Testing Cross-Device Applications

Master Thesis

Nina Heyder
<heydern@student.ethz.ch>

Prof. Dr. Moira C. Norrie
Maria Husmann

Global Information Systems Group
Institute of Information Systems
Department of Computer Science
ETH Zurich

11th September 2015

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

globis

Abstract

Here comes the abstract.

Contents

1
Introduction

1.1 Problem Statement

The abundance of devices nowadays have made it desirable to have cross-device applications where the interface and content is distributed among multiple devices. The emergence of new web technologies such as Devices API and WebRTC have made it possible to to create a new generation of web-based frameworks that facilitate the development of cross-device applications. Such applications typically run on any device that has access to a modern web browser. Despite the large number of frameworks for developing cross-device applications, none of them have focused on testing and debugging cross-device applications.

However, there are already plenty of practical tools for testing and debugging web applications in general and many of them can be accessed directly from modern browsers. Today's devices have many different characteristics, mainly in terms of screen size, but also concerning their input capabilities and connectivity. This diversity of devices has made it a requirement to develop websites that are functional and appealing on all devices. This goal can be achieved by following the principles of responsive design. Many tools have emerged that support testing of such websites; some are already built into modern web browsers while others can be accessed through a website or by installing a program on a desktop PC. In summary, those tools use two different approaches for testing responsive websites: First, different devices can be emulated on a desktop computer. Second, a varied set of actual devices can be used. Google Chrome provides extensive support for emulating devices; apart from simply emulating the screen size, it can also emulate touch, varying network conditions, location, and more. Other browsers also provide basic facilities for device emulation. When using multiple devices, the developer has to refresh all devices individually when changing the web application. However, there exist a number of tools to facilitate this. Some allow the developer to reload all devices at once, while others automatically reload devices when files change. Some of those tools even allow developers to simultaneously browse the web application on multiple devices. Apart from those tools, there are also web services for testing websites across a number of devices and platforms. Such web services typically include a screenshot generation service that renders a given website on a large number of devices.

Those tools are already a good starting point for testing and debugging cross-device applications. However, cross-device applications and web applications also have some fundamental differences that are not accounted for by those tools. In cross-device applications, multiple devices are typically used simultaneously and in a coordinated manner. Also, different devices do not necessarily show the same thing in cross-device applications, which limits the use of mirroring interactions from one device to all other devices. Furthermore, most of the tools for emulating devices focus on emulating one device at a time, which requires the developer to open multiple browser windows, possibly with different user profiles or in incognito mode, at a time, as cross-device applications only become useful when multiple devices are involved at once. Finally, all those tools focus either on emulating devices or on using real devices, but with cross-device applications, it might be desirable to combine those two approaches.

1.2 Goal Statement

Our project aims to make it easier to test and debug cross-device applications. Our first goal is to analyze what features might be useful for achieving this. This analysis is based on the current work in cross-device application development and web application testing. We will analyze existing tools for testing responsive web applications and try to find their limitations regarding cross-device application testing. From this, we will gather requirements for improvements or more suitable tools.

After analyzing existing tools, we will design and implement a new set of tools for testing and debugging cross-device applications following the requirements gathered before. Both emulating devices and connecting actual devices should be supported.

Finally, we will conduct a user study to evaluate the usefulness and suitability of the tool, especially comparing to traditional methods of testing web applications.

Furthermore, we will also develop two sample applications. This will allow us to further test our set of tools and gain helpful insights on further improvements for the tools.

1.3 Structure of this Document

We conclude this chapter by giving an overview of the structure of this document:

In Chapter 2, we present the background and related work of our project. In particular, we will describe existing tools for testing web applications as well as cross-device application development frameworks.

In Chapter 3, we describe our approach at achieving our goals and the requirements gathered during the analysis of related work.

In Chapter 4, we will describe the implementation of our set of tools in detail.

In Chapter 5, we will describe the sample applications that we developed and the insights we gained from them.

In Chapter 6, we will describe the user study and explain the results of it.

In Chapter 7, we conclude the work by showing what was achieved and what problems remain, and address possible future work that might be based on our existing set of tools.

2

Background and Related Work

2.1 Web Application Testing

2.1.1 Chrome DevTools

Google Chrome already provides a wide range of features for testing and debugging web applications. First of all, it lets the developer inspect the DOM tree and allows inspection and on-the-fly editing of DOM elements. Furthermore, new CSS rules or properties can be added or existing ones can be modified. Another useful feature is the JavaScript Console. It has two main purposes: First, it can be used to log diagnostic information in the development process. Second, it is a shell prompt which can be used to interact with the document and DevTools. Chrome DevTools can also be used to debug JavaScript. In the DevTools, you can see all the scripts that are part of the inspected page. Breakpoints can be set in the scripts and standard controls to pause, resume and step through code are provided. All features mentioned before are also available for debugging remote devices. Remote debugging can be used by connecting a device to the desktop PC with a cable.

Another feature is Device Mode which is especially useful for testing responsive web applications. In Chrome Device Mode, the user can select an arbitrary device from a large list of predefined devices or create a custom device. They can also enable network throttling, touch emulation and location emulation.

Chrome DevTools provides many more features useful for debugging, but explaining them all would exceed the scope of this document. While all those features are very useful for testing and debugging web applications, the main disadvantage is that they can only be used to debug one device and one website at a time. For cross-device applications, it would be desirable to debug multiple devices at a time without having to navigate between windows all the time.

2.1.2 Fire Crystal - Understanding Interactive Behaviors in Dynamic Web Pages

2.1.3 Invariant-Based Automatic Testing of Ajax User Interfaces

2.1.4 State-Based Testing of Ajax Web Applications

2.1.5 Testing Web-Based Applications - The State of the Art and Future Trends

2.1.6 Visual and Textual Consistency Checking Tools for Graphical User Interfaces

2.1.7 Summary

2.2 Responsive Web Application Testing

There are a number of tools for testing responsive websites. In the following subsections we will describe some of them.

2.2.1 BrowserStack

BrowserStack[1] allows developers to select browsers and devices and then generate screenshots. It is also possible to live test one device at a time. Furthermore, there are developer tools for remote devices and Selenium cloud testing is possible. Advantages of BrowserStack are that real iOS devices are used for screenshots and interactions can be tested automatically using Selenium. A major disadvantage is that only emulated Android devices are available, which limits the usefulness of the tool. Furthermore, only one device can be live tested at a time which is a huge disadvantage for cross-device application testing.

2.2.2 CrossBrowserTesting

With CrossBrowserTesting[2], developers can select a number of devices as well as the operating system, browser and resolution and generate screenshots. The layout differences between different devices can then automatically be analyzed. Furthermore, websites can also be live tested and Selenium automated testing is available as well. The main advantage of CrossBrowserTesting is that all screenshots are generated on real devices and a very wide variety of devices is available. Also, interactions can be tested. The usefulness of the tool is again limited by the fact that live testing is only possible on one device at once. Additionally, while detecting layout differences is a useful feature in general, it is not yet very mature and some layout differences that are detected seem rather trivial (the body element of a larger device is larger), while other differences are not noticed at all.

2.2.3 Remote Preview

Remote Preview[3] allows synchronizing URLs across multiple devices. This allows fast previewing of a website on multiple devices. In cross-device scenarios, it may provide especially useful for quick connecting of devices in applications where devices are connected simply by copying the same URL to all devices. However, the fact that this tool only provides one feature, namely URL synchronizing, limits its usefulness.

