues ECMA called the scripting language ECMAScript and documented the standards in ECMA-262 specification. Other languages that implement ECMAScript specifications are Adobe’s ActionScript and Microsoft’s JScript. Despite the different names the language is popularly known as JavaScript.

All popular browsers implement ECMAScript standards but every browser vendor interprets the standards differently. For example all modern browsers support the ‘addEventListener’ ECMAScript standard except IE 8. Microsoft then added this feature from IE 9 onwards.

JavaScript gained popularity with the advent of AJAX in 2005. AJAX used JavaScript to retrieve information asynchronously and update a segment of the web page without having to reload the entire page.

Because JavaScript is a loosely typed language developers are prone to make programming errors. These programming errors could be syntax errors, runtime errors, permission denied errors or null exception errors. Despite these errors browsers will render the web page but the functionality of the web page is affected.
The flexibility and dynamic nature of JavaScript allows a developer to write an incomplete code and such issues would be found only much later after the code has executed. Some errors like the syntax errors can be avoided if the developer uses tools like JSLint to do a sanity check before deploying the code. But runtime errors cannot be caught before the browser loads the web page. Such errors have unpredictable results on different browsers and browser versions.

Different browsers handle the DOM construction of invalid HTML document differently, resulting in structural differences in browser DOMs. Due to this it is impossible to determine beforehand if the script will execute without any issues.

For example when you access groovycandies site (www.groovycandies.com) on all the latest browser versions the site looks and behaves the same on all browsers. In the following example, the site is viewed on IE 11 browser.

In Figure 2.6 and 2.7 when groovycandies site (www.groovycandies.com) is accessed on all the latest browser versions, the site looks and behaves the same on all browsers.

Figure 2.6. Screenshot of GroovyCandies site when viewed on IE 11 browser.
But when the same site is accessed on IE 8 or IE 9 the images of the candies are not rendered and no error messages are displayed by IE. On further analysis it was found that a JavaScript error caused the rendering issue on IE 8 and IE 9. Runtime errors like these are captured using 3rd browser developer tools. Every modern browser has their own developer tools which makes it easy to analyze and debug the issues. Firefox uses Firebug, Chrome uses DevTools etc.

IE’s developer tool shows that the rendering issue was caused by JavaScript runtime error (see Figure 2.8). The innerHTML property was introduced by Microsoft in IE to provide an easy way to access entire HTML content at once. Other browsers adopted this property but implemented it differently because it was not part of web standards then. In IE8 and IE9 some elements is read-only on COL, COLGROUP, FRAMESET, HEAD, HTML, STYLE, TABLE, TBODY, TFOOT, THEAD, TITLE and TR objects [9]. So the contents of the TR elements cannot be changed directly. Because other browsers did not have this restriction the images were rendered. To prevent further incompatibilities Microsoft removed this restriction from IE 10 and upwards.
The core functionality of client-side JavaScript is to detect and react to the user interaction on the web page. Any user interaction with the web application is an event that has an associated function known as event handler. Mouseover, click, key press, focus on a form field, loading of a page etc. are all different kinds of events that trigger the JavaScript event handler to do something.

There are two types of event models – event capturing and event bubbling. Event capturing starts from the root of the DOM to the event that occurs on a HTML element. Event bubbling starts from the event on a HTML element and works its way up to the root of the DOM.

During the browser wars Netscape and Microsoft handled the events differently. Netscape Navigator 4.0 and earlier supported only event capturing whereas Microsoft IE 6 and earlier supported only event bubbling. This resulted in developers having to check what type of event model the browser supported.

With the release of DOM Level 2 specifications in 2000, W3C introduced a new model to control events in a web page. The new way to apply events to HTML elements looked like this:

```javascript
document.getElementById("my-link").addEventListener("click", myFunction, false);
```
In this model the first parameter of the addEventListener method is the name of the event, the second parameter is a reference to the function to call when the event occurs and the third parameter checks if event capturing or event bubbling should be used.

While most modern browsers used this method to register event handler Microsoft did not implement DOM Level 2 event model, instead Microsoft has its own proprietary attachEvent method to register the event handler

```javascript
document.getElementById("my-link").attachEvent("onClick", myFunction);
```

This method supported only event bubbling. Microsoft implemented this proprietary method up from IE 5 till IE 9.

