Browser Compatibility: the State of the Art from a Historical Perspective

Joachim Roeleveld
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
m.j.roeleveld@student.utwente.nl

ABSTRACT
Browser compatibility is an aspect of web development that reaches back to the beginning of the internet. Because browsers have been marketed by various vendors in a competitive landscape, fragmentation was inevitable. This creates technical challenges for web developers whom have incentive to create websites that perform predictably on different browsers, browser versions and rendering engines. Over the years the browser landscape has changed and so has mutual compatibility. Until now, no research has been conducted that provides insight in the developments of this area. Little is therefore known about the state of the art relative to the historical context. This paper provides this missing information by using available data sources. The outcomes show that developers should be less concerned with supporting legacy browsers and versions and can be more confident about feature support because of an increase in use of more compatible browsers. However, it was also found that feature support among browsers is decreasing.

Keywords
Browsers, Browser compatibility, Browser features, Browser vendors, Rendering engines, Web development, Web standards

1. INTRODUCTION
1.1 Background
From the beginnings of the internet around 1970, internet access has grown from a dexterity to a commodity and even a human right [15]. Browsers in the meantime have become the de facto interface to browse the web for the average user.

From then on, new browsers and rendering engines have been developed by parties that tried to conquer a share of the market. The first popular web browser Mosaic was followed by Netscape, Opera, Internet Explorer, Safari, Firefox and Chrome, which have been the more widely used browsers since then [27, p1]. Being one of the first parties involved in building a web browser, Netscape and Microsoft both tried to build the most popular browser by implementing new features rather than fixing bugs that existed in their current feature set. During these days, vendors were not concerned with staying mutually compatible.

Developers at that time had difficulty creating websites for these browsers because of confusion around many features’ support [27, p2]. In 1994 the World Wide Web Consortium (W3C) was founded with the idea to standardize the protocols and technologies used in building the web. Its vision was to allow the web to further grow by mitigating inconsistencies and defining standards.

Consequently, several standards were constituted, including HTML 4.01, the format for PNG images and Cascading Style Sheets (CSS) 1 and 2. However, browser vendors did not recognize these standards and neither adhered to them. Because the W3C did not enforce its recommendations and browser vendors had no direct benefit from adopting them, incompatibilities continued to exist and be added [27, p3]. Only after the new millennium, certain developments elevated the W3C recommendations to standards that were also respected by browser vendors. From then on, web development became more focused on creating compliant websites that render well on all browsers, platforms and devices.

The creation of the HTML5 specification was one of these developments, which ensured interoperability and backwards compatibility [27, p4].

1.2 Problem statement
Despite the progress that had been made in the area of compatibility and web standards, developers still had to take into account browser incompatibilities. Although conflicting and mutual exclusive functionality had decreased, vendors still had divergent implementation debts of adopted specifications and features.

An historically exemplary and contentious browser is Internet Explorer 6, which did in some aspects not adhere to web standards. This browser once had a large user base and partly because it had been shipped as Windows’ default browser. Later on, when new and superior versions of browsers were released, the user base only slowly shrunk. Most developers were therefore forced to make their websites backwards compatible, which could be a time-consuming task [10]. The usage of Internet Explorer 6 has nowadays decreased to a minimum [18]. Also, browsers are updated more frequently and at smaller intervals than they were before [24]. This could imply that the need to develop for desktop legacy browsers may have been diminished.
In the meantime however, other challenges have arisen. The increasing popularity of mobile platforms has resulted in more browsers and devices that websites can be visited on, which makes for possibly additional complications through fragmentation [21, 22].

Simultaneously, browsers are becoming increasingly interactive platforms as more native software is brought to the web. Originally native-reserved technical features can nowadays be found in web browsers too [26].

1.3 Research goal
Many internet resources provide and have provided a glance on the current state of browser compatibility from a specific perspective. These include blogs from authors who are involved in web development, whom argue to which extend their fellow developers should take browser compatibility seriously. The argumentation is often based on a current data-snapshot presented by different online data sources, such as browser usage statistics websites.

However, a more historical overview of how this landscape has changed and how this affected web development over recent years is a missing piece of information. Also, there is no elaborate overview of the state of the art which combines all relevant aspects and sources of information.

Using the available data to extract information about the state of the art and its historical context is the primary goal of this paper. This will benefit the web development community with a more in-depth, informative and contextual resource and it can serve as a reference for further investigation.

2. RESEARCH QUESTIONS
In order to achieve the research goal, the following research questions will need to be answered:

1. How has browser feature support changed over time?
A browser feature is a specified piece of functionality, introduced by either a browser vendor or a independent body, with the original intent to be ultimately supported by major browsers. This includes most HTML specifications, JavaScript APIs and CSS properties. Browser feature support is a measure for the amount of features a browser fully supports.

2. How has browser fragmentation changed over time?
Browser fragmentation is a measure for the different browsers and browser versions that are around and actively used.

3. How important are compatibility considerations for web development relative to the created (historical) context?

3. APPROACH
The answers on the research questions lie in data that is available on browser compatibility. Therefore, firstly, several indicators are identified that together provide views on the data that lead to answers on the research questions. Subsequently, the data sources are analyzed and prepared. Finally, executing queries and formatting the outputs leads to insights into the indicators.

