Masterarbeit

Testen von mobilen HTML5 Web-Applikationen

Stefan Mayer

stefan.mayer@student.tugraz.at

Institut für Softwaretechnologie (IST)
Technische Universität Graz
Inffeldgasse 16B/II,
8010 Graz, Österreich

Begutachter und Betreuer: Univ.-Prof. Dipl.-Ing. Dr. techn. Wolfgang Slany

Graz, Februar 2012

Master's Thesis

Testing of Mobile HTML5 Web-Applications

Stefan Mayer

stefan.mayer@student.tugraz.at

Institute for Software Technology (IST)
Graz University of Technology
Inffeldgasse 16B/II,
8010 Graz, Austria

Graz University of Technology

Assessor and supervisor: Univ.-Prof. Dipl.-Ing. Dr. techn. Wolfgang Slany

Graz, February 2012

Abstract

Over the past years, smartphone sales grew exponentially. These phones allow to install custom applications in an easy way and additionally bring a good web experience to mobile devices. They allow working with the web on the go and using even desktop web pages on a small screen. Especially the WebKit based mobile browsers support many of the new HTML5 and CSS3 features.

Within the rise of the smartphones, the mobile commerce has grown. Nearly every operating system on mobile devices uses a different programming language for their applications. Companies had to choose a limited amount of supported systems, as the development of such apps is expensive. An application developed for one system is incompatible with the others. The solution to this problem is HTML5 and CSS3. They allow to create native alike web applications which run on nearly all smartphone browsers. This allows to write applications for many different mobile devices without the need to reimplement it in different programming languages.

This thesis describes the new features HTML5 and CSS3 offer. They allow to create native alike mobile web applications and access smartphone specific functions like the geolocation or the compass within the browser. Additionally some promising mobile frameworks are presented, which allow to use the new features in an easier way. They also smooth out some of the differences between the browsers on different devices. Tools and methods are described how to test such mobile web applications. The document covers test driven development, behaviour driven development, continuous integrations, nightly builds and GUI testing. For each of those different approaches, the appropriate tools and strategies are presented.

Keywords: HTML5, CSS3, JavaScript, mobile, web, framework, testing, iPhone, Android, BlackBerry, Windows Phone 7, test driven development (TDD), behaviour driven development (BDD), continuous integration (CI), GUI testing

Kurzfassung

In den letzten Jahren stiegen die Smartphone Absatzzahlen exponentiell. Sie erlaubten es endlich das Internet auch auf mobilen Geräten mit einem kleinen Bildschirm einfach und bedienerfreundlich zu nutzen. Vor allem die auf WebKit basierenden mobilen Browser ermöglichen neuste Technologien wie HTML5 und CSS3 auf mobilen Endgeräten.

Mit dem Boom der Smartphones ist auch das Interesse der Unternehmen am mobilen e-Commerce gestiegen. Jedoch verwenden die unterschiedlichen Betriebssysteme verschiedene Programmiersprachen für die Applikationsentwicklung. Firmen mussten daher aus Kostengründen entscheiden für welche Plattformen sie ihre Applikationen entwickeln lassen wollten. Das Entwickeln einer mobilen Anwendung ist sehr kostspielig und die Programme für ein bestimmtes mobiles Betriebssystem sind inkompatibel zu den anderen. Die Lösung dieses Problems sind HTML5 und CSS3. Diese beiden Technologien erlauben das Entwickeln von mobilen Web-Anwendungen, die sich wie eine native Applikation anfühlen. Die Anwendungen sind danach auf den Browsern der verschiedenen Smartphones lauffähig und müssen nicht neu implementiert werden.

Die vorliegende Arbeit konzentriert sich auf das Erstellen dieser so genannten mobilen Web-Apps und wie sie getestet werden können. Zuerst werden die neuen Features von HTML5 und CSS3 erklärt und gezeigt wie sie eingesetzt werden können um Smartphone-spezifische Funktionen wie die Bestimmung des Aufenthaltsortes oder den Kompass im Browser zu verwenden. Zudem werden Frameworks vorgestellt, die das einfache Nutzen dieser Features erlauben und Unterschiede zwischen den Browsern korrigieren. Danach folgen Programme, Frameworks und Strategien welche das Testen dieser mobilen Web-Apps ermöglichen. In der Arbeit werden Strategien und Programme für Test-Driven Development, Behaviour Driven Development, Continuous Integration, Nightly Builds und das Testen der GUI vorgestellt.

Schlüsselwörter: HTML5, CSS3, JavaScript, Web, mobil, Frameworks, Testen, iPhone, Android, Blackberry, Windows Phone 7, Test Driven Development (TDD), Behaviour Driven Development (BDD), Continuous Integration (CI), GUI Tests

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Ort	Datum	Unterschrift

Acknowledgements

First of all I would like to thank Martin Bachler, my adviser at the NETCONOMY Software & Consulting GmbH, and Johann Blauensteiner for their input and help during the work on this master thesis. I would also like to thank my working colleagues at the company for creating a friendly and special working atmosphere. Most importantly I would like to thank my friends and family for their support.

Graz, February 2012 Stefan Mayer

Contents

List of Tables

List of Figures

1 Introduction

In the last years the growth of smartphone sales was exponential. In November 2010 a former Morgan Stanley analyst predicted that more smartphones than PCs would be sold in 2012. It already happened in the last quarter of 2010. The traffic to mobile websites grew 600% in that year. This happened because the smartphones' web browsers offered a better mobile web experience than classic mobile phones. Figure 1.1 shows the mobile data traffic over time of the AT&T network as the iPhone was released. At this time AT&T was the exclusive carrier of the iPhone in the US. According to Cisco the data usage of an average iPhone user was 4 times higher than any other smartphone platform at the beginning of 2010. By the end of the same year an iPhone user had only 1.75 times more data usage than an average Android user. The other platforms were catching up fast as their browsers were getting better. The average smartphone data usage doubled in 2010 and Cisco predicts that till 2015 the smartphone data traffic will increase 16-fold to 1.3GB per month for the average user. [Wro11], [Cis11]

Within the rise of smartphones in the last years, the mobile commerce has grown. The downside of this boom is that there are a lot of different operating systems for smartphones right now. The applications programmed for one operating system do not work on another and even the programming language is different on nearly every system. See Table 1.1 for comparison. Therefore large companies had to choose two or a maximum of three supported operating systems. Smaller companies could afford programming an application for only one system. The solution to this problem is HTML5. Modern browser both on desktop and mobile devices support HTML5 already well. With the new features of this markup language combined with JavaScript and CSS3, a web application can be developed that works on all the important smartphone operating systems and still has the look and feel of a native application. Adobe realised the power of HTML5 and announced on the 9th of November 2011 that they are going to stop supporting flash for mobile devices in favour of HTML5. [Ado11], [Wro11]

The HTML5 buzzword is commonly understood not only as the markup language, but as a combination of HTML5, CSS3 and JavaScript which enable the development of native alike

Mobile OS	Native Programming Language
Android	Java
Bada	C++
Blackberry	Java, WebWorks, Adobe Air
iOS	Objective C
Windows Phone 7	Silverlight

Table 1.1: Mobile OS Programming Languages [Wro11]

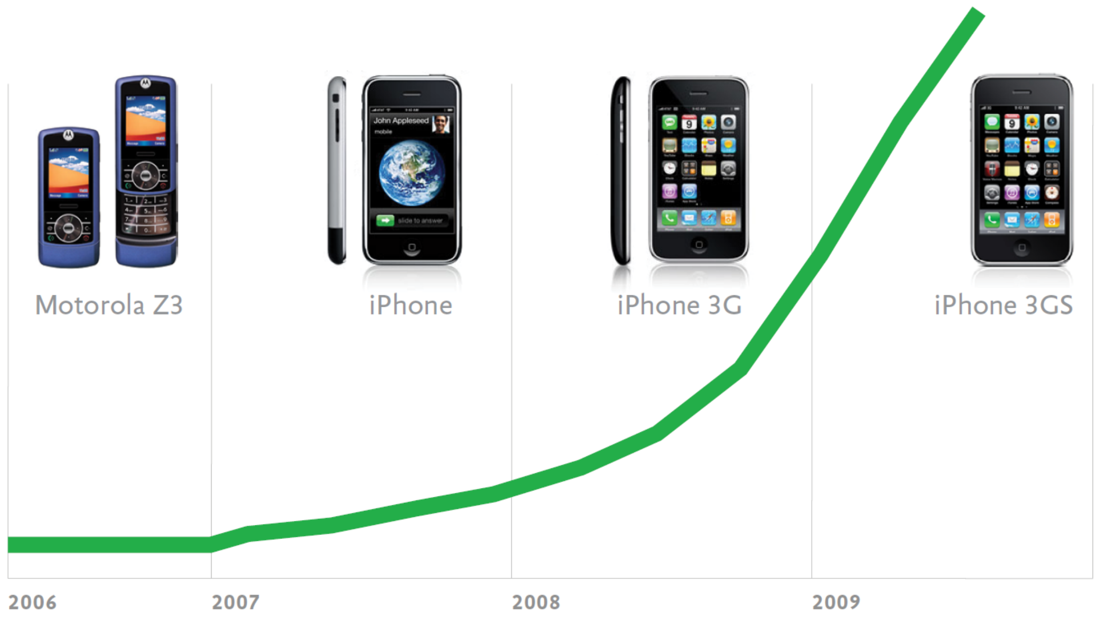


Figure 1.1: AT&T's Rise in Mobile Data Traffic with the Introduction of the iPhone [Wro11]

web applications. HTML5 itself allows, among other things, new input tags that can make use of virtual touch-keyboards as they can change their layout depending on the input type. Data can be stored in a manageable way on the client device and location based services are possible thanks to the geolocation API (not part of the HTML5 standard, but also a W3C standard[1]). The new video and audio tags allow the browser to handle multimedia files without additional plugins and the Application Cache in combination with the local storage offers the possibility to use the web application on the browser even without an active internet connection. With the use of CSS3 animations, smooth transition between pages are possible analogue to native mobile applications. 2D and 3D transformations of elements allow a completely different look and feel than classical web sites. Not all features are fully supported by the current devices, but the support is already quite good and keeps getting better with every new version of the operating systems.

Programming a mobile web application means moving a lot of the business logic to the client side, because in order to use the new features offered by HTML5 and CSS3, one has to work with the JavaScript APIs. As the mobile Internet access is not always stable and fast, the application should need as few requests as possible. Therefore, some work on the data has to be moved to the client side and only the results are sent to the server. This allows to use the application even if the smartphone does not have an active internet connection. Testing of the client side code was always important, but with web applications it becomes essential. Modern web applications have a lot more lines of code on the client side, than classical web sites. The approach how to test these client side code is different than testing on the desktop or for simple websites. Companies already started to develop frameworks to help testing

[1] http://dev.w3.org/geo/api/spec-source.html

these web applications. [Joh11]

The main target of this master thesis is to explain the basics of HTML5 and CSS3 and how to test mobile web apps programmed using the features these technologies offer.

1.1 Outline

The new features of HTML5 and CSS 3 are explained in detail first and afterwards the following chapter focuses on mobile browsers and their capability to handle those new technologies. Chapter 4 focuses on general testing and explains the basics of Agile testing, Test-Driven development and Behaviour-Driven development. To use the full power of HTML5 the average programmer will use a framework to work with the technologies. Chapter 5 introduces some promising Frameworks which enable programming of mobile web applications. In order to deliver well designed and maintainable source code, the applications have to be tested. Testing frameworks for JavaScript and GUI-Testing on mobile devices are explained in Chapter 6. The last chapter describes how to use the introduced testing frameworks for different kinds of testing methods like TDD, GUI-testing and others.

During the creation of this thesis, a mobile web shop was developed using the latest technologies. The testing methods and frameworks presented were used to test the web shop. All examples mentioned in this document refer to the developed application, if not otherwise stated.

1.2 Scope

The focus of this master thesis lies on the new technologies that enable mobile web applications and how to test them. General testing methods are explained in Chapter 4. The tools, frameworks and techniques presented in this document are restricted to the mobile web. Techniques like exploratory testing, security testing or other general testing techniques that are the same for desktop and mobile web applications are not mentioned as there is already plenty of literature available. To get more information on testing websites and web applications that is not unique to mobile devices, see books and articles covering this topic for the desktop like Exploratory Software Testing from Whittaker [Whi09], Web Security Testing Cookbook from Hope and Walther [HW08] or Performance Testing Guidance for Web Applications from Microsoft [Mic07].

2 What is HTML5

The term HTML5 is widely used and refers to more than just the new version of the markup language most of the time. It has become the buzzword for websites that seem like native applications and run on every modern browser. HTML5, CSS3 and JavaScript together allow the development of these new so called web applications. [SH10]

2.1 HTML5

The development on the HTML5 standard started with the arise of the WHATWG in 2004. The working group was formed because the W3C refused to work on an extension to HTML and CSS for Web Applications. As the W3C continued to mainly work on the XHTML2 specification, the WHATWG meanwhile worked on a specification called "Web Applications 1.0". Because the WHATWG members were mainly web browser manufacturers and interested parties, many features of their specification were already implemented by October 2006, while XHTML2 was still an unimplemented draft. From there on the W3C and the WHATWG worked together. One of the first things they did was renaming "Web Applications 1.0" to "HTML5". In October 2009 the XHTML2 working group was shut down. [Pil10]

This chapter lists and explains the most important new features and changes on existing elements of HTML5. See Chapter 3 for information on the mobile browser support of the certain tags and APIs. HTML5 is defined in a way that all HTML4 pages still work. To allow this, the standard separates requirements for authors and user agents. For example, an author is not allowed to use the `plaintext` element on the page he/she is designing, while the browser still has to support this element for backwards compatibility. [W3C11h]

Please keep in mind that this is not a complete list of all new features and changes. Only the most important and common ones are listed in this document. At the time of writing, the standard was still a working draft and some elements may have changed. Please see `http://www.w3.org/TR/html5/` for the official document. Other W3C specifications, that are connected with the buzzword HTML5 in the public, are mentioned under this section as well.

2.1.1 Doctype

Internet Explorer 5 for Mac was one of the first browsers that supported the standards so well, that old websites, which were not programmed properly, actually did not work anymore. Microsoft introduced as solution a special doctype triggering a specific renderer in the Internet

Explorer. That way older pages were rendered differently than pages with the new doctype. As the standard was developed further, more doctypes were introduced to trigger different rendering modes. IE8 already had 4 different modes depending on the doctype of the page. In Figure 2.1 you can see a diagram of this behaviour drawn by Henri Sivonen.

To get rid of the complex doctypes and the different modes in the browsers' engines, HTML5 introduced a new simple doctype that is meant to stay the same even for future versions. The standard is designed with backwards compatibility in mind, to avoid the need of triggering different rendering engines. Listing 2.1 shows a simple page with valid HTML5 syntax and the new doctype. [Pil10]

The second syntax that can be used for HTML5 is XML. Please refer to `http://www.w3.org/TR/html5/` for further information about the XML-Syntax. A simple example can be seen in Listing 2.2. [W3C11h]

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Example document</title>
  </head>
  <body>
    <p>Example paragraph</p>
  </body>
</html>
```

Listing 2.1: Valid HTML5 Syntax [W3C11h]

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Example document</title>
  </head>
  <body>
    <p>Example paragraph</p>
  </body>
</html>
```

Listing 2.2: Valid HTML5 XML-Syntax [W3C11h]

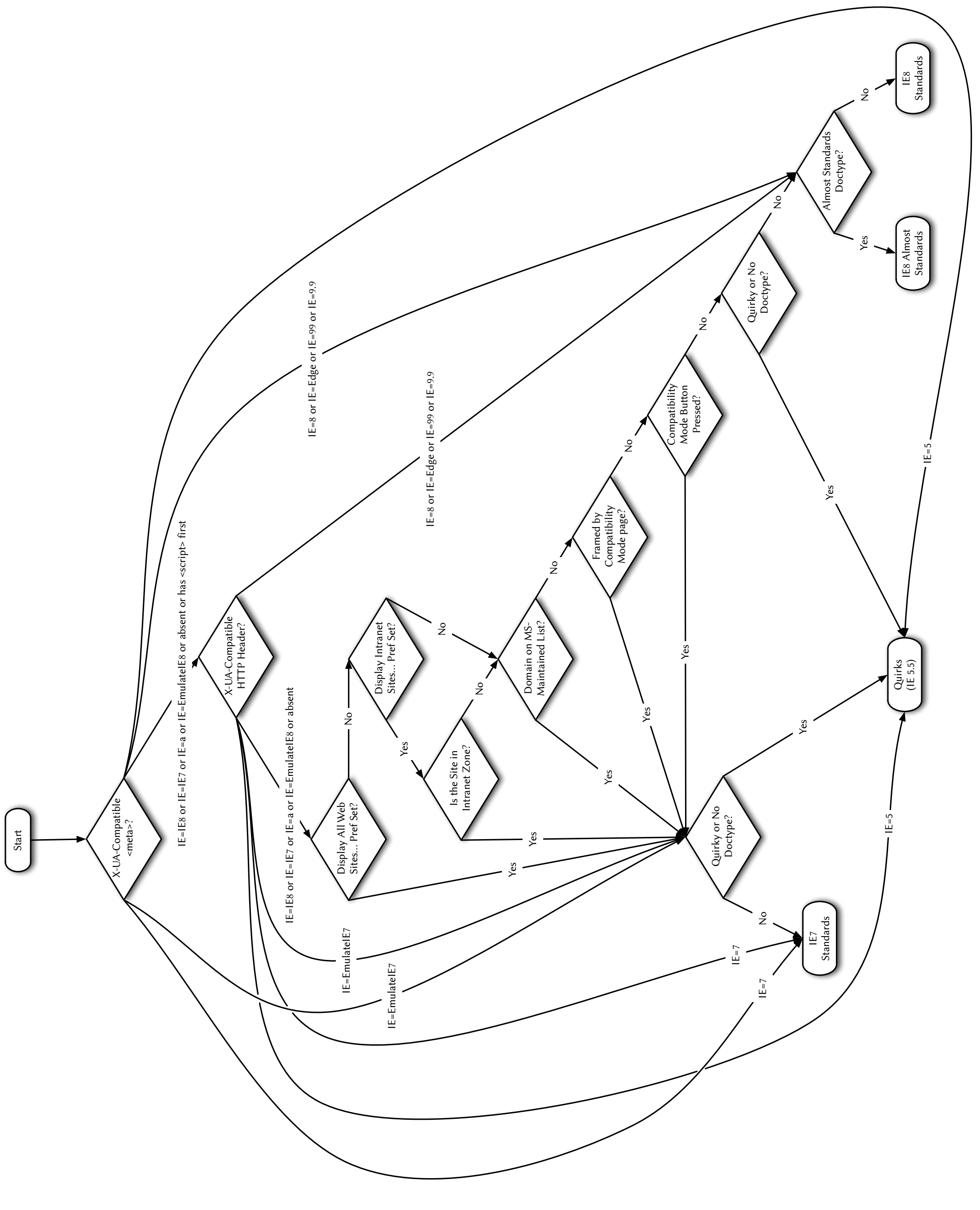

Start
X-UA-Compatible <meta>?
IE=IE8 or IE=IE7 or IE=a or IE=EmulateIE8 or absent or has <script> first
X-UA-Compatible HTTP Header?
IE=8 or IE=Edge or IE=99 or IE=9.9
IE=8 or IE=Edge or IE=99 or IE=9.9
IE=IE8 or IE=IE7 or IE=a or IE=EmulateIE8 or absent
Display All Web Sites... Pref Set?
No
Yes
Display Intranet Sites... Pref Set?
Yes
No
Is the Site in Intranet Zone?
No
Yes
Domain on MS-Maintained List?
No
Yes
Framed by Compatibility Mode page?
No
Yes
Compatibility Mode Button Pressed?
No
Yes
Quirky or No Doctype?
No
Yes
Almost Standards Doctype?
Yes
No
IE8 Almost Standards
IE8 Standards
Quirky or No Doctype?
Yes
No
Quirks (IE 5.5)
IE7 Standards
IE=EmulateIE7
IE=EmulateIE7
IE=7
IE=7
IE=5
IE=5

Figure 2.1: IE8 Rendering Modes Algorithm [Siv10]

2.1.2 New Tags

Table 2.1 lists the tags the HTML5 specification is introducing.

Table 2.1: New HTML5 Tags

Tag	**Description**	**Example**
Section	Used to structure the document. Every section can have their own h1-h6 elements.	`<section>...</section>`
Article	Represents an independent piece of information. Meant to be used for news or blog entries amongst other things.	`<article>...</article>`
Aside	Information that is only slightly related to the rest of the content.	`<aside>...</aside>`
Hgroup	Groups set of h1-h6 elements.	`<hgroup>...</hgroup>`
Header	Contains the header of a section (title, navigation etc.).	`<header>...</header>`
Footer	Footer of a section (copyright, author, etc.).	`<footer>...</footer>`
Navigation	Meant to include navigation elements.	`<nav>...</nav>`
Figure	Contains for example images or videos within a text flow.	`<figure>` `<video src="example.webm">` `</video>` `<figcaption>Example` `</figcaption>` `</figure>`

Continued on Next Page…

Table 2.1 – Continued

Tag	**Description**	**Example**

Audio	Used for audio content. The element provides an API that can be used to control it via JavaScript. See Section 2.1.5 for details.
	
	Because the standard is not defining any codecs that have to be supported as a minimum, every browser supports different ones. Fortunately the element supports multiple sources per audio tag. The browser is going to choose the first one it is supporting. The format of the audio file is determined through the MIME-type of the file. The control parameter allows to hide or show control elements for this tag.
		`<audio controls="controls">`
		`  <source src="example.oga" />`
		`  <source src="example.mp3" />`
		`  <p>`
		`    Sorry, your browser`
		`    does not support any of`
		`    the provided audio`
		`    formats!`
		`  </p>`
		`</audio>`

Continued on Next Page…

Table 2.1 – Continued

Tag	**Description**	**Example**

Video	Element to present videos. Like the audio element it provides a JavaScript API. See Section 2.1.5 for details to the API.
	The standard does not state which codecs have to be supported as a minimum for this element. As well as the audio tag, the video element can therefore have multiple sources from which the browser can choose one file it supports. The type argument is optional, but has the advantage that the browser does not have to load the video file first to determine the used codec. Every browser handles the video tag differently. For example the iPhone 4 (iOS5) opens the video in the native video player, while Android 4 plays the video embedded in the website.
		`<video controls="controls">`
		`  <source src="movie.mp4"`
		`    type="video/mp4";`
		`    codecs="mp4a.40.2" />`
		`  <source src="movie.ogv"`
		`    type="video/ogg";`
		`    codecs="theora, vorbis"/>`
		`  <p>`
		`    Sorry, your browser`
		`    does not support any of`
		`    the provided video`
		`    formats!`
		`  </p>`
		`</video>`

Canvas	Used to render bitmap graphics on the fly. The JavaScript API allows the interactive creation of graphical objects for games, graphs or other types of dynamic images.	`<canvas id="myCanvas">`
		` Your browser does not`
		` support HTML5-Canvas!`
		`</canvas>`

Time	Represents a date and/or time.	`<time datetime="2007-10-05">`
		` October 5`
		`</time>`

MathML and SVG	This new elements allows the use of SVG and MathML elements inside the document. They allow to create scalable vector graphics respectively mathematical expression.	`<svg>`
		` <circle r="50" cx="50`
		` cy="50" fill="green"/>`
		`</svg>`

Table 2.1: New HTML5 Tags Continued

As already mentioned, the `audio` and `video` tag specifications are not containing a minimum of supported codecs. Table 2.2 lists the codecs which are supported by the current platforms.

Additional elements that support the semantics of a website like `progress`, `meter`, `details`, `summary` and `output` exist. For a complete list see the official document at `http://www.w3.org/TR/html5/`.

[W3C11h], [SH10]

Platform	Audio Codecs	Video Codecs
iOS	AAC, MP3, AIF, WAVE	MPEG-4 (H.264)
Android	AAC, MP3, WAVE, MP4, OGG	WebM(VP8), MPEG-4 (H.263 & H.264)
Blackberry	AAC, MP3, WMA, OGG	MPEG-4(H.263 & H.264), WMV9
Windows Phone	MP3, MP4, WebM	MP4, WebM

Table 2.2: Supported Audio and Video Codecs [App11],[Goo12b],[Lim11],[Mic12]

2.1.3 Changes to Existing Tags

All changes to existing tags have been chosen carefully, because they are already widely supported or have a reasonable fallback. For example all modern browsers treat CSS as the default language for stylesheets and JavaScript as the default scripting language. Now for the `link` and the `script` tag the `type` attribute is optional. If it is not present the default languages CSS and JavaScript are chosen. [FR11]

Meta Tag

In order to set the character encoding of the HTML site, the `meta` tag was simplified. Instead of `<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">` (still valid) in HTML5 `<meta charset="UTF-8">` can be used. [W3C11h]

Not included in the HTML5 standard, but a working draft at the W3C, too, is the viewport meta element. First introduced by Apple for the mobile Safari on iOS, the element got soon implemented by other browser manufacturer. It allows to set the viewport properties with a CSS syntax. To display desktop sites properly, mobile browsers set the viewport to a size used by desktop browser and then allow the user to zoom into the areas of interest. The `viewport` property allows to set the width, height and zoom values of the viewport. Additionally, the user-zoom functionality can be disabled completely. [W3C11b]

Input Tag

The input element got a bunch of new types. Browsers that are not supporting the special type have a fallback to the default input type `text`. The new types are meant to make specific inputs easier for the user and additional attributes allow input checking by the browser. For

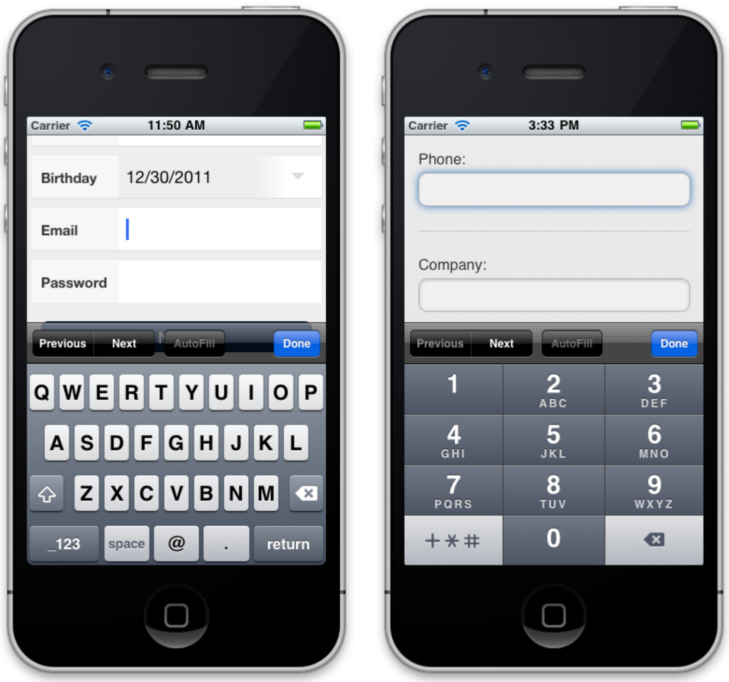


Figure 2.2: Virtual Keyboard Layout Depending on the HTML5 Input Field

example, the `number` type supports minimum and maximum values and the `tel` type offers a pattern attribute to restrict the input. `Tel` and `email` input fields can be validated by the browser because of their semantics.