Compatibility issues will continue to exist if browser vendors continue to support browser specific event models. To overcome these issues browser vendors are now implementing the W3C event model in current browser versions.
CHAPTER 3

UNDERLYING CAUSES OF CROSS-BROWSER ISSUES

3.1 RENDERING ENGINES

Every browser has a standard set of components: User interface, browsing engine, rendering engine, networking interface, JavaScript interpreter, XML parser and persistency engine. A reference architecture for web browsers proposed by is reproduced in Figure 3.1 below [10].

![Figure 3.1. Reference architecture for Web Browser.](image)

Essential to the browser's architecture is the rendering engine (also called a "layout engine"). Rendering engines figure out how to interpret the HTML, CSS and Javascript that makes up a page, and decide how to layout the page. Each browser has its own rendering engine. The four main rendering engines are

- Blink used by Chrome and latest version of Opera browser
- Gecko used by Firefox browser
WebKit used by Safari browsers
Trident used by Internet Explorer

Let’s look at the basic flow [8] of the rendering engine after the web page is downloaded by the browser.

Figure 3.2 displays the basic flow [5] of the rendering engine after the web page is downloaded by the browser.

![Flow of the rendering engine](image)

**Figure 3.2. Flow of the rendering engine.**

HTML parser is tightly integrated in the rendering engine. The rendering engine parses the HTML elements in the web page and constructs a DOM tree as shown in figure 4.

Each tag in the HTML document is a node in the DOM tree. At the top of the DOM tree is the document node which is called a root node it then branches into 2 element nodes `<head>` and `<body>` which then branches into other children. After the DOM tree has been constructed and CSS style sheets have been loaded the rendering engine parses the CSS elements into another tree called the “Render tree” (Firefox calls it frames while Webkit calls it Renderer or Render Object). Render tree is the visual part of the DOM tree and it’s not one to one relation with the DOM tree. Each node in the DOM would have zero or more boxes. Example - HTML elements having style as display :none will not be represented in the render tree.

After the render tree is constructed the layout of the render tree determines where each node should appear in the display window and gives the exact coordinates for each node. Finally the render tree is traversed and each node is painted on the screen.

Rendering engine will try to display the content as soon as possible. It does not wait until all the HTML elements are parsed before starting to build and layout the render tree. Parts of the web document will be parsed and displayed while the process continues with rest of the document that keeps coming from the web server.

Unlike CSS parsers, HTML cannot be parsed by regular parsers because browsers support invalid and non-standard HTML. That is why we don’t see an invalid HTML syntax
error because custom HTML parsers created by browser vendors fixes the invalid HTML errors like HTML tags not closed, using deprecated tags etc. The error handling mechanism is internal. Due to this HTML parsers are tightly integrated with rendering engine for performance reasons.

Browser vendors have reused rendering engine but with alternative design decisions for user level interfaces. Apple reused KHTML browser engine of KDE and made some modifications to create WebKit rendering engine for their Safari browser. Last year Google announced that it’s forking from Webkit and creating its own rendering engine called “Blink”. For users, growing differences in how the rendering engines work could mean that viewing the same site with different browsers will give different results.

3.2 AJAX

AJAX is short for Asynchronous JavaScript + XML. It is not a technology but a programming technique that uses combination of technologies to create interactive web applications. Even before the term AJAX [11] was coined data from web sites were exchanged to the web server asynchronously using XMLHttpRequest.

XMLHTTP was introduced as Microsoft ActiveX object in 1999 and it was implemented in IE 5.0. A year later Mozilla adopted this feature in its Firefox browser, they used JavaScript object to mimic Microsoft’s XMLHTTP and called it XMLHttpRequest. Soon Safari, Opera and Google duplicated XMLHttpRequest in their browsers. From IE 7 onwards Microsoft also implemented XMLHttpRequest as JavaScript object. Except for older browsers today all modern browsers support XMLHttpRequest feature.

