From Large Touch Walls to Tablets – a Responsive Web Framework for Large Multi-Touch Wall Applications

1st Author Name
Affiliation
City, Country
e-mail address

2nd Author Name
Affiliation
City, Country
e-mail address

3rd Author Name
Affiliation
City, Country
e-mail address

ABSTRACT

Agile software development is a highly collaborative and communication intensive method in modern software development. One of its characteristics is the usage of large pin boards with cards where the team does the whole project planning. Based on *aWall*, a web based agile collaboration platform using large multi-touch wall systems, we present a responsive web design approach that supports application interaction as a team and as an individual using both the large multi-touch wall system and as well as tablets. Combining these two interaction methods present unique challenges with respect to the UI and the flow of information. We describe *aWall*'s Web UI architecture and implementation for a concrete case study to showcase the collaborative nature of the application with special focus on its responsive design approach, interaction and information dissemination in the multi-device environment.

ACM Classification Keywords
H.5.2. Information Interfaces and Presentation (e.g. HCI): User Interfaces – *Interaction styles*; H.5.3. Information Interfaces and Presentation (e.g. HCI): Group and Organization Interfaces – *Computer-supported cooperative work*; H.5.4. Information Interfaces and Presentation (e.g. HCI): Hypertext/Hypermedia – *Architectures*

Author Keywords
Collaborative work and interaction; multi-touch; wall; tablets; responsive design; agile; sprint planning 2; web components

INTRODUCTION

In agile software development physical pin boards, so called task boards, are the main tool for agile project planning. We developed a software-based solution, the *aWall* web application, intended to replace the paper-based pin board. The system consists of two distinct device types: The wall as a direct replacement for the pin board (see Figure 1) and tablets for more input-oriented tasks.

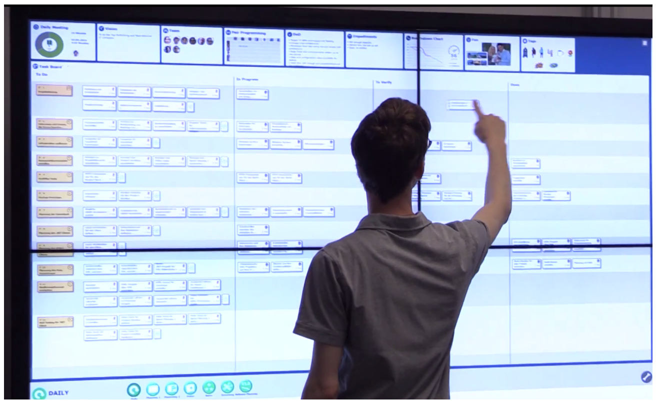


Figure 1: The *aWall* application running on a large multi-touch wall composed of 2x2 displays.

This presents unique challenges regarding the interaction with the system as a team with the wall or as an individual with the tablet. In the following sections, we give some background, present the challenges in detail and describe how we solved them.

While a physical card wall offers many advantages in respect of ease of use and flexibility, it also has major disadvantages compared to digital solutions. However, recent studies [13, 18] show that current desktop-based digital solutions can hardly fulfill the requirements concerning agile collaboration. Large digital multi-touch wall systems have the potential to be able to replace the physical boards and offering even more possibilities, amongst are also support for distributed team collaboration. However as [13] and [18] show, they must provide the natural interaction and ease of use as physical boards.

In [18] we present a concept for a large multi-touch wall based agile collaboration platform. The agile software development process comprises of different kinds of meetings, which can all be held at the wall. For some of the meetings a direct interaction with the wall, respectively with the artifacts represented at the wall, seem less appropriate. Other, smaller, input devices like individual developers tablets or even smartphones might be more appropriate. These devices allow multiple people to directly and independently interact with the artifacts on the wall and enter or modify data during the meeting more comfortable; still everybody immediately seeing the changes made.