Some browsers already offer special input controls for `color`, `date`, `range` and `number`. Mobile browsers with a virtual keyboard can change the keyboard layout for faster input depending on the type. For example, number fields only get a keyboard with numbers and the email field has a designated key for the @ symbol. As hardware keyboards have a fixed layout, they can still switch to the number input mode if the browser has the focus on a number field. Figure 2.2 shows an example of the different modes a virtual keyboard supports depending on the input field.

Table 2.3 shows the new input types and the allowed attributes for every type. Most attributes are self-explanatory, while others, like `list`, are hard to guess. The attributes are explained in Table 2.4.

[W3C11g], [W3C11h]

Iframe Tag

Iframes can be limited in their functionality with the attribute values `sandbox`, `seamless` and `srcdoc`. For example, sandboxed iframes are treated as being from a different origin and all scripts and form submissions are disabled for security reasons. [W3C11h]

Type	Description	Supported Arguments
tel	Phone-number	autocomplete, list, maxlength, pattern, placeholder, readonly, required, size
search	Search fields. Allows the browser to visually distinguish this element.	autocomplete, dirname, list, maxlength, pattern, placeholder, readonly, required, size
url	Url	autocomplete, list, maxlength, pattern, placeholder, readonly, required, size
email	Email	autocomplete, list, maxlength, multiple, pattern, placeholder, readonly, required, size
datetime	Date and time	autocomplete, list, max, min, readonly, required, step
date	Date only	autocomplete, list, max, min, readonly, required, step
month	Month only	autocomplete, list, max, min, readonly, required, step
week	Week only	autocomplete, list, max, min, readonly, required, step
time	Time only	autocomplete, list, max, min, readonly, required, step
datetime-local	Date and time without timezone information	autocomplete, list, max, min, readonly, required, step
number	Floating point numbers	autocomplete, list, max, min, readonly, required, step
range	Numerical range	autocomplete, list, max, min, step
color	Colour. Browser can increase usability with a special colour picker.	autocomplete, list

Table 2.3: New Input Types

Attribute	Description
autocomplete	Disables the autocomplete feature of the browser.
dirname	To define the directionality of the input. E.g. right-to-left text.
list	Used for elements which list predefined options to the user.
max	Maximum value of an element.
maxlength	Restricts the length of the input.
min	Minimum value of an element.
multiple	Allows to specify more than one value (e.g. email addresses).
pattern	Regular expression to validate input.
placeholder	A small hint that should be displayed to the user for this element.
readonly	Boolean to control if the field can be edited.
required	Marks required fields.
size	Sets the amount of characters a user is going to see while editing the field.
step	Defines the granularity of the input by limiting allowed values.

Table 2.4: Input Type Attributes

Additional Attributes

The elements `input`, `select`, `textarea` and `button` support the new `autofocus` attribute to specify which element should get the focus after the page has loaded. The `input` and `textarea` tag additionally got the new `placeholder` attribute. This attribute allows to specify a text which is displayed before the user types something into the element.

The `form` attribute is supported by the elements `input`, `output`, `select`, `textarea`, `button`, `label`, `object` and `fieldset`. It allows to associate an element with a form that can be anywhere on the page (the element does not have to be a descendant of the form element).

`Input`, `select` and `textarea` got the attribute `required`. The browser should only submit the form if all fields marked with `required` have a valid value.

One important new global attribute is `draggable` and `dropzone`, which allow in combination with the JavaScript API drag & drop. Elements marked as `draggable` can be dragged into a `dropzone`. See Section 2.1.5 for more details on the JavaScript API.

[W3C11h]

2.1.4 Application Cache

A very interesting new feature is the Application Cache. It allows to define how the files of the web application are cached. This information is contained in the cache manifest file. The kind of caching can be specified for single files or whole directories. The different cache types which can be defined for a file or directory are `CACHE`, `NETWORK` and `FALLBACK`. `CACHE` means that this file or directory should be cached and only requested via network if the cache is not up to date or not present. `FALLBACK` means that the resource is requested via network and if not possible (e.g. no internet connection), the defined fallback file is used. `NETWORK` means the resource has to be requested all the time. If not available, a 404 error will occur.

To check if the cache is up to date, the manifest file is checked byte-by-byte. This means the cache manifest has to be changed in order to trigger an update. Therefore, it is recommendable to include a version number in the file. If an update is necessary, the download of the resources happen in the background. When the download is complete, the website is not refreshed automatically. Instead the new cache will be used after the next browser refresh.

It is recommended to include * in the `NETWORK` part, otherwise every file has to be present in one of the sections. Files not mentioned will not be loaded, except the page that holds the cache manifest, which is automatically included and cached. Therefore, websites that need to be requested from the network all the time, should not have a link to the manifest. Listing 2.3 shows an example of the file.

See the next chapter for information about how to use the JavaScript API of the Application Cache.

```
CACHE MANIFEST
#Version 1.0.15

CACHE:
images/
css/
js/
index
products
shopping_cart

NETWORK:
*

FALLBACK:
online_version offline_version
```

Listing 2.3: Cache Manifest Example

2.1.5 JavaScript APIs

As JavaScript is now the official default scripting language, the specification defines APIs only for this language. Nearly every new element got its own API. Additional APIs for native objects like the history, geolocation or the cache exist as well. Most of the APIs are already supported well. For details on the support see Table 3.1 in Chapter 3.

Touch Events

Not included in the HTML5 standard, but defined in its own[1], is the Touch Events specification. Touch enabled devices finally got their own JavaScript events. The devices had to emulate mouse click events in order to work properly on older pages. Now the programmer might choose a different behaviour for a touch or a click event. Many browsers are still sending both events in order to stay compatible with older pages. This has to be taken into account when working with the new events. The official events are `touchstart`, `touchend`, `touchmove` and `touchcancel`. [W3C11m]

Multimedia

Besides the HTML attributes, like `preload`, `autoplay`, `loop`, `muted`, etc., the multimedia elements `video` and `audio` offer an API that allows to control them with JavaScript.

It offers various methods, properties and events to interact with the multimedia elements. For example, the video or audio can be played, paused, loaded or muted programmatically. Some properties offer information about the file like the width and height of a video or the current time and duration. Events like `play`, `pause`, `ended`, `waiting`, `volumechange` and others can be caught via JavaScript. The API allows, for example, to program particular controls for the element or even define a playlist. With the method `canPlayType()` it is possible to determine the codecs of video and audio files the browser might be able to play. The method returns an empty string if the user agent is confident it cannot render the video file or play the audio, `probably` if it is confident it is possible and `maybe` otherwise. [FR11], [W3C11g]

Application Cache

The main functionality of the Application Cache is described in Section 2.1.4. This section focuses on the JavaScript part. The API offers only 2 methods, `update()` and `swapCache()`. `Update()` forces a check of the cache manifest and `swapCache()` swaps the cache already before a reload of the page happens. This does not mean that the page is going to change, but that resources loaded after `swapCache()` was executed, will be received from the new cache.

Additionally, the status of the cache can be checked with the attribute `status` of the global Application Cache object. The available constants are `UNCACHED`, `IDLE`, `CHECKING`, `DOWNLOADING`, `UPDATEREADY` and `OBSOLETE`. Every state change of the cache, including errors, fires an event, which can be handled programmatically.

[SH10]

[1] http://www.w3.org/TR/touch-events/

Online / Offline events

The navigator provides a property concerning the connection state. If the state changes, the `online` or `offline` event is fired. This is no guaranty that there is a stable internet connection if the property says online. For example flaky connections cannot be detected by the device. [W3C10c]

Drag and Drop

An element marked as `draggable` will fire the `dragstart` event if dragged. The element can be dragged into a drop target which has to be marked as `dropzone` and listen to `drop` events. The MIME-type the dropzone accepts can be specified as an attribute as well. [W3C11g]

History

As pages loaded with Ajax are very common today and it is necessary in order to allow animated page transitions, the history API offers to add this pages to the browser history. The method `pushState(data, title [, url ] )` allows to add data to the state, a page title and optional a URL. In comparison `replaceState(data, title [, url ] )` overrides the actual history state. This allows to change the actual data, title and/or URL of the displayed page without reloading it. [W3C11g]

Canvas

The canvas element allows to draw resolution dependent images on the fly. Those images can be exported as a Blob or a data URL. The API to draw the image depends on the context. As of this writing, only the 2D and the WebGL context were available. For example the 2D context allows transformations of the image, drawing of rectangles, circles, lines, paths, texts, other images and so on. The WebGL context allows to create, transform and animate 3D objects on the fly. [W3C11g], [W3C11e], [WHA11]

Geolocation

The geolocation API is not included in the HTML5 standard, but it states an own standard. It is a none transparent interface for the hardware location sensors, therefore the programmer does not know how the location is determined by the device. It supports one-shot requests and repeated position updates.

For one-shot requests the API offers the method `getCurrentPosition()`. As arguments it accepts a callback for successful requests, an error callback and an options argument. Possible options are the Boolean `enableHighAccuracy`, the timeout and the maximum age of the position data. The first argument tells the device to use the most accurate sensor, the timeout defines how long the application is willing to wait for data and the last argument allows to request a cached position.

For repeated position updates the method `watchPosition()` is used. It has the same arguments than `getCurrentPosition()`. The difference is that it calls the success callback on every position change until `clearWatch()` is called.

The received position object contains the coordinate object and the timestamp when the location was determined. The coordinate object holds the latitude, longitude and accuracy of the position. It can also provide the altitude, altitude accuracy, heading and speed if the device supports it, else these values are null.

[W3C10a]

Orientation and Motion Event

Not included in the HTML5 standard, but another W3C standard, is the `device orientation event` standard. The name is confusing as it includes, besides the orientation event, a motion event as well.

The orientation event provides 3D information on how the device is oriented in relation to the earth, while the motion event provides information about the acceleration and the current rotation rate. The `compassneedscalibration` event tells the website that the compass used for the orientation event needs calibration.

[W3C11c]

Web Storage

Web storage was earlier included in the HTML5 standard, but now it is developed further as a stand-alone specification. This client side storage is meant to replace cookies. Two different types of client side storages exist: The session storage, which gets destroyed as soon as the session is destroyed and the local storage which stays permanently. The storages offer methods such as setting, getting and removing items, clearing the whole storage, getting the amount of items or finding the key of the `n`th element. The storages can only be accessed within the same script origin for security reasons, but from every window with the same session. The specification recommends a limit of 5MB per origin which most devices apply.

[W3C11h], [W3C11o]

Web SQL Database

The W3C stopped working on this specification because of a lack of independent implementations (all browser vendors used SQLite as the backend).

WebSQL is meant to bring SQL to the client side of a webpage. It allows to use SQL databases within JavaScript to work with structured data.

[W3C10d]

IndexedDB

Like the Web SQL database it is meant for large amounts of structured data. As an advantage over the SQL database it supports high performance searches, but on the downside it only supports key value pairs. Instead of tables, IndexedDB uses object storages with indexes. In comparison to the simple web storage, it supports relations, range searches for keys and allows iterating over a result set. See Listing 2.4 for an example. [W3C11i], [RW10]

```
// Create object store
db.createIndex("FriendNames", "name", false);
var index = db.openIndex('FriendNames');
var id = index.get('Eric');

// Restrict to names beginning A-E
var range = new KeyRange.bound('A', 'E');
var cursor = index.openObjectCursor(range);

while (cursor.continue()) {
  console.log(cursor.value.name);
}
```

Listing 2.4: Valid HTML5 Syntax [Gre11]

Server-Sent Events

This specification allows the server to push data to the client after the client had first initialized a connection. It is not yet well supported on mobile devices, but on desktop solutions. It has been taken out of the HTML5 specification and is now developed as a stand-alone specification. [W3C11j]

Web Sockets

Allows the client to initialize a bidirectional communication with the server. Once the socket is opened the client can call the `send()` method in order to send data. If the server has sent data to the client, the `onmessage` event is triggered. It is a stand-alone specification since 2009, too. [W3C11l]

Web Workers

As more and more logic is moved to the client side, performance gets an issue. Most modern computers and even smartphones have multicore processors, but JavaScript is single threaded. Web workers run in their own thread independently of the main JavaScript thread. This allows to execute long running processes and still respond to user input events.

In order to keep the DOM operations efficient only the main thread is allowed to change it. Web workers and the main thread communicate via messages. Primitive types, arrays and even JSON objects can be exchanged, but functions cannot. This restriction is in place, because no shared references are allowed between a worker and the main thread. The only shared object is the web storage, which is thread safe.

[W3C11p]

Network Information

This specification provides an API to enable determining the underlying network type of the device. The working draft document defines 7 types at the time of writing. `unknown`, `ethernet`, `wifi`, `2g`, `3g`, `4g` and `none`. Android uses integer constants instead of the defined strings for the connection type at the moment. [W3C11k]

File API

This API allows to access local files (on the device) with JavaScript. This is useful, for example, to manipulate local files or sent multiple files to the server. For security reasons the script can only access files the user has provided access to. [HL11], [W3C11d]

Media Capture

The media capture feature extends the file upload field. It is meant to allow easier access to capture devices like cameras or microphones. The specification defines the arguments `accept` to define the type of media and `capture` for telling the browser from where the media should come. Valid values for `accept` are `image/*`, `audio/*`, or `video/*`. `Capture` is an enumerated attribute which can take one of the following values: `camera`, `camcorder`, `microphone` or `filesystem`.

Right now only Android 3+ devices support this API. As fallback all other browsers open the default file upload dialog.

```
<input type="file" accept="image/*" capture="camera" id="capture">
```

Listing 2.5: Media capture example [W3C11f]

[HL11], [W3C11f]

Future APIs

Other APIs to access for example the battery status, the calendar or the contacts on the device are work in progress. They are already a working draft at the W3C, but not yet implemented. [HL11]

2.2 CSS3

It has nothing to do with HTML intrinsically, but still CSS3 always appears in combination with the HTML5 buzzword. This is because only with the new version of CSS the features of HTML5 can be presented in an attractive visual way. Additionally, CSS3 offers with the media queries a good way to design a page for different screen resolutions.

CSS3 brings a lot of new features and this document does not mention all of them. Only the features which bring special value to mobile web applications are mentioned here.

2.2.1 Media Queries

One of the most important new feature is the possibility to use CSS media queries. They allow to trigger different styles for specific resolutions, size and other screen related values.

Media queries define a rule which has to be true in order to apply the style. Media queries can be used within the `link` tag to restrict the inclusion of a specific CSS-file or within the CSS for style expressions.

All media queries mentioned in this document (not a complete list), except the orientation query, accept the `min` and `max` prefixes. They can be combined and allow to define styles for specific ranges. All media queries can be combined using the `and` operator.

[Gas11]

Size

The `width` and the `height` of a screen can be used as a media query. [W3C10b]

```
@media screen and (min-width: 400px)  { rules }
@media media and (min-device-width:800px) { rules }
```

Listing 2.6: Media Query Examples for the Width Keyword [Gas11]

These keywords represent the width and the height of the browser window. To work with the resolution of the device itself CSS3 provides the `device-width` and `device-height` keywords. [Gas11]

2.2.2 Orientation and Aspect Ratio

This media query is quite helpful on mobile phones which can display the page in landscape or portrait mode depending on how the user is holding the device. Allowed values are `portrait` and `landscape`. Besides `orientation`, `aspect-ratio` and `device-aspect-ratio` are available as well. They allow to define styles for specific screen aspect ratios. [Gas11]

```
@media all and (orientation:portrait) { ... }
@media all and (orientation:landscape) { ... }
```

Listing 2.7: Media Query Examples for the Orientation Keyword [W3C10b]

2.2.3 Resolution and Pixel Ratio

Modern smartphone devices already have a quite high resolution regarding their screen size. For example, the iPhone 4 with Retina display has a resolution of 960 x 640 pixels on a 3.5 inch screen [2] and the Samsung Galaxy Nexus has a resolution of 1280 x 720 pixels on 4.65 inch [3]. In order to display websites in a size the user still can work with, the CSS resolution is less than the physical one. The ratio of the actual devices pixel to the CSS pixels is called device pixel ratio. The two previously mentioned devices both have a device pixel ratio of 2. This means one pixel in CSS corresponds to 4 physical pixels on the screen.

For example, to display high resolution images on these screens, two media queries exist for this purpose. The `resolution` media feature describes the density of the pixels of the device. The `-webkit-device-pixel-ratio` query is only defined for WebKit browsers, but should be added in the future to the specification. It describes the mentioned ratio between CSS and physical pixels.

[Gas11], [W3C10b]

```
@media print and (min-resolution: 300dpi) { ... }
@media all and (-webkit-min-device-pixel-ratio: 1.5) { ... }
```

Listing 2.8: Media Query Examples for the Resolution and Device-Pixel-Ratio Keyword [W3C10b]

2.2.4 Decorative Elements

With CSS3 a lot of decoration can be added to boxes, borders and texts. Images previously used to achieve the effect are not necessary any more. This saves bandwidth and allows mobile devices with a slow internet connection to load the page faster.

[2] http://support.apple.com/kb/SP587

[3] http://www.samsung.com/at/consumer/mobile-phone/mobile-phone/smartphones/BGT-I9250-spec?subsubtype=galaxy

Shadows

Shadows can be added to texts and boxes. The syntax allows to set the position, the blur radius and the colour of the shadow. Even multiple shadows per element can be defined.

`div { text-shadow: x y blur-radius color; }`

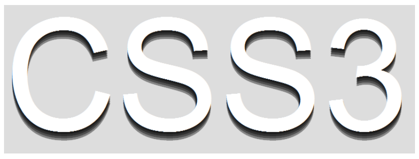


Figure 2.3: CSS3 Text Shadow

Figure 2.4: CSS3 Box Shadow

Rounded Corners

In order to get visually attractive buttons, background-images had to be used. With the ability to use rounded corners on every element and in combination with the other features, this is not necessary anymore.

Different values for the rounded corner effect can be defined for every single corner of the element. A code example can be seen in Listing 2.9 and the result is shown in Figure 2.5.

```
div {
    border-top-left-radius: 20px;
    border-top-right-radius: 20px;
    border-bottom-right-radius: 20px;
    border-bottom-left-radius: 20px;
}
```

Listing 2.9: CSS3 Rounded Corners Example [Gas11]

Gradients

With this feature some background images might become obsolete. It allows to define a gradient mathematically. Two different types are possible: the linear gradient and the radial gradient. Both allow to define the angle and the colours. A minimum of two colours is

Figure 2.5: Rounded Corners

required, but any amount is possible. Additionally, the radial gradient has the arguments `shape` and `size`.

At the time of this writing, the gradients needed still the browser specific prefixes like `-webkit-` or `-moz-`. Listing 2.10 shows some example code and the results are displayed in Figure 2.6, 2.7 and 2.8.

```
background: -webkit-linear-gradient(135deg, black, white);
background: -webkit-radial-gradient(80% 50%, circle closest-side, white,
    black);
background: -webkit-radial-gradient(left, circle farthest-side, white, black
    25%, white 75%, black);
```

Listing 2.10: CSS3 Rounded Corners Example [Gas11]

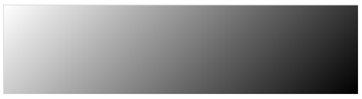


Figure 2.6: Linear Gradient [Gas11]

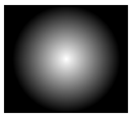


Figure 2.7: Radial Gradient [Gas11]

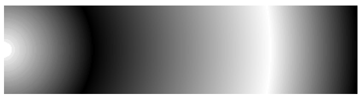


Figure 2.8: Multiple Gradients [Gas11]

Opacity

The opacity of elements can be defined in two ways: there is a stand-alone attribute `opacity` and the attribute `rgba` which allows to define the alpha channel in combination with the other colour channels.

```
.opacity { opacity: 0.5; }
.rgba { background-color: rgba(255,255,255,0.5); }
```

Listing 2.11: Opacity Syntax [Gas11]

Text Overflow and Line Wrapping

As the screens on mobile devices are quite small, longer texts have to be clipped sometimes. The new `text-overflow` feature allows to define the behaviour of this clipping.

```
p {
    overflow: hidden;
    text-overflow: ellipsis;
    white-space: nowrap;
}
```

Listing 2.12: Text Overflow Example [Gas11]

Listing 2.12 shows a possible way to achieve clipping. The style `overflow: hidden` prevents the text from being displayed outside the element's border, `white-space: nowrap` prevents the text to span over multiple lines and the new element `text-overflow: ellipsis` adds an ellipsis character, which is basically 3 dots, where the text got clipped. Instead of `ellipsis` other values are possible. The keyword value `clip` cuts the text without adding any additional characters. At the time of writing, the other values for this property are marked as at risk to be dropped out of the specification.

With the new `word-wrap: <keyword>` feature, the browser can be authorized to break words, to fit into the surrounding element.

[Gas11], [W3C11a]

2.2.5 Transformations

CSS3 introduces 2D and 3D transformations for elements. They can be rotated, translated or scaled. It is even possible to apply a complex matrix to an element. This allows in combination with transitions, fancy animations of elements or even whole pages rendered by the browser itself. Listing 2.13 shows some usage examples.

```
div { transform: rotate(value); }
div {
    transform: translateX(20px);
    transform: translateY(15%);
}
div { transform: skew(skewX,skewY); }
div { transform: scale(scaleX,scaleY); }
div { transform: matrix(cos(angle),sin(angle),-sin(angle),cos(angle),X,Y); }

div { transform: translate3d(translateX,translateY,translateZ); }
div { transform: matrix3d(
    m01,m02,m03,m04,
    m05,m06,m07,m08,
    m09,m10,m11,m12,
    m13,m14,m15,m16
); }
```

Listing 2.13: Transformation Examples [Gas11]

2.2.6 Transitions

With the transition feature the behaviour between two states of an element can be defined. The default is that the change is applied immediately. For example if a class defines a bigger font size and is applied to an element, the font changes instantly. Transitions allow a smooth and defined change between the two states.

The `transition-property` defines the CSS property the transition is applied to. `Transition-delay` and `transition-duration` control when the transition begins and how long it lasts. Additional control is offered by the `transition-timing-function` attribute. Possible values are `ease`, `linear`, `ease-in`, `ease-out`, `ease-in-out` and `cubic-bezier`. The last one allows to customize the transition timing function with the use of a cubic Bézier curve.

```
h1 {
    font-size: 150%;
    transition-property: font-size;
    transition-duration: 2s;
    transition-timing-function: ease-out;
    transition-delay: 250ms;
}
```

Listing 2.14: Transition Example [Gas11]

2.2.7 Animations

The animations are using the same syntax as the transitions. Additional arguments allow more functionality. The biggest difference to transitions is, that an animation can have an unlimited amount of keyframes, while a transition only has two states (start and end). This means an animation consists of multiple transitions. Additionally, the repetition can be configured. A simple example can be seen in Listing 2.15. The example animation consists of three keyframes. The first two define the state of the border and the third also changes the size of the div. The duration of the animation is 6s, with infinite repeat and alternating animation sequence.

```
@keyframes 'expand' {
  0% { border-width: 4px; }
  50% { border-width: 12px; }
  100% {
    border-width: 4px;
    height: 130px;
    width: 150px;
  }
}
div {
    border: 4px solid black;
    height: 100px;
    width: 100px;
    box-sizing: border-box;
    animation: 'expand' 6s ease 0 infinite alternate;
}
```

Listing 2.15: Animation Example [Gas11]

2.3 URI schemes

Although it is no W3C standard and does not have anything to do with HTML5, URI schemes are still an important feature for mobile devices. There are certain RFCs regarding special

URI schemes that are well supported on mobile devices. For not supported schemes the browser displays a meaningful error message to the user.

tel

This scheme is used for telephone numbers. A mobile phone which supports this scheme allows the user with a simple tap on the link to call the number. Devices which do not support calling, for example a tablet, open the address book to allow adding of the number.

```
<a href="tel: +43 316 12 45 678">Call +43 316 12 45 678</a>
```

sms

The SMS-scheme is supposed to define a message and one or multiple phone numbers. Right now, Apple supports only a single number which opens the SMS-app without any text. Android 4+ supports the full RFC.

```
<a href="sms:123456789,+123456789?body=Hello SMS">Please send me a text</a>
```

mailto

Already supported by every desktop browser since years, mobile browsers support this scheme as well. It is possible to define the recipients, cc, bcc, the subject and the body. This is supported by all devices.

```
<a href="mailto:q@a.com?subject=Testing&cc=no.one@b.com&body=Hello>Mail</a>
```

Google maps

It is not a RFC nor a special scheme, but it still opens an application on some devices. A simple link to a Google maps page is sufficient to open the installed Google maps application. Browser which do not support this special behaviour open the Google maps page in the browser. Some devices support to open the navigation application as well.

```
<a href="http://maps.google.com/maps?q=TU+Graz,+Austria">TU Graz</a>
```

[W3C10c],[SH10]

3 Current Smartphone Browsers

As many websites do not offer a mobile version, smartphone browsers also have to render desktop websites correctly. Additionally, smartphone browsers support special tags to limit the zooming or interpret additional URL schemes to call phone numbers directly. Unfortunately, the vendors (except Apple) do not offer official documents that describe the capabilities of their mobile browsers. The only sources are testing platforms to test the support of certain tags and features of the browser like `html5test.com` or `caniuse.com`. Other pages gather the information from this testing platforms and list supported features for the individual browsers. This sources are not 100% reliable, as they often only test the capability of a browser to handle a certain tag, but not if it works as expected. For example, the video and audio tags are tested if the browser knows them, but not if it can handle any codecs or if the JavaScript API works correctly. If possible, I tested the tags on the smartphone browsers by myself and compared them to the results on the websites. As I did not have access to all devices, for some I had to rely on the accuracy of the websites. Table 3.1 shows an overview of the browsers and their feature support.