XMLHttpRequest object is used to exchange data with a web server without having to do a full page refresh. It can be used to retrieve any type of data like XML, JSON, HTML or plain text. XMLHttpRequest gained popularity in 2005 when Google used this feature in its Gmail, Google suggest and Google maps web applications. Jesse James Garret [11] called this approach AJAX and since then it has been referred to as AJAX.

AJAX incorporates the following [12]

- XHTML (or HTML) and CSS, for marking up and styling information.
- The DOM accessed with a client-side scripting language, especially ECMAScript implementations such as JavaScript and JScript, to dynamically display and interact with the information presented.
The XMLHttpRequest object is used to exchange data asynchronously with the web server.

XML is sometimes used as the format for transferring data between the server and client, although any format will work, including preformatted HTML, plain text and JSON. These files may be created dynamically by some form of server-side scripting.

Using AJAX a user interface component in a web page can be manipulated or updated by exchanging data with the web server asynchronously using XMLHttpRequest and then JavaScript is used to display the response from the server, without having to reload the entire page. This is achieved using AJAX frameworks, which is an engine [11] introduced between the user and the web server.

AJAX frameworks consists of collections of pre-written JavaScript function that you can incorporate into your web application. It also provides a library of rich user interface components and a development environment for developing AJAX-based web applications. The advantages of using frameworks are as follows

- Help developers to build rich web applications that can look and act like a desktop application. These web applications provide better user experience, faster rendering of the pages and improved response time to user interactions. Netflix, Gmail, Youtube, Facebook etc are some of the popular AJAX-based web applications.

- Hide the incompatibilities between different browsers and platforms

- Makes the development lot easier by hiding the client server communication complexities.

AJAX frameworks include client-side and server-side components. There are numerous AJAX frameworks available and they range from small general purpose frameworks that can be downloaded for free to large commercial frameworks. Examples of popular client-side frameworks include JQuery, YUI Library MooTools, Dojo toolkit etc. For the server-side the most popular AJAX frameworks are Google’s GWT, Backbase, AribaWeb etc.

Despite the popularity of AJAX based web applications, AJAX does add more problems to the existing cross browser issues. With so many frameworks available in the market, a developer has to spend a considerable amount of time to figure out which framework to use because the underlying technologies differ.
The major drawback of AJAX framework is its dependency on JavaScript. Because every browser implements JavaScript differently AJAX may not work consistently across all browsers.

Another drawback is, if a user disables the JavaScript support on their browser then AJAX no longer works causing the website to not function properly. For example when I disabled JavaScript support on Chrome browser version 37 and tried to access a web application like Gmail that heavily uses AJAX, the following message in Figure 3.3 is displayed by the browser.

![Figure 3.3. Browser displays a message when JavaScript support is disabled.](image)

The same application will work fine in Firefox or any other browser if JavaScript has not been disabled in the browser. Also AJAX is not supported on browsers on mobile phones, so developers have to keep this mind and provide an alternative solution. This is achieved by using NOSCRIPT tags. This leads to additional development effort.

Frameworks were created to hide the complexity of developing an AJAX-based web application and the incompatibilities between different browsers. Despite this AJAX interfaces makes it difficult to implement, debug and maintain web pages. There is a need to check the browser version and if it’s IE 5 or IE 6 then XMLHTTPRequest is created using ActiveX object. AJAX interfaces can also increase the number of requests to the web server and databases and this leads to slow response time and timeouts and it makes your web site prone to security threats.

There is no browsing history when AJAX is used to update and manipulate a user interface component like forms in a web page. This means users cannot bookmark or navigate to and from the web page using the back and forward buttons on the browser because the URL does not change. This causes the browsers to behave differently from its original design and creates a challenge for search engines to interpret such pages.
Search engines parse HTML but not JavaScript code. And AJAX uses JavaScript to display the dynamic content on the web page. This dynamic content is invisible; if you view the source code of your page you will not find the HTML code for the dynamic content that is displayed on the page. There are various ways to change the URL and make the forward and back buttons work when a page is updated using AJAX but if proper coding practices are not followed then AJAX-based web pages will not be detected by search engines. And this could cause loss in businesses for companies.