2.2.4 BrowserSync

BrowserSync[4] provides a large number of features for testing websites. It allows remote debugging of HTML and CSS, can add CSS outlines or box shadows to all elements and add a CSS grid overlay. It can also load a URL on all devices or refresh all devices as well as automatically refresh devices when files are changed. Furthermore, it allows synchronizing interaction between devices, i.e. clicks, scrolls, form submits, form inputs and form toggles. It also provides network throttling. The advantages of BrowserSync are that it provides a wide range of features, including synchronizing interactions, which is a feature that distinguishes it from other tools. However, for cross-device application testing, synchronizing interactions

[1] https://www.browserstack.com/
[2] http://crossbrowsertesting.com/
[3] https://github.com/viljamis/Remote-Preview
[4] http://www.browsersync.io/

among all devices is of limited usefulness, as different devices have different roles and thus also different responsibilities.

2.2.5 Adobe Edge Inspect CC

Adobe Edge Inspect CC[5] allows developers to take screenshots on all connected devices simultaneously. The screenshots are then automatically transferred to a folder on the desktop PC. It can also refresh all devices simultaneously. Furthermore, the URL that is opened on the desktop PC is loaded on all other connected devices. It also allows remote HTML and CSS debugging using weinre. The main advantage that distinguishes Adobe Edge Inspect CC from other tools is that is simply synchronizes the URL that the developer is currently looking at on the desktop PC, thus it works even if the developer switches tabs or browser windows. However, the fact that the URL cannot be changed from devices other than the desktop PC could be a disadvantage in some scenarios. Also, the installation process is rather extensive: A program needs to be installed on the desktop PC as well as a Chrome extension and an app needs to be installed on all mobile devices that the developer wants to connect. Regarding cross-device applications, again, refreshing all devices at once can be useful as well as synchronizing URLs in some cases, but other than that, it does not provide any features that help with cross-device application testing.

2.2.6 Ghostlab

Ghostlab[6] is one of the more mature tools for website testing and provides features similar to Browser Sync. It can also load a URL on all devices or refresh all devices at once and refresh automatically when files are changed. It provides means for synchronized browsing as well as synchronized HTML and CSS inspection on multiple devices. Furthermore, it can automatically fill out forms and provides remote Javascript debugging. Synchronization can be turned on and off on a per-device basis, which makes the tool more useful than tools that simply synchronize all devices. However, the usefulness is still limited because constantly changing the devices that should be synchronized is time-consuming and error-prone. Also, interactions on cross-device applications typically do not happen in a synchronous fashion.

2.2.7 Summary

While the tools described above already provide a wide variety of features that are immensely useful for responsive web application testing as well as web application testing in general, the distinguishing characteristics of cross-device applications lead to a limited usefulness of those tools. First of all, there are two different approaches to live testing in the tools described above. The first is to test on real devices, but through a web service. In those tools, live testing is possible on one device at a time, but in cross-device scenarios, multiple devices are typically involved. The second approach is to let the developer connect their own devices and synchronize interactions among all or some devices. This is also problematic for testing

[5] `https://www.adobe.com/ch_de/products/edge-inspect.html`
[6] `http://www.vanamco.com/ghostlab/`

cross-device applications because not all devices perform the same interactions and those that do, do not necessarily perform them at exactly the same time. Two features that some of the tools mentioned above provide and that would definitely also be useful for cross-device applications are refreshing all devices at once and loading a URL on all devices. Remote HTML, CSS and Javascript debugging are also desirable in a cross-device scenario, but this is already covered by Google Chrome anyways. Something similar to Selenium testing would clearly also be useful in cross-device scenarios. However, depending on the device, different tests would be needed and it should be possible to run those tests in parallel.

2.3 Cross-Device Application Development Frameworks

In the following subsections, we will describe some frameworks that facilitate the development of cross-device applications.

2.3.1 XD-MVC

XD-MVC[7] is a framework that combines cross-device capabilities with MVC frameworks. It can be used as a plain JavaScript library or in combination with Polymer. The framework consists of a server-side and a client-side part. For communicating, either a peer-to-peer or a client-server approach can be used.

2.3.2 Connichiwa

Connichiwa is a framework for developing cross-device web applications developed by Schreiner at al. [7]. It runs local web applications on one of the devices without requiring an existing network or internet connection. It has four key goals:

- Integration of existing devices: Devices should be supported without the need to augment them with additional hardware, markers, or tags
- Independence of network infrastructure: No remote server is used, therefore the device neither needs to be in the same network nor have a permanent internet connection
- Usability of its API

A native helper-application runs a web server on-demand. The native application automatically detects other devices using Bluetooth Low Energy, which are then connected by sending the IP address of the local web server over Bluetooth. Connichiwa's JavaScript API gives easy access to common functions like device detection and connection and it also provides JavaScript events to notify about device detection and connection.

[7] `https://github.com/mhusm/XD-MVC`

2.3.3 DireWolf

DireWolf is a framework for distributed web applications based on widgets. It was developed by Kovachev et al. [4] Widgets can be shared, reused, mashed up and personalized between applications. Splitting the interface into separate widgets and enabling them to exchange information allows the development of customizable web applications. It makes the following contributions:

- They provide a framework for easy browser-based distribution of Web widgets between multiple devices.
- They facilitate extended multi-modal real-time interactions on a federation of personal computing devices
- They provide continuous state-preserving widget migration

DireWolf helps managing a set of devices and handles communication and control of distributed parts of the web application. Local inter-widget communication is used to communicate with widgets running in the same browser context and remote inter-widget communication provides the message-exchange mechanism for widgets located at different devices.

2.3.4 XDStudio

In [6], Nebeling et al. present their web-based GUI builder, XDStudio. XDStudio is designed to support interactive development of cross-device web interfaces. It has two complementary authoring modes: Simulated authoring allows designing for a multi-device environment on a single device by simulating other target devices. On-device authoring allows the design process itself to be distributed over multiple devices. The design process is still coordinated by a main device, but directly involves target devices. The user can switch between two different modes:

- Use mode: The user can interact normally with the interface loaded into the editor
- Design mode: The user can manipulate the interface directly

XDStudio makes the following contributions:

- It explores two scenarios thatt were developed as interesting use cases of multi-device, distributed user interfaces and used to drive the design and evaluation of XDStudio
- The notion of distribution profiles at the core of XDStudio extends existing context models to multiple devices and users
- It provides cross-device authoring concepts and tools that cater for cases where not all devices and users are available, or where they are even different, at deisgn and run-time
- User study to evaluate XDStudio

Distribution profiles can be specified in terms of involved devices, users and target user interfaces. The user can either select which user interface widgets and other elements of the source to include in the distribution by dragging and dropping them from the source interface to the corresponding target interface or click a button available for each target interface to insert the full version of the source interface as a starting point, and then include/exclude interface elements from the distribution.