3.1 iOS

On the iOS operating system (used on iPhones, iPods and iPads) the default web browser is a mobile version of Apples Safari. Like the most default mobile browsers and its desktop equivalent, the mobile Safari is WebKit based.

Firtman [Fir11] and Deveria [Dev11] only list the features for iOS version 3.2 and up. The 3.2 version of the mobile Safari is already capable of the most HTML5 features, like Application Cache, web storage, web SQL, geolocation, video tag, canvas, input depended keyboard layouts and touch events. Most of the CSS3 features, like rounded corners, opacity, 2D and 3D transformations, are already implemented. With iOS 4.1, Server-Sent Events got possible. iOS 4.2 introduced Web Sockets and the Motion Sensor API which allows using the accelerometer and gyroscope on the website. [Fir11], [Dev11]

The newest version, introduced with iOS 5.0, improved the performance of JavaScript, HTML/CSS rendering and HTML5 canvas. New features of this version are Web Workers and XMLHttpRequest level 2. The new input field types like date, time, month and range got their own controls and keyboard layouts. The CSS element `position: fixed` finally got supported with iOS5, which allows static header and footer bars as in native applications. Additionally, the compass can now be used via the device orientation events. [Ban11]

Currently, only the iOS version of Safari has the possibility to define an icon for start screen bookmarks. If started from that bookmark, the web application can be run in fullscreen mode and with a splash screen image at start-up. Even the browser controls are not present for a real native app feeling. A disadvantage is that the website has to include a back button or navigation options, as the default browser controls are hidden. JavaScript has the ability to determine if the web application is started in the browser or in fullscreen mode. [App11]

Apple provides a documentation about Safari which is valid for both the desktop and mobile version. Differences between the version are mentioned in the articles of the documentation[1].

3.2 Android

With every major OS update, a new version of the mobile browser is released as well. Unfortunately, some browser vendors use their own version of the Android browser and are not updating older devices anymore. For example, the HTC version of the Android 2.x browser has a bug and ignores the viewport meta tag which allows disabling user zooming.

Like most mobile browsers it is WebKit based and uses the same JavaScript engine as Google's Chrome. Since Android 2.1, already many of the new features like the web storage, web SQL, geolocation, the video tag, 2D transformations and CSS animations are supported. Froyo (2.2) added a network information API, which allows the web application to determine the current connection type. Additionally, `position: fixed` is supported if the user zoom is disabled. Gingerbread finally added support for the audio tag.

Android 3 is only available for tablets, but it introduced a new version of the Android browser. Furthermore, this version supports SVG, motion sensors, 3D transformations and the file API. It has special keyboard layouts for the new HTML5 input types and allows to capture images and videos with the camera or audio with the microphone. Ice Cream Sandwich (Android 4) brought the tablet browser version finally to the mobile phones.

[Fir11], [Dev11], [Goo12a]

3.3 Windows Phone

The first browser shipped with Windows Phone 7 was the Internet Explorer Mobile 7 which was based on the desktop browser Internet Explorer 7. This version did not had any HTML5 support. With the update to Mango (7.5) the phones got Internet Explorer Mobile 9, which is based on IE9.

Unfortunately, this version still does not support as many of the new HTML5 and CSS3 features as for example the iOS or Android stock browser. It supports the client-side web storage, but neither web SQL nor the Application Cache. One can make use of the geolocation API and the multimedia tags video and audio on Windows Phone 7. Like other mobile

[1] http://developer.apple.com/library/safari/navigation/index.html

browsers it supports different keyboard layouts for the new input types, HTML5 canvas and SVG. The supported CSS3 features are only the basic ones like opacity, rounded corners or 2D transformations. Internet Explorer Mobile lacks in advanced CSS3 features as 3D transformations and animations and does not support the JavaScript APIs for Web Workers and Motion Sensors.

[Fir11], [Par11]

3.4 BlackBerry

BlackBerry smartphones and tablets supports HTML5 and CSS3 features already very well. Additionally, it is possible to deploy HTML5 web applications as native apps to a BlackBerry device and make use of APIs else only available to native apps. Since version 6 the browser is WebKit based as well.

Browser version 6 and below were Mango based and only supported the canvas API, SVG and different keyboard layouts for the new input types. Version 6 introduced support for the Application Cache, web storage, web SQL, geolocation, web workers and CSS3 features as opacity, rounded corners, 2D transformations, transitions and animations. A minor update (6.1) added web sockets and touch event support.

The current version 7 finally added video and audio tag support and is the only stock mobile web browser that supports remote debugging. This allows to connect a desktop developer tool like the one included in Chrome to the mobile browser and see the DOM, change CSS on the fly and debug JavaScript code. The tablet version has the same capabilities as the smartphone version of the browser, but adds additionally 3D transformations and, since version 2, WebGL support.

[Fir11]

3.5 MeeGo

At the time of writing, the only available smartphones with MeeGo are the Nokia N9 and the developer phone N950. MeeGos browser on this smartphones is WebKit based and supports nearly all features available on Android and iOS. Version 1.2 already supports the Application Cache, the web storage, web SQL, geolocation, the video and audio tags, web workers, canvas, SVG, the motion sensor API and XMLHttpRequest level 2.

Only few things are not yet supported, like Web Sockets, special controls for Date, Time, Month and Range, the remote debugger API, WebGL, the network information API, the file API and the media capture feature. Except of WebGL, all main CSS3 features like 2D and 3D transformations, animations, transitions, shadows, rounded corners and opacity are supported. See Table 3.1 for a detailed list. [Fir11]

3.6 WebOS

Most statistics do not even list this operating system explicitly, because of its low market share. For example, Gartner [Gar11], Perez [Per12] and StatCounter [Sta12] do not list webOS (its included in others).

When it comes to HTML5 support, HP's webOS browser lags in comparison with the other mobile browsers behind. It is WebKit based and supports the Application Cache, web storage, web SQL, geolocation, the video and audio tags, canvas, 2D transformations, transitions, animations and the basic CSS3 features like shadows, rounded corners and opacity.

Unfortunately more advanced features like web sockets, web workers, the motion sensor API, touch events or the network information API are not yet supported. As HP announced at the end of 2011 to stop developing webOS devices and to release it as an open-source project, it is uncertain if the devices will get any future updates.

[Fir11], [Pan11]

3.7 Cross Platform Browser

Firefox for mobile and Opera Mobile are available on multiple platforms. They can be installed in addition to the stock browser and are using their own rendering engines. Only the main cross platform browsers are listed here. Others like Dolphin for Android are based on the stock browser and differ only at the GUI level, or ones like Opera mini have only a very limited JavaScript and HTML5 support and cannot be used for mobile web applications with a native look and feel and are therefore not described here.

3.7.1 Firefox for Mobile

Like its desktop equivalent, Firefox for mobile is Gecko based. It is available on Android, MeeGo and iOS. Since version 6 it supports the Application Cache, web storage, geolocation, the video and audio tags, server sent events, web workers, canvas, SVG, motion sensors, input type dependent keyboard layouts and additional controls for inputs like date, time, etc., touch events, the notification API and as the only one of the mobile browsers IndexedDB.

CSS3 features, like opacity, shadows, rounded corners, 2D transformations, transitions, WebGL and animations, are supported. Version 7 introduced additional support for web sockets. Only few features like SQL storage, 3D transformations, remote debugger, XMLHttpRequest level 2, the media capture feature and the network information API do not work in the current version. [Fir11]

3.7.2 Opera Mobile

Opera mobile is available for Android, Symbian, MeeGo and Windows Mobile. At the time of writing, the Windows Mobile version lagged the other platforms behind. No version was available for Windows Phone or Blackberry devices.

Like the desktop version, it uses the Presto engine to render the websites and supports since version 11 the Application Cache, web storage, web SQL, geolocation, video and audio tags, server sent events, web sockets, web workers, canvas, SVG, virtual keyboard layouts and specific controls for the new input types, touch events and the remote debugger features that allows to connect with Operas DragonFly. See Section 7.4 for details on the remote debugger feature.

Unfortunately it does not support advanced CSS3 features like animations or 3D transformations, but features like opacity, rounded corners, text shadows, 2D transformations and transitions.

[Fir11], [Dev11]

Feature	Safari on iOS	Android Browser		BlackBerry Browser		Nokia Browser	Internet Explorer	Opera	Firefox	webOS Browser
Version tested	iPhone, iPad	Phones(1+2.3, 4.0)	Tablet (3.0+)	Phones	Tablet	MeeGo - Nokia N9	Windows Phone	Mobile	for mobile	
Minimum version tested	3.2	1.5	3.0	5.0	1.0	1.2	9	11	6	1.4
Application cache	✓	✓2.1+	✓	✓6.0+	✓	✓		✓	✓	✓
Web storage	✓	✓2.0+	✓	✓6.0+	✓	✓	✓	✓	✓	✓
Web SQL storage	✓	✓2.0+	✓	✓6.0+	✓	✓		✓		✓
Geolocation	✓	✓2.0+	✓	✓6.0+	✓	✓	✓	✓	✓	✓
Video	✓	✓2.3+	✓	✓7.0+	✓	✓	✓	✓	✓	✓
Audio	✓4.0+	✓2.3+	✓	✓7.0+	✓	✓	✓	✓	✓	✓
Server-Sent Events	✓4.1+					✓		✓	✓	
Web Sockets	✓4.2+			✓6.1+	✓			✓	✓7+	
Web Workers	✓5.0+			✓6.0+	✓	✓		✓	✓	
Canvas API	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SVG	✓	✓4.0+	✓	✓	✓	✓	✓	✓	✓	
Motion Sensors	✓4.2	✓4.0+	✓			✓			✓moz	
HTML5 Form Virtual Keyboards	✓			✓	✓	✓	✓	✓	✓	
HTML5 Form New Controls	✓5.0+		✓					✓	✓	
Touch Events	✓	✓2.1+	✓	✓6.1+	✓	✓		✓(android)	✓	
Basic CSS3	✓	✓	✓	✓6.0	✓	✓	✓	✓	✓	✓
CSS3 Transforms 2D	✓	✓2.0+	✓	✓6.0	✓	✓	✓	✓	✓	✓
CSS 3 Transforms 3D	✓	✓4.0+	✓		✓	✓				
CSS 3 Transitions	✓	✓2.0+	✓	✓6.0	✓	✓		✓	✓	✓
CSS 3 Animations	✓	✓2.0+	✓	✓6.0	✓	✓			✓	✓
Viewport definition	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Position: fixed support	✓5.0+	✓2.2+	✓3.1+	✓7.0+	✓	✓		✓		
XMLHttpRequest 2.0	✓5.0+	✓4.0+	✓			✓				
WebGL		✓2.3 Sony Xperia			✓2.0 beta					
Network Information API		✓2.2+	✓							
File API		✓4.0+	✓					✓11.1+		
HTML Media Capture		✓4.0+	✓							
IndexedDB									✓	

Table 3.1: Mobile Html5 Feature List, status as December 2011 [Fir11], [Dev11]

4 Testing

The mobile web is a fast changing place. To keep up with new technologies, devices, etc. the development process of new applications has to react fast on this changes. In order to achieve such a goal, agile practices are needed. This chapter focuses on testing in agile teams, because of this reason. Other development processes can use the same testing techniques in a different order or only fractions of them. For example, the waterfall model will use the same regression and usability testing techniques, only at the very end of the development process, instead of after every iteration cycle.

The agile development processes fits to mobile development because of the high productivity, the faster time to market and the high quality of the outcome. [Coh10]

4.1 Agile Testing

In comparison to linear development processes, like the waterfall model, testing is not postponed until development has finished. If the testing phase is the last phase, it gets squished or skipped quite often as time is running short. Agile testing is iterative and incremental. As tests are already finished before the actual coding, errors are caught very early. In early development stages errors can be fixed a lot easier and thanks to the incremental steps the code part causing the error can be isolated. Testing in an agile process is not only writing tests to see if the code has no errors, it also includes verifying that the customer requirements are met, that the end-user is able to use the application and non-functional requirements like performance and security. It is not possible to automate all tests. Different projects might not need heavy testing on all the mentioned areas, but at least all testing areas should be considered and tested appropriate. Lisa Crispin and Janet Gregory divide the different tests into 4 areas, as seen in Figure 4.1. The tests on the left hand side (Q1 and Q2) describe tests that are driving the development. Those tests are written before the actual coding and support the team during the development process. They are executed every time the code changes. The tests on the right side (Q3 and Q4) critique the application. They should be run for every released feature to catch errors as soon as possible.

The tests that are used to support the team and the ones that critique the product are divided into two parts. Technology facing tests (Q1 and Q4), which are not meant for business experts and customers. They are highly technical and need knowledge of the used technologies. Business facing tests (Q2 and Q3) are also meant for business experts and customers. They reflect the requirements and are easy to understand. [CG09]

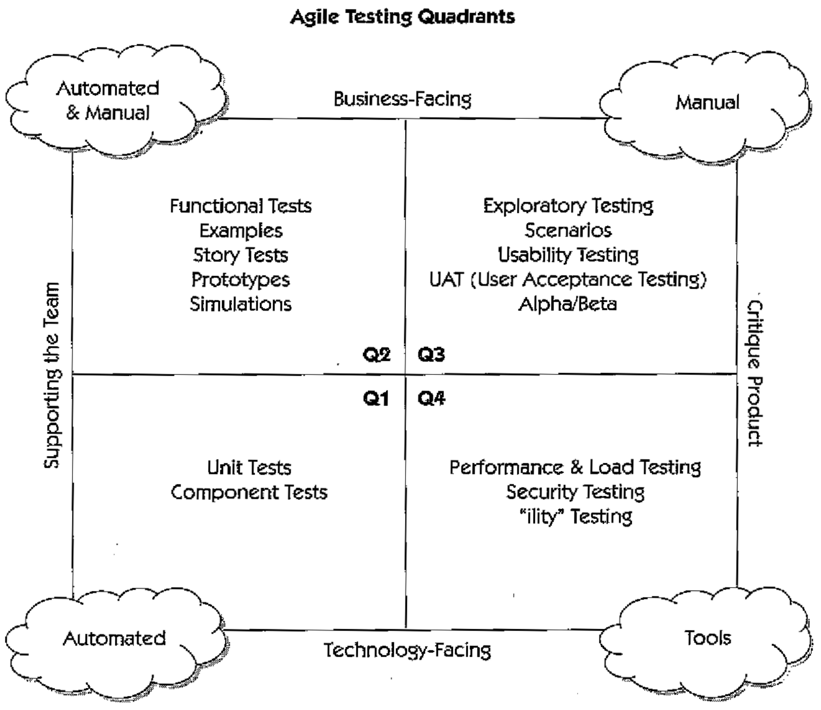


Figure 4.1: Agile Testing Quadrants [CG09]

4.1.1 Technology-Facing Tests that Support the Team

These tests are mainly written by the programmers themselves. For every piece of functionality a programmer wants to implement, a test has to be written first. Those unit tests drive design and development. More Details on Test-Driven Development can be found in Section 4.2. Tests are usually automated and are for internal use only. Therefore, they need not to be understandable for customers. They are a factor for the internal quality of the code. This quadrant also includes regression and component tests which test the interaction between the functional pieces tested with the unit tests. [CG09]

4.1.2 Business-Facing Tests that Support the Team

The tests from quadrant 2 are of a higher level. They should be written in a language the business experts can understand and should be created with close contact to the customer as they reflect the customer's requirements. Ideally they are written by the customer directly. Those tests are used to drive development as well. See Section 4.2 for information about Behaviour-Driven Development (BDD). To accomplish this task they have to be automated and run at the same interval than the unit tests the programmers are writing themselves. Quadrant 2 tests are a part of the so called acceptance tests. A story is not finished until all test are passed and all functionality is tested accordingly. [CG09]

4.1.3 Business-Facing Tests that Critique the Application

The majority of the tests from this quadrant are executed manually. Including acceptance tests, usability tests that are run by actual user and exploratory testing. The tests in quadrant 3 have the purpose to reveal requirements that were misinterpreted by the programmers or not well formulated. The usability may feel different on the real product than in the head of the customer during the concept phase and can be tested only after some features are already usable. Exploratory tests are tests executed by professional testers to reveal errors on some edge cases or not tested combinations of events the developers have not thought about. Because the systems code is already well tested with an automated testing suite consisting of the tests from quadrant 1 and 2, the testers have more time to look for bugs that are not that obvious or are hard to catch. As agile development releases every iteration a working product, those tests can be run as soon as a feature is released, to fix errors or misunderstood requirements as early as possible in the process. Even though most tests are run manually, an automation can be used for the test setup, like navigating to a special point in the application. If a test can be automated and does not take a lot of time to run, it should be included in the automated test suite the developers are running on every code change. [CG09]

4.1.4 Technology-Facing Tests that Critique the Application

Tests from this quadrant include non-functional requirements such as security, performance, memory management, robustness, etc. For example, the system should be tested if it can handle a lot of data or more than a certain amount of user at the same time. The results of these tests should be used as feedback for the tests in quadrant 1 and 2. Automated tests and story cards should be written concerning this issues, hence they can be used to drive the design of the application as well.

The tests from this quadrant might require special tools or specialists. The development team has to consider which external resources the tests might need and which cases they can handle themselves. For example the robustness of an application might be tested using the automated test suite from quadrant 1 and 2 and run multiple times against the system. These type of tests should include end-to-end functional tests, if not already handled by the regression tests from quadrant 1. Duplication should always be avoided. That applies to test cases, too. [CG09]

4.2 Test Driven Development

Test-Driven development (TDD) means that the tests are used to drive development and design. To accomplish this task, the tests have to be written first. Gerard Meszaros ([CG09] P.113) explained that the main difference between Test-First development and Test-Driven development is that Test-First development does not say that the production code is developed passing one test at a time. This means that it is essential to TDD to write a test for a tiny bit of functionality, see it fail, implement this tiny bit of functionality till the test passes and start with the test for the next bit of functionality. Using this pattern every piece of code is tested and the developer has to think in advance about the functionality he/she wants to implement. Additionally, letting the test fail first, is a small test for the test. There might be a typo and the test passes before the actual functionality is implemented. Only the happy path of a functionality can drive the development. Unlikely edge cases do not help driving the design. Of course, they should be tested, too, but it is appropriate to test them afterwards, to see if the code is robust. [CG09]

Because every programmer has an automated test suite whcih is running every time they want to change the code and after they changed something, they cannot affect the other developers. The code that gets checked in is tested and working. That way it is unlikely to crash the code on the development system. But because the tests are run so often, they have to be quick. It is essential to write easy, fast running tests. If the test suite needs too long to give feedback, it is unlikely that the developer will use it all the time. Therefore, there should be at least two different building processes: one that runs the fast running tests and one that executes the longer ones. That way the fast test suite can be used to drive development and the other suite ensures nothing breaks before the check-in. Afterwards, integration tests have to be written to ensure that the code pieces are working together. [CG09]

When using TDD the developers do not have to worry about unintended changes to the system. Additionally, they can refactor the system and get quick feedback if they broke something. With TDD the internal code quality rises. Madeyski found that using TDD leads to a significantly less coupled code and has an positive impact on the defect rate of an application. The resulting code has a modular design which leads to easier reuse and maintainability. [Mad10]

4.2.1 Test Double Pattern

To get quick and exact feedback from the unit test suite, Mocking and/or Stubbing has to be used. This means the function under test has to be isolated from other influences. In that case, if a test fails, the function under test is causing the error and not one of its dependencies. Another value of using this pattern is that tests with dependencies, which need a long time to run, can be speed up. The objects that replace these influences are referred to as test doubles. There are different types of test doubles which are explained in the following subsections. [Mes07] [Joh11]

Dummy Object

A dummy object is used for function calls or objects that are not under test, but will throw an error if not present. They are usually empty functions or objects. [Joh11]

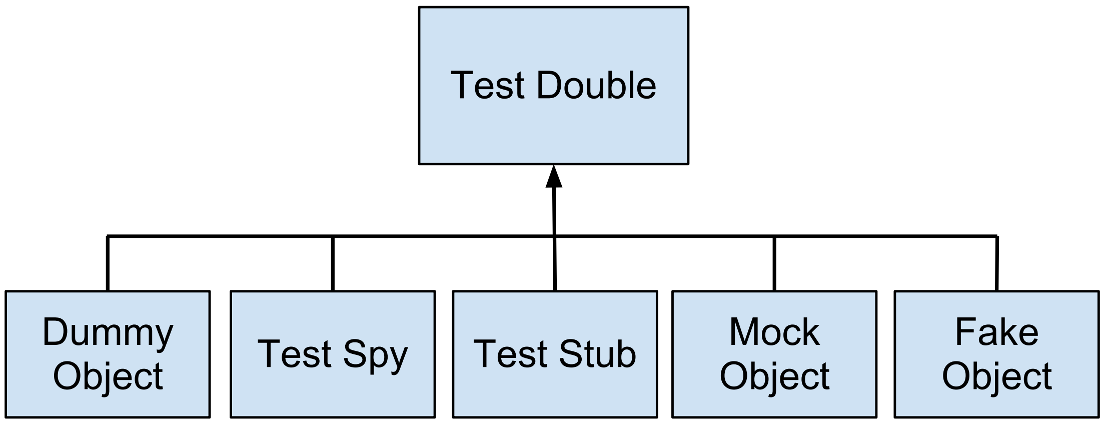


Figure 4.2: Test Double Types[Mes07]

Test Spy

A test spy is used to record the information about its usage. The indirect output of a function can be tested with the use of this object. Gerard Meszaros [Mes07] describes a test spy as a more capable version of a test stub (see Subsection Test Stub), while Christian Johansen [Joh11] does not mention the spy having to stub the functionality. Testing frameworks like Sinon.JS [1] define spies as object that log the usage and delegate the call to the original implementation. Sinon.JS-Stubs are equipped with spy functionality as well. The common feature of all those spies is the ability to log the usage.

Test Stub

Stubbing is used to isolate an interface from its dependencies. A test stub is a double with pre-programmed behaviour. It can be used to define specific return values or may throw an exception depending on the given arguments. With test stubs certain code paths can be forced and the code can be tested if it can handle unexpected behaviour. [Mes07] [Joh11]

Mock Object

A mock object is more complex than a stub. It defines not only pre-programmed behaviour, but also pre-programmed expectations and behaviour verification. The mock has an exact expectation how it should get used. If it gets used in a different way, it fails. Mock objects should not be used to silence dependencies as they might break a test. Every use for a mock object has to be defined beforehand, because mock object only support behaviour verification. In difference a test stub supports behaviour and state verification. [Joh11]

[1] http://sinonjs.org/

Fake Object

A fake object implements the same functionality then the replaced one, but in a simpler way. This can be due to various reasons. Maybe the implementation is not yet available, too slow or has some side effects. Fake objects are not used to verify some behaviour, their only purpose is to replace the original implementation. One example is an I/O-Object. It can be too slow to write data to the disk. Instead, a fake object writing the data to the RAM can be used for testing. [Mes07]

4.3 Behaviour Driven Development

The main idea behind BDD is that writing the tests should feel more natural. A test describes the behaviour of the code and it is actually the behaviour that drives the development. With this thinking, how to write a test should be more clear. The test method names should be sentences that describe the behaviour, so they are easier to understand and can be transformed into real sentences with a simple script. Business experts and other people not familiar with programming are then able to read the tests. Another advantage is, if a test fails, one can directly read what the test should do and then derive if the code had a bug or maybe the behaviour of the code has changed on purpose and the test has to be fixed. With thinking the system has to do X, the programmers focus more on the business values, because they think beforehand what the system should do (the behaviour it should implement). When Dan North started developing the idea of BDD, he was trying to use all the advantages of TDD and avoid the pitfalls. He discovered when he tried to teach about TDD, people had a hard time to get the idea behind it. That is why BDD is an extension to TDD which is easier to understand how it is meant to work. It encourages a thinking about the tests as requirements for the system. [Nor06]

5 Frameworks

HTML5 offers a lot of new features, but unfortunately they are not all supported on every device yet. Sometimes the features are already implemented, but still have a vendor prefix. Frameworks help to smooth out some of the differences between the browsers, provide fallbacks and allow easier access of the functionality. They do a lot of work to ease development for the programmers. For example, most of the frameworks allow animated page transitions with a simple argument or handle the browser history. In that way the developers do not need full knowledge of every single functionality offered by HTML5 and CSS3.

This thesis only covers the most promising frameworks at the time of writing. Most of the books, blogs and people mentioned only the frameworks listed below.