By using AJAX frameworks a developer will learn how a framework works but will not understand the different technologies used to build an AJAX based web application. Without this knowledge it is difficult to build a web application that works consistently across all browsers.

3.3 Lack of Uniform Support for Web Standards

The World Wide Web Consortium (W3C), along with other groups and standards bodies define and describe the core technologies (HTML, CSS, DOM and ECMAScript) for creating and interpreting web-based content [13]. By building web applications that are standard compliant web developers can avoid using proprietary technology and improve the overall functionality, interoperability of the web application and also achieve browser independence.

Browser vendors are part of the W3C committee and have been involved in creating the web standards. And yet these vendors don’t implement the web standard specifications as quickly or they choose not to implement certain specifications.

There are two reasons for this situation –

1. Supporting backward compatibility for older browser versions.
2. During the 90s and early 2000 very few websites were standards compliant and they were built to work on IE 6 and Netscape 4.0 browsers, these browsers were known for deviating from web standards. So when modern browsers started implementing web standards these websites would not render or function correctly. Likewise today’s web applications that are standard compliant will not work well on older browsers. This means browsers had to support forward compatibility and backward compatibility to get websites working. To overcome this issue browsers started supporting two different rendering modes – quirks mode for backward compatibility and standards mode for forward compatibility.
3. The problem with supporting backward compatibility is browser vendors have to continue supporting the bugs and quirks that existed in older browsers. It is a challenge for developers who wish to improve the standard for older websites, since it would mean that they have to continue to use various hacks, invalid specifications and maintain browser specific code. In the fast pace of doing business, companies are generally inclined to provide functionality which improves accessibility of their content on all browsers rather than maintaining standards compliance. Due to these inherent problems, we find a lack of effort from the development community in building a fully standard compliant web application.

4. Although browser vendors do understand the importance of web standards, no browser is fully standard compliant because browser vendors are always innovating and implementing new features that are not yet standardized. This allows them to compete with other browser vendors. Due to this competitive nature of browser vendors what emerges is a Web that is evolving at a rapid pace while W3C and other standards bodies are not able to keep pace with this growth.

5. There was no governing body to enforce browser vendors and web developers to follow the web standards.

6. W3C has not developed a certification program or authorized any other program to certify conformance to W3C standards. Conformance testing is defined in ISO/IEC Guide 2, "as any activity concerned with determining directly or indirectly that relevant requirements are fulfilled" [14]. Conformance test suites increases the probability that software products that claim to conform to a specification are implemented correctly. The benefits of conformance testing are portability and interoperability where -

Portability ensures that a website can be accessed from any browser and on any platform.

Interoperability ensures that a standard compliant website does not use proprietary features of a browser

7. In order to test that a specification is faithfully adhered to, one must carefully examine the specification and compare the implementation's result against the syntax and semantics contained in the specification. Often times, errors or ambiguities in the specification are discovered. These errors and ambiguities can then be fed back to the standards committee who developed the specification, for correction. For software developers, using conformance test suites early-on in the development process can help improve the quality of their implementations by identifying areas in which they conform as well as those areas in which they do not. [15]

8. Because W3C does not have any such conformance testing, browser vendors and web developers do not adhere to web standards completely. So a group of developers formed the Web Standards Project (WaSP) group in 1998. This group developed the conformance tests and called it Acid1, Acid2 and Acid3 tests. The Acid tests are a battery of 100 tests that check a given browser for enforcements
of various W3C and ECMA standards. A score out of 100 is rendered if the web browser complies with all the 100 tests. Although the latest versions of all the modern browsers score 100, it does not imply that browser vendors are fully adhering to W3C standards. This is because these tests are not comprehensive and test only specific limited selection of browser features.

9. Acid3 tests are not the only tests that check the browser conformance to web standards, there are other test suites like –

   DOM Test suite developed by National Institute of Standards and Technology (NIST)

   CSS2.1, HTML and SVG test suites developed by W3C working groups

Since W3C does not have a formal testing and certification program, it is important that browser vendors and web developers spend more time to test the browsers and web applications adherence to web standards.