2.3.5 Panelrama

Panelrama, developed by Yang et al. [8], is a web-based framework for the construction of applications using distributed user interfaces. It introduces a new XML element, ”panel”, which may be placed around groupings of control and it facilitates the distribution and synchronization of panels among the connected devices. The developer can specify the state information that should be synchronized across devices as well as the suitability of panels to different types of devices. An optimization algorithm distributes panels to devices that maximize their match for the developer's intent. An existing application can be converted to Panelrama using the following steps:

1. Wrap grouping of HTML UI elements within a ”panel” tag
2. Complete a Panelrama definition including by selecting the state information for synchronization and rating each panel for its needs with respect to various device characteristics
3. Modify the business logic such that it accesses the new synchronized state information

2.3.6 Polychrome

Polychrome is a web application framework for creating web-based collaborative visualizations that can span multiple devices. It was developed by Badam et al. [1]. It supports:

- Co-browsing new web applications as well as legacy websites with no migration costs
- An API to develop new web applications that can synchronize the UI state on multiple devices to support synchronous and asynchronous collaboration
- Maintenance of state and input events on a server to handle common issues with distributed applications such as consistency management, conflict resolution, and undo operations

Polychrome provides the interaction and display space distribution mechanisms to create new collaborative web visualizations that utilize multiple devices and it provides framework modules to store the user interaction. Combined with the initial state of the website, the interaction logs are useful for synchronizing devices within the collaborative environment, consistency management and interaction replay. There are three different modes for sharing:

- Explicit sharing: The user decides when the operations should be shared, which operations to share, and whom to share them with

- Implicit sharing: Operations are automatically shared with all connected devices
- Unilateral sharing: One device (the leader) always shares its operations automatically, while the other devices only listen to the operations shared by the leader

2.3.7 XDSession

XDSession, developed by Nebeling et al. [5], is a framework for cross-device application development based on a concept of cross-device sessions which is also useful for logging and debugging. The session controller supports management and testing of cross-device sessions with connected or simulated devices at run time. The session inspector enables inspection and analysis of multi-device/multi-user sessions with support for deterministic record/replay of cross-device sessions. A session consists of users, devices, and information. When a device joins a session, it receives the whole or missing part of the data belonging to the selected session. When a device leaves a session, all data of that session that is not shared with any other session is removed from the device.

2.3.8 WatchConnect

In [3], Houben et al. present WatchConnect. WatchConnect is a toolkit for rapidly prototyping cross-device applications and interaction techniques with smartwatches. It provides an extendable hardware platform that emulates a smartwatch, a UI framework that integrates with an existing UI builder and a rich set of input and output events using a range of built-in sensor mappings.

2.3.9 Weave

Weave is a web-based framework for creating cross-device wearable interaction by scripting, developed by Chi et al. [2]. It provides a set of high-level APIs for developers to easily distribute UI output and combine sensing events and user input across mobile and wearable devices. Devices can be manipulated regarding their capabilities and affordances, rather than low-level specifications. Weave also has an integrated authoring environment for developers to program and test cross-device behaviors. Developers can test their scripts based on a set of simulated or real wearable devices. Weave's APIs capture affordances of wearable devices and provide mechanisms for distributing output and combining sensing events and user input across multiple devices.

2.3.10 Summary

In the sections above, we have described several different frameworks for developing cross-device applications. Some of them are only useful for specific types of applications, while others are suited for cross-device applications of all types. Despite this abundance of frameworks available, most of the papers describing the frameworks do not mention anything about testing and debugging cross-device applications. XDSession is the only framework that has some features related to testing, but those features are mainly limited to testing and replaying

sessions. Thus, a proper tool for testing and debugging cross-device applications is clearly still missing.

3
Approach

3.1 Device Emulation

4
Implementation

4.1 Architecture

4.2 Choice of Technologies

4.3 General Features

4.4 Device Emulation

4.5 Connecting Real Devices

4.6 Shared JavaScript Console

4.7 Function Debugging

4.8 Shared CSS Editor

4.9 Record/Replay

5
Sample Applications

5.1 XDCinema

5.2 XDYouTube

5.3 Insights

6
Evaluation

6.1 Setup

The study was carried out in a room of the GlobIS group. The participants were sitting in front of a 27-inch screen with a 2560x1280 and had access to an English (US) keyboard and a mouse. They also had access to two real devices: A Nexus 7 and an HTC M9. Furthermore, they were given a tutorial sheet about Javascript that contained some functions that are useful for DOM modification as well as arrays. During the study, the instructor was sitting next to the participants and was available to answer any questions that occurred during the study.

6.1.1 Participants

We recruited 12 participants that all were university members in the department of computer science at ETH Zurich. Most participants were either PhD or Master students, but there were also some Bachelor students. It was required that all participants have at least basic knowledge about front-end web technologies (i.e. HTML, CSS, Javascript). The age of the participants ranged from 23 to 33 and the median age was 26.

Previous Experience

We asked all participants about their previous experience with web application development and Javascript in particular, as well as about their previous experience with responsive web applications and cross-device web applications. Furthermore, we asked them about whether they have used Chrome DevTools before and how they used them. The participants rated their skills in web application development and Javascript on a 5-point Likert scale (see Figure 6.1) from basic to proficient and also gave the numbers of years in experience (see Figure 6.2) that they had in web application development and Javascript.

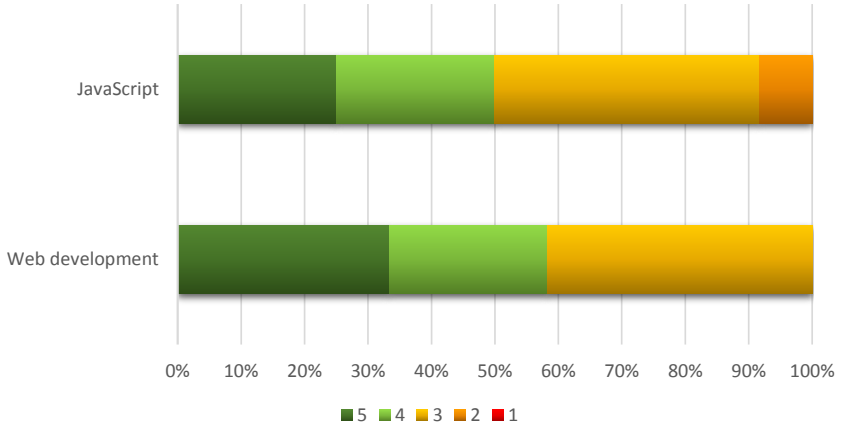

Figure 6.1: Previous experience

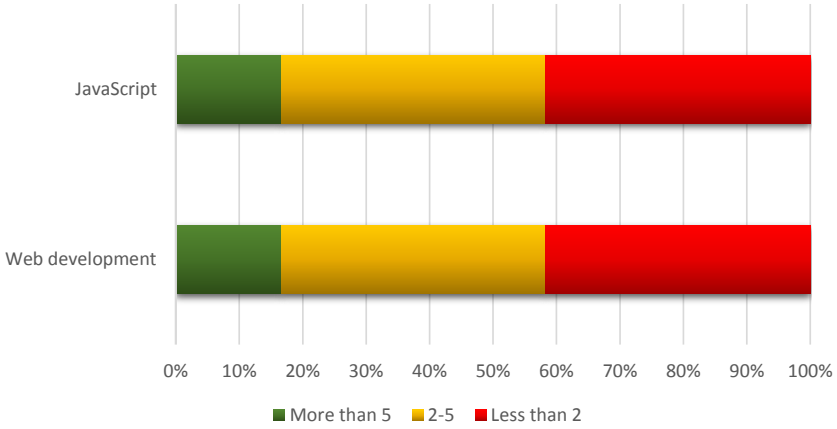

Figure 6.2: Years of experience

Eight out of the twelve participants stated that they already had some experience with developing responsive web applications. All of them either used browser tools for emulating devices to test their applications or real devices. Some also used both.

Seven participants already had some experiences with cross-device application development. Again, most of them used browser tools for emulating devices or real devices for testing their applications. Four of them either used multiple browsers, multiple browser profiles or incognito modes to simulate multiple devices on one device. Thus, about half of the participants that already had some cross-device experience constantly used multiple devices to test their applications.