5.1 jQuery Mobile

jQuery is a very popular and powerful JavaScript library. It normalizes the differences between web browsers and has an easy to use API which helps with a lot of tasks. But jQuery only helps with programmatic tasks and does not have any specific functionality for mobile devices. Here jQuery Mobile comes into play. It is built on top of jQuery and is optimized for mobile devices with a touch screen.

The framework supports a wide range of mobile devices like Android, iOS, Blackberry, Palm, Windows Phone and even Amazon's Kindle[1]. For older devices it has fallback modes to still stay functional. The framework offers touch optimized widgets, layouts and animations which are specifically designed for smaller screens and touch-input.

jQuery Mobile makes use of the custom `data-*` attribute allowed on every html element. With this attribute the programmer can tell the framework, what should be the header, the footer, the main content, a button, etc. The framework then generates additional styles and markup for this elements.

In order to use the animated page transitions, the new page has to be loaded via Ajax. This can take some time and might let the page appear frozen. There are two options to avoid this, displaying a loading message, or, the alternative to Ajax offered by the framework, defining multiple pages within one HTML document. A page for jQuery Mobile is a `div` with the data attribute `page` and an unique id. Listing 5.1 shows a simple HTML file with two pages. In order not to break the browser's back button, the framework makes heavy use of the history API offered by HTML5. When it comes to the structure of a page, jQuery Mobile does not

[1] http://jquerymobile.com/gbs/

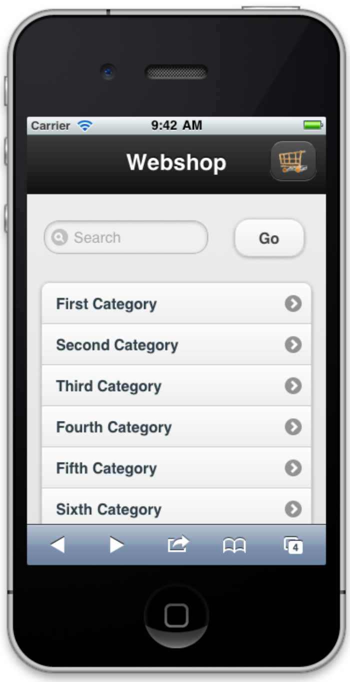


Figure 5.1: Simple Page Created with jQuery Mobile

rely on the HTML5 elements like header or footer, but on the data-role attribute. The header and footer tag can be switched to simple **divs** and the page will still look the same.

```
...
<body>
 <div id="mainPage" data-role="page">
   <header data-role="header">
     <h1>Webshop</h1>
   </header>
   <div data-role="content">
     ...
   </div>
   <footer data-role="footer">
     ...
   </footer>
 </div>
 <div id="secondPage" data-role="page">
   ...
  </div>
</body>
...
```

Listing 5.1: A simple jQuery Mobile page

In Figure 5.1 you can see a jQuery Mobile page with only a little of custom CSS.

One of the biggest advantages of jQuery Mobile is that it is very easy to use and needs only

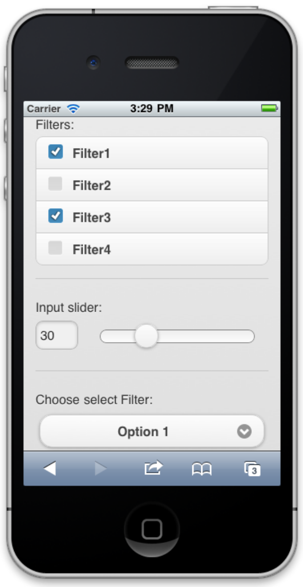


Figure 5.2: A Form with jQuery Mobile Widgets

little JavaScript knowledge to create good looking mobile web apps. The framework offers many additional widgets like custom sliders, select fields or checkboxes, which can be used only by setting the specific `data-*` attribute on the HTML element. Figure 5.2 shows an excerpt of the available widgets.

jQuery Mobile is focusing on the main tasks like navigation and the look, but as it uses the jQuery core library, it has all of its advantages. Custom events can be fired, the DOM can be manipulated with an easy to use API and all the jQuery plugins are available as well. For example neither jQuery nor jQuery Mobile have any functionality regarding client side storage, but there are a lot of plugins available for this task. Other plugins allow to swipe through pictures using the swipe events offered by jQuery Mobile.

The framework is free of charge and might be used under the terms of either the MIT Licence or GNU General Public License (GPL) Version 2 [2].

[FR11], [jQu10], [Rei11], [Thea]

5.2 Sencha Touch

Sencha Touch in comparison with jQuery Mobile is taking a completely different approach. While jQuery Mobile enriched HTML markup, Sencha Touch applications are entirely written in JavaScript. This requires some learning time of the framework's functions, objects and structure. Additionally, a good knowledge of the JavaScript programming language is recommendable for bigger applications.

[2] http://jquery.org/license/

On the 14th of June 2010 the Ext JS team joined with jQTouch and Raphaël and changed the name of the company to Sencha. The new knowledge was used to create the framework Sencha Touch and the first public beta was already released on the 17th of June 2010. Five month later Sencha Touch 1.0 was ready including a free commercial license. In March 2011 the version 1.1 supported, additionally to Android and iOS, BlackBerry 6. Since 2.0 it is possible to convert a Sencha Touch app into a native iOS or Android application with a single command. With this feature some native device APIs are now available within the framework. Additionally, the new class loading system, first introduced for Ext JS 4, was ported to this framework.

Sencha Touch is developed to work best on WebKit based browsers. It makes use of all the capabilities this browser engine offers. This has the advantage that the applications developed with Sencha Touch look like native apps, but the disadvantage is that they do not work on other engines. Future releases are planned to be based on the CSS3 specification without needing any WebKit exclusive features.

In order to develop applications with this framework a lot of JavaScript knowledge is necessary, but only little of HTML code. Listing 5.2 shows all needed HTML code. Everything else is coded in JavaScript.

```html
<!DOCTYPE html>
<html>
  <head lang="en">
    <title>Webshop Sencha Touch</title>
    <meta charset="utf-8">
    <meta name="author" content="Stefan Mayer">
<script src="lib/touch/sencha-touch.js" ></script>
<script src="app.js"></script>

<link rel="stylesheet" href="css/sencha-touch.css" />
<link rel="stylesheet" href="css/my.css" />
</head>
  <body>

  </body>
</html>
```

Listing 5.2: Sencha Touch HTML code

Other JavaScript files can be loaded using the new class loader. The application needs therefore a specific structure in order to find the class files. Views have to be in the `view` directory, controllers in the `controller` directory and so on. The application is initialized in the app.js file. This file only consist of the application name and the used controllers. The class-loader configuration happens in this file as well. An example can be seen in Listing 5.3.

```
Ext.Loader.setConfig({
    enabled: true
});
Ext.application({
    name: 'Webshop',
    controllers: ['Main']
});
```

Listing 5.3: Sencha Touch app.js

Sencha Touch applications are meant to work with the MVC-Pattern, but it is not mandatory. In order to have a maintainable code, the classes offered by the frameworks can be extended, as known from other object oriented programming languages. Sencha Touch offers its own functions to extend, define and instantiate classes. Examples are shown in Listings 5.4 and 5.5.

```
Ext.define('Webshop.view.Cart', {
    extend: 'Ext.Panel',
    alias: 'widget.cartView',
    config: {
      ...
    }
});
```

Listing 5.4: Sencha Touch Class Extending

```
var shoppingCart = Ext.create('Webshop.view.Cart', {
    id: 'senchaCart'
});
```

Listing 5.5: Sencha Touch Class Instantiation

Sencha Touch offers a lot of widgets, like toolbars, styled buttons, tab navigation, lists, form inputs and different overlay dialogs. In order to allow custom styles for those widgets the framework uses Sass. It is an extension of CSS3 and allows to work with nested rules, variables and inheritance. The Sass files can be compiled to CSS afterwards. Sencha Touch makes use of variables to allow custom theming. With replacing some values, the used colour of all widgets can be changed and the framework still has a consistent look. Predefined styles not needed for the current application (if some widgets or icons are not used) can easily be removed in order to reduce the size of the final file. An example is shown in Listing 5.6. Figure 5.3 shows the framework in action.

Sencha Touch is available under different licenses depending on the type of use. The general commercial license is free, but in case the framework is used to develop other commercial frameworks, a paid license is needed. Additionally a special open source license is available for open source projects.

[Sen11]

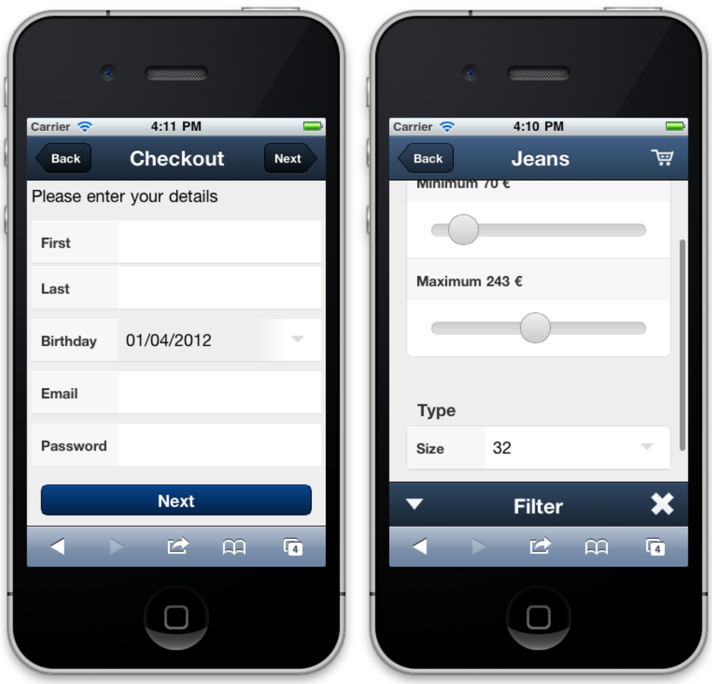


Figure 5.3: Sencha Touch Widgets

```
$body-bg-color: #000;

$base-color: #333;
$base-gradient: 'matte';
$active-color: #B2DF1E;

$tabs-dark: $base-color;
$tabs-light: #555;

$tabs-bottom-radius: .4em;
$tabs-bottom-gradient: 'bevel';
$tabs-bar-gradient: 'matte';
$tabs-bottom-active-gradient: 'recessed';

$toolbar-gradient: 'glossy';
```

Listing 5.6: Sencha Touch Theming with Sass

5.3 Other Frameworks

This section includes smaller, less important frameworks. They do not have an equally big and active community as the two frameworks listed above. They support less features and have a more simple look, but might be still a good choice for smaller applications because of their small filesize or easier to use API.

5.3.1 jQTouch

It was first developed as a jQuery plugin focusing on allowing to create mobile web applications using a simple syntax. Later jQTouch b4 was released with two options: using jQuery as the core or Zepto.js. Zepto.js is a project which recreates the jQuery API optimized for WebKit browsers. As jQTouch also focuses on WebKit and Zepto.js has a smaller filesize and a faster loading speed, it is going to be the default library for future versions. [Kan11]

jQTouch offers not as many widgets as Sencha Touch or jQuery Mobile, but is therefore smaller in filesize. It runs only on WebKit based browsers, while for example jQuery Mobile is optimized for a broader range of browsers. This framework is meant for developing smaller, less complex applications for mobile devices with a WebKit based browser.

The framework works quite similar than jQuery Mobile, but instead of using the `data-*` attribute, it looks for specific classes. The elements marked with the specific CSS classes are enriched with special markup and CSS. Like Sencha Touch it works with Sass, which allows easier change of the global styles. [Pea11]

A code sample of jQTouch can be seen in Listing 5.7.

```
...
<body>
  <div id="mainPage">
    <div class="toolbar">
      <h1>Webshop</h1>
    </div>
    <ul class="edgetoedge scroll">
      ...
    </ul>
    <div class="info">
      ...
    </div>
  </div>
  <div id="secondPage">
    ...
  </div>
</body>
...
```

Listing 5.7: Multiple jQTouch Pages

The current version (1 beta 4rc) of the framework works well on iOS devices, but has some issues on Android phones. On Android you can observe how the framework is adding the additional markup and the performance of scrolling is very poor. Even the widgets look different on the devices. A screenshot is displayed in Figure 5.4. The framework is available for free under the MIT License. [3]

[3] http://jqtouch.com/

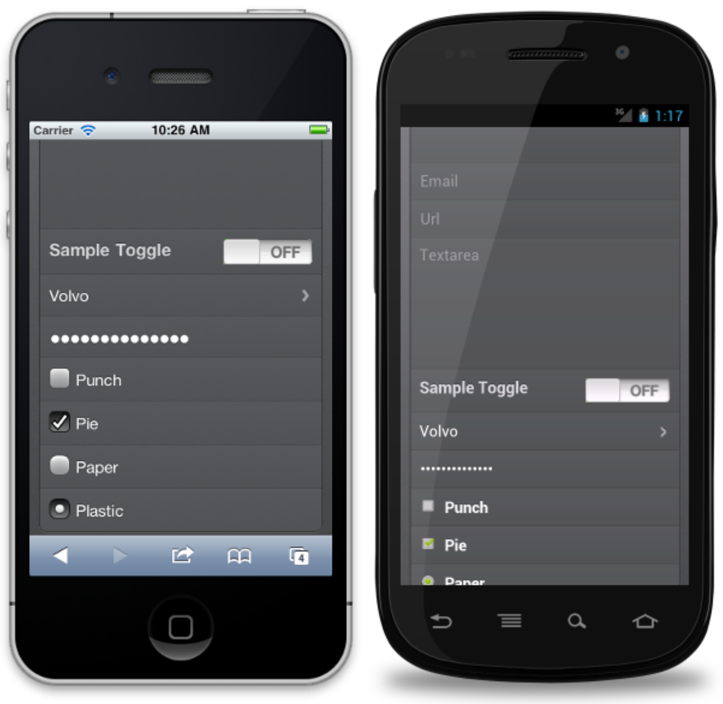


Figure 5.4: jQTouch Kitchen Sink (`http://jqtouch.com/`) Browser Comparison

5.3.2 Jo

Jo is a lightweight mobile UI-framework without any dependencies to other libraries, which has a positive impact on the filesize. On the other hand, the default look of the widgets is not as fancy as of the other frameworks, nor is it as comprehensive as for example jQuery Mobile or Sencha Touch. The framework is tested on iOS, Android, webOS and Blackberry.

Like with Sencha Touch, the entire application is written in JavaScript. No further page loads are needed, once the page is ready. Jo is available under the BSD License. An example can be seen in Figure 5.5.

[OB11]

5.4 Summary

From the current point of view, jQuery Mobile and Sencha Touch are the most promising frameworks. They offer a lot of widgets and functions to develop native alike mobile web applications. Both frameworks have a big active community and are under active development. All frameworks still have issues with animations and transformations on Android, but jQuery Mobile and Sencha Touch are putting in great efforts for a better performance. Every new release has an improved Android support. Other devices do not have a big market share and

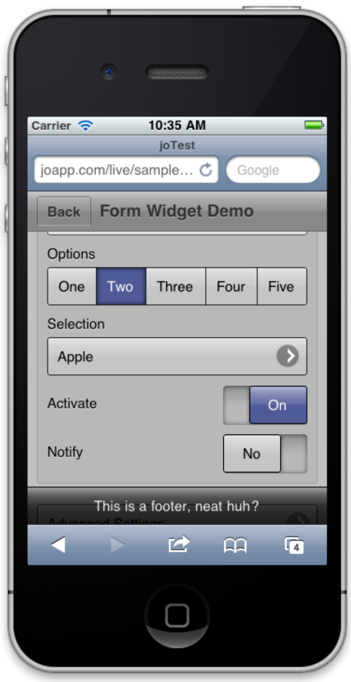


Figure 5.5: Jo Kitchen Sink from `http://joapp.com/demos.html`

are not a top priority for the framework developers. For small applications jQTouch might be appropriate because of its its small filesize and easy syntax, but more complex applications should rely on the two big frameworks. Depending on the preferences jQuery mobile and Sencha Touch have both their advantages and disadvantages.

6 Testing-Frameworks

Mobile web applications have a lot of JavaScript code in order to make use of the features offered by HTML5 and CSS3. Therefore, the client side code gets more and more important and the web applications get more complex. This makes manual testing tedious and error-prone. Additionally, most developers do not like testing and companies are trying to keep costs short. Testing-frameworks help to automate testing and allow to spend as few time as possible on writing and executing tests. They help to run the tests continuously, compare results, make screenshots and enable running the test suite on multiple devices at the same time.

In which cases the specific frameworks make most sense is explained in Chapter 7. This chapter list the advantages and limitations of the frameworks. The chapter is divided into in-browser testing-frameworks, headless testing frameworks, test drivers and stubbing and mocking libraries. All this frameworks test the JavaScript business logic for mobile web applications or the GUI. Backend testing is not covered within this document.

[Joh11], [CS11]

6.1 In-Browser Testing-Frameworks

This type of testing-frameworks run within the browser. The advantage of this approach is, that the tests are run within a real environment. The disadvantage is that on their own, they cannot be run automated and repeatedly. Some of them do not offer a command line tool to start the tests or to evaluate the results. It might be hard to introduce continuous integration with some of these testing-frameworks. [Joh11]

6.1.1 QUnit

QUnit is written and developed by the jQuery Project. Because it is an in-browser testing-framework, a HTML page is needed to run the tests. HTML elements mandatory for the tests are placed within a div with the id `"qunit-fixture"`. Listing 6.1 shows the template HTML page.

```html
<body>
  <h1 id="qunit-header">Webshop Tests</h1>
  <h2 id="qunit-banner"></h2>
  <div id="qunit-testrunner-toolbar"></div>
  <h2 id="qunit-userAgent"></h2>
  <ol id="qunit-tests"></ol>
  <div id="qunit-fixture">
    ...
  </div>
</body>
```

Listing 6.1: QUnit HTML Template

In order to run the tests the framework files, the code that should be tested and the test code have to be included in the HTML file. The tests itself can be categorized with the function `module()` and the actual test is coded within the `test()` function. A sample can be seen in Listing 6.2. The result of the tests are displayed visually. Failed tests are displayed in red including an error message, while passed tests are blue. See an example in Figure 6.1.

```javascript
module("Helper Tests");
test("should be object", function() {
  expect(1);
  equal( typeof(Webshop.helper), "object", "should be an object");
});
test("should have isAppleMobile function", function() {
  expect(1);
  equal( typeof(Webshop.helper.isAppleMobile), "function", "should be a
      function" );
});
```

Listing 6.2: QUnit Test Example

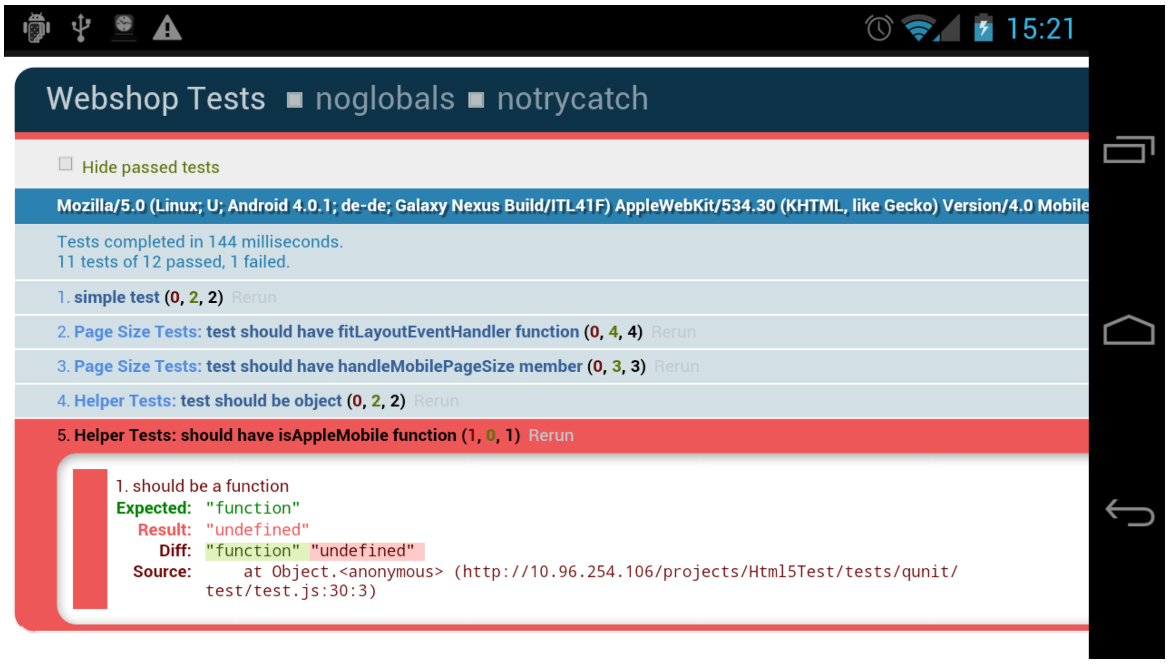
The `expect()` function tells QUnit how many assertions are expected to be called. Supported assertions are `ok()`, for variables that should be true, `equal()`, `notEqual()`, `deepEqual()`, `notDeepEqual()`, `strictEqual()` and `notStrictEqual()`, for comparison and `raises()` to test for exceptions. Asynchronous tests are supported as well. This allows the testing of Ajax calls or timeouts.

To allow integration with automation tools, QUnit provides a simple microformat as output for easier parsing. QUnit is meant for unit and regression testing. It can be combined with various automated test runners and other libraries. A big advantage is, that the tests can be run in every browser with access to the server. Therefore only the test-URL has to be typed into the browser in order to run the tests on a mobile device or emulator.

[CS11], [Mac11], [Theb]

6.1.2 YUI Test

YUI is a JavaScript and CSS library from Yahoo!. As other big frameworks it offers a testing library as well. YUI Test is an in-browser testing-framework that offers a lot of



```
15:21
Webshop Tests ■ noglobals ■ notrycatch
☐ Hide passed tests
Mozilla/5.0 (Linux; U; Android 4.0.1; de-de; Galaxy Nexus Build/ITL41F) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile
Tests completed in 144 milliseconds.
11 tests of 12 passed, 1 failed.
1. simple test (0, 2, 2) Rerun
2. Page Size Tests: test should have fitLayoutEventHandler function (0, 4, 4) Rerun
3. Page Size Tests: test should have handleMobilePageSize member (0, 3, 3) Rerun
4. Helper Tests: test should be object (0, 2, 2) Rerun
5. Helper Tests: should have isAppleMobile function (1, 0, 1) Rerun
   1. should be a function
   Expected: "function"
     Result: "undefined"
       Diff: "function" "undefined"
     Source:     at Object.<anonymous> (http://10.96.254.106/projects/Html5Test/tests/qunit/
             test/test.js:30:3)
```

Figure 6.1: QUnit Result Page on an Android Device

features. Besides a test runner and assertions it includes a mocking library, allows testing of asynchronous functions and can simulate mouse events. The framework has various ways to present the results. It is possible to render the results in a more user friendly way, but also machine readable formats like XML or JSON are possible.

Like QUnit, YUI Test requires an HTML file to start the tests. The HTML file only has to include the framework files and the test files. DOM fixtures can be included into the HTML file if needed. The Tests are run manually as soon as a browser opens the HTML file. Unfortunately the simulated user actions are quite limited in their functionality, therefore other frameworks should be used for such kind of tests.

Some example code can be found in Listing 6.3. All test cases have to be added manually to the test runner or else will not be executed.

[Joh11], [BW10], [Yah11]

```
YUI.webshop.test.HelperTestCase = new YUI.Test.Case({

    name: "Helper Tests",

    "test should be object": function () {
        YUI.Assert.isObject(Webshop.helper);
    },

    "test should have isAppleMobile function": function () {
        YUI.Assert.isFunction(Webshop.helper.isAppleMobile);
    }
});
```

Listing 6.3: YUI Test Example

6.1.3 Jasmin

Jasmin is developed especially for behaviour driven development. It does not require any other libraries, frameworks or a DOM, which has a positive impact on the file size, performance and can run in every environment.

Such as the other frameworks, it needs a HTML file to load and execute the JavaScript code. Listing 6.4 shows an example test case.

```
describe("Helper Tests", function() {
    it( "should be an object", function () {
        expect( typeof(Webshop.helper)).toBe( "object" );
    });

    it( "should have isAppleMobile function", function () {
        expect( typeof(Webshop.helper.isAppleMobile)).toBe( "function" );
    });
});
```

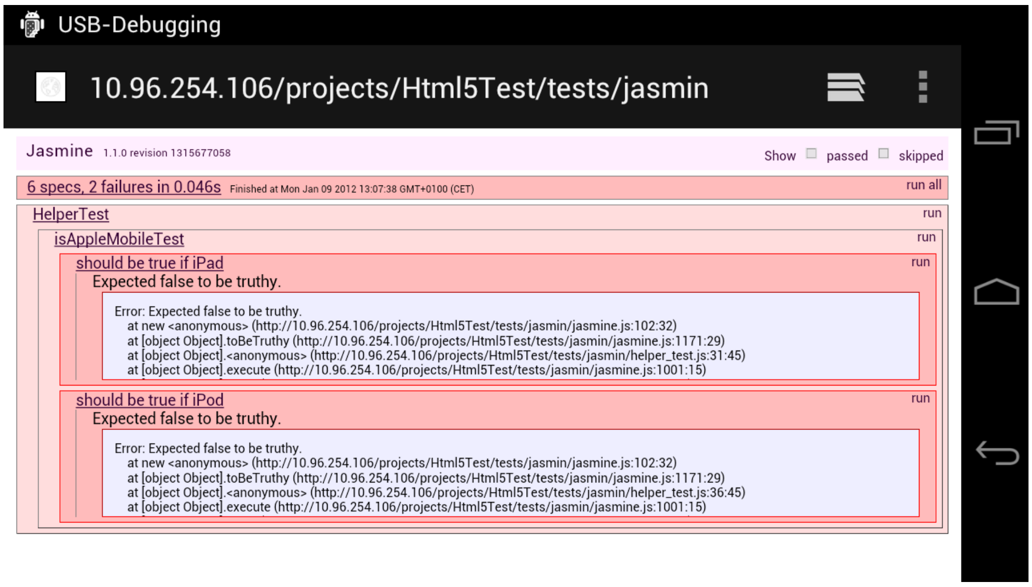
Listing 6.4: Jasmin Test Example

As known from Chapter 4.3, BDD has a more natural language like syntax. Jasmin uses for grouping the tests the function `describe()` and the tests are run within the function `it()`. This allows sentences like `it("should have ...)`.