### 3.4 Lack of Validation

“W3C validation is the process of checking your website against a formal formatting standard put up by W3C” [16]. Following are the benefits of validating a website

1. Displaying properly and consistently on all modern browsers
2. Rendering of the pages are faster because the browser does not have to spend time fixing your errors
3. Will continue to work well with newer technologies
4. Reduce the amount of code on your page making it easier for search engines to index the page.

The W3C validation tools (HTML validator, CSS Validator) and various browser extensions (HTML Tidy, Web developer etc.) validates the code and pinpoint the errors on the page which otherwise would not be displayed by the browsers. Validation tools does not point out the best practices in web development but it makes debugging lot easier. For example unclosed tags can go undetected and it can mess up the page layout. Without these tools it would be difficult to find these errors.

So why do most websites fail validation? The screenshots in Figure 3.4 (http://validator.w3.org/check) display the validations results of few popular and commonly accessed websites.
Figure 3.4. Validation results of various sites.

Despite these errors sites like Facebook, youtube, amazon works on all browsers. So is it important to validate your website? In reality if a website functions correctly and the
content is accessible on all modern browsers then users don’t care if there are errors on the website. Even browsers don’t care about the errors on the website. As discussed in the previous chapters the in-built error handling mechanism in browsers fix most of the errors and render the page as best as it can.

Only web developers care about validating their site. But they are not too concerned if the site fails validation. The most common reasons why sites do not validate are:

1. Many large corporates use content management systems to build their websites and many of these systems generate invalid HTML code.

2. Developers continue to use browser specific hacks to support backward compatibility.

3. Developers use vendor specific prefixes, and new features in their code which have not been standardized yet. In Figure 3.5 (http://validator.w3.org/check) yahoo.com fails validation because it uses an attribute called aria-role and as per W3C it is in working draft status.

![Errors found while checking this document as HTML5!](image)

**Figure 3.5. Validation result of yahoo.com.**

Its good practice to validate a web page during development and fix all the errors or as many as possible. This ensures that the web page is standard compliant and will work across all browsers consistently. But it is not critical to achieve 100% validated site because as long as the web site is displayed correctly on all browsers, users accessing the site will not check to see if the site passed validation. Ultimately W3C validators are tools to help developers code better.
**3.5 DOCTYPE**

DOCTYPE short for Document Type Declaration (DTD) informs the browsers and validators on the HTML version used to code the web page. Every web page includes the DOCTYPE in the first line of its HTML code.

Over the last decade browsers have become mostly standards compliant but there are lots of websites that were built in the 90s that still exist and do not adhere to web standards. These sites were built on older browsers like Netscape 4, IE 4, 5 and 6 which had lot of bugs, so the sites expected these bugs and non-standards to exist in the browser in order to render the page. But as browsers started implementing the standards correctly and became standard complaint 2 situations arose.

Older websites work well only on older browsers and does not render correctly on modern browsers.

Websites that are standard compliant did not work well on older browsers but had no rendering issues on modern browsers

This became a major problem for browser vendors. They didn’t want users complaining that their site didn’t work well on the browsers and at the same time they wanted to adhere to web standards.

To overcome this problem and support both backward and forward compatibility, browser vendors decided to use 2 different rendering modes

quirks mode – emulates the quirks of older browsers with incomplete or incorrect implementation of web standards to render the page.

standards mode – renders the pages according to web standards.

To trigger the appropriate rendering mode, browser vendors decided to use the DOCTYPE in the HTML document. For example when a browser sees the following DOCTYPE it renders the web page in a standards mode.

```html
<!DOCTYPE html>
OR
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```
If no DOCTYPE is used in the HTML document or if a DOCTYPE with incorrect syntax is used then the web page is rendered in quirks mode.

This is known as DOCTYPE switching or DOCTYPE sniffing. DOCTYPE switching was first implemented on Mac platform by IE 5 in 2000 and then quickly adopted by other browser vendors.

The major flaw with DOCTYPE switching is the developer’s failure to understand the role of DOCTYPE in rendering a page. Developers who don’t follow the web standards strictly, fail to pay attention to the DOCTYPE they use in their HTML code. They end up adding incorrect DOCTYPE or forget to add it, causing the browser to render the page in quirks mode. This might be okay if the web page was built on an older browser, but if the web page was built on a modern browser then the page is rendered incorrectly resulting in debugging the issue for hours.

Another problem is DOCTYPE switching in IE version 8 and above operates differently from other modern browsers. Lot of websites in IE 6 had valid DOCTYPE but these sites also used CSS hacks to render correctly on IE 6. When IE 7 was released, Microsoft had fixed lot of the bugs that existed in IE 6 due to this the sites broke on IE 7. To counter this problem Microsoft decided to use both the DOCTYPE and the META element in the head section of the HTML code to switch to quirks mode. For example