Multiple internet resources exist which maintain a collection of this data, but they largely differ in focus, coverage and accuracy. Two databases will be used as data source.

3.1 Data sources
CanIUse.com [8] is an often-cited internet resource which offers support tables for features in browsers since early 2012 (licensed under Creative Commons BY 4.0 [7]). These support tables show browser support for HTML, CSS and JavaScript properties and more. It provides a visual interface for the dataset that is maintained on GitHub [13]. This data is regularly updated through a manual process wherein standard procedures are being followed [6]. Priorities are managed by voting through GitHub issues [5].

Features that make it to the database are assessed on the following criteria [6]:

1. Useful to web developers.
   This is a desirable characteristic, since this paper is aimed at this group of people.

2. Likely to be eventually implemented by the majority of browsers.
   Many discussions are conducted around the introduction of browser features and sometimes vendors introduce features that have little to no backing from other vendors. Because of these features’ lack of support it is unlikely they will be eventually implemented into browsers. Data for these features would therefore add noise to the dataset.

3. Not already fully supported in even old browsers.
   This paper asks questions about the current state of browser compatibility and fragmentation. To answer these questions, it is unnecessary to include features which are already fully supported since this data is solely contextual. It may be of value to see the data of well-supported features in relation to the results presented in this paper, but that is beyond scope for this paper.

Since the data from CanIUse is not a machine-generated dataset, it is important to note that this paper tries to discover trends. Also, analysis is only performed after aggregation of data. Since the granularity of this data is low, small invariances that may be introduced by the manual process have minimal impact.

The second data collection which is used is StatCounter [24]. StatCounter has a tracking code installed on 3 million member websites, which accumulate to more than 15 billion page views per month. With this tracking code, browser data about these website’s visitors is sent to StatCounter [23].

It can be argued that StatCounter has the most accurate and representative browser statistics available. Other websites that provide similar data sets have a smaller sample size and lesser pageviews; have less to no diversified members; apply artificial weighting; track unique visitors instead of page views or do other data transformations that skew the dataset [23].

StatCounter data is already incorporated in the CanIUse database (for its own reference). It is therefore unnecessary to scrape this data separately in order to include it in the dataset.

3.2 Process
The CanIUse database is contained in a single JSON file that is versioned through Git, available via its GitHub repository [9]. Since the data is raw and not queryable, it needs to be parsed. For this, a scraper script was built that iterates over Git commits while inserting rows into another database. Since this resulted in 20 million entries,
The ubiquity of mobile devices also contributes to the ad-
application and have more functionality [26]. Few years ago, the web was only used grown over the years.
The curve is in agreement with the browser’s already men-
tures accelerates. Browser feature specifications thus have database over time. The path shows a clear rise in num-
5. RESULTS
The identified indicators which were assessed to be accu-
rate and feasible to be extracted from the available data set are feature count, feature support debt and fragmen-
tation. Together, they delineate the state of the art of browser compatibility and its historical context.

5.1 Feature count
The amount of features is a relevant metric that provides insight in the complexity of browser API specifications. As these specifications grow, more incompatibilities may be introduced when vendors implement them. This indicator is also interesting in relation to the feature support debt indicator (section 5.2).

Figure 1 plots the amount of features registered in the database over time. The path shows a clear rise in numbers for more recent dates. Also, the rate of added features accelerates. Browser feature specifications thus have grown over the years.

The curve is in agreement with the browser’s already men-
tioned role shift. Few years ago, the web was only used for serving static content. Nowadays, browsers run many applications and have more functionality [26].

The ubiquity of mobile devices also contributes to the ad-
dition of features, because they give need to the introduc-
tion of new hardware APIs such as Notifications, Device Orientation and Battery Status [20].

5.2 Feature support debt
5.2.1 Per browser
Note: Firefox for Android, Blackberry and and Opera Mobile have been omitted from this subsection for the sake of clarity and relev-
ancy, as these browsers have a negligible usage share.

Figure 2 clearly distinguishes the progressive from the lag-
gging browsers. Chrome and Opera share the top position. Their feature support is similar because they have the same rendering engine built in: Blink. The Firefox desktop browser is just behind and also scores high sup-
port rates. Safari, Edge and Internet Explorer are lagging behind.

The fact that Internet Explorer has the lowest support fig-
ures will not surprise many web developers, as the browser has always been criticized for this [10].

Safari and Edge (Internet Explorer’s successor) however, do not share this criticism as much as Internet Explorer does. This figures may suggest that Apple and Microsoft, being the companies behind these browsers, are less pro-
gressive when it comes to supporting features.

Furthermore, mobile browsers are mostly on par with their desktop browser variants. This is due to the fact that many mobile browsers’ codebases are a fork of their desk-
top counterparts. In this way, features can be largely syn-
chronized between the variants [11, 19].