Most of the participants already had experience with Chrome DevTools, only three participants indicated that they had never used them before. We asked participants how often they used certain features of Chrome DevTools, in particular Device Mode, HTML and CSS inspection, Javascript debugging and the console (see Figure 6.3). Device Mode was rarely used by the participants, which is no surprise, given that it is a rather new feature. All participants stated that they often use HTML and CSS inspection, thus this seems to be the most popular feature. The console was also used rather often. Surprisingly, Javascript debugging was not that popular, less than half of the participants stated that they often use it.

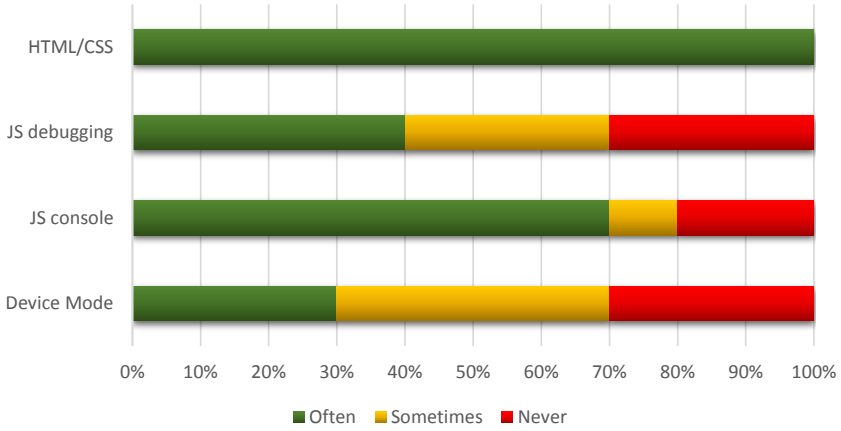

Figure 6.3: Previous experience with Chrome DevTools

6.1.2 Tasks

We used two different applications for the tasks. For each application, there were two tasks; one was about finding and fixing a bug in the code, and the other one was about implementing a new feature. The maximum time for the tasks where the participants fixed a bug was 15 minutes and the maximum time for implementing a feature was 30 minutes. After this time, we aborted the task unless it was clear that the participants would finish within the next 2 to 3 minutes. Each participant had to complete all four of the tasks; the tasks of one application with our tools, the tasks of the other application without them. The order of the applications as well as whether the first two tasks were with or without the tools was random. One of the applications was a cross-device YouTube application, called XDYouTube. The other application was a cross-device cinema application, called XDCinema.

XDYouTube

XDYouTube allows users to use their personal devices to search for videos and add them to a queue. The videos from the queue were then played one after the other on the largest of the devices. Users could also see the title and description of the currently playing video as well as the videos that are still in the queue by switching their device into landscape mode.

The first task with XDYouTube was to fix a bug concerning the video queue. As soon as one video finishes playing, the next video was dequeued from the queue and started playing. However, when no video was in the queue, a Javascript error occurred and caused the next video that was added to the queue not to play. The users were given a description of the task and then had to reproduce and fix it.

For the second task, we asked participants to implement a remote control that could play and pause the current video. The participants had to implement two functions: One was called when the remote control button was clicked (the button as well as the event handler were already implemented), the other was called when a shared variable that states whether the video is paused or playing is changed. Thus, the participants had to change the shared

variable whenever the button was clicked and to react accordingly on all devices if the shared variable changes, i.e. they had to pause or play the video on the device that plays the video and they had to change the text of the remote control button on all other devices. Furthermore, they had to change the CSS of the button such that it looked similar to a picture of a button that was given to them.

XDCinema

XDCinema allows users to search for a city and date on one device. The device then shows a list of movies that play in this city on that date as well as the cinemas where the movie is played and the time that the movie starts. If the user clicks on a cinema, a summary of the movie as well as other information about the movie is shown on another device. If the user clicks on a cinema, the location of the cinema is shown on another device.

The first task was to fix a bug where the location of most cinemas was displayed wrongly, even though the information in the database is correct. The bug was that in one function, ”j” was used instead of ”i”, which caused a wrong location to be returned.

In the second task, the participants first had to complete the implementation of a function that shows the prices of each cinema where the movie plays below the description of the cinema. A skeleton for this function was already given where a loop over all cinemas that show the movie was already implemented, the participants only had to fill in the body of the loop. The second part was about first highlighting the correct price when the user clicks on a cinema in the search view and improving the CSS for highlighting.

6.1.3 Evaluation Methods

Questionnaires

At the beginning of the study, each participant had to fill out a questionnaire about their background information. After every task, the participant had to fill out another questionnaire with the following questions:

- It was easy to complete the task with the tools I had access to.
- I felt efficient completing the task with the tools I had access to.
- It was challenging to complete the task with the tools I had access to.
- The tools I had access to were well suited for completing the task.

The questions could be answered on a 5-level Likert scale from ”Strongly Disagree” to ”Strongly Agree”. In the tasks where the participants had access to our tools, we also asked them to rate the usefulness of the individual features of the tool on a 5-level Likert scale.

After completing all tasks, the participants had to fill out a final questionnaire where they could answer some questions that compare our tools to the usual Chrome browser tools. They had to answer the following questions:

- Did you find it easier to debug with or without the tool?
- Did you feel more efficient debugging with or without the tool?
- Did you prefer debugging with or without the tool?
- Did you find it easier to implement a feature with or without the tool?
- Did you feel more efficient implementing a feature with or without the tool?
- Did you prefer implementing a feature with or without the tool?

In addition, they also answered some general questions about our tool?

- It was easy to learn how to use the tool.
- I felt confident using the tool.
- The tool was unnecessarily complex.
- The tool would be useful for debugging cross-device applications.
- The tool would be useful for implementing cross-device applications.
- I would use the tool for debugging cross-device applications.
- I would use the tool for implementing cross-device applications.

Those questions could again be answered on a 5-level Likert scale from ”strongly disagree” to ”strongly agree”.

Finally, the participants could state which features of the tool they would use for debugging and implementing cross-device applications and they could also write some comments about the tool if they wanted to.

Video Recording

In addition to letting participants fill out questionnaires, we also used a video camera to record the participants while completing the tasks. This was mainly done to make sure that no important information was lost and so some strategies for solving tasks could be extracted from the videos.

Personal Feedback

At the end of the study, participants were encouraged to share any comments that they still wanted to mention and to give their opinion about the tool. Any comments that the participants had given during the study were also noted.

Time Measuring

For each participant, the time required for completing each task was measured. This was mainly done to detect any major discrepancies between completion times with and without the tool. However, exact times are not considered relevant for evaluation because they highly depend on the participant and on the hints given by the instructor during the study.

6.2 Results

In the following sections, we will present the results from the individual tasks as well as the more general results. For each task, we will compare how people answered the questions in the per-task questionnaires with and without our tools.

6.2.1 XDCinema: Fixing a Bug

The results for the task where the participants had to fix a bug in XDCinema can be seen in Figure 6.4. The figure shows the median values for the questions asked after the task with and without our tools. For the question that asks about how challenging it was to complete the task with the tools the participant had access to, a lower value is better; for all other questions, a higher value is considered better. The median values for the suitability of the tools has a rather big difference, while the differences are smaller for the other tasks. Concerning easiness and efficiency, only a very small difference in favor of our tools can be seen. One reason for the bigger difference in the suitability could be that unlike easiness and efficiency, suitability does not really depend on the task itself. In other words, if a task is difficult, the easiness will be rated lower regardless of whether our tools are used or not. On the other hand, the suitability of the tools for the task does not change with the difficulty of the task.