Instead of the usual asserts, Jasmin uses more natural keywords, like `expect(..).toEqual(..)` or `expect(..).toBeTruthy(..)`. Additionally, it offers `spies`. These Jasmin spies are not equivalent to the definition in Chapter 4.2.1. They are a combination of spies and mocks.

Figure 6.2 shows the visual results of some executed tests run on an Android device. Jasmin can easily be integrated into scripts or test runners for continuous integration or other automated test suites. It is meant to work with Ruby, Maven, JsTestDriver and Django.

[Mac11], [Piv11]



```
USB-Debugging
10.96.254.106/projects/Html5Test/tests/jasmin

Jasmine 1.1.0 revision 1315677058                                    Show ☐ passed ☐ skipped
6 specs, 2 failures in 0.046s Finished at Mon Jan 09 2012 13:07:38 GMT+0100 (CET)   run all
HelperTest                                                                       run
  isAppleMobileTest                                                              run
    should be true if iPad                                                       run
      Expected false to be truthy.
        Error: Expected false to be truthy.
          at new <anonymous> (http://10.96.254.106/projects/Html5Test/tests/jasmin/jasmine.js:102:32)
          at [object Object].toBeTruthy (http://10.96.254.106/projects/Html5Test/tests/jasmin/jasmine.js:1171:29)
          at [object Object].<anonymous> (http://10.96.254.106/projects/Html5Test/tests/jasmin/helper_test.js:31:45)
          at [object Object].execute (http://10.96.254.106/projects/Html5Test/tests/jasmin/jasmine.js:1001:15)
    should be true if iPod                                                       run
      Expected false to be truthy.
        Error: Expected false to be truthy.
          at new <anonymous> (http://10.96.254.106/projects/Html5Test/tests/jasmin/jasmine.js:102:32)
          at [object Object].toBeTruthy (http://10.96.254.106/projects/Html5Test/tests/jasmin/jasmine.js:1171:29)
          at [object Object].<anonymous> (http://10.96.254.106/projects/Html5Test/tests/jasmin/helper_test.js:36:45)
          at [object Object].execute (http://10.96.254.106/projects/Html5Test/tests/jasmin/jasmine.js:1001:15)
```

Figure 6.2: Jasmin Result Page on an Android Device

6.1.4 Siesta

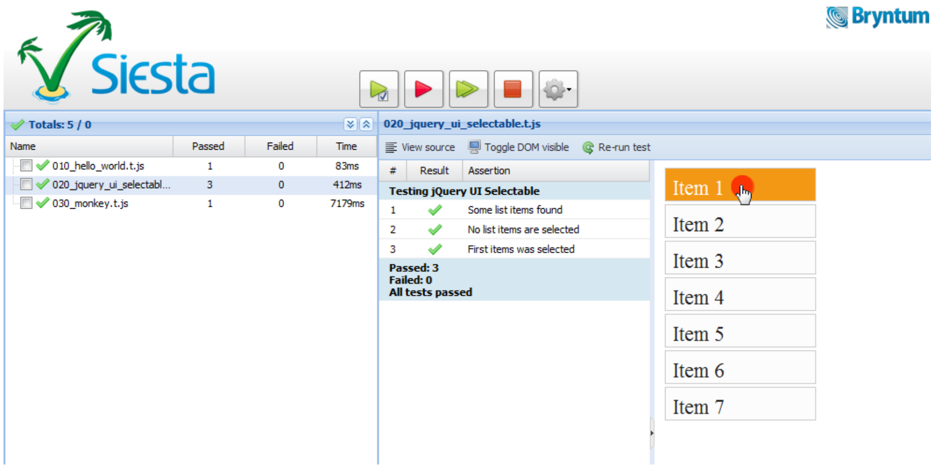
Siesta is developed especially for the ExtJS framework, which is the desktop version of Sencha Touch. Besides ExtJS it supports jQuery applications, too. The framework allows testing the JavaScript code, DOM manipulations and can simulate user actions like clicks, text inputs and drag and drop.

It offers a wide range of functionality like identifying unexpected global variables or detecting page refreshes, by letting the JavaScript code run in a separate context.

The tests can be run within the browser and the results are presented using ExtJS. This allows to rerun the tests, see the source code of each single test and observe the user actions. A screenshot can be seen in Figure 6.3. Besides running the test within the browser Siesta offers to run them headlessly with PhantomJS or on multiple browser via Selenium. PhantomJS is a headless WebKit implementation. See Section 6.2 for more information about headless testing. Selenium is described in Chapter 6.3.2.

The framework can be used for any kind of web applications, but it offers special features for ExtJS and jQuery apps. Future releases of this framework will be even more interesting for mobile web applications, as official support for Sencha Touch has been announced.

[Bry11], [Bry12]



Totals: 5 / 0

Name	Passed	Failed	Time
✓ 010_hello_world.t.js	1	0	83ms
✓ 020_jquery_ui_selectabl...	3	0	412ms
✓ 030_monkey.t.js	1	0	7179ms

020_jquery_ui_selectable.t.js

#	Result	Assertion
Testing jQuery UI Selectable		
1	✓	Some list items found
2	✓	No list items are selected
3	✓	First items was selected

Passed: 3
Failed: 0
All tests passed

Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7

Figure 6.3: Siesta Example [Bry12]

6.2 Headless Testing-Frameworks

Headless testing-frameworks are command line tools that emulate a browser. Because of the lack of a GUI, the tests can be executed very fast and run even on servers without a graphic card. This kind of frameworks can easily be integrated with continuous integration tools and can be run frequently. The downside is that they only emulate a browser. If the tests passes within the emulated browser, there is no guarantee that they are going to pass in a real environment. Even if a real browser is emulated, the tools cannot keep up with the fast update cycles of modern browsers and cannot emulate the real counterpiece perfectly. [Joh11]

6.2.1 Envjs and Rhino

Rhino is a JavaScript implementation in Java developed by Mozilla. It allows to compile JavaScript code into Java bytecode or to interpret it during runtime. Envjs is built on top of Rhino and implements a browser and a DOM API. The combination of both plus a testing framework like QUnit or Jasmin allows to execute the JavaScript tests headless. [Mac11], [Joh11]

6.2.2 Zombie.js and Node.js

Another server side implementation of JavaScript is Node.js. It uses Google V8 JavaScript engine, which is used in Chrome and Android. Zombie.js runs on top of Node.js, implements a browser API and executes the tests. The test and the client code are run in separate contexts. Hence, global variables cannot affect the test code. Additionally, tests can run parallel in different contexts and make use of multicore CPUs.

Zombie.js keeps the state of the browser object between page transitions for testing of cookies, web storage and the history. Tests can even interact with the page, like filling in and submitting forms.

[Mac11]

6.2.3 PhantomJS

It is a headless implementation of WebKit, the engine used in most mobile web browsers. It offers a DOM API, CSS selectors, Canvas and SVGs. PhantomJS itself does not offer testing functionality, but it is compatible with other testing frameworks like Jasmin or QUnit. The implementation allows to render the website into an image, which can be used for comparing the results. [Hid11]

6.3 Test Driver

Till now all the introduced frameworks are run only within one environment. The in-browser testing-frameworks can be run within any browser manually, but therefore one person has to execute them and compare the results. Test Drivers do exactly this task. They run a test suite in multiple environments and collect and interpret the results. [Joh11], [Mac11]

Again, only tools which work with mobile devices are mentioned. Popular test drivers like Watir[1] work only with desktop browsers at the time of writing and are therefore not described in this document.

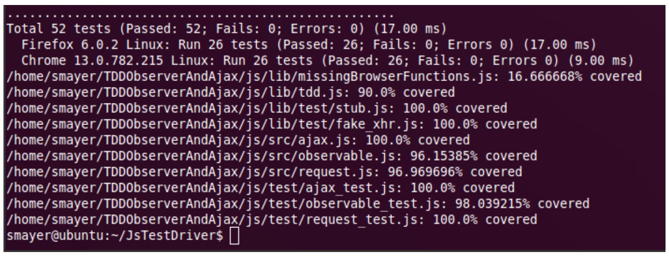
6.3.1 JsTestDriver

JsTestDriver is a tool provided by Google to run unit-tests on different browsers at the same time. It consists of two parts, a server and a client. The server runs on one machine and can handle multiple clients running on the same machine or different ones. Every device with network access to the server can connect it's browsers. They stay connected as long as the server is running and the network connection is available. The developers run their test suite with the help of the client. The unit-tests are distributed to the connected browsers and run in parallel on all of them.

The big advantage of this system is, that code can be tested easily on different machines and browsers at the same time. For example, if the goal is to develop a web application mainly for iPhone and Android, their browsers can be connected to the server and the unit tests can be run on both of them with a single command.

JsTestDriver also offers a code coverage and an some IDE plugins. Unfortunately, they cannot be combined yet. If projects already use other testing frameworks like QUnit or YUI, they

[1] `http://watir.com/`



```
....................................................
Total 52 tests (Passed: 52; Fails: 0; Errors: 0) (17.00 ms)
  Firefox 6.0.2 Linux: Run 26 tests (Passed: 26; Fails: 0; Errors 0) (17.00 ms)
  Chrome 13.0.782.215 Linux: Run 26 tests (Passed: 26; Fails: 0; Errors 0) (9.00 ms)
/home/smayer/TDDObserverAndAjax/js/lib/missingBrowserFunctions.js: 16.666668% covered
/home/smayer/TDDObserverAndAjax/js/lib/tdd.js: 90.0% covered
/home/smayer/TDDObserverAndAjax/js/lib/test/stub.js: 100.0% covered
/home/smayer/TDDObserverAndAjax/js/lib/test/fake_xhr.js: 100.0% covered
/home/smayer/TDDObserverAndAjax/js/src/ajax.js: 100.0% covered
/home/smayer/TDDObserverAndAjax/js/src/observable.js: 96.15385% covered
/home/smayer/TDDObserverAndAjax/js/src/request.js: 96.969696% covered
/home/smayer/TDDObserverAndAjax/js/test/ajax_test.js: 100.0% covered
/home/smayer/TDDObserverAndAjax/js/test/observable_test.js: 98.039215% covered
/home/smayer/TDDObserverAndAjax/js/test/request_test.js: 100.0% covered
smayer@ubuntu:~/JsTestDriver$ 
```

Figure 6.4: JsTestDriver Code Coverage Output

can be run with the use of adapters. On the homepage of JsTestDriver[2] are adapters listed for the most important frameworks and others might be added soon or found elsewhere on the Internet. Additionally, the test driver differentiates between normal and strict mode. If the browser is captured in strict mode, all scripts are distributed with the `"use strict"` prefix. This allows to trigger the strict mode provided by ECMAScript 5. This special mode throws errors on unsafe or error-prone code parts. As JsTestDriver host the JavaScript files, Ajax requests are directed to the test driver's server, too. In order to allow communication with a backend server, it allows to configure gateways. They redirect a certain URL path to a specific server.

One big advantage of JsTestDriver is that the client is able to connect to any JsTestDriver server. This allows to run for example two different servers. One with only the main target devices, which allows to execute the tests fast and a company test server that has all browsers connected the company wants to support. The developers can run local tests against the main supported browsers on a regularly base and use the company test server before deployment to test against all browsers.

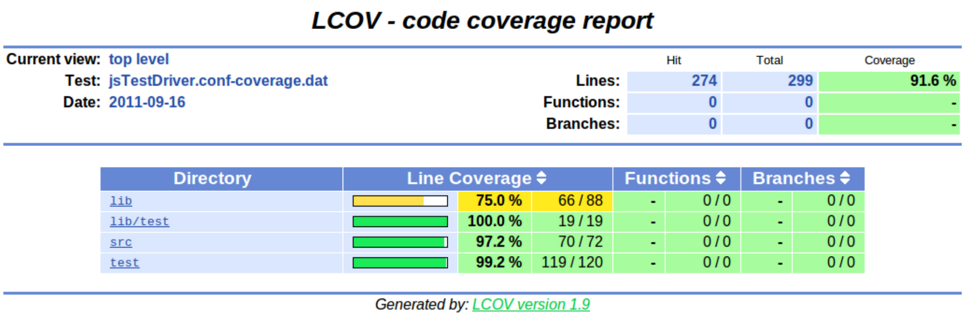
For DOM interactions JsTestDriver offers to write HTML-elements into JavaScript variables without rendering them. It is also possible to add HTML to the global DOM, but this renders the elements and might slow down the tests.

[Joh11]

Code Coverage Plugin

If this plugin is included in the configuration file, it is run every time the test suite is executed. For every file the aggregated coverage percentage is displayed. This gives a good overview about the code health. Even the coverage of the test cases is displayed. Figure 6.4 shows the output information. The plugin allows to write the coverage information into a file by setting the `--testOutput` flag. This file is compatible with the LCOV visualizer[3]. Figure 6.5 and 6.6 show how the line coverage is visualised. With the visualised information, missing test cases can be found. [Goo11]

[2] `http://code.google.com/p/js-test-driver/`
[3] `http://ltp.sourceforge.net/coverage/lcov.php`



LCOV - code coverage report

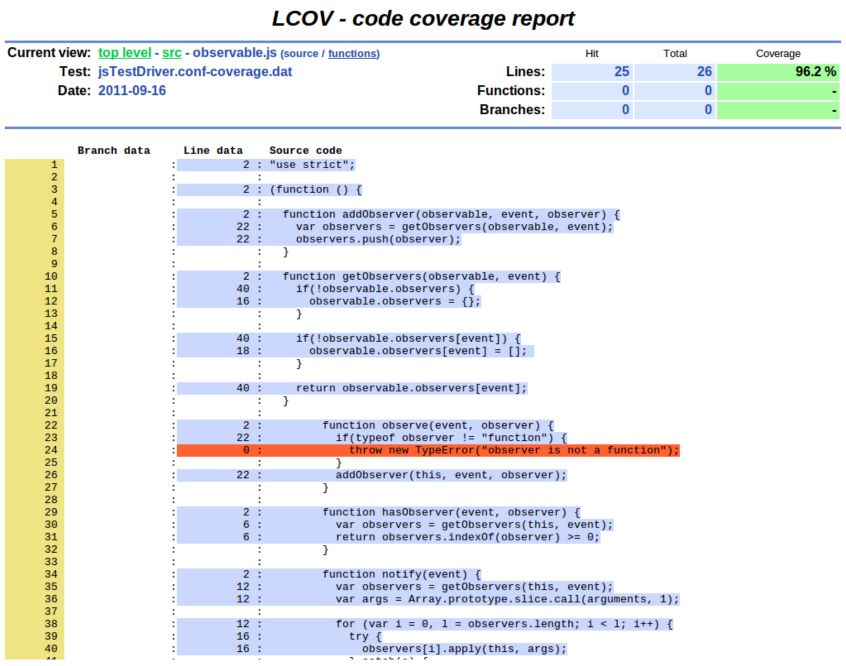
Current view: top level
Test: jsTestDriver.conf-coverage.dat
Date: 2011-09-16

	Hit	Total	Coverage
Lines:	274	299	91.6 %
Functions:	0	0	-
Branches:	0	0	-

Directory	Line Coverage			Functions		Branches	
lib		75.0 %	66 / 88	-	0 / 0	-	0 / 0
lib/test		100.0 %	19 / 19	-	0 / 0	-	0 / 0
src		97.2 %	70 / 72	-	0 / 0	-	0 / 0
test		99.2 %	119 / 120	-	0 / 0	-	0 / 0

Generated by: LCOV version 1.9

Figure 6.5: LCOV Output



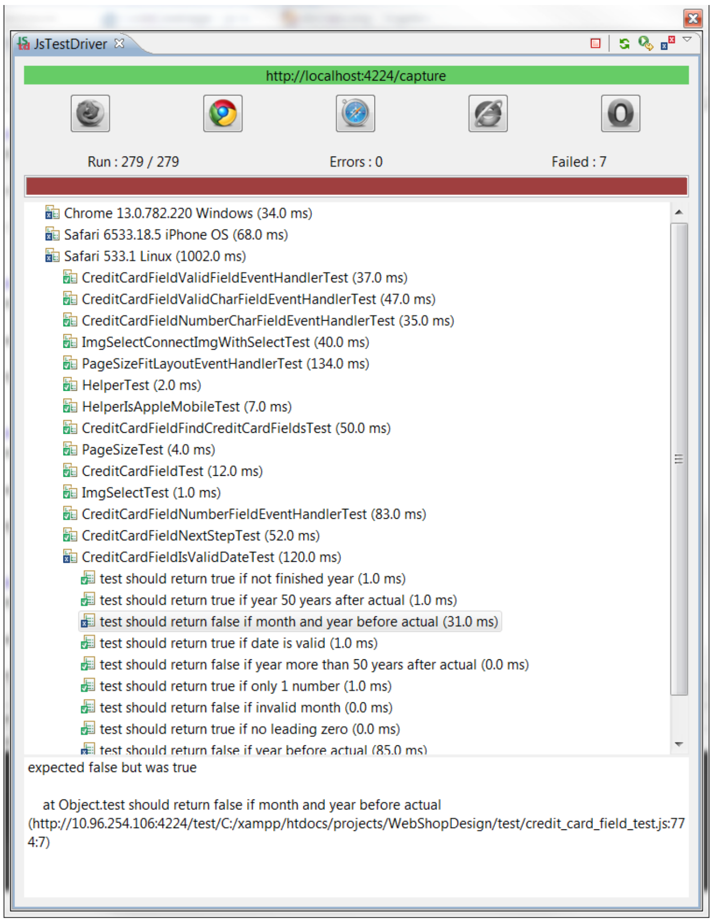
LCOV - code coverage report

Current view: top level - src - observable.js (source / functions)
Test: jsTestDriver.conf-coverage.dat
Date: 2011-09-16

	Hit	Total	Coverage
Lines:	25	26	96.2 %
Functions:	0	0	-
Branches:	0	0	-

```
  Branch data     Line data    Source code
              :           2 : "use strict";
              :             :
              :           2 : (function () {
              :             :
              :           2 :   function addObserver(observable, event, observer) {
              :          22 :     var observers = getObservers(observable, event);
              :          22 :     observers.push(observer);
              :             :   }
              :             :
              :           2 :   function getObservers(observable, event) {
              :          40 :     if(!observable.observers) {
              :          16 :       observable.observers = {};
              :             :     }
              :             :
              :          40 :     if(!observable.observers[event]) {
              :          18 :       observable.observers[event] = [];
              :             :     }
              :             :
              :          40 :     return observable.observers[event];
              :             :   }
              :             :
              :           2 :         function observe(event, observer) {
              :          22 :           if(typeof observer != "function") {
              :           0 :             throw new TypeError("observer is not a function");
              :             :           }
              :          22 :           addObserver(this, event, observer);
              :             :         }
              :             :
              :           2 :         function hasObserver(event, observer) {
              :           6 :           var observers = getObservers(this, event);
              :           6 :           return observers.indexOf(observer) >= 0;
              :             :         }
              :             :
              :           2 :         function notify(event) {
              :          12 :           var observers = getObservers(this, event);
              :          12 :           var args = Array.prototype.slice.call(arguments, 1);
              :             :
              :          12 :           for (var i = 0, l = observers.length; i < l; i++) {
              :          16 :             try {
              :          16 :               observers[i].apply(this, args);
```

Figure 6.6: LCOV Detailed Output



```
JsTestDriver
http://localhost:4224/capture

Run : 279 / 279          Errors : 0          Failed : 7

Chrome 13.0.782.220 Windows (34.0 ms)
Safari 6533.18.5 iPhone OS (68.0 ms)
Safari 533.1 Linux (1002.0 ms)
  CreditCardFieldValidFieldEventHandlerTest (37.0 ms)
  CreditCardFieldValidCharFieldEventHandlerTest (47.0 ms)
  CreditCardFieldNumberCharFieldEventHandlerTest (35.0 ms)
  ImgSelectConnectImgWithSelectTest (40.0 ms)
  PageSizeFitLayoutEventHandlerTest (134.0 ms)
  HelperTest (2.0 ms)
  HelperIsAppleMobileTest (7.0 ms)
  CreditCardFieldFindCreditCardFieldsTest (50.0 ms)
  PageSizeTest (4.0 ms)
  CreditCardFieldTest (12.0 ms)
  ImgSelectTest (1.0 ms)
  CreditCardFieldNumberFieldEventHandlerTest (83.0 ms)
  CreditCardFieldNextStepTest (52.0 ms)
  CreditCardFieldIsValidDateTest (120.0 ms)
    test should return true if not finished year (1.0 ms)
    test should return true if year 50 years after actual (1.0 ms)
    test should return false if month and year before actual (31.0 ms)
    test should return true if date is valid (1.0 ms)
    test should return false if year more than 50 years after actual (0.0 ms)
    test should return true if only 1 number (1.0 ms)
    test should return false if invalid month (0.0 ms)
    test should return true if no leading zero (0.0 ms)
    test should return false if year before actual (85.0 ms)
expected false but was true

  at Object.test should return false if month and year before actual
(http://10.96.254.106:4224/test/C:/xampp/htdocs/projects/WebShopDesign/test/credit_card_field_test.js:77
4:7)
```

Figure 6.7: JsTestDriver Eclipse Plugin

Eclipse and IntelliJ IDEA Plugin

This plugins allow easy integration of JsTestDriver with Eclipse or IntelliJ IDEA. The Eclipse version of the plugin can be configured to run the tests after every save. The plugins itself also contain the server. The advantage of this plugins is the easy integration. With the run on every save option, not even a button has to be pressed (except of starting the server once with a single click). The results are directly visible within the IDE. In combination with Test Driven Development this is a powerful tool.

The disadvantage is, that it does not always support the most recent version. An alternative approach is to define a builder for the Eclipse IDE or use Apache Maven. An additional advantage is that the code coverage plugin works with this method. The results are afterwards presented in the console of the IDE. Figure 6.7 displays a screenshot of the plugin in action.

[Goo11]

JsTestDriver Adapters

As already mentioned before, JsTestDriver allows to run test suites from other testing frameworks, like YUI Test, QUnit or Jasmin. Additionally, it offers adapters to work with continuous integration servers as Ruby Auto Test or Team City.

Adding an adapter that allows to execute a test suite programmed using another testing-framework is as simple as including an additional JavaScript file. With this method test suites from different frameworks can be combined. At the time of this writing the test coverage plugin did not work with the adapters.

[Goo11]

6.3.2 Selenium

Selenium is a well known and popular open-source project that allows to test the GUI of web applications. Selenium 1 interacted with the websites through JavaScript. Selenium 2 emerged when the Selenium 1 and the WebDriver project merged. It combines the advantages of both Frameworks. WebDriver takes a different approach when it comes to interactions. It passes the commands from the OS to the browser and has therefore more options to interact with the page and a more fine grained control of how the interaction takes place. The disadvantage of this approach is that Selenium 2 needs some adaption for every new browser.

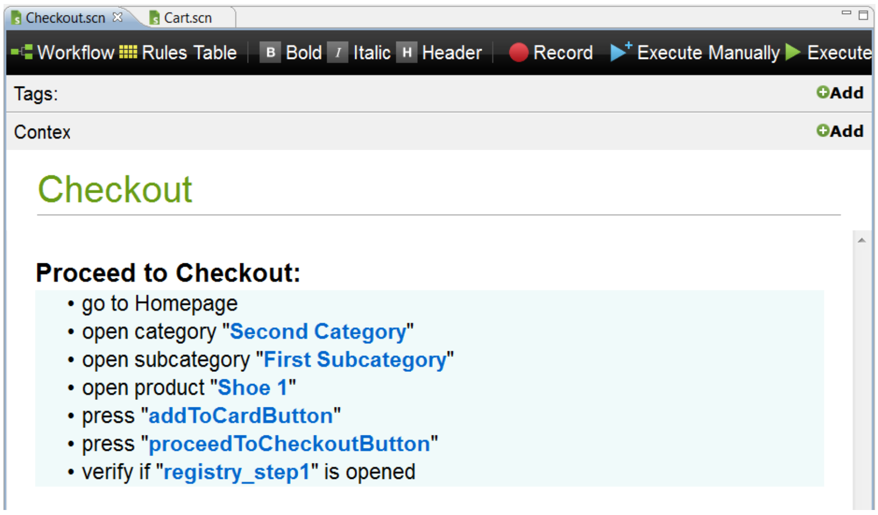
Selenium 1 did not support mobile browsers explicitly, but as it uses JavaScript to interact with the pages, at least for Android tricks exist to get it working[4]. Selenium 2 has an official support for iPhone and Android. A dedicated BlackBerry version of WebDriver is available as well. Unfortunately the iPhone driver is not in the Apple App store. Because of this, the tests can only be executed on a real device with a paid developer account. Testing on the emulator is possible with a free developer account. As Android allows to install applications from outside the Android Market, it is possible to run the tests on every real device. The Android WebDriver is officially supported by Google and ships with the Android SDK.

Selenium communicates with the native driver applications via HTTP. Therefore the device can by anywhere as long as it has network access to the hosting machine. Selenium also offers additional tools to distribute the tests to multiple machines. In combination with a headless testing framework Selenium can also make use of the computation power of cloud computing.