```
<meta http-equiv="X-UA-Compatible" content = "IE=8" />
```

To make matters worse Microsoft introduced more than 2 rendering modes in IE browser. IE 8 has 4 modes – IE 5.5 (quirks mode), IE 7 (default standards mode in IE 8), IE 8 (almost standards mode) and IE 8 (standards mode) [17]. IE 9 has 7 rendering modes, while IE 10 has 11 rendering modes and latest IE 11 has 16 rendering modes. As you can see with every browser release the number of rendering modes increased. Rather than providing a simple solution to support backward compatibility Microsoft now made it more complicated.

Thankfully no other browser vendors has adopted Microsoft’s meta element switch. To maintain interoperability with other browsers, developers should not include the meta tag on a web page being built on IE 8 and above. Instead the meta tag should be used only if an old site needs to be rendered correctly on the IE 8 and above browsers. This will achieve browsers to behave in a consistent manner.
CHAPTER 4

CURRENT SOLUTIONS

In the recent years, the major browser vendors have adopted web standards and have strived to achieve similar functionality amongst the browsers. While some browsers may be missing certain HTML5 APIs, the core JavaScript rule of DOM traversal, manipulation, event handling, server communication and CSS effects are well implemented and documented. Despite this, there are still inconsistencies between different browsers because of the different rendering engines, so every browser can still exhibit the look and functionality of websites differently.

To ensure that a website works in a similar manner across all major browsers, web developers currently use the following techniques in web development

- Creating multiple stylesheets for various browsers
  Among the major browsers, the older version of IE browsers have been the most problematic. So it’s easier for developers to add IE conditional statements to detect the user’s IE browser version to load the specific CSS stylesheet for IE browsers. The drawback with this technique is, developers now have to maintain various versions of stylesheets for different browsers.

- Using browser vendor prefixes
  To stay competitive, browser vendors add new features in their browsers that are not part of the web standards yet. Since W3C standards are constantly evolving, it is important that you ensure that your styles have sensible fallbacks for when a browser doesn’t support a CSS property. Older browsers will understand those fallbacks and skip the code it cannot understand, while newer browsers will understand both the fallbacks and the newer properties.

  To implement these new CSS features web developers use a proactive approach by adding browser vendor prefixes to the new CSS property. For example the correct way to use the new CSS3 transition property is to list all the prefixes first and then end with W3C standard property.
<table>
<thead>
<tr>
<th>Browser</th>
<th>Vendor Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chrome, Safari</td>
<td>webkit-transition</td>
</tr>
<tr>
<td>Firefox</td>
<td>moz-transition</td>
</tr>
<tr>
<td>Microsoft</td>
<td>ms-transition</td>
</tr>
<tr>
<td>Opera</td>
<td>o-transition</td>
</tr>
</tbody>
</table>

- With this approach the web developers achieve two things –
  They can use the new browser features to develop their web applications
  Code is now both backward and forward compatible. Older browsers that support
  their own vendor prefixes will use that property while browsers that follow W3C
  standards will use the standard property and ignore the rest.
  The drawback with this approach is that you have to write this property 4-5 times
  in a stylesheet to get it to work in all the browsers, and later when this feature
  becomes a standard developers do not revisit their code to remove these prefixes.
  And browser vendors will still continue to support the prefixed version of the
  property rather than discontinuing the support.

- Using frameworks
  Compared to the above two methods mentioned above, front-end frameworks
  offer better solution to tackle layout and functionality issues. A framework is a set
  of CSS, HTML and JavaScript files which can be used as a basis to start the
  development of a web application. Biggest advantage of using frameworks is,
  they reduce the web development time while improving the code readability,
  standardizes the layout code and solves many issues like browser specific bugs.
  The rise of responsive web design techniques that can adapt to various resolutions
  for different mobile and desktop devices, is the other reason leading to the
  emergence of responsive frameworks. There are lot of frameworks out there and
  each is slightly different in its own way, but the two most popular frameworks are
  Bootstrap from Twitter and Zurb’s Foundation.

  Bootstrap is the most popular and widely used front-end framework and it is