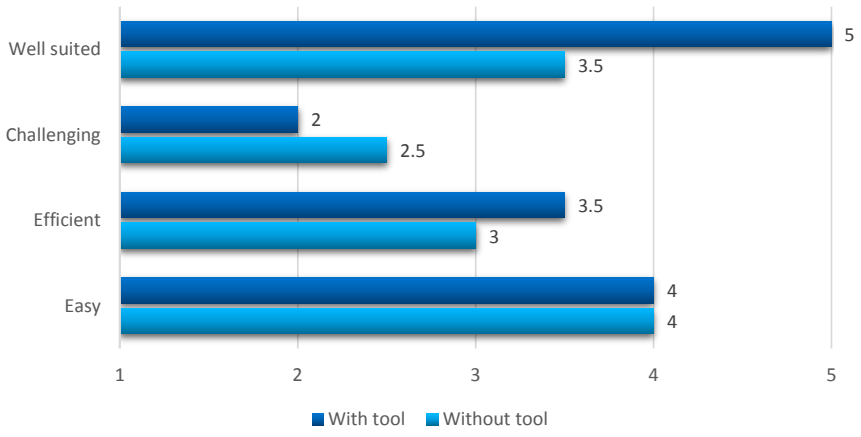

Figure 6.4: XDCinema bug task - Comparison

Figure 6.5 shows how many participants used the individual features of our tools and how useful they found them. Obviously, the figure only includes the participants that had access

to our tools. None of the participants used real devices and all of them used device emulation instead. However, two participants had a neutral opinion about device emulation. This could be due to the fact that the bug they had to fix is actually rather trivial and could maybe be solved faster by just looking at the code. The connection features and function debugging were used by all except one participant and were very appreciated by the participants. The shared JavaScript console was rather unpopular for this task, probably also due to the simplicity of the task and due to the fact that the bug produced no JavaScript errors that would be displayed in the console.

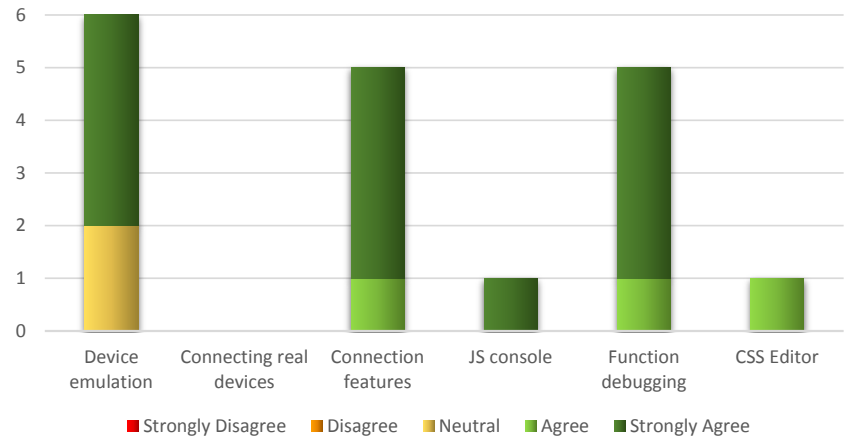

Figure 6.5: XDCinema bug task - Features used

6.2.2 XDCinema: Implementing a Feature

In Figure 6.6, the results for the implementation task in XDCinema can be seen. Again, the figure shows the median values. In this task, the difference in suitability is less pronounced than in other tasks, whereas the difference in rather large. In general, the task was considered as rather easy independent of the tools the participant had access to, although participants that had access to the tools perceived it even as a bit easier. In conclusion, it seems that the task is easy anyways, but our tools makes participants feel much more efficient when completing the task. Surprisingly, if we compare the average or median completion times for this task, the participants that had access to our tools were considerably slower. In fact, this is the only task where the difference in completion times with and without our tools is noticeable; the completion times for all other tasks are almost equivalent. However, those two facts do not necessarily contradict each other, as there were participants with very different experience levels and as it is random which participants have access to our tools and which not and the number of participants is rather low, it is possible that almost all participants with low experience fall into the same category. In general, the completion times should not be considered as especially relevant, after all the instructor also gave some hints during the study and this can also distort completion times significantly. However, it would be interesting to see how completion times differ if there is a larger number of participants.

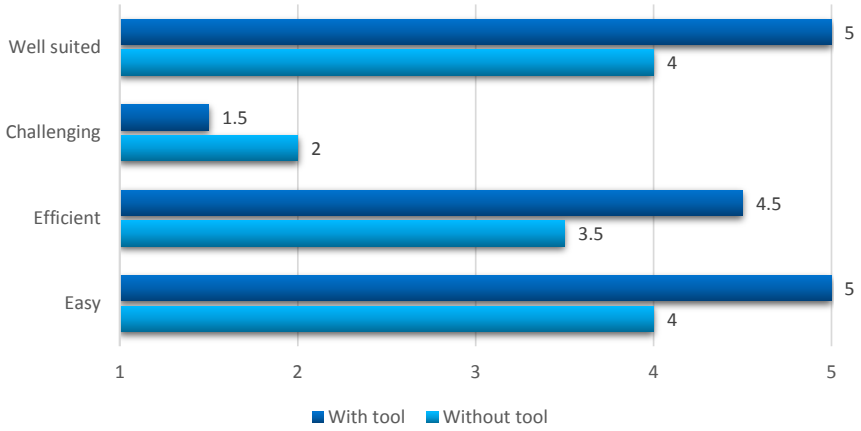

Figure 6.6: XDCinema implementation task - Comparison

Figure 6.7 again shows the use and the ratings of the individual features. Again, no participant used the real devices. All participants used device emulation and the connection features and in contrast to the bug task in XDCinema, device emulation was rated as useful by all participants. Function debugging was used a bit less than in the bug task, but the shared JavaScript console was used much more. It makes sense that function debugging is used more when fixing a bug; if one implements a feature and it works immediately when testing it, there is no need to debug a function, but if one has to fix a bug, there obviously must be a bug in a function and thus it makes much more sense to debug functions. The shared JavaScript console was rarely used to send commands and most participants did not use logging for solving the task, but many participants had some syntax errors when first testing the feature and noticed the error messages in the console. This also explains why the console was used more in the implementation task than in the bug task: Generally, console outputs are very useful for debugging, but in this specific bug, there were no errors in the console in contrast to the implementation task, where syntax errors were shown in the console. Finally, the CSS editor was also used by some participants in this task. However, some completed the CSS part of the task using only the CSS file. This may be because they did not think of the CSS editor at this specific moment, or because they know CSS so well that they can just write everything down immediately.