The tests itself can be written in many different languages. Selenium offers support for Java, C#, Python, Ruby, PHP and Perl. The framework offers functions to click on certain elements, enter text, drag & drop, navigate through the history and to manage cookies. It is open source and available under the Apache 2.0 License.

[Mac11], [Bur10], [Sel12]

[4] http://artofsolving.com/node/48



Tags:

Contex

Checkout

Proceed to Checkout:

- go to Homepage
- open category "Second Category"
- open subcategory "First Subcategory"
- open product "Shoe 1"
- press "addToCardButton"
- press "proceedToCheckoutButton"
- verify if "registry_step1" is opened

Figure 6.8: Twist Example

6.3.3 Twist

Twist is developed by Thoughtworks Studios and allows to write business requirements in plain English and execute them as tests. This should help with the communication between business user and developers. Every sentence is associated with a test function. This test functions include the actual test code. Twist supports Selenium or Sahi for programming the actual behaviour of a test function. Sahi is an alternative to Selenium, but only supports Android at the time of writing and has a smaller community then Selenium.

Twist allows to run the tests with different sets of data. With this feature it is possible to switch the input data of a test without the need to change the actual testing code behind it. Because every sentence is matched to a function, they can be reused easily. Figure 6.8 shows an example of a Twist test.

[Tho11]

6.3.4 EventRecorder

Developed by Sencha Inc., EventRecorder allows to record the interactions with a web site on a real Android device or emulator. The recordings are saved as a python script. To replay the recording the python script has to be run. It is possible to set points in the script where a screenshot should be taken. Additionally, the output of the JavaScript function `Console.log()` are logged in a file. These two outputs can be used to compare if the test passed. JavaScript can be evaluated as well, which allows to write test cases in combination with the `Console.log()` function. The EventRecorder works already very well, but is still in an early development stage.

The recorded actions are replayed independently of the state. If something goes wrong the script does not recognize this. The only possibility to detect a failure are the screenshots and the log file. Unfortunately the generated scripts only works on a device with the same resolution, because the screen coordinates are used for the user interactions. Therefore, one generated script cannot be run on multiple different devices.

[Cor11]

6.4 Stubbing and Mocking Libraries

Not all of the introduced frameworks support mocking or stubbing of JavaScript objects and functions. Fortunately there exist specific libraries to fill this gap. A lot of libraries are available like qmock, JsMockito or JSMock. All the mentioned ones only offer mocking functionality. Therefore I am going to introduce Sinon.JS, because it offers stubbing and mocking and has some additional functionality as well. Besides that, all libraries offer quite similar mocking functionality with only a different syntax.

6.4.1 Sinon.JS

Sinon.JS is a standalone library that offers test spies, stubs and mocks and can be combined with any testing-framework. The stubs used in this library are a combination of stubs and spies regarding to the definition in Chapter 4.2.1.

Spies, stubs and mocks overwrite the original function and the program under test does not notice the difference. Because another test might need the original function, the changes can be reverted by simply calling `restore()` on the spy, stub or mock. Sinon.JS offers another possibility to control the lifecycle of its objects. The tests can be wrapped within the `sinon.test()` function. All overwritten functions will be restored automatically after the `sinon.test()` function is finished. Alternatively, multiple tests can be included in a `sinon.testCase`. After all tests in such a test case are executed, the functions are restored. Examples can be seen in Listing 6.5 and Listing 6.6.

[Joh11]

```
...
  "test should retrieve element with provided id": sinon.test(function () {
    var jQuerySpy = this.spy(jQuery.fn, "init");

    jQuery.nc.fitLayoutEventHandler(this.event);

    assertEquals('#1', jQuerySpy.getCall(0).args[0]);
  }),
...
```

Listing 6.5: Sinon.JS Spy Example

```
...
  var dataMock = sinon.mock();
  dataMock.atLeast(1).withExactArgs('role');
  this.dataReturnObject.data = dataMock;
  this.jQueryStub = sinon.stub(jQuery.fn, "init");
  this.jQueryStub.withArgs(this.element).returns(this.dataReturnObject);

  jQuery.nc.creditCardField.findCreditCardFields(this.event);

  dataMock.verify();
...
```

Listing 6.6: Sinon.JS Stub and Mock Example

7 Testing Approaches

While the programming language used on the server side runs in a controlled environment, the client side JavaScript has to work in many different browsers and operating systems. Therefore, testing the client side code is a lot harder.

The server side code was always number one priority for testing, as a website with a non working server side code is broken. If JavaScript fails, most conventional websites are still usable. JavaScript fails silently and recovers after an error as soon as a new event occurs. Modern web applications depend heavily on the JavaScript part and do not work properly with faulty code anymore. Therefore, server and client side code should be equally handled when it comes to testing.

JavaScript is a lot different to most common server-side programming languages. It allows to add, overwrite or remove object properties on the fly and functions can be passed as values. It is possible to access the objects methods and properties using different syntaxes and uninitialized variables or properties are not `null`, they are `undefined`. JavaScript is also event driven. It gets executed as soon as the page is loaded and afterwards it can react on events, like clicks, swipes, network changes, etc. Additionally, native browser objects, like the history or the location object, are available. Depending on the browser it is sometimes not possible to overwrite (e.g. stub or mock) them. The native objects differ from browser to browser and specific features are not supported on all of them. To test for example fallbacks, the test has to be executed on a browser that does not support the feature or a browser that allows to overwrite native objects in order to remove the feature support.

This chapter describes how the HTML5 features itself can be tested and how the introduced testing frameworks can be used for TDD, continuous integration or GUI testing. The summary includes a final evaluation figure of how the testing-frameworks work with jQuery Mobile and Sencha Touch. The other frameworks described in Chapter 6 are similar to either jQuery Mobile or Sencha Touch. Therefore, the presented methods can be applied to the other frameworks as well.

[Joh11], [FR11], [Eug10]

7.1 Testing the HTML5 Features

7.1.1 Feature Detection

As described in Chapter 3, not every mobile browser supports all of the new HTML5 features. If a feature is not supported, a fallback should be implemented or at least the user should be notified. Proper feature detection is not easy, because some browsers have a faulty or incomplete implementation of a feature. For example, Android supports since version 2.1 the audio tag, but does not have any supported codecs. Because of this, libraries like Modernizr (see following section), help detecting the support of certain features, like video, audio, storage, CSS3 transformations and more. Libraries like Modernizr will remain important as long as the HTML5 specification is still under development and not all mobile devices are supporting an equal level of HTML5 and CSS3 features. Only Modernizr is presented in this document as at the time of this writing no other feature detection library existed that focused on HTML5 and CSS3.

Modernizr

Modernizr is meant to detect the browser support for the new HTML5 and CSS3 features. This library can be used in order to detect if the current browser is able to support the needed features and provide fallback if not. It supports a broad range of desktop and mobile browsers and offers an easy to use API for feature detection. It consists of a small JavaScript file which has to be included on the website. The library can be configured to include code only needed to detected certain features. This keeps the filesize small. A code example can be seen in Listing 7.1. [FR11]

```
if (Modernizr.geolocation) {...}
if (Modernizr.audio.ogg) {...}
```

Listing 7.1: Feature Detection with Modernizr

7.1.2 Offline Mode

The Application Cache offers an API that states if the cache is working correctly and which state it is in, but it cannot tell if all needed resources are cached. Additionally, Ajax requests targeted to not cached resources will fail. Therefore, all asynchronous requests have to be tested on the unit-level for proper error handling. All other unit-tests will not behave differently as in online mode, as no additional connections are needed to execute them.

The application needs a defined behaviour if the network connection is lost. In order to test if the web application works as expected without a network connection, GUI tests have to be used. Depending on how the behaviour differs from the online case, the same tests might be used or special test cases have to be added. With tools like Selenium, the browser only needs to have access to the Selenium server, but not to the server delivering the resources. The GUI test suite can then be run on the resources from the Application Cache.

7.1.3 Storage

Testing the client side storages, like web storage or web SQL, is straight forward. The native objects can be mocked and therefore the correct usage be verified. Integration tests can

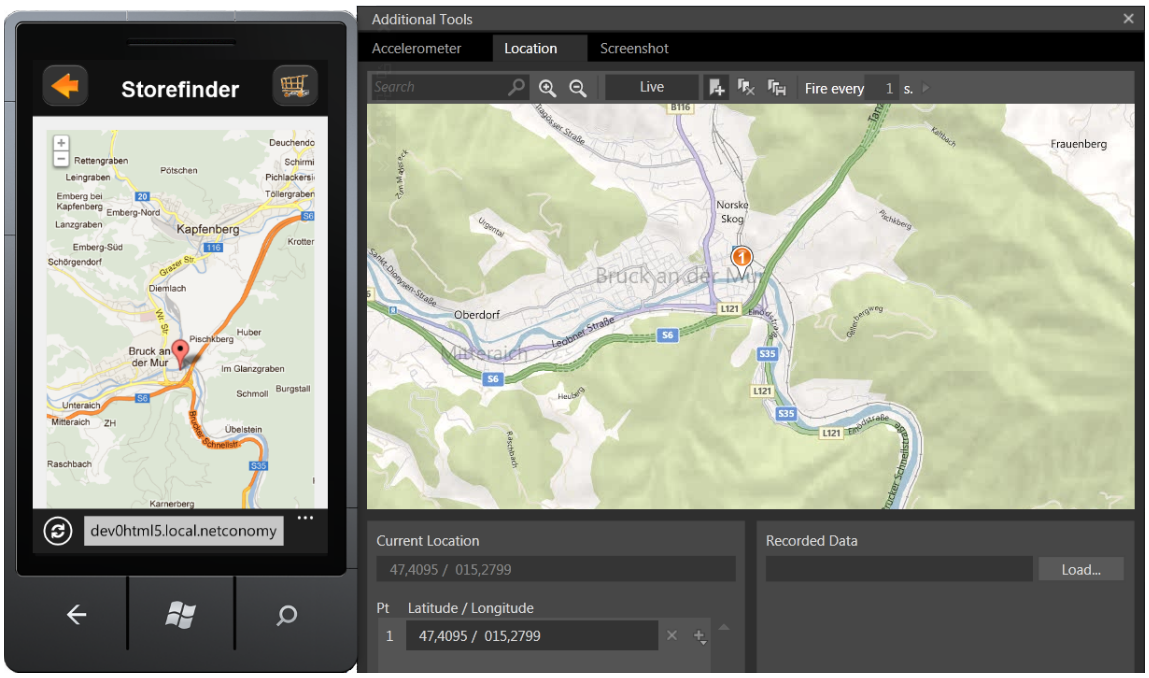


Figure 7.1: Windows Phone 7 Location Mocking

test the real storage on the supported browser in order to detect faulty implementations or missing feature detection. Additionally, the fallbacks need to be tested with integration tests, too. Therefore, a browser that does not support the feature should be used or the native storage objects should be stubbed.

7.1.4 Geolocation

Integration testing of the Geolocation is difficult. The location API is not accessible on the Android, BlackBerry nor the iPhone emulator. Only native apps can access the geo location on the emulators. The Windows Phone emulator is the only one allowing to set the browser location. The location on real devices is not fixed and therefore cannot be used for repeatable automated testing. Android allows to fake locations on real devices and emulators for native applications. As Selenium accesses the web application through its own app, it can make use of faked locations. The iPhone emulator provides a fixed location for native applications which cannot be changed. The only other possibility is trying to fake the Geolocation object within JavaScript for testing purposes, which might not be possible on all devices. See Figure 7.1 how the location can be set for the Windows Phone emulator.

7.1.5 Motion Sensors

Unfortunately the current version of the iPhone simulator does not allow to simulate any movements. The emulator can only be shaken and rotated left or right. The Android emulator

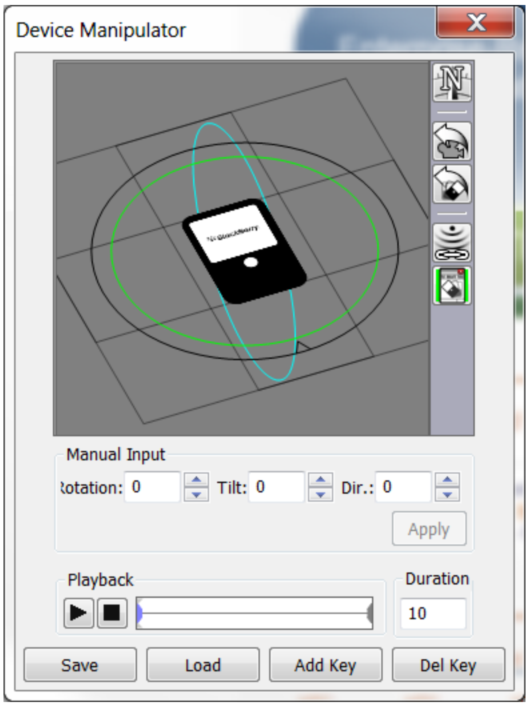


Figure 7.2: Blackberry Accelerometer Simulation

is even more limited and allows no movement at all. For both emulators exist custom programs (iPhone[1], Android[2]) that allow to inject sensor data into native applications. Unfortunately, the applications have to include some extra code to support it.

The Blackberry and the Windows Phone emulator both allow to set the sensor data and even to replay recorded data. See Figure 7.2 and 7.3 for emulator screenshots.

7.1.6 Network

The Android emulator allows to control the network speed (type) and the latency. The BlackBerry emulator provides options to set the network type, but the option does not affect the network speed. All other emulators do not allow setting any network options. Therefore, the Network Information API (see page 19) can only be tested on the Android emulator. Running tools, that allow to control the network speed and latency on the hosting machine, affect the emulators and allow for example testing of timeouts.

[1] http://code.google.com/p/accelerometer-simulator
[2] http://code.google.com/p/openintents/wiki/SensorSimulator

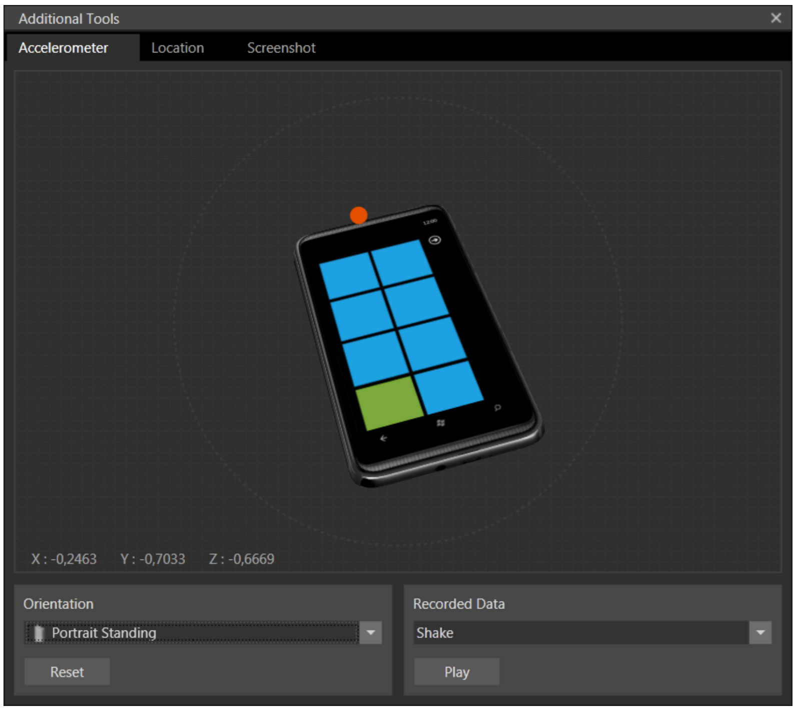


Figure 7.3: Windows Phone 7 Accelerometer Simulation

7.2 Continuous Integration and Test Driven Development

Continuous Integration and Test Driven Development require the same test qualities. They have to give quick and exact feedback. Because they are run very frequently, developers want to get quick results. In order to get an exact description of an error, unit-tests are the best choice. Every unit-test only tests a small bit of functionality, allowing to narrow down defect code. As described in Chapter 4.2 at least two different test suites should exist. One that runs fast and covers the most important tests and one that perhaps takes a longer time, but covers more aspects. Additionally fast-running integration tests should be run frequently. Depending on the time it takes to complete them, they should be executed at least before every commit, to ensure a working code base for the other team members.

Because of the good performance of headless testing frameworks they can be used for test driven development. But as the tests are not run in a real environment, at least some cross-browser integration tests are needed to ensure the code works within the targeted browsers [Mac11]. In order to ensure the tests pass also in a real environment, in-browser tests or test drivers are recommendable. If performance is not an issue, the test suite should be executed using a test driver in at least the main targeted browsers. This allows to catch device dependent errors in an early phase. Functions delaying the test executions like timers or AJAX requests should be stubbed, if not relevant for the test.

7.2.1 jQuery Mobile

Testing of jQuery Mobile like applications is quite easy. A jQuery Mobile project mainly consists only of JavaScript code written by the developers. For unit testing the jQuery functions should be stubbed or mocked and the rest of the code can be tested using one of the testing-frameworks from Chapter 6. Best practise would be to use a test driver and run the test suite frequently on the main targeted platforms.

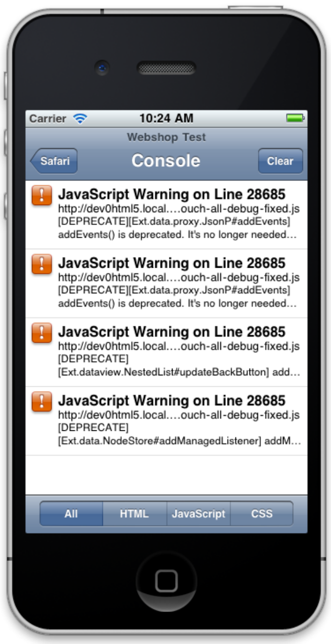
7.2.2 Sencha Touch

TDD on Sencha Touch like frameworks is hard. Their applications have a lot of JavaScript code overhead. Class extending, defining and instantiation needs additional code and is typically distributed over many files. The frameworks do not offer any specific testing tools and handle the object instantiation themselves. The objects created by the frameworks are hard to test, because of their complex structure. Additionally, the structure might change if the version of the framework is upgraded. Therefore, it seems to be best to have the business logic in its own files and custom objects and only use them from within the framework. This way the logic is easy to test with TDD and integration tests handle the rest of the framework. The only tool Sencha offers for testing is the EventRecorder(Chapter 6.3.4). One can create small GUI integration tests and replay them with a script in order to verify the functionality of the application. Unfortunately those tests are, comparing to unit-tests, slow and time-consuming to create. These tests might not be appropriate for TDD, but can be run on a continuous integration server and provide feedback after a commit.

7.3 Testing Nightly Builds

Nightly Builds do not need to have a quick feedback and can therefore execute more complex test-suites. This includes of course the same test-suite the developers are running frequently plus additional regression tests and GUI tests. As time is not a big issue, the test-suite should be run on as many different browsers and devices as possible. Test Drivers are the first choice. The integration tests should stub or mock as little as possible.

All introduced test drivers (except Twist, which is only a wrapper for a test driver) can be controlled via command line and can be run automated by a build-server. The browsers which should be tested, have to be connected to JsTestDriver only once and Selenium needs a network connection to the devices and starts the test automated. The EventRecorder needs a USB connection to the Android devices. Problems might occur if more test drivers are used on the same device. This might cause problems, as JsTestDriver needs a running browser and the other ones are starting their own apps. For example EventRecorder is not closing its app and the device might close the browser if it needs more resources. It is not possible for JsTestDriver then to test on this devices. Additional custom scripts, that verify that the browser is opened and are able to start the browser with the correct address if necessary, can solve this issue.



```
JavaScript Warning on Line 28685
http://dev0html5.local....ouch-all-debug-fixed.js
[DEPRECATE][Ext.data.proxy.JsonP#addEvents]
addEvents() is deprecated. It's no longer needed...

JavaScript Warning on Line 28685
http://dev0html5.local....ouch-all-debug-fixed.js
[DEPRECATE][Ext.data.proxy.JsonP#addEvents]
addEvents() is deprecated. It's no longer needed...

JavaScript Warning on Line 28685
http://dev0html5.local....ouch-all-debug-fixed.js
[DEPRECATE]
[Ext.dataview.NestedList#updateBackButton] add...

JavaScript Warning on Line 28685
http://dev0html5.local....ouch-all-debug-fixed.js
[DEPRECATE]
[Ext.data.NodeStore#addManagedListener] addM...
```

Figure 7.4: Mobile Safaris Console on an iPhone

The results of the tests should be collected, combined and evaluated. They should be accessible to monitor progress of development and failed test need to be reported to the developers automatically.

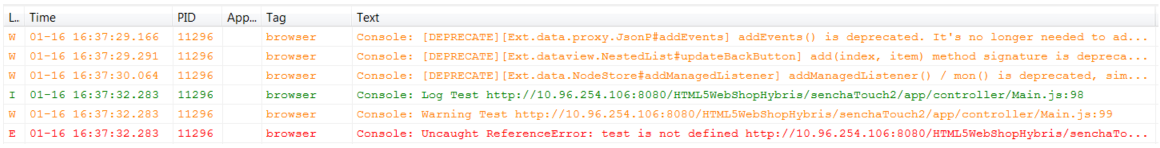
7.4 Debugging

The desktop browsers offer developer tools which allow to debug the JavaScript, CSS and web requests. Because of the limited screen size this would not be the best solution on mobile browsers. Some mobile devices offer limited debugging directly on the phone, others via network on a desktop application and others even do not support any debugging of the mobile browser at all.

7.4.1 In Browser Debugger

The mobile Safari on iOS devices allow to activate a console which displays HTML, JavaScript and CSS errors. JavaScript can log messages to the console with the use of the `console` object. The amount of information in the console is displayed directly under the address bar and if tapped, the console is opened in fullscreen including the whole message text. Figure 7.4 shows the fullscreen view of the console.

Android does not have such a console within the browser, but all errors are logged in the logcat application. Logcat is the default Android logging system and shows debug information



```
L.. Time                PID    App... Tag      Text
W   01-16 16:37:29.166  11296         browser  Console: [DEPRECATE][Ext.data.proxy.JsonP#addEvents] addEvents() is deprecated. It's no longer needed to ad...
W   01-16 16:37:29.291  11296         browser  Console: [DEPRECATE][Ext.dataview.NestedList#updateBackButton] add(index, item) method signature is depreca...
W   01-16 16:37:30.064  11296         browser  Console: [DEPRECATE][Ext.data.NodeStore#addManagedListener] addManagedListener() / mon() is deprecated, sim...
I   01-16 16:37:32.283  11296         browser  Console: Log Test http://10.96.254.106:8080/HTML5WebShopHybris/senchaTouch2/app/controller/Main.js:98
W   01-16 16:37:32.283  11296         browser  Console: Warning Test http://10.96.254.106:8080/HTML5WebShopHybris/senchaTouch2/app/controller/Main.js:99
E   01-16 16:37:32.283  11296         browser  Console: Uncaught ReferenceError: test is not defined http://10.96.254.106:8080/HTML5WebShopHybris/senchaTo...
```

Figure 7.5: Android logcat Debugging

from all applications including the browser. Like on iOS a `console` object is available within JavaScript for debugging and logging purposes. The log can be accessed via the Android debug bridge for any connected device. The output of the log can be filtered, for example, to display only the browser messages as shown in Figure 7.5.

The mobile Opera allows to connect via network to the Opera Dragonfly debugging toolbar of its desktop equivalent. Figure 7.6 and 7.7 display how this combination looks like. Additionally, to the console similar to Android and iOS, Dragonfly allows to access the DOM, see Ajax requests and debug the JavaScript.