  compatible with the latest versions of the major browsers. It is a free open-source
  framework that contains an extensive set of standardized CSS classes and
  JavaScript components, allowing developers to select the components they wish
  to use in their project. Bootstrap also offers configurable features like styled
  typography, HTML templates, navigation, buttons, set of forms etc. With this
  modular approach, a developer can save considerable amount of time and effort
  resulting in rapid development of a web application. Also web applications today
  need to be responsive, slickly designed, accessible on mobile devices and fast.
  These requirements are achieved by Bootstrap’s core feature, responsive grid
  system. It helps developers to create web applications that can automatically
adjust itself to look good on all devices like mobile phones, smart TVs, tablets and desktop.

The goal of Bootstrap is to abstract routine tasks into generic modules that can be reused on all major browsers, so there is consistency and the code can be easily passed on to another developer for maintenance. Bootstrap has well-written documentation with examples that makes it easier for developers lacking significant web experience to understand the framework and build a web application without much effort.

Bootstrap has an active community of developers involved in the project, so the framework is constantly updated and extensively tested on all major browsers, and most of the cross browser issues have been fixed. This reduces the debugging time for developers. However, there are some browser bugs that Bootstrap is unable to solve. Bootstrap maintains a documentation called “wall of browser bugs” [18] that list the known bugs in browsers.

Disadvantages of using a Bootstrap or any framework

- Impedes learning and better understanding of the core underlying technology

  One needs a sound understanding of CSS, HTML and JavaScript and how these interwork in a project architecture. The abstract nature of a framework limits the developers from understanding the architecture of a web application.

- Unused and/or bloated code

  In many cases, a framework provides more functionality than the requirements from a project. As such, there is often a large percentage of code like CSS classes and JavaScript libraries that never get executed. This can greatly degrade the performance of a framework.

Despite the disadvantages, developers prefer using frameworks because of its inherent capability to get a web application running quickly with minimum debugging. It cuts down the time required to create a website, which will make the website cost a bit less. The use of a framework will go a long way towards building powerful, responsive application with standardized and well documented code that can be easily passed on and maintained by other developers too.
CHAPTER 5
CONCLUSION

In this thesis, we have looked at the history and evolution of web browsers and identified several underlying causes of cross browser compatibilities. While modern browsers are similar in design and functionality, major browser vendors have been proactive in introducing and implementing new features that are not finalized by W3C standards. This approach benefits web developers since they can use the new features in the web applications, but it opens the possibility that the specification of the new features might change once it gets finalized by W3C. Users then see different browser behaviors.

Getting a web application to behave consistently across all browsers and diverse platforms has proven to be a major challenge for developers. To ensure compatibility, developers have to spend a lot of time to manually inspect the web pages and their behaviors on different browsers. There are existing commercial tools like BrowserStack, SauceLabs, BrowserShots etc. that help in detecting the compatibility issues but these tools do not identify the cause of these issues. So developers still have to spend a considerable amount of time to identify and understand the underlying issues and fix the issues manually.

To ensure compatibility, developers must assume nothing about browser support and must have a suitable fallback in place for when it’s lacking. Usually some kind of workaround is required in such cases. Using vendor prefixes for new features and adding W3C property as a fallback is such a workaround. Another alternative is usingdoctype switching if older browsers are not standard compliant.

Today, most developers prefer to use frameworks for developing web applications because they have been tested extensively on all major browsers. Currently there are lot of frameworks available and each one is slightly different from the other. But they all share a common goal to display the web application on all major browsers and platforms consistently.
REFERENCES