The user requirements for the software system are described in *user stories*, a brief description of a feature in everyday language. All *user stories* are stored in a *product backlog* and are maintained by a *product owner*, who represents the customer to the development team and who is in charge of all requirements. Each sprint starts with a *sprint planning meeting*, which is divided in two sub meetings. In the sprint planning 1 (SP1) meeting, the team and the product owner select those user stories, which will be implemented in the next iteration. The selected user stories are stored in a list called the *sprint backlog*. In the second sprint planning (SP2) meeting, the team then splits the selected user stories from the *sprint backlog* into smaller technical tasks, which describe what the developer has to do to implement the user story. These tasks should not take longer than one workday to complete.

The remainder of the paper is structured as follows. First, related work is presented. Then the UI design and the challenges we faced when adapting a web application designed for large screens to smaller tablet-sized screens. In the next section, we present the architecture of the system we developed and how we tackled the challenges. The following section describes the implementation and in the end, we discuss the work and provide an outlook into our future work.

AWALL WEB APPLICATION

aWall is a web application developed for large multi-touch walls and for tablet sized computers. The application offers several interaction possibilities, ranging from multiple people working on the wall simultaneously to people using the application with tablets and the wall together. With the server of the system offering real-time updates to changes made with the application, all connected application-instances receive the modifications and are able to update the data in-place without reloading the website.

Figure 1 shows our setup using 2x2 full-HD monitors forming a 4K display. Interactions can span monitors and are detected using an infra-red overlay in the frame around the four monitors.

It is based of configurable workspaces, that are composed of smaller widgets and one large main widget. The main widget is always tailored to the a specific task that coincides with a meeting of the agile process. The smaller widgets offer additional functionality and interaction possibilities.

RELATED WORK

An often mentioned and typical setting used in the agile process is the taskboard. It shows the current progress of the project iteration at a glance and is used in the daily stand-up meetings. A digital tabletop application is introduced for agile planning meetings in [9]. The tabletop only has a single-touch display. Thus only one person can interact with the system at a time and without advanced gestures that involve multiple fingers like pinch-to-zoom. In [21], a cooperative multi-touch taskboard for large displays is proposed. The paper discusses how interactive tabletops and surfaces can be applied to the agile taskboard for the daily stand-up and other meetings and evaluates gestures known from smartphones for big tabletops and group settings. [14] introduces a task assignment system for tabletop computers that consists of two components: the tabletop application and a separate Android companion application. The companion application shows changes made on the tabletop in real-time and offers some limited interaction capabilities.

A multi-surface communication and collaboration platform using a wall, tabletops and tablets for an emergency operations center is presented in [5]. The system is spatially aware of all the devices and people in the room using multiple Kinects. This allows the use of gestures like a flick on a tablet towards the wall to share information. The device types are used for different scenarios: The tablet is used for personal and private interaction with colleagues. Whereas the tabletop is a semi-private workspace for collaboration and the wall is a public information radiator, aggregating multiple data sources. The authors developed native applications for the wall, the tabletop and the tablet. Thus multiple independent applications were developed with different code bases using in part the same information provided by servers. With *aWall*, we do not want to maintain multiple code bases but one application to handle all the devices. This includes the viewing experience corresponding to the device type of what is shown and how it is presented.

One of the most significant challenges with the emergence of all the new devices like smartphones, tablets and phablets in the last years, is device and platform fragmentation. Developing a unique experience for each device would mean to have an application per device type. This also means multiple times the work if something needs to be changed. By using a single code base that serves a multitude of devices, a maintenance nightmare can be avoided. For the web, HTML5 and CSS3 bring many advancements to create a single code base for different device and screen sizes. The approach using a single application is called responsive web design (RWD). In [19], the author presents a case study of a mobile-first design process used to create a hybrid smartphone and tablet application for transaction banking.

What responsive design does not take into account is the proximity of the user. The authors in [23] developed an example implementation of a responsive web application that adjusts the information displayed according to the user's distance to the display.

In [4], an web-based application framework is presented for collaborative visualizations across multiple devices. The framework allows to synchronize UI state of new or legacy web applications with minimal overhead.