Although mobile browsers always seem to lag a bit behind in support compared to their desktop variant, this feature support discrepancy is constant. In most cases, this is caused by the limitation of mobile platforms, which make certain features challenging or impossible to implement. Examples are drag and drop and webGL. It will either take a long time for vendors to implement the features or it will never happen, because it is impossible. A few years ago, Android and IE Mobile were exceptions on this rule, but in the meantime also those browsers have been aligned to the desktop versions [19].

When observing the graph lines’ walk, several major events can be recognized. One is Opera’s switch from layout en-
gine Presto to Blink. Blink was far more compatible than its in-house developed engine, which explains the major spike in the graph [3].

Another similar event causes the dramatic rise in feature support for the Android browser. During this period, An-
droid Lollipop was introduced, which made it possible for Android’s stock browser’s rendering engine to be updated separately from the Android OS version. This caused a large share of Android users to be suddenly using a recent version of the Chromium engine, contrary to the highly incompatible rendering engines dominating before [12].

5.2.2 Mobile versus desktop
As can be seen in figure 3, feature support for both desktop and mobile has seen a clear rise over time. At first sight, the graph’s lines may seem in contradiction with figure 2. However, the charts are different because of the use of weighted averages.

Mobile browsers in figure 3 appear to support more fea-
tures than desktop browsers in general. But, since these feature support data points are weighted by their corre-
sponding browser version’s usage, the higher feature sup-
port for mobile reveals that the mostly used mobile browsers
are generally more compatible. On the other hand, desktop feature support is held back by the weight of less compatible browsers such as Safari, Edge and Internet Explorer.

Also, when observing the per-browser graph, feature support is actually rising for most browsers until around January 2014. After this date, a declining trend is visible. This path, though, seems not to be reflected in figure 3. Again, this can be attributed to a usage increase of more compatible browsers (Chrome, Opera, Firefox etcetera) in favor of less compatible browsers (Internet Explorer in particular).

Furthermore, figure 3a is very similar to figure 3c. Initially, this was caused by the dominance of desktop browsers. Accordingly, weighted averages for this period are heavily determined by desktop browsers. Later on, when mobile browser usage rose, desktop browser’s and mobile browser’s feature support became more synchronized as rendering engines were aligned between desktop and mobile versions (section 5.2.1). Combined, this results in overall analogous paths for both platforms.

5.3 Fragmentation
Browser fragmentation is another indicator which characterizes the browser landscape. When fragmentation of browsers and versions rises, users collectively use a larger amount of different browsers and versions. Because each of these has its own incompatibilities and quirks, this adds complexity when building compatible websites.

5.3.1 Concurrent version usage
Figure 4 shows an aggregated, chronological overview of
Mostly used versions over time

For Chrome, the mostly used browser version follows the latest available version closely. For most browsers, a similar pattern is recognizable. A few browsers deviate from this pattern.

Chrome for Android (fig. 5c), just as Firefox for Android, tends to have the latest version on a major share of devices immediately after it is made available. These mobile browsers outpace their desktop versions in rate of adoption.

Safari’s update adoption pattern (fig. 5d) can be explained by looking at the release cycle of OS X. Apple’s desktop operating system. Because new versions of Safari are shipped with new OS X versions, Safari is only updated after a user performs an operating system update [1]. Obviously, operating system upgrades are more time-consuming than browser updates, which a user is normally not involved in anyway. Therefore, users often delay these updates, which is why these installations get outdated consequently.

Lastly, Android stock browser’s mostly used versions (fig. 5e) lag behind due to Android’s operating system fragmentation [25]. A large share of the mobile operating system landscape is still dominated by legacy versions of Android. Accordingly, usage of legacy browser versions (currently Android stock browser version 4) is higher than for later versions.

6. CONCLUSIONS

The results show several trends. Firstly, browser API specifications steadily grow more complex with more features being added over time. Moreover, feature support for both for desktop and mobile seems to decrease. Yet some browsers such as Chrome and Firefox are clearly more engaged in supporting features, whereas Safari and Edge are lagging in this respect.

Contradictory, when including browser usage into the data, the browser landscape has become more compatible over time. This is caused by an enlarging number of users that is using more compatible browsers. The diminishing usage of Internet Explorer, a traditionally incompatible browser, also contributes to this trend.

Furthermore, the browser landscape has become more uniform with less different browsers and versions being actively and simultaneously used. This might be attributed to better auto-updating mechanisms which minimize users with outdated browsers, as most browsers now show high adoption rates for the latest releases.

Hence, web developers should be less concerned with supporting legacy browsers and/or browser versions. Also, more recent features can be considered to be used in projects, as more internet users are now using more compatible browsers. On the other hand, developers should still carefully review which features are and which are not yet ready for use in their projects, because compatibility among browsers decreases.

7. REFERENCES


How people build software


Creative Commons. CC Attribution 4.0 International License. 2013. URL: https://creativecommons.org/licenses/by/4.0/.

A. Daveria. Can I use... Support tables for HTML5, CSS3, etc. 2016. URL: http://caniuse.com/.


Microsoft. Developer Resources : Microsoft Edge Dev. URL: https://dev.windows.com/en-us/microsoft-edge/ie6countdown/.