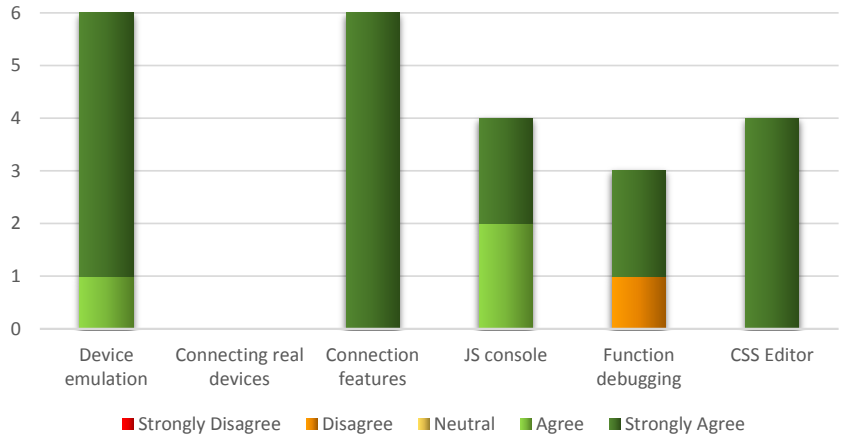

Figure 6.7: XDCinema implementation task - Features used

6.2.3 XDYouTube: Fixing a Bug

Figure 6.8 shows the results for the bug task in XDYouTube. The difference in suitability between our tools and the usual browser tools was most significant in this task with a difference of 2. The difference in efficiency is also rather large. Surprisingly, there is no difference in easiness and the task was also rated as almost equally challenging with and without our tools despite the large differences in the other questions. It seems that for this task, our tools did not make the task any easier to solve, but the participants felt more efficient when completing it. During the study, we noticed that most participants had problems reproducing the bug and almost all participants required some hints and finished the task more or less around the time limit. This could explain why the task was perceived as equally difficult with and without our tools: The participants did not really find the bug without help anyway, independent of whether they had access to our tools, so it makes sense that they would consider the task as difficult in general.

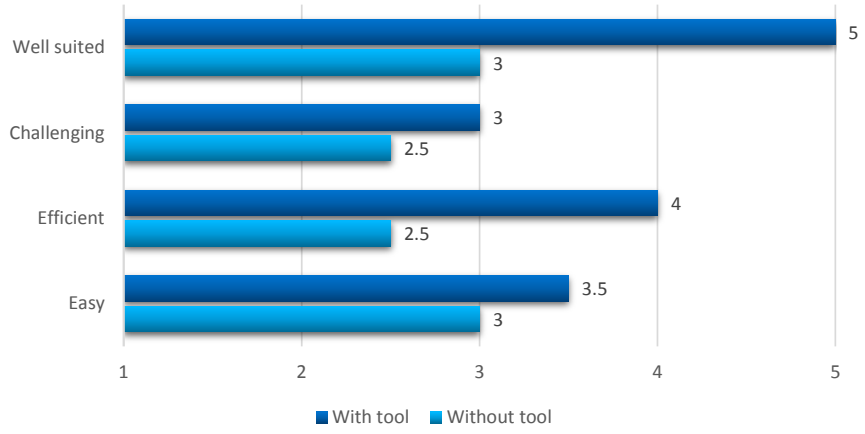

Figure 6.8: XDYouTube bug task - Comparison

In Figure 6.9; the use and ratings of the individual features can be seen. All of the participants used device emulation and connection features and rated them well. Function debugging was also used by almost all participants, probably because it was difficult to reproduce the bug and the participants wanted to see what was going on in the functions. About half the participants used the shared JavaScript console, mainly to see the error produced in the function that caused the bug. One participant connected the Nexus 7 to our tools and liked the feature, but no statement about the general usefulness of the feature can be made from just one participant.

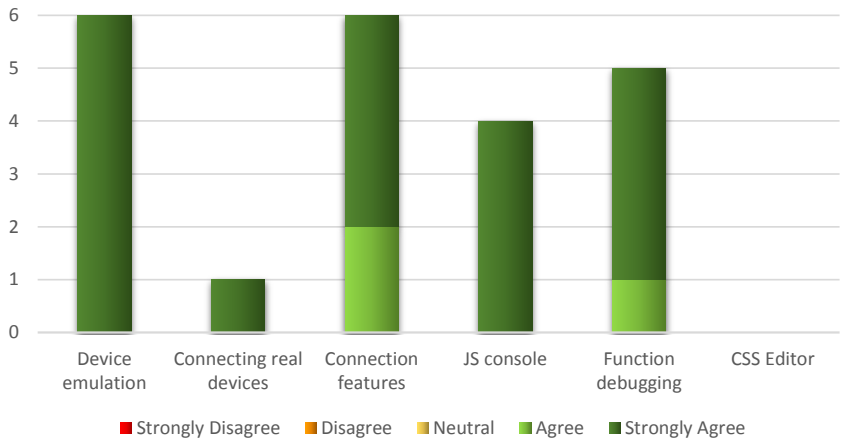

Figure 6.9: XDYouTube bug task - Features used

6.2.4 XDYouTube: Implementing a Feature

Figure 6.10 shows the results for the implementation task in XDYouTube. In this task, all questions were clearly rated in favor of our tools. While the participant answered the questions in a rather neutral way when they did not have access to our tools, they clearly stated that the task was easy to complete and felt efficient to complete with our tools. This task differs from the others a bit, because all other tasks had questions where the difference in median was 0.5 or 0, whereas the difference is at least 1 in this task for every question.

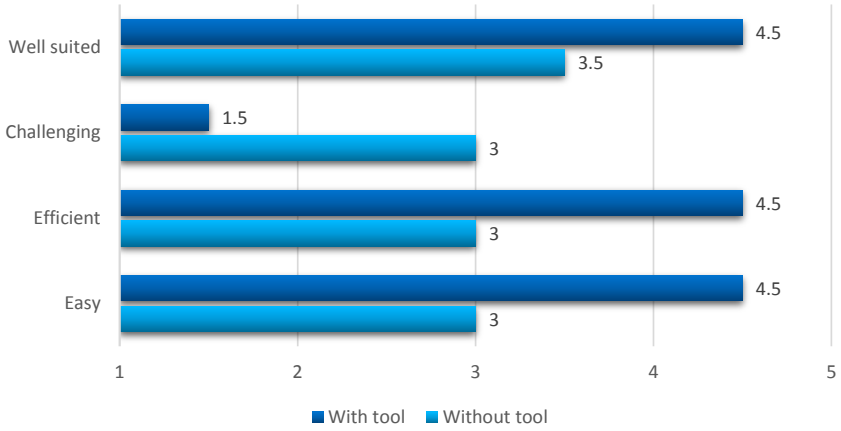

Figure 6.10: XDYouTube implementing task - Comparison

In Figure 6.11; the use and ratings of the individual features can be seen. Once again, device emulation and the connection features were used by every participant. The shared JavaScript console and CSS editor were about equally popular and rated as very useful except for one participant that had a neutral opinion on the console. Function debugging was rarely used for this task. This is rather surprising because many participants had a bug where they had switched playing and pausing the video at the beginning and this could probably have been solved easily by debugging the function. However, most participants just got stuck at the bug and required some hints to fix it instead of debugging their functions. Those participants were in the same group as the one that required significantly longer for solving the implementation task in XDCinema which could again indicate that the participants in this group had a bit lower experience in web application development. When comparing the web application development experience stated in the general questionnaire by the two groups, the other group has a median experience of 4.5 while this one has a median experience of 3.5, so there indeed seems to be some difference in experience levels.