UI DESIGN AND CHALLENGES

rwd: diff. between normal sites and aWall

tablet is to create input, not only for consumption

collaborative aspect: see changes of others

howto bring to tablet

Responsive Design

The RWD [17] approach offers a solution to create one application that provides an optimal viewing experience across a wide variety of different devices with various display-resolutions. An important concept of RWD is mobile-first [24]. It is an approach to give the mobile site a higher priority and design it with the constraints and capabilities of small-screen devices in mind. The approach is based on the assumption that adding content to a website designed for small-screen devices is easier than removing content from the large desktop layout to fit the smaller screen.

For the UI, the web application uses an approach called RWD that builds on three cornerstones [17]:

- Relative units: The style and layout definitions should use relative units like percent instead of fixed units like pixel. A pixel is on a 1080p smartphone display does not have the same physical size as a pixel on a 1080p 24" desktop display.

- Flexible images: The images are sized in relative units but there may also be multiple versions of the same image. For example, a low resolution image for small-screen devices and a higher resolution image for larger devices.

- Media Queries: Media queries are the main cornerstone of RWD. They allow us to define breakpoints with conditional statements in the stylesheet at which different CSS style definitions take effect.

Media Queries have been introduced in CSS3 and became a W3C standard in June 2012 [20]. They allow the content of a website to adapt to the conditions of the web browser. The most prominent example of such a condition is the screen resolution. By defining break-points, using for example the width of the viewport, the statement defines the style of the given Document Object Mode (DOM) element. Let's assume we have a simple element with a title in a h1-element and a text in a p-element. The media query could define that the p-element is hidden when the height of the viewport is less than 500 pixels.

General Layout

The agile process has multiple collaborative meetings that deal with different aspects of the project. The UI of *aWall* in Figure 2 is built around workspaces configured with independent widgets that can be used by several workspaces. For each meeting there is a workspace configured with the most appropriate widgets. Each workspace consists of a main widget that takes most of the available space, numerous independent widgets in a panel at the top of the screen, called info-view, and the workspace-menu with options to configure the workspace.

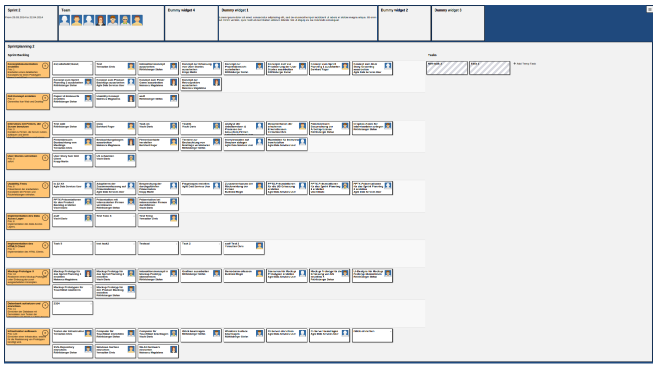


Figure 2: UI layout of *aWall* with a main widget for SP2 as seen on the multi-touch wall.

The main widget occupies the largest part of the UI and is the main working area. For each meeting of the agile process there is a specialized main widget that shows exactly what is needed and offers tailored interaction methods. Supplemental interaction with additional artifacts is provided by the widgets in the info-view. Each main widget is responsible for the responsive behavior of its content to show the appropriate amount of information depending on the available space.

The info-view is the horizontal panel at the top with the smaller widgets. The widgets in the info-view can have different widths from standard width to multiple-times the standard-width. The number of widgets to display is calculated by the application and depends on the width of the viewport. If a currently hidden widget is enabled, the widget in the last position is replaced with the enabled widget. This ensures that the widgets always have enough space. Widgets that do not fit can be accessed by the workspace-menu, where each widget can be enabled or disabled.

The widgets in Figure 2 are in a docked position but can be undocked by dragging them out of the info-view. When a widget is dragged out of the info-view (undocked) the space is closed whereby all widgets to the right move one position to the left. If a widget is to be docked again, the widget's space is freed up with all widgets to the right moving one position to the right. With this, the widgets can be freely arranged by the user. Once undocked, the widget can be moved around the viewport and resized by either using the mouse or the pinch-to-zoom gesture on a multi-touch display.