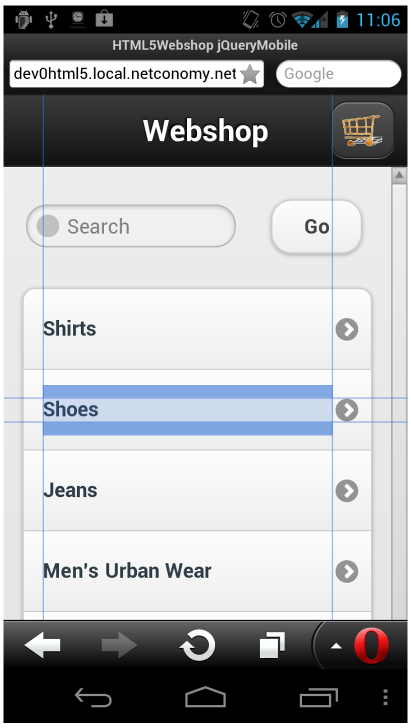
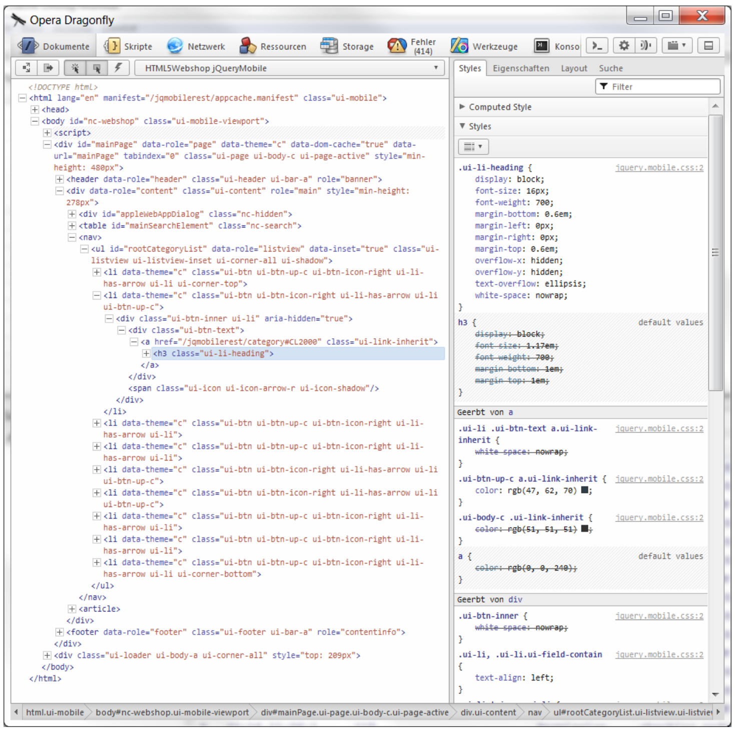


Figure 7.6: Opera Mobile



```
<!DOCTYPE html>
<html lang="en" manifest="/jqmobilerest/appcache.manifest" class="ui-mobile">
  <head>
  <body id="nc-webshop" class="ui-mobile-viewport">
    <script>
    <div id="mainPage" data-role="page" data-theme="c" data-dom-cache="true" data-
    url="mainPage" tabindex="0" class="ui-page ui-body-c ui-page-active" style="min-
    height: 480px">
      <header data-role="header" class="ui-header ui-bar-a" role="banner">
      <div data-role="content" class="ui-content" role="main" style="min-height:
      278px">
        <div id="appleWebAppDialog" class="nc-hidden">
        <table id="mainSearchElement" class="nc-search">
        <nav>
          <ul id="rootCategoryList" data-role="listview" data-inset="true" class="ui-
          listview ui-listview-inset ui-corner-all ui-shadow">
            <li data-theme="c" class="ui-btn ui-btn-up-c ui-btn-icon-right ui-li-
            has-arrow ui-li ui-corner-top">
            <li data-theme="c" class="ui-btn ui-btn-icon-right ui-li-has-arrow ui-li
            ui-btn-up-c">
              <div class="ui-btn-inner ui-li" aria-hidden="true">
                <div class="ui-btn-text">
                  <a href="/jqmobilerest/category#CL2000" class="ui-link-inherit">
                    <h3 class="ui-li-heading">
                  </a>
                </div>
                <span class="ui-icon ui-icon-arrow-r ui-icon-shadow"/>
              </div>
            </li>
            <li data-theme="c" class="ui-btn ui-btn-up-c ui-btn-icon-right ui-li-
            has-arrow ui-li">
            <li data-theme="c" class="ui-btn ui-btn-up-c ui-btn-icon-right ui-li-
            has-arrow ui-li">
            <li data-theme="c" class="ui-btn ui-btn-icon-right ui-li-has-arrow ui-li
            ui-btn-up-c">
            <li data-theme="c" class="ui-btn ui-btn-icon-right ui-li-has-arrow ui-li
            ui-btn-up-c">
            <li data-theme="c" class="ui-btn ui-btn-up-c ui-btn-icon-right ui-li-
            has-arrow ui-li">
            <li data-theme="c" class="ui-btn ui-btn-up-c ui-btn-icon-right ui-li-
            has-arrow ui-li">
            <li data-theme="c" class="ui-btn ui-btn-up-c ui-btn-icon-right ui-li-
            has-arrow ui-li ui-corner-bottom">
          </ul>
        </nav>
        <article>
      </div>
      <footer data-role="footer" class="ui-footer ui-bar-a" role="contentinfo">
    </div>
    <div class="ui-loader ui-body-a ui-corner-all" style="top: 209px">
  </body>
</html>
```

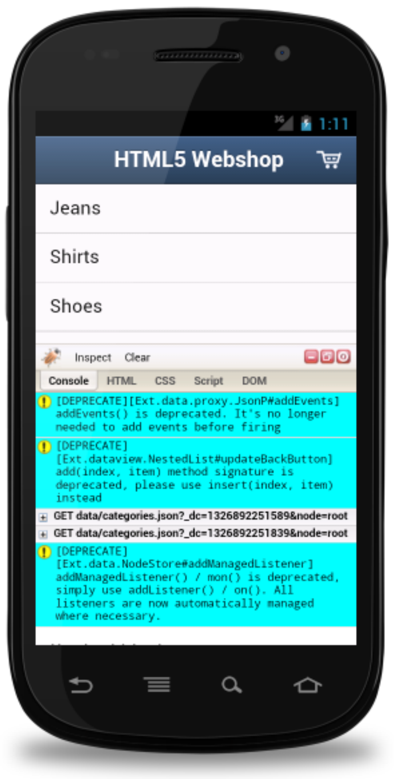
```
html.ui-mobile > body#nc-webshop.ui-mobile-viewport > div#mainPage.ui-page.ui-body-c.ui-page-active > div.ui-content > nav > ul#rootCategoryList.ui-listview.ui-listvie
```

```
.ui-li-heading {                          jquery.mobile.css:2
    display: block;
    font-size: 16px;
    font-weight: 700;
    margin-bottom: 0.6em;
    margin-left: 0px;
    margin-right: 0px;
    margin-top: 0.6em;
    overflow-x: hidden;
    overflow-y: hidden;
    text-overflow: ellipsis;
    white-space: nowrap;
}
h3 {                                         default values
    ~~display: block;~~
    ~~font-size: 1.17em;~~
    ~~font-weight: 700;~~
    ~~margin-bottom: 1em;~~
    ~~margin-top: 1em;~~
}
Geerbt von a
.ui-li .ui-btn-text a.ui-link-            jquery.mobile.css:2
inherit {
    ~~white-space: nowrap;~~
}
.ui-btn-up-c a.ui-link-inherit {          jquery.mobile.css:2
    color: rgb(47, 62, 70) ■;
}
.ui-body-c .ui-link-inherit {             jquery.mobile.css:2
    ~~color: rgb(51, 51, 51) ■;~~
}
a {                                          default values
    ~~color: rgb(0, 0, 240);~~
}
Geerbt von div
.ui-btn-inner {                           jquery.mobile.css:2
    ~~white-space: nowrap;~~
}
.ui-li, .ui-li.ui-field-contain           jquery.mobile.css:2
{
    text-align: left;
}
```

Figure 7.7: Opera Dragonfly

Blackberry is offering an Eclipse and Visual Studio plugin for its devices, which provide similar functionality like Dragonfly. Unfortunately no debugger is available on Windows Phone 7 at the time of this writing.

[Fir10]



```
[DEPRECATE][Ext.data.proxy.JsonP#addEvents]
addEvents() is deprecated. It's no longer
needed to add events before firing
[DEPRECATE]
[Ext.dataview.NestedList#updateBackButton]
add(index, item) method signature is
deprecated, please use insert(index, item)
instead
GET data/categories.json?_dc=1326892251589&node=root
GET data/categories.json?_dc=1326892251839&node=root
[DEPRECATE]
[Ext.data.NodeStore#addManagedListener]
addManagedListener() / mon() is deprecated,
simply use addListener() / on(). All
listeners are now automatically managed
where necessary.
```

Figure 7.8: Firebug Lite

7.4.2 JavaScript Based Debugger

This type of debuggers only need a JavaScript file added to the page and allow to debug directly on the device or via Ajax on the desktop. They are not as powerful as for example Opera Dragonfly, because of the limitations of the JavaScript language.

Firebug Lite

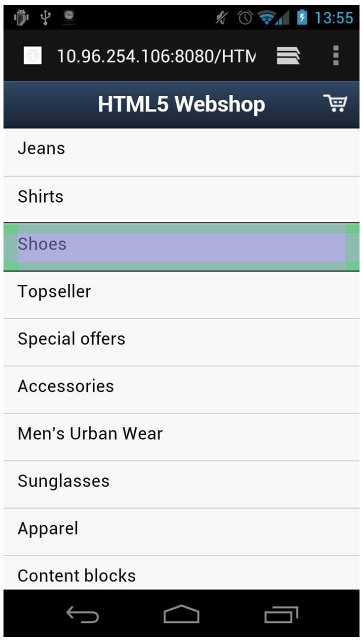
Firebug Lite has the look and feel from the popular desktop debugger Firebug, but does not need any plugins at the browser. It runs directly within the mobile browsers window. Firebug Lite offers a console that displays all warnings and errors and allows to execute custom JavaScript code. The tools include an inspector that displays the HTML code and CSS styles applied to an element. The CSS styles can be edited in real-time. Because it is programmed in JavaScript, it is not possible to debug the websites JavaScript code. Unfortunately it is designed for desktop usage and is not optimised for mobile devices and their small screen. Figure 7.8 shows Firebug Lite on an Android device.

[Moz11], [Fir10]

Weinre

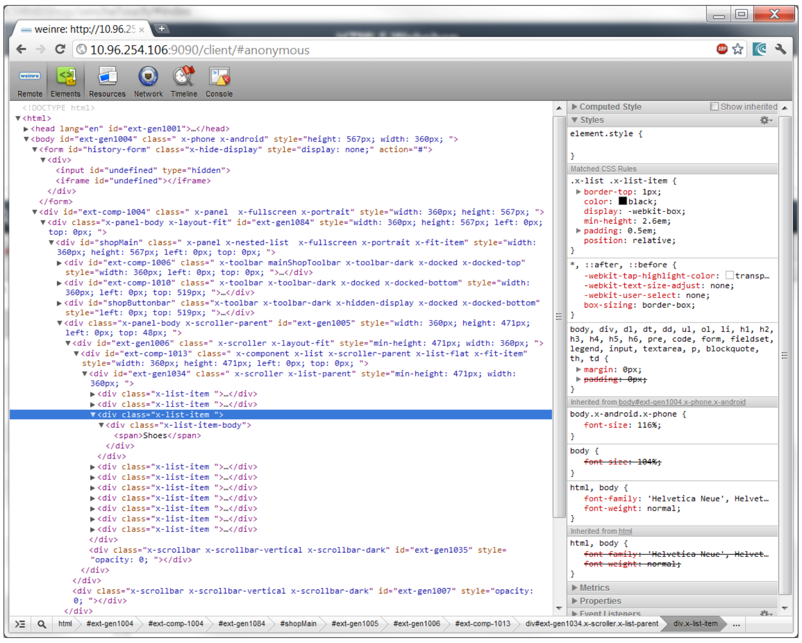
Weinre is an open-source tool with a similar target as Firebug Lite, but it does not display the debug view on the device itself, instead it works via Ajax and allows to browse the DOM and CSS on the desktop. The look and feel is quite similar to Firebug Lite and it offers nearly the same functionality, but also has the limitation when it comes to JavaScript debugging. It allows to view the JavaScript source code and to execute custom code.

In order to use Weinre, the Weinre server has to be started on the desktop. The server serves a JavaScript file, that has to be included in the page to enable debugging. This way, a connection is automatically created between the mobile browser and the Weinre server. Figure 7.9 and 7.10 shows Weinre in action.



Jeans
Shirts
Shoes
Topseller
Special offers
Accessories
Men's Urban Wear
Sunglasses
Apparel
Content blocks

Figure 7.9: Weinre on the Mobile Device



```
<!DOCTYPE html>
▼<html>
 ▶<head lang="en" id="ext-gen1001">…</head>
 ▼<body id="ext-gen1004" class=" x-phone x-android" style="height: 567px; width: 360px; ">
  ▼<form id="history-form" class="x-hide-display" style="display: none;" action="#">
   ▼<div>
      <input id="undefined" type="hidden">
      <iframe id="undefined"></iframe>
    </div>
   </form>
  ▼<div id="ext-comp-1004" class=" x-panel  x-fullscreen x-portrait" style="width: 360px; height: 567px; ">
   ▼<div class="x-panel-body x-layout-fit" id="ext-gen1084" style="width: 360px; height: 567px; left: 0px;
     top: 0px; ">
    ▼<div id="shopMain" class=" x-panel x-nested-list  x-fullscreen x-portrait x-fit-item" style="width:
      360px; height: 567px; left: 0px; top: 0px; ">
     ▶<div id="ext-comp-1006" class=" x-toolbar mainShopToolbar x-toolbar-dark x-docked x-docked-top"
       style="width: 360px; left: 0px; top: 0px; ">…</div>
     ▶<div id="ext-comp-1010" class=" x-toolbar x-toolbar-dark x-docked x-docked-bottom" style="width:
       360px; left: 0px; top: 519px; ">…</div>
     ▶<div id="shopButtonbar" class="x-toolbar x-toolbar-dark x-hidden-display x-docked x-docked-bottom"
       style="left: 0px; top: 519px; ">…</div>
     ▼<div class="x-panel-body x-scroller-parent" id="ext-gen1005" style="width: 360px; height: 471px;
       left: 0px; top: 48px; ">
      ▼<div id="ext-gen1006" class=" x-scroller x-layout-fit" style="min-height: 471px; width: 360px; ">
       ▼<div id="ext-comp-1013" class=" x-component x-list x-scroller-parent x-list-flat x-fit-item"
         style="width: 360px; height: 471px; left: 0px; top: 0px; ">
        ▼<div id="ext-gen1034" class=" x-scroller x-list-parent" style="min-height: 471px; width:
          360px; ">
         ▶<div class="x-list-item ">…</div>
         ▶<div class="x-list-item ">…</div>
         ▼<div class="x-list-item ">
          ▼<div class="x-list-item-body">
             <span>Shoes</span>
           </div>
          </div>
         ▶<div class="x-list-item ">…</div>
         ▶<div class="x-list-item ">…</div>
         ▶<div class="x-list-item ">…</div>
         ▶<div class="x-list-item ">…</div>
         ▶<div class="x-list-item ">…</div>
         ▶<div class="x-list-item ">…</div>
         ▶<div class="x-list-item ">…</div>
         </div>
         <div class="x-scrollbar x-scrollbar-vertical x-scrollbar-dark" id="ext-gen1035" style=
         "opacity: 0; "></div>
        </div>
       </div>
       <div class="x-scrollbar x-scrollbar-vertical x-scrollbar-dark" id="ext-gen1007" style="opacity:
       0; "></div>
      </div>
```

```
html  #ext-gen1004  #ext-comp-1004  #ext-gen1084  #shopMain  #ext-gen1005  #ext-gen1006  #ext-comp-1013  div#ext-gen1034.x-scroller.x-list-parent  div.x-list-item  ...
```

```
element.style {

}
Matched CSS Rules
.x-list .x-list-item {
 ▶ border-top: 1px;
   color: ■black;
   display: -webkit-box;
   min-height: 2.6em;
 ▶ padding: 0.5em;
   position: relative;
}
*, ::after, ::before {
   -webkit-tap-highlight-color: □transp…
   -webkit-text-size-adjust: none;
   -webkit-user-select: none;
   box-sizing: border-box;
}
body, div, dl, dt, dd, ul, ol, li, h1, h2,
h3, h4, h5, h6, pre, code, form, fieldset,
legend, input, textarea, p, blockquote,
th, td {
 ▶ margin: 0px;
 ▶ ~~padding: 0px;~~
}
Inherited from body#ext-gen1004.x-phone.x-android
body.x-android.x-phone {
   font-size: 116%;
}
body {
   ~~font-size: 104%;~~
}
html, body {
   font-family: 'Helvetica Neue', Helvet…
   font-weight: normal;
}
Inherited from html
html, body {
   ~~font-family: 'Helvetica Neue', Helvet…~~
   ~~font-weight: normal;~~
}
```

Figure 7.10: Weinre Inspector with a Selected Element Viewed on a Desktop Browser

[GG11]

7.4.3 RemoteJS

Sencha Labs developed a remote debugging tool for Android using the Android Debug Bridge (adb). A custom app is installed on the device displaying only a `webview`. Two versions exist from the desktop client, a GUI version and a python version. The GUI version displays a console within a desktop browser allowing to see logged messages and to execute custom JavaScript code. This tool overwrites the Android console object in JavaScript and provides its own. The python version has the same capabilities as the GUI version, but additionally allows to automate the scripting via python. Sencha Labs is planning to release an iPhone version in the future. [Cor10]

7.5 GUI Testing

Testing the GUI manually or automated is quite similar to testing desktop websites. Tools like Selenium for example work on desktop websites and offer plugins to work on mobile devices, too.

In Chapter 6.3 three automated GUI testing tools were introduced. As Twist uses Selenium to drive the browser it has the same capabilities, only displays the tests in an easy to understand way. They work at the time of writing only on Android and iOS devices. The third tool, EventRecorder from Sencha Labs is a lot simpler, less powerful and only works on Android right now. An advantage of the EventRecorder is that test cases can be created very quickly, but they only work on the device they are created on.

Manual testing on real physical devices can become very expensive as some top-end smartphones costs more than 600€. Testing on emulators is cheaper, but some of them offer a poor performance and cannot emulate the real devices 100%. The following chapter describes a trade-off between testing on emulators and real devices.

7.5.1 Remote Labs

Remote Labs allow to access real physical devices over the internet. This allows to test the mobile web application on many devices without the need of purchasing them. Most remote labs allow to capture a screenshot or a video during the test. Depending on the internet connection and the remote lab, the feedback from control events might be delayed.

Nokia Remote Device Access (RDA)

At the time of writing, Nokias remote lab only offers Symbian and Maemo based phones. The service is free of charge, offers only a few devices and the time how long one user can access a device is limited. Registered Nokia developers have access to this lab and get 8 hours per day to access the devices. The software allows to make screenshots, change the screen orientation and to set the location data for the device. Unfortunately no automation of tests is available. [Fir10], [Nok12]

Samsung Lab.Dev

This is Samsungs equivalent to Nokias RDA. The service is free of charge for user with a Samsung Mobile Innovator user account. Every user gets 10 credits per day, which are valid for 2.5h of using a device. Premium members can get more credits and have a broader range of devices. At the time of writing only a limited amount of Android phones and tablets

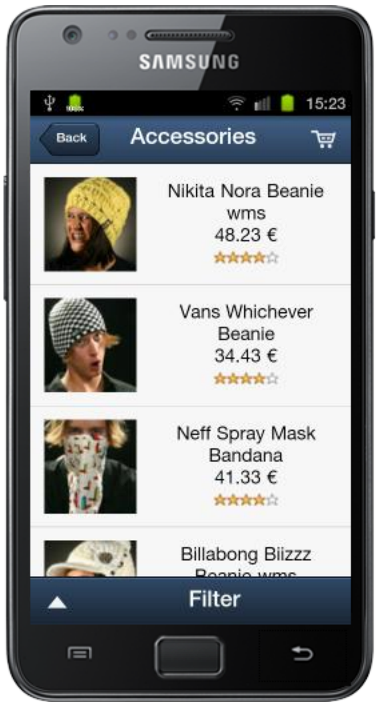


Figure 7.11: Samsung Lab.Dev

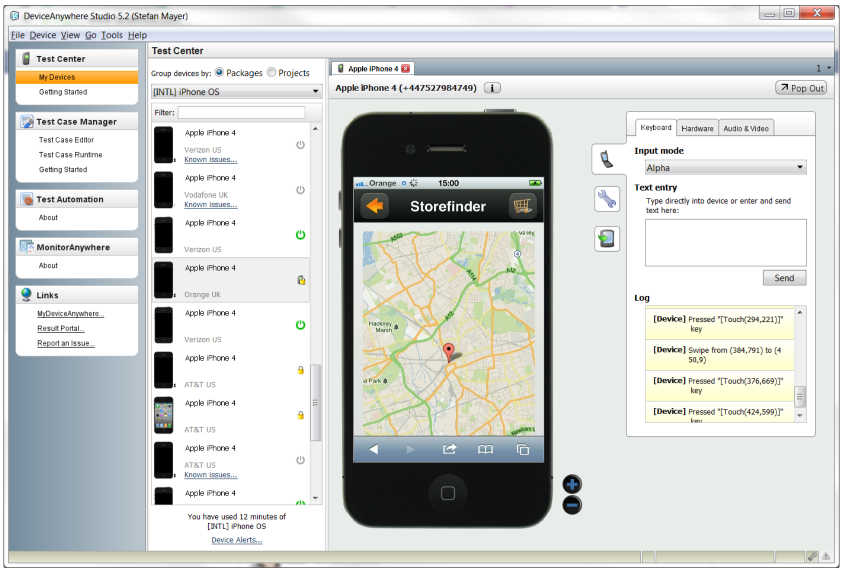
were available. Additionally, the data on the phones is not deleted after logging out and is available to the next user. Figure 7.11 shows a Samsung Galaxy S II accessed via Lab.Dev. It is possible to transfer files to the device and to take screenshots or record a video. [Fir10], [Sam12]

DeviceAnywhere

DeviceAnywhere is a commercial service to access real mobile devices from different vendors and with different operating systems. The service offers much more functionality than the free services from Samsung and Nokia. It supports to capture videos or screenshots and also plays back the audio of the device. Additionally, a proxy records and displays all network requests and offers a DOM inspector. The software allows to use and control hardware components like the accelerometer, compass or the camera of the device. It is also possible to connect Android devices to the Android Debug Bridge (adb) as if they were local. This allows to use Androids logcat or other programs requiring adb.

Besides manual testing, the test center software allows to create automated test cases. Testers can add instructions and the types of proof for the test case. The created test cases can be scheduled and run on multiple devices at the same time. Afterwards the results, including screenshots and videos, are stored centrally and can be accessed by the whole team. A screenshot of the software can be seen in Figure 7.12.

[Fir10], [Key11]



```
[Device] Pressed "[Touch(294,221)]" key
[Device] Swipe from (384,791) to (4 50,9)
[Device] Pressed "[Touch(376,669)]" key
[Device] Pressed "[Touch(424,599)]" key
```

Figure 7.12: DeviceAnywhere

Perfecto Mobile

Perfecto Mobile is a direct competitor to DeviceAnywhere. It offers the same services, like accessing and controlling real smartphones via the internet. Like with DeviceAnywhere it is possible to share the live session with other user or to record it. Tests can be automated using scripts, image recognition and OCR. This allows to tell the script to wait for a certain image or text to appear. The software allows to write custom commands and reuse them in other tests.

An advantage of Perfecto Mobile over DeviceAnywhere is, that it runs completely in the browser and does not require any additional software. Also a unique feature is Website Validation. It runs a single URL on multiple devices and takes screenshots on all of them. Afterwards the screenshots are displayed side by side on one page for comparison. Contrary to its main competitor, Perfecto Mobile does not allow access or control of any of the hardware components and does not offer a proxy to access the DOM on the device. Figure 7.13 shows a screenshot of the software.

[Fir10], [Per11]

7.6 Summary

There is no tool that can handle all kinds of the different test types on mobile devices right now, but with a combination of different tools one can cover the different test cases at least on Android and iOS. Blackberry and Windows Phone 7 is not yet a top priority for the vendors of testing tools, but this might change in the future. The biggest problem is the

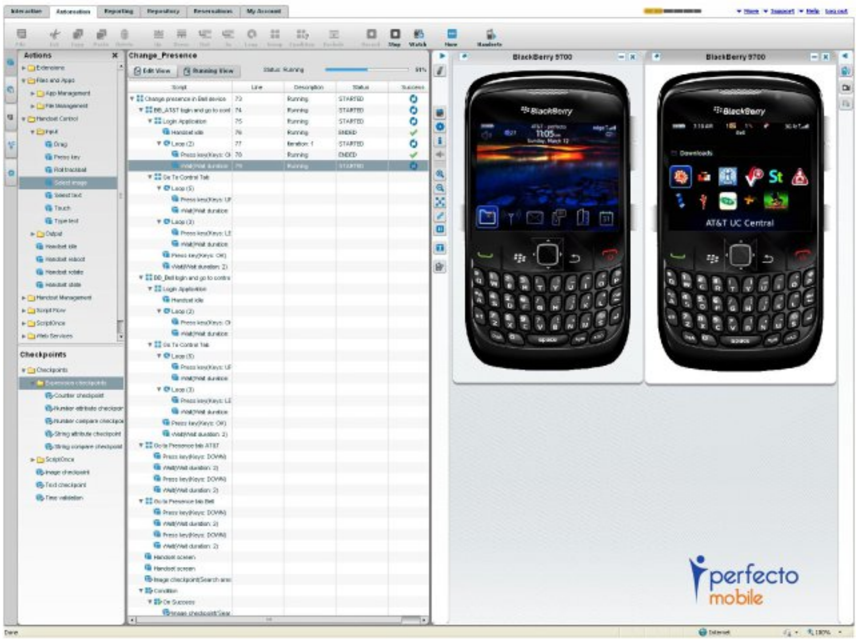


Figure 7.13: Perfecto Mobile [Per11]

missing debugger on Windows Phone 7. If a JavaScript error occurs uniquely on Windows Phone devices, it is hard to find the cause of the problem.

Nearly all of the introduced test drivers work on both Android and iOS devices. This allows to run unit, regression and GUI tests on multiple of this devices at the same time. Test drivers can be run manually or integrated into automated scripts for continuous integration and nightly build tests.

Headless testing frameworks do have a performance advantage over the other frameworks, but cannot emulate a real browser on a physical mobile device 100%. Additionally, they cannot keep up with the fast release of new versions of the browsers and may lag behind. Simulators do have the same problem and some of them are quite slow. A trade-off are the presented remote labs. They allow to test remotely on real physical devices. Test can be automated and run parallel on many devices for a quick feedback.

Table 7.1 gives an overview how the testing frameworks work together with either jQuery Mobile or Sencha Touch. This table recommends which testing framework should be chosen for a specific testing type and a used mobile web application framework. A value of 1 means this testing framework is not a good choice for this case and 5 for a perfect match. The different test cases are TDD, CI, nightly builds and GUI testing. TDD has priority on a fast feedback and easy and fast test creation, while CI on the other hand needs the ability to call the tests via script and compare the results programmatically. Nightly builds need frameworks that allow to run the test suites on real devices via command line, while quick feedback is not an issue. GUI testing also needs to be done on real devices or emulators, while real environments are preferred. Priorities at GUI testing are automation, repeatability and automated test validation.