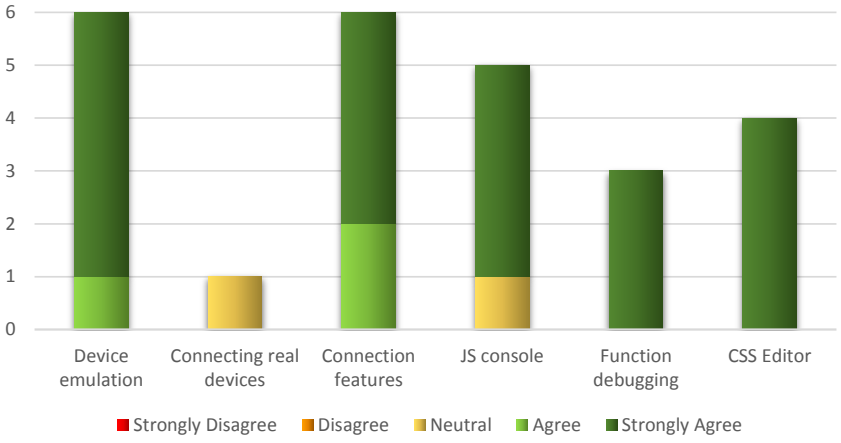

Figure 6.11: XDYouTube implementing task - Features used

6.2.5 General

In general, our tools were rated as very useful for all tasks with a median value of 5 for every task. In contrast, the usefulness of the usual browser tools was rated with median values between 3 and 4, thus we consider our tools as useful for implementing and debugging cross-device applications. The efficiency was also rated as significantly better for all tasks except the bug task in XDCinema, where the difference was only 0.5. This indicates that our tools make the process of testing cross-device applications more efficient. The difference in easiness seems to depend partially on the type of task; in the tasks where the participants had to implement a feature, completing the task with out tools was clearly considered easier, whereas the difference was smaller for the tasks where the participants had to fix a bug. Surprisingly, the results for the question about how challenging it was to complete the task do not necessarily relate to the results of the question about easiness. For the bug task in XDCinema and the implementation task in XDYouTube, our tools seem to make the task less challenging, but for the other two tasks, the differences are very minor. In the XDYouTube bug task, completing the task with our tools is even rated as slightly more challenging.

6.3 Discussion

Figure 6.12 shows that about three quarters of all participants considered implementing a feature easier with our tools. Only one participant found it easier to implement a feature without our tools. However, in principle, this option is redundant as the participants still have access to the browser tools even when they have access to our tools. Thus the relevant conclusion is that about one quarter of the participants did not see any gain in easiness from using our tools. The same applies to the question about whether implementing a feature feels more efficient with our tools (see Figure 6.13); this figure shows exactly the same results as the figure about easiness. However, all except one participant preferred implementing a feature with our tools (see Figure 6.14). Thus, it seems that participants like to have access to our tools even if they cannot directly relate them to a decrease in difficulty or to an increase in efficiency.

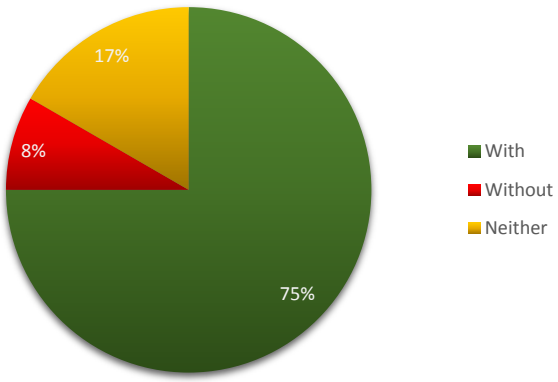

Figure 6.12: Easiness of implementing a feature

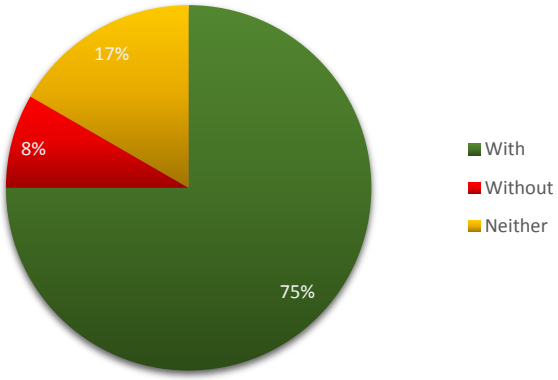

Figure 6.13: Efficiency of implementing a feature

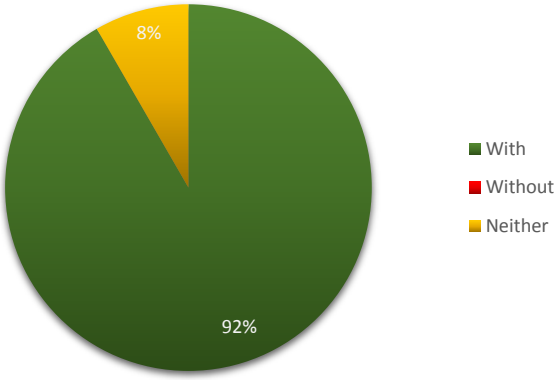

Figure 6.14: Preference for implementing a feature

Figure 6.15 shows that most participants found it easier to debug a cross-device application with our tools. The results get even more clear if we look at Figure 6.16 and Figure **??**. Those two figures show that all participants felt more efficient when debugging with our tools and also preferred debugging with our tools.

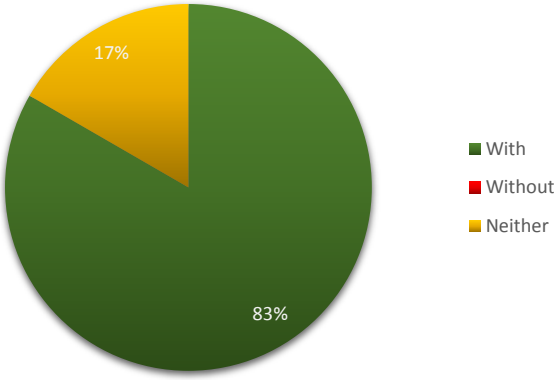

Figure 6.15: Easiness of debugging

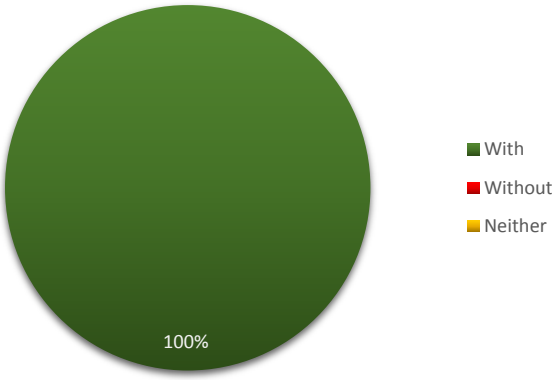

Figure 6.16: Efficiency of debugging

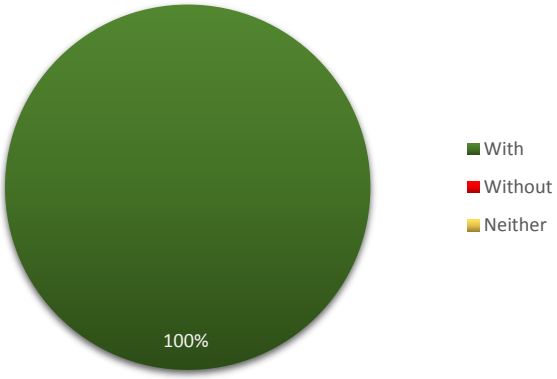

Figure 6.17: Preference for debugging

We also asked the participants if they would use our tools for debugging and implementing cross-device applications and if they think that our tools would be useful. The results can be seen in figure 6.18. Almost all participants think that our tool would be very useful for implementing as well as debugging cross-device applications and the remaining few also think that they would be useful. All participants would use our tools for implementing cross-device applications and all except one participant would use them for debugging cross-device applications.