The widgets can have multiple views for different sizes when being resized. In the docked position in the info-view the widgets show the default view on larger screens. On smaller screens like on a tablet, only the title of the widget is shown, henceforward called title-only view. The default size with the same functionality as on larger screens is shown when it is undocked. Besides the title-only and the default view, a widget can offer an additional full view and an edit mode. The full view is intended to show more information on the widget's subject or offer additional interaction functionality. It is triggered by specifying the breakpoints for the width and height. Once the widget has exceeded both breakpoints while resizing, the full view is shown. If a widget has an edit mode, an edit icon is displayed in the top-right corner of the widget. There are two more buttons on all widgets: one to reset the size to the default view and one that docks the widget in the info-view.

The workspace-menu is the small button in the top right corner. It allows to control which widgets are enabled and disabled with toggles, to switch to other workspaces for the other meetings and to save or reset the current workspace configuration. As mentioned above, the widgets in the info-view can be rearranged. The new arrangement can be saved by the user or reset to the default.

SP2 Layout

The main widget for the SP2 meeting, as depicted in Figure 2, shows the *sprint backlog* on the left and a panel on the right for unassigned tasks. Unassigned tasks are specific to our solution and they represent the paper-based card written by a team member that has not been discussed by the team and thus has not been assigned to a user story yet. This is where the tablets comes into play. Each team member has a tablet and creates tasks with it. As soon as the task is created on a tablet, it is sent to all devices connected to the system and pops up on the wall. Then it is discussed and finally either assigned to a user story or discarded.

On the wall, everything including the *sprint backlog* and the unassigned tasks can be shown simultaneously. The tablet does not have enough space to show all the information like on the wall. Thus we split the widget into two views: The first view (Figure 4a) shows all the user stories of the *sprint backlog* in a grid. When the user clicks, respectively selects a user story, the second view is shown, called detail view (Figure 4b). There the selected user story with all its tasks and the list of unassigned tasks is displayed.

Challenges

concurrent modifications

platform independence

The two device types the system uses, namely the wall and hand-held tablets, have significantly different screen resolutions and physical dimensions. An important goal driving our design was the ability to see everything without scrolling. Supporting these different devices with one code base and adhere to the no scrolling requirement presents some challenges:

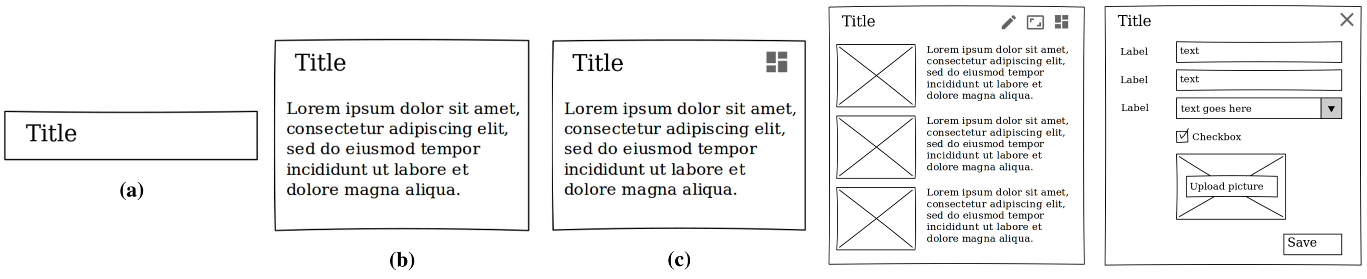
- **Retain functionality:** Even though the screen of a tablet is not as small as a smartphones', the available space is significantly smaller than on the large multi-touch wall. A big challenge was to retain the same functionality on the tablet as on the wall. Often when going from a large screen to a smaller one, functionality is sacrificed in order to keep it organized and legible.
- **Reduce space of non-essential elements:** A lot of vertical space is used up by the info-view widgets. They offer auxiliary functionality, but it is not absolutely essential that they are completely visible the whole time in their default view. Thus, we created the title-only view for the widgets. That way, the widgets are still in the same place and visible, but have to be dragged out of the info-view to be usable. Once the widget is undocked, it enlarges to the default size and it can be used like on the wall.
- **Viewing experience: focused vs. overview:** In our collaborative scenario of the SP2 meeting, the wall and the tablet serve different needs. The wall's duty is to give the whole team the overview of the whole sprint, while the tablet's focus lies with the individual to create new tasks for the currently discussed user story. This requires the information displayed on the device to be specific to its purpose. While the complete *sprint backlog* is important on the wall, it is not relevant for creating a task for a user story. Thus, we split the main widget for the SP2 meeting into two views on smaller screens as depicted in Figure 4.