Testing Method	Testing-Framework	jQuery Mobile	Sencha Touch
TDD	QUnit	5	4
	YUI Test	4	3
	Jasmin	5	4
	Siesta	4	4
	Envjs and Rhino	4	4
	Zombie.js and Node.js	4	4
	PhantomJS	4	4
	JsTestDriver	5	4
	Selenium	2	2
	Twist	2	2
	EventRecorder	4	4
CI	QUnit	5	4
	YUI Test	4	3
	Jasmin	5	4
	Siesta	5	5
	Envjs and Rhino	5	5
	Zombie.js and Node.js	5	5
	PhantomJS	5	5
	JsTestDriver	5	5
	Selenium	4	4
	Twist	4	4
	EventRecorder	2	2
Nightly Builds	QUnit	5	4
	YUI Test	5	4
	Jasmin	5	4
	Siesta	5	5
	Envjs and Rhino	2	2
	Zombie.js and Node.js	2	2
	PhantomJS	2	2
	JsTestDriver	5	5
	Selenium	5	5
	Twist	5	5
	EventRecorder	1	1
GUI Tests	QUnit	1	1
	YUI Test	1	1
	Jasmin	1	1
	Siesta	4	4
	Envjs and Rhino	1	1
	Zombie.js and Node.js	1	1
	PhantomJS	1	1
	JsTestDriver	2	2
	Selenium	4	4
	Twist	5	5
	EventRecorder	3	3

Table 7.1: Testing Framework Comparison

8 Conclusion

The HTML5 and CSS3 standards are still a subject to change. During the development of the specifications, elements were removed, new ones were added and some even got their own specification. At the time of writing the target date for the W3C recommendation of HTML5 is 2014 [W3C11n]. Nevertheless many of the new features are already implemented in the biggest part of the mobile browsers. Android, Blackberry and iOS support the new features already well and allow the development of native alike mobile web applications.

At the end of 2012, iOS and Android were the most popular operating systems for mobile devices. Figure 8.1 shows that mainly Android and iOS devices are bought currently. This is reflecting in the statistics shown in Figure 8.2, which displays the current share of mobile OSs in Europe, and in Figure 8.3, which describes the share of mobile browsers in Europe in 2011. The statistics tell us that most user in Europe use Android and iOS devices to access the mobile web and mainly stay with the stock mobile browsers. Current HTML5 frameworks focus therefore on this two devices. Blackberry switched with version 6 to a WebKit based browser and has now the same underlying engine as the Android and iOS browser. This allows Blackberry devices to render iPhone or Android optimized pages. Windows Phone 7 is at the end of 2011 not widely used and has, together with webOS, the worst HTML5 support of the current smartphone operating systems.

The most promising mobile HTML5 frameworks are Sencha Touch and jQuery Mobile at the time of writing. They allow to develop mobile web applications with a native look and feel. Both of them have their advantages and disadvantages. jQuery Mobile for example has the broader range of supported mobile devices, while Sencha Touch focuses on a real native feeling of the web applications, but uses for that reason some WebKit unique features. The frameworks work very well on iOS devices, but still have some problems with animations and performance on Android devices. However, because of the increasing market share of Android the developers are now focusing to solve the problems on this platform. Every new version of the frameworks have an increased Android support.

Testing of mobile web applications is harder then testing classical desktop websites because of the big diversity of mobile devices and the lack of tools. Only a few tools exists that are specifically designed for testing on mobile devices. Unfortunately most of them only work on Android and iOS devices. Additionally, not every mobile browser supports debugging of the HTML and JavaScript code. This can make troubleshooting difficult.

When it comes to testing the JavaScript code, many options exist. Headless testing-frameworks emulate a real browser and allow executing the test suite very fast, but are not a guarantee that the code will work in a real environment. In-browser testing-frameworks can be run manually on the mobile device, but are hard to automate. Test drivers like JsTestDriver

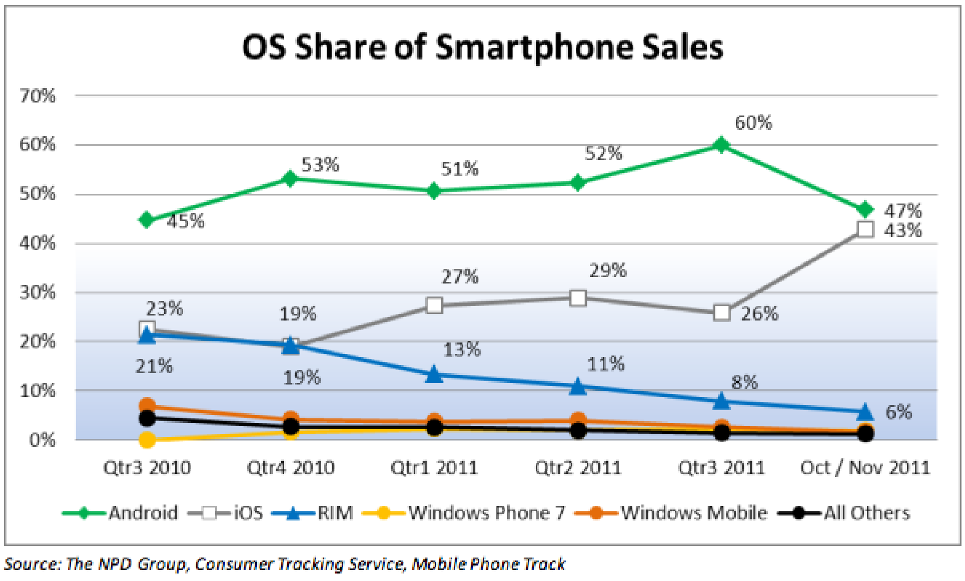


Figure 8.1: OS Share of Smartphone Sales [Per12]

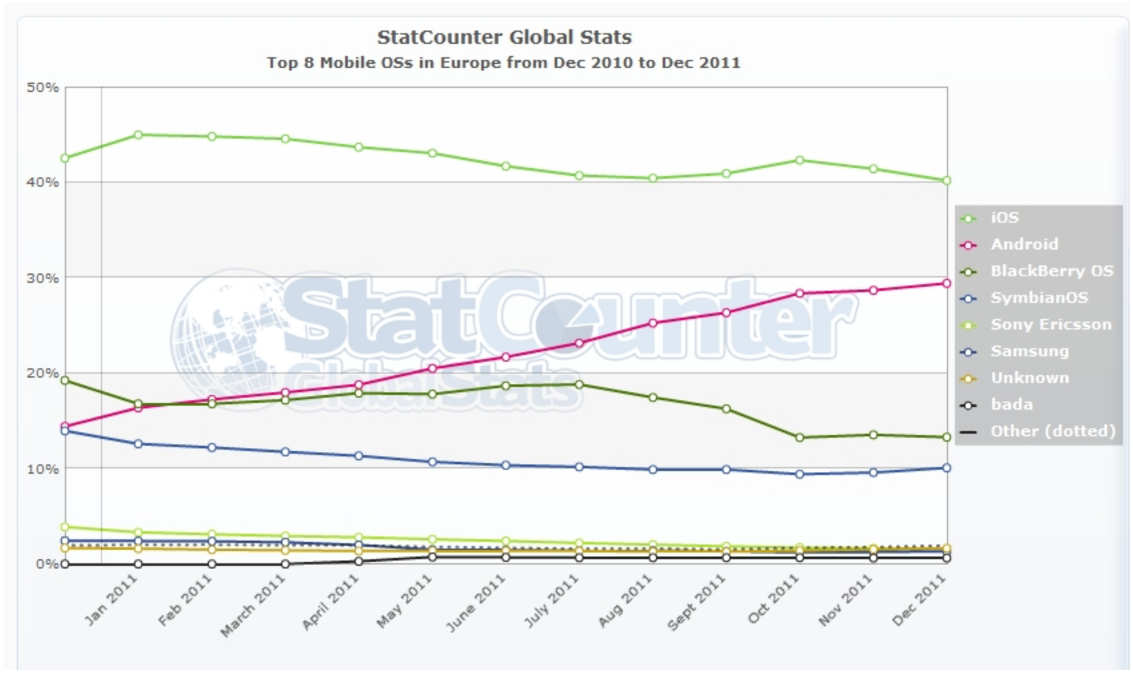


Figure 8.2: Top 8 Mobile OSs in Europe in 2011 [Sta12]

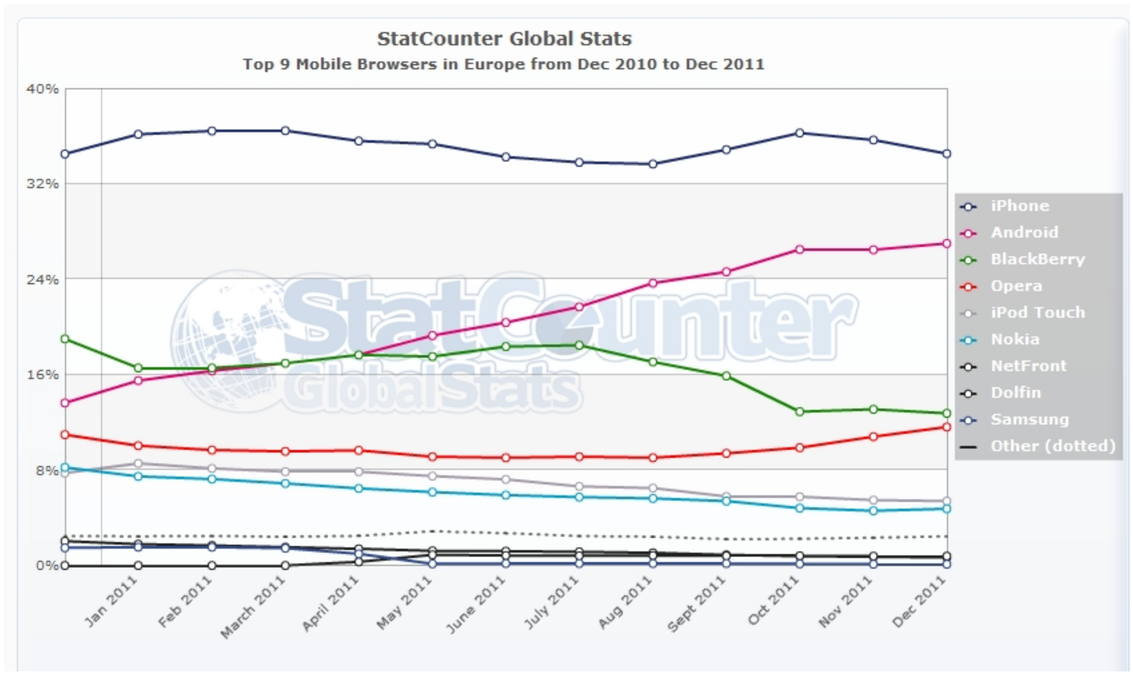


Figure 8.3: Top 9 Mobile Browsers in Europe in 2011 [Sta12]

allow to execute the test suite on many connected devices at the same time and to collect and evaluate the results on a central node. Even testing on real devices is not a guarantee that it will work on every device available. Purchasing smartphones is expensive and testing on a lot of them is time consuming. A trade-off has to be found.

GUI testing mobile web applications is more difficult than testing only the JavaScript code. Only few tools, as for example Selenium, allow automated GUI testing on mobile devices. At the time of writing, only Android and iOS devices are supported. Additionally, it is not possible to test on all available Android devices, because they are expensive and it is very time consuming. Remote labs allow to test the applications on real devices over the internet without the need of purchasing them. Depending on the used service, some of them allow test automation as well. This services are expensive and cannot replace testing on a real device, as the usability, like controlling it over the internet with a mouse, is different to holding it in the own hands. Nevertheless, they are a good complement to testing on real devices.

With combining some of the introduced testing tools, a good test coverage can already be achieved. As the mobile web will become even more important in the future, new testing tools will become available.

List of Abbreviations

adb	Android Debug Bridge
API	Application Programming Interface
bcc	Blind carbon copy
BDD	Behaviour Driven Development
Blob	Binary large object
BSD	Berkeley Software Distribution
cc	Carbon copy
CI	Continuous Integration
CSS	Cascading Style Sheets
DOM	Document Object Model
GB	Gigabyte
GPL	GNU General Public License
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
IE	Internet Explorer
JSON	JavaScript Object Notation
MIT	Massachusetts Institute of Technology
MVC	Model View Controller
OCR	Optical Character Recognition
OS	Operating System
RDA	Remote Device Access
RFC	Request for Comments
Sass	Syntactically Awesome Stylesheets
SMS	Short Message Service
SQL	Structured Query Language
SVG	Scalable Vector Graphics
TDD	Test Driven Development
URI	Uniform resource identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WebGL	Web Graphics Library
YUI	Yahoo! User Interface

Bibliography

[Ado11] Adobe. Aggressively Contribute to HTML5, 2011. Available online at `http://blogs.adobe.com/conversations/2011/11/flash-focus.html`; visited on December 20th 2011.

[App11] Apple Inc. Safari Developer Library, 2011. Available online at `http://developer.apple.com/library/safari/`; visited on January 23th 2012.

[Ban11] Aditya Bansod. Apple iOS 5: HTML5 Developer Scorecard, 2011. Available online at `http://www.sencha.com/blog/apple-ios-5-html5-developer-scorecard/`; visited on January 20th 2012.

[Bry11] Mats Bryntse. Introducing Siesta: A Testing Tool for Ext JS, 2011. Available online at `http://www.sencha.com/blog/introducing-siesta-a-testing-tool-for-ext-js/`; visited on January 9th 2012.

[Bry12] Bryntum AB. Siesta, 2012. Available online at `http://www.bryntum.com/products/siesta/`; visited on January 9th 2012.

[Bur10] David Burns. *Selenium 1.0 Testing Tools.* Packt Publishing, Birmingham, 2010.

[BW10] Daniel Barreiro and Dan Wellman. *YUI 2.8 Learning the Library.* Packt Publishing, Birmingham, 2010.

[CG09] Lisa Crispin and Janet Gregory. *Agile Testing.* Addison-Wesley, Boston, 1st edition, 2009.

[Cis11] Cisco. Cisco Visual Networking Index : Global Mobile Data Traffic Forecast Update , 2010 - 2015. Technical report, Cisco, 2011.

[Coh10] Mike Cohn. *Succeeding with Agile, Software Development Using Scrum.* Addison-Wesley, Boston, 2010.

[Cor10] Helder Correia. Remote JavaScript Debugging on Android, 2010. Available online at `http://www.sencha.com/blog/remote-javascript-debugging-on-android/`; visited on January 19th 2012.

[Cor11] Helder Correia. EventRecorder for Android Web Applications, 2011. Available online at `http://www.sencha.com/blog/event-recorder-for-android-web-applications`; visited on January 13th 2012.

[CS11] Jonathan Chaffer and Karl Swedberg. *Learning jQuery Third Edition*. Packt Publishing, Birmingham, 2011.

[Dev11] Alexis Deveria. When can I use... Support tables for HTML5, CSS3, etc, 2011. Available online at `http://caniuse.com`; visited on January 24th 2012.

[Eug10] Liang Yuxian Eugene. *JavaScript Testing*. Packt Publishing, Birmingham, 2010.

[Fir10] Maximiliano Firtman. *Programming the Mobile Web*. O'Reilly, Sebastopol, 2010.

[Fir11] Maximiliano Firtman. Mobile Html5, 2011. Available online at `http://mobilehtml5.org/`; visited on December 19th 2011.

[FR11] Eric Freeman and Elisabeth Robson. *Head First HTML5 Programming*. O'Reilly, Sebastopol, 1st edition, 2011.

[Gar11] Gartner. Gartner Says Sales of Mobile Devices in Second Quarter of 2011 Grew 16.5 Percent Year-on-Year; Smartphone Sales Grew 74 Percent, 2011. Available online at `http://www.gartner.com/it/page.jsp?id=1764714`; visited on January 30th 2012.

[Gas11] Peter Gasston. *The Book of CSS3*. No Starch Press, Inc., San Francisco, 2011.

[GG11] Lyza Danger Gardner and Jason Grigsby. *Head First Mobile Web*. O'Reilly, Sebastopol, 2011.

[Goo11] Google. JsTestDriver, 2011. Available online at `http://code.google.com/p/js-test-driver/`; visited on January 10th 2012.

[Goo12a] Google. Android Developers, 2012. Available online at `http://developer.android.com`; visited on January 20th 2012.

[Goo12b] Google. Android Supported Media Formats, 2012. Available online at `http://developer.android.com/guide/appendix/media-formats.html`; visited on January 23th 2012.

[Gre11] Ido Green. Offline, 2011. Available online at `http://offline-11.appspot.com`; visited on December 29th 2011.

[Hid11] Ariya Hidayat. PhantomJS, 2011. Available online at `http://www.phantomjs.org/`; visited on January 10th 2012.

[HL11] Chuck Hudson and Tom Leadbetter. *HTML5 Developer's Cookbook*. Addison-Wesley, Boston, 2011.

[HW08] Paco Hope and Ben Walther. *Web Security Testing Cookbook*. O'Reilly, Sebastopol, 2008.

[Joh11] Christian Johansen. *Test-Driven JavaScript Development*. Pearson Education, Inc., Boston, 2011.

[jQu10] jQuery Community Experts. *jQuery Cookbook*. O'Reilly, Sebastopol, 2010.

[Kan11] David Kaneda. jQTouch, 2011. Available online at `http://blog.jqtouch.com/`; visited on January 4th 2012.

[Key11] Keynote DeviceAnywhere. DeviceAnywhere, 2011. Available online at `http://www.deviceanywhere.com/`; visited on January 18th 2012.

[Lim11] Research In Motion Limited. Manuals and Guides for BlackBerry Users, 2011. Available online at `http://www.blackberry.com/docs/smartphones`; visited on January 23th 2012.

[Mac11] Alex MacCaw. *JavaScript Web Applications*. O'Reilly, Sebastopol, first edition, 2011.

[Mad10] Lech Madeyski. *Test-Driven Development: An Empirical Evaluation of Agile Practice*. Springer Berlin Heidelberg, Wroclaw, 1st edition, 2010.

[Mes07] Gerard Meszaros. *xUnit Test Patterns*. Addison-Wesley, Boston, 2007.

[Mic07] Microsoft Corporation. *Performance Testing Guidance for Web Applications*. Microsoft Press, 2007.

[Mic12] Microsoft. How to use HTML5 to Add an Audio Player to your Webpage, 2012. Available online at `http://msdn.microsoft.com/en-us/library/gg589483(v=VS.85).aspx`; visited on January 23th 2012.

[Moz11] Mozilla. Firebug Lite, 2011. Available online at `http://getfirebug.com/firebuglite`; visited on January 18th 2012.

[Nok12] Nokia. Remote device access, 2012. Available online at `http://www.developer.nokia.com/Devices/Remote_device_access/`; visited on January 18th 2012.

[Nor06] Dan North. Introducing BDD, 2006. Available online at `http://dannorth.net/introducing-bdd/`; visited on December 19th 2011.

[OB11] Damon Oehlman and Sébastien Blanc. *Pro Android Web Apps: Develop for Android Using HTML5, CSS3 & JavaScript*. Apress, New York, 2011.

[Pan11] Matthew Panzarino. HP announces it will discontinue TouchPad, Pre phones, stop webOS device development, 2011. Available online at http://thenextweb.com/insider/2011/08/18/hp-announces-it-will-discontinue-touchpad-stop-webos-device-development/ visited on January 30th 2012.

[Par11] Alejandro Parjus. IE9 For Windows Phone Is "Code Complete", 2011. Available online at `http://www.everythingwm.com/ie9-for-windows-phone-is-code-complete/2011/05/25/`; visited on January 30th 2012.

[Pea11] James Pearce. *Professional Mobile Web Development with WordPress, Joomla!, and Drupal*. Wiley Publishing, Inc., Indianapolis, 2011.

[Per11] Perfecto Mobile. Mobile Application Testing on Real Devices, 2011. Available online at `http://www.perfectomobile.com/`; visited on January 19th 2012.

[Per12] Sarah Perez. iOS Market Share Up From 26% In Q3 To 43% In Oct/Nov 2011, 2012. Available online at `http://techcrunch.com/2012/01/09/ios-marketshare-up-from-26-in-q3-to-43-in-octnov-2011/`; visited on January 23th 2012.

[Pil10] Mark Pilgrim. *Html5 up & running*. O'Reilly, Sebastopol, 1st edition, 2010.

[Piv11] Pivotal Labs. Jasmin, 2011. Available online at `https://github.com/pivotal/jasmine/wiki`; visited on January 9th 2012.

[Rei11] Jon Reid. *jQuery Mobile*. O'Reilly, Sebastopol, 1st edition, 2011.

[RW10] Arun Ranganathan and Shawn Wilsher. Firefox 4: An early walk-through of IndexedDB, 2010. Available online at `http://hacks.mozilla.org/2010/06/comparing-indexeddb-and-webdatabase/`; visited on December 29th 2011.

[Sam12] Samsung Electronics Co. Ltd. Lab.dev, 2012. Available online at `http://innovator.samsungmobile.com/bbs/lab/view.do?platformId=1`; visited on January 18th 2012.

[Sel12] Selenium. SeleniumHQ, 2012. Available online at `http://seleniumhq.org/`; visited on January 11th 2012.

[Sen11] Sencha Inc. Sencha Touch, 2011. Available online at `http://www.sencha.com/products/touch/`; visited on January 4th 2012.

[SH10] Markus Spiering and Sven Haiges. *HTML5-Apps für iPhone und Android*. Franzis Verlag GmbH, Poing, 2010.

[Siv10] Henri Sivonen. Activating Browser Modes with Doctype, 2010. Available online at `http://hsivonen.iki.fi/doctype/`; visited on December 21th 2011.

[Sta12] StatCounter. Top 9 Mobile Browsers in Europe from Dec 2010 to Dec 2011, 2012. Available online at `http://gs.statcounter.com/#mobile_browser-eu-monthly-201012-201112`; visited on January 23th 2012.

[Thea] The jQuery Project. jQuery Mobile. Available online at `http://jquerymobile.com`; visited on December 29th 2011.

[Theb] The jQuery Project. QUnit. Available online at `http://docs.jquery.com/QUnit`; visited on January 5th 2012.

[Tho11] Thoughtworks Studios. Twist, 2011. Available online at `http://www.thoughtworks-studios.com/agile-test-automation`; visited on November 11th 2011.

[W3C10a] W3C. Geolocation API Specification, 2010. Available online at `http://www.w3.org/TR/2010/CR-geolocation-API-20100907/`; visited on December 28th 2011.

[W3C10b] W3C. Media Queries, 2010. Available online at `http://www.w3.org/TR/2010/CR-css3-mediaqueries-20100727/`; visited on January 3rd 2012.

[W3C10c] W3C. Mobile Web Application Best Practices, 2010. Available online at http://www.w3.org/TR/2010/REC-mwabp-20101214/; visited on December 22th 2011.

[W3C10d] W3C. Web SQL Database, 2010. Available online at http://www.w3.org/TR/2010/NOTE-webdatabase-20101118/; visited on December 29th 2011.

[W3C11a] W3C. CSS Basic User Interface Module Level 3, 2011. Available online at http://dev.w3.org/csswg/css3-ui/; visited on January 3rd 2012.

[W3C11b] W3C. CSS Device Adaptation, 2011. Available online at http://www.w3.org/TR/2011/WD-css-device-adapt-20110915/; visited on December 21th 2011.

[W3C11c] W3C. DeviceOrientation Event Specification, 2011. Available online at http://www.w3.org/TR/2011/WD-orientation-event-20111201/; visited on December 28th 2011.

[W3C11d] W3C. File API, 2011. Available online at http://www.w3.org/TR/2011/WD-FileAPI-20111020/; visited on December 29th 2011.

[W3C11e] W3C. HTML Canvas 2D Context, 2011. Available online at http://www.w3.org/TR/2011/WD-2dcontext-20110525/; visited on December 28th 2011.

[W3C11f] W3C. HTML Media Capture, 2011. Available online at http://www.w3.org/TR/2011/WD-html-media-capture-20110414/; visited on December 29th 2011.

[W3C11g] W3C. HTML5, 2011. Available online at http://www.w3.org/TR/2011/WD-html5-20110525/; visited on December 23th 2011.

[W3C11h] W3C. HTML5 differences from HTML4, 2011. Available online at http://www.w3.org/TR/2011/WD-html5-diff-20110525/; visited on December 21th 2011.

[W3C11i] W3C. Indexed Database API, 2011. Available online at http://www.w3.org/TR/2011/WD-IndexedDB-20111206/; visited on December 29th 2011.

[W3C11j] W3C. Server-Sent Events, 2011. Available online at http://www.w3.org/TR/2011/WD-eventsource-20111020/; visited on December 29th 2011.

[W3C11k] W3C. The Network Information API, 2011. Available online at http://www.w3.org/TR/2011/WD-netinfo-api-20110607/; visited on December 29th 2011.

[W3C11l] W3C. The WebSocket API, 2011. Available online at http://www.w3.org/TR/2011/CR-websockets-20111208/; visited on December 29th 2011.

[W3C11m] W3C. Touch Events version 1, 2011. Available online at http://www.w3.org/TR/2011/CR-touch-events-20111215/; visited on December 23th 2011.

[W3C11n] W3C. W3C Confirms May 2011 for HTML5 Last Call, Targets 2014 for HTML5 Standard, 2011. Available online at http://www.w3.org/2011/02/htmlwg-pr.html; visited on January 23th 2012.

[W3C11o] W3C. Web Storage, 2011. Available online at http://www.w3.org/TR/2011/CR-webstorage-20111208/; visited on December 28th 2011.

[W3C11p] W3C. Web Workers, 2011. Available online at `http://www.w3.org/TR/2011/WD-workers-20110901/`; visited on December 29th 2011.

[WHA11] WHATWG. CanvasContexts, 2011. Available online at `http://wiki.whatwg.org/wiki/CanvasContexts`; visited on December 28th 2011.

[Whi09] James A. Whittaker. *Exploratory Software Testing.* Pearson Education, Inc., Boston, 2009.

[Wro11] Luke Wroblewski. *Mobile First.* Jeffrey Zeldman, New York, 2011.

[Yah11] Yahoo! Inc. YUI Test, 2011. Available online at `http://yuilibrary.com/projects/yuitest/`; visited on January 9th 2012.