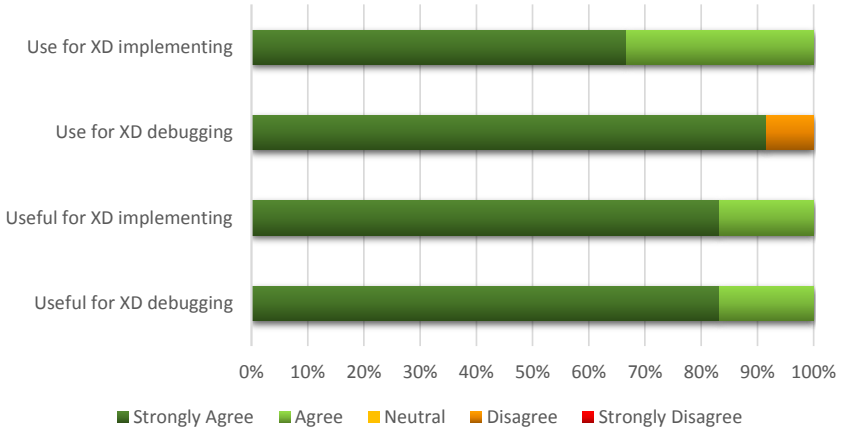

Figure 6.18: Usefulness of our tools

Finally, you can see the results for the general questions about our tools in Figure 6.19. The figure shows that our tools were perceived as easy to learn despite the many different features and the participants also felt rather confident using our tools. Our tools are not considered as unnecessarily complex at all. This indicates that the interface of our tools is generally well-structured and it would be easy for developers to get used to our tools.

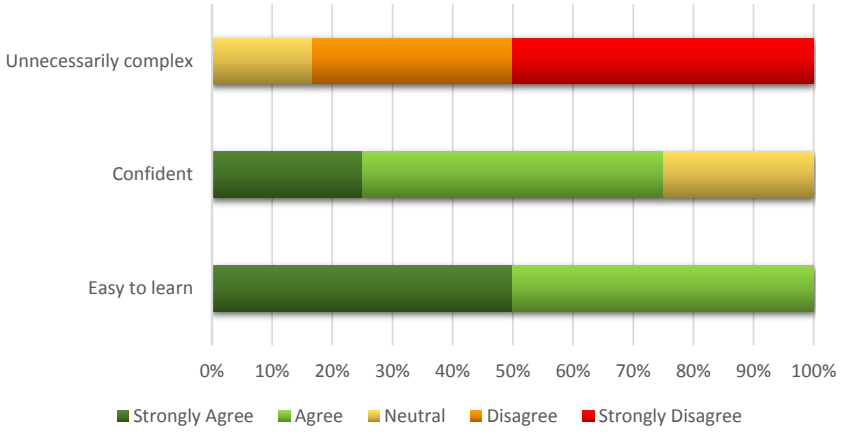

Figure 6.19: General evaluation of our tools

Figure 6.20 shows which features the participants would use for implementing and debugging cross-device applications. In general, people would rather use emulated devices rather than connecting real devices for both implementing and debugging. This is understandable as most things can be done just as well with emulated devices as with real devices. Some participants mentioned that they would not really use real devices during implementing, but that they would test their application on real devices after finishing implementing to make sure the application works fine on them. The connection features are almost unavoidable to use, thus it is surprising that some participants state that they would not use them. However, for some parts of debugging and implementing, one device might be sufficient for testing

and no connection features would be required. One participant mentioned that the connection features seem very natural and that there is no point in asking about their usefulness because it is obvious that they are useful. This indicates that the feature was indeed greatly appreciated by some participants. The shared JavaScript console is equally popular for debugging and implementing and would be used by almost all participants. Function debugging is more popular for debugging than for implementing. This corresponds to the actual results of the study and has been elaborated before. Apart from connecting real devices, the shared CSS editor is the least popular. This may be due to the fact that browsers already have quite mature CSS editors and it might be possible to test CSS on one device at a time in many cases. Also, the CSS parts of our tasks were rather simply, thus the real value of such a feature might not be obvious to the participants. While things like changing the background color of a button can easily be done on just one device, more complex CSS problems like positioning elements require more effort and look much different on different devices.

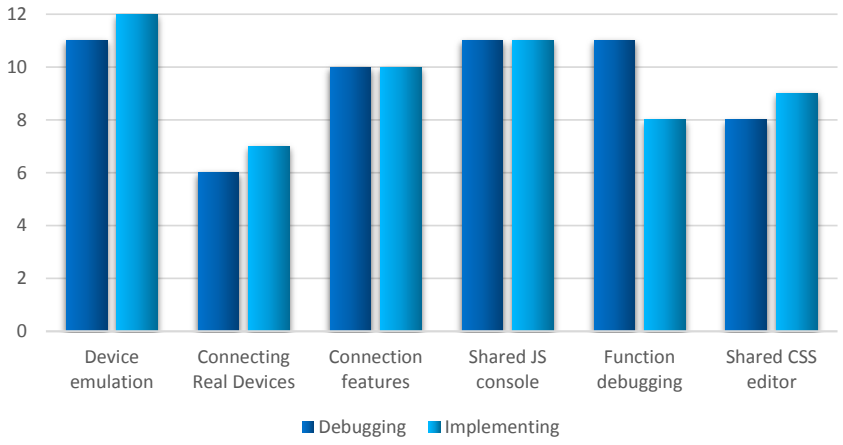

Figure 6.20: Features that the participants would use

During the study, some participants mentioned that they found device emulation very useful because they can have all devices in one place instead of having to manage different browser windows and profiles. Reloading all devices at once was also considered very useful. With multiple browser windows, each window has to be reloaded individually which makes this a much more tedious task. It was also appreciated that the devices are still connected after reloading, though this is also the case with multiple browser windows. The participants really liked function debugging, the only critique was that it does not show on which device a function is called which was fixed after the study. In general, the output aggregation of the shared JavaScript console was considered more useful than sending commands. One participant even mentioned that he would not use it to send commands, but the aggregation is very useful.

Record/replay was disabled for the user study, but one participant that had attended a presentation about our tools before, mentioned that they would find it immensely useful. They consider it as a powerful feature that could be very helpful for replicating bugs in an application and for regression testing. Often, it is not clear how to reach a bug and being able to record the set of interactions that lead to the bug and then replay them can simplify this process.

7
Conclusion

7.1 Future Work

List of Figures

List of Tables

Acknowledgements

Bibliography

[1] Sriram Karthik Badam and Niklas Elmqvist. Polychrome: A cross-device framework for collaborative web visualization. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*, pages 109–118. ACM, 2014.

[2] Pei-Yu Peggy Chi and Yang Li. Weave: Scripting cross-device wearable interaction. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3923–3932. ACM, 2015.

[3] Steven Houben and Nicolai Marquardt. Watchconnect: A toolkit for prototyping smartwatch-centric cross-device applications. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 1247–1256. ACM, 2015.

[4] Dejan Kovachev, Dominik Renzel, Petru Nicolaescu, and Ralf Klamma. Direwolf-distributing and migrating user interfaces for widget-based web applications. In *Web engineering*, pages 99–113. Springer, 2013.

[5] Michael Nebeling, Maria Husmann, Christoph Zimmerli, Giulio Valente, and Moira C Norrie. Xdsession: integrated development and testing of cross-device applications. *Proc. EICS*, 2015.

[6] Michael Nebeling, Theano Mintsi, Maria Husmann, and Moira Norrie. Interactive development of cross-device user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2793–2802. ACM, 2014.

[7] Mario Schreiner, Roman Rädle, Hans-Christian Jetter, and Harald Reiterer. Connichiwa: A framework for cross-device web applications. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, pages 2163–2168. ACM, 2015.

[8] Jishuo Yang and Daniel Wigdor. Panelrama: enabling easy specification of cross-device web applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2783–2792. ACM, 2014.