IMPLEMENTATION

Frameworks

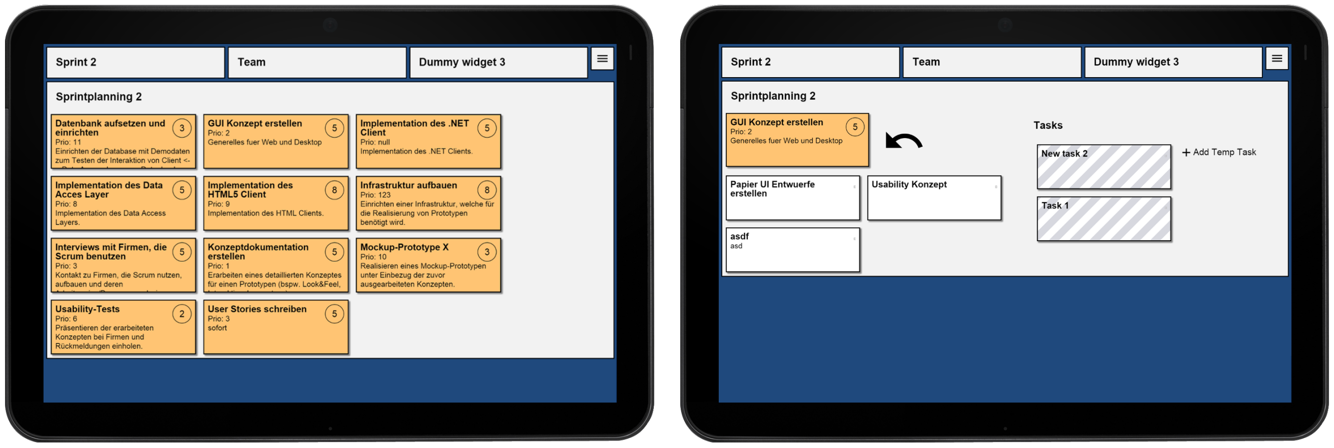
The *aWall* web application uses Polymer [3], a library that abstracts WebComponents and allows to define custom elements using declarative HTML tags instead of pure Javascript code. This makes it easy to write new custom elements without the huge amount of boilerplate code that is required when using the native Javascript APIs directly. For multi-touch gestures like pinch-to-zoom, drag&drop and resizability of elements, *aWall* uses the interact.js [1] library.

Responsive Widgets


Title
Title
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Title
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Title
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Title
Label
text
Label
text
Label
text goes here
Checkbox
Upload picture
Save

(a) **(b)** **(c)** **(d)** **(e)**

Figure 3: The different views for info-view widgets.



Datenbank aufsetzen und einrichten 3
Prio: 11
Einrichten der Database mit Demodaten zum Testen der Interaktion von Client <-

GUI Konzept erstellen 5
Prio: 2
Generelles fuer Web und Desktop

Implementation des .NET Client 5
Prio: null
Implementation des .NET Clients.

Implementation des Data Acces Layer 5
Prio: 8
Implementation des Data Access Layers.

Implementation des HTML5 Client 8
Prio: 9
Implementation des HTML Clients.

Infrastruktur aufbauen 8
Prio: 123
Einrichten einer Infrastruktur, welche für die Realisierung von Prototypen benötigt wird.

Interviews mit Firmen, die Scrum benutzen 5
Prio: 3
Kontakt zu Firmen, die Scrum nutzen, aufbauen und deren

Konzeptdokumentation erstellen 5
Prio: 1
Erarbeiten eines detaillierten Konzeptes für einen Prototypen (bspw. Look&Feel,

Mockup-Prototype X 3
Prio: 10
Realisieren eines Mockup-Prototypen unter Einbezug der zuvor ausgearbeiteten Konzepten.

Usability-Tests 2
Prio: 6
Präsentieren der erarbeiteten Konzepten bei Firmen und Rückmeldungen einholen.

User Stories schreiben 5
Prio: 3
sofort

GUI Konzept erstellen 5
Prio: 2
Generelles fuer Web und Desktop

Papier UI Entwuerfe erstellen

Usability Konzept

asdf
asd

New task 2

Task 1

(a) Overview **(b)** Detailed view

Figure 4: The two views of the split up SP2 user interface for smaller displays.

All the widgets use the custom element *awall-widget*. It gives all the widgets the features like drag & drop and resizability. The four views of the widgets are also implemented by custom elements, namely *awall-widget-title*, *awall-widget-size-default*, *awall-widget-size-full* and *awall-widget-editable*. Listing 1 shows the pattern on how a widget is implemented using the aforementioned elements. The title for a widget is given as the parameter to the attribute *value* for the element *awall-widget-title*. For the other three view elements, the content is more comprehensive than just a string and thus is enclosed in the element's tags as indicated using HTML comment syntax.

```
<awall-widget>
  <awall-widget-title value="Team" />

  <awall-widget-size-default>
    <!-- Content for default view -->
  </awall-widget-size-default>

  <awall-widget-size-full showHeight="500"
      showWidth="400">
    <!-- Content for full view -->
  </awall-widget-size-full>

  <awall-widget-editable showHeight="500"
      showWidth="400">
    <!-- Content for edit mode -->
  </awall-widget-editable>
</awall-widget>
```

Listing 1: HTML elements for the widget's different views.

Not all the view elements must be used of course. A widget does not need to have a full view or an edit mode. Both are optional and can be configured using the two parameters *showHeight* and *showWidth* to define their breakpoints at which point they activate and the default view is hidden. The edit mode is not shown automatically, but is represented by an edit button appearing when both breakpoints have been exceeded.

The *awall-widget* custom element implements the features that all widgets have and controls when which view is displayed. Listing 2 depicts part of the template of the *awall-widget* custom element. The *content* element is used in Shadow DOM and represents an insertion point for DOM-trees. That means that everything between the start and end tag of *awall-widget-size-default* in Listing 1 is inserted in the position of the content tag that selects the *awall-widget-size-default* in line 4 of Listing 2.

```
1 <content select="awall-widget-title" />
2
3 <div id="extendedContent">
4   <content select="awall-widget-size-default" />
5   <content select="awall-widget-size-full" />
6   <content select="awall-widget-editable" />
7 </div>
```

Listing 2: Part of the awall-widget custom element's template using insertion points for the different HTML tags.

All elements in the template in Listing 2 except *awall-widget-title* are enclosed in an div-element whose visibility is controlled by the media query shown in Listing 3. As long as the browsers viewport height is smaller than the defined *max-height* breakpoint, only the title of the widget is shown. When the widget is dragged out of the info-view, the default view becomes visible. The widget can now be resized using the pinch-to-zoom gesture or the mouse. During resizing, the width and height of the widget are compared to the *showHeight* and *showWidth* breakpoints defined for the full view and the edit mode.

```
@media only screen and (max-height: 40rem) {
  #extendedContent {
    display: none;
  }
}
```

Listing 3: Media Query hiding all the views except the title when the screen is small enough.

SP2 Widget

As mentioned before, each widget is responsible for its own responsive behavior. The main widgets use the *awall-widget* but with drag & drop and resizability disabled. The main widget for the SP2 workspace, as depicted in Listing 4, has two templates that are controlled by the boolean variable *isSmallScreenHeight*. The variable is set by the application's Javascript code and is accessible through Polymer's data binding mechanism. Depending on the value of the variable one of the two inactive templates is activated. The views are all compartmentalized into their own custom elements and thus the two templates in Listing 4 only contain one element (results in Figure 4a) for the small-screen view and two elements for the large-screen view, that results in Figure 2. The *awall-userstory-cardlist* element for small screens handles its behavior to switch to the detailed view (Figure 4b) itself. The title for the widget at the top of the listing is visible on any screen and thus is not enclosed in a template.

```
<h2>Sprintplanning 2</h2>

<template if="{{isSmallScreenHeight}}">
  <awall-userstory-cardlist />
</template>

<template if="{{!isSmallScreenHeight}}">
  <awall-sprintbacklog />
  <awall-temptasks />
</template>
```

Listing 4: How the SP2 main widget decides which view to show.

DISCUSSION

Hardware

bezel between the monitors in our setup

inaccurate touch-point tracking

UI Design

orientation changes on tablets

handwriting

virtual keyboard hiding content

Responsive Design

In combination with new best practices like RWD, enabled by the new standards, the development of new and adaptable web applications becomes more approachable. Especially media queries allow us to detect and react to changes of factors like the screen resolution that are important for the rendering of content. They allow us to either hide elements or render them using a different style (e.g. smaller font). Together with the HTML template element, that allows to create inactive DOM-trees, conditional templates can be defined that are shown depending on a boolean variable set by a media query.

The main widget for the SP2 workspace presented in this paper uses such conditional templates to present its content depending on the size of the browsers viewport dimensions. The views including their specific behavior for both display sizes we implemented, namely the large wall and the tablet, have to be developed separately. Thanks to the use of custom elements even for the REST API, the separate views only have to include a HTML element in their HTML code to access it. The data is then available in the whole element using data binding provided by Polymer.

Implementation

The framework we developed with *aWall* shows, that current web technologies and emerging new web standards make it possible to build an application that serves different devices with disparate screen resolutions and physical size using a single code base.

Through its design of using loosely-coupled web components, specifically custom elements with Shadow DOM, there are no dependencies between the widgets. This allows a developer to create new widgets without worrying about whether there are going to be problems with other widgets like giving an element an ID that has already been used or having its stylesheet definitions (CSS) affect other elements of other widgets.

By working with the newest technologies, there is always the problem with rapidly evolving and changing APIs and libraries. Since we started to develop *aWall*, the Polymer library evolved and reached version 1.0 with lots of changes. It is going to take some time to port the current application to the new version.

Another inconvenience of using the newest web technologies is that the number of browsers implementing all those new features is limited. Currently, *aWall* only runs natively on the Chrome and Chromium web browsers. Most other browsers are in progress of implementing the specifications for web components. For all those browsers not supporting any or all the features, there are so called polyfills available. Polyfills are Javascript libraries, that add the missing APIs to the browser. One big disadvantage of using polyfills is their lower performance compared to the native implementations.

CONCLUSION AND FUTURE WORK

In this paper we presented how we designed and developed a responsive web application using a single code base that provides the appropriate viewing experience on the large wall as well as the much smaller screen of a tablet without sacrificing functionality. As an example, we used the SP2 meeting of the agile process, where multiple people create tasks for a user story which are then discussed together on the wall, to showcase the responsive and collaborative nature of *aWall*.

After the design of the large-screen UI, we conducted some live-action interviews with developers to discover if the wall could be something they want to work with and to get input on how to improve it. In the future, we are going to build on the input we received and would also like to extend the wall to work with distributed teams, where the movement of a task for instance is mirrored on other walls.

REFERENCES

1. Taye Adeyemi. 2015. interact.js. `http://interactjs.io`. (2015). Accessed: 2015-05-30.
2. Atlassian. 2015. JIRA – The flexible and scalable issue tracker for software teams. `https://www.atlassian.com/software/jira`. (2015). Accessed: 2015-06-16.
3. Polymer Authors. 2015. Polymer. `https://www.polymer-project.org`. (2015). Accessed: 2015-05-30.

4. Sriram Karthik Badam and Niklas Elmqvist. 2014. PolyChrome: A Cross-Device Framework for Collaborative Web Visualization. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces (ITS '14)*. ACM, New York, NY, USA, 109–118. DOI: http://dx.doi.org/10.1145/2669485.2669518

5. Apoorve Chokshi, Teddy Seyed, Francisco Marinho Rodrigues, and Frank Maurer. 2014. ePlan Multi-Surface: A Multi-Surface Environment for Emergency Response Planning Exercises. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces (ITS '14)*. ACM, New York, NY, USA, 219–228. DOI: http://dx.doi.org/10.1145/2669485.2669520

6. Alistair Cockburn. 2001. *Agile Software Development*. Addison-Wesley Professional.

7. WebComponents.org contributors. 2015. WebComponents.org – a place to discuss and evolve web component best-practices. http://webcomponents.org. (2015). Accessed: 2015-05-27.

8. I. Fette and A. Melnikov. 2011. The WebSocket Protocol. https://tools.ietf.org/html/rfc6455. (December 2011). Accessed: 2015-05-27.

9. Y. Ghanam, Xin Wang, and F. Maurer. 2008. Utilizing Digital Tabletops in Collocated Agile Planning Meetings. In *Agile, 2008. AGILE '08. Conference*. 51–62. DOI: http://dx.doi.org/10.1109/Agile.2008.13

10. Dimitri Glazkov. 2014. Custom Elements – W3C Working Draft. http://www.w3.org/TR/custom-elements. (16 December 2014). Accessed: 2015-06-17.

11. Dimitri Glazkov and Hayato Ito. 2014. Shadow DOM – W3C Working Draft. http://www.w3.org/TR/shadow-dom. (17 June 2014). Accessed: 2015-06-17.

12. Dimitri Glazkov and Hajime Morrita. 2014. HTML Imports – W3C Working Draft. http://www.w3.org/TR/html-imports. (11 March 2014). Accessed: 2015-06-17.

13. Stevenson Gossage. 2014. Understanding the Digital Card Wall for Agile Software Development. (May 2014). Master thesis.

14. Benedikt Haas and Florian Echtler. 2014. Task Assignment and Visualization on Tabletops and Smartphones. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces (ITS '14)*. ACM, New York, NY, USA, 311–316. DOI: http://dx.doi.org/10.1145/2669485.2669538

15. Ian Hickson, Robin Berjon, Steve Faulkner, Travis Leithead, Erika Doyle Navara, Edward O'Connor, and Silvia Pfeiffer. 2014. HTML5 – W3C Recommendation. http://www.w3.org/TR/html5/scripting-1.html#the-template-element. (28 October 2014). Accessed: 2015-06-17.

16. Martin Kropp and Andreas Meier. 2015. Swiss Agile Study 2014. http://www.swissagilestudy.ch/files/2015/05/SwissAgileStudy2014.pdf. (May 2015). Accessed: 2015-05-29.

17. Ethan Marcotte. 2011. *Responsive Web Design*. A Book Apart.

18. Magdalena Mateescu, Martin Kropp, Roger Burkhard, and Carmen Zahn. 2015. aWall Designing Socio-Cognitive Tools for agile team collaboration with large Multi-Touch Wall Systems. (October 2015). To be submitted.

19. Sumit Pandey. 2013. Responsive Design for Transaction Banking - a Responsible Approach. In *Proceedings of the 11th Asia Pacific Conference on Computer Human Interaction (APCHI '13)*. ACM, New York, NY, USA, 291–295. DOI: http://dx.doi.org/10.1145/2525194.2525271

20. Florian Rivoal, Hkon Wium Lie, Tantek elik, Daniel Glazman, and Anne van Kesteren. 2012. Media Queries – W3C Recommendation. http://www.w3.org/TR/css3-mediaqueries. (19 June 2012). Accessed: 2015-05-27.

21. Jessica Rubart. 2014. A Cooperative Multitouch Scrum Task Board for Synchronous Face-to-Face Collaboration. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces (ITS '14)*. ACM, New York, NY, USA, 387–392. DOI: http://dx.doi.org/10.1145/2669485.2669551

22. Andrew Stellman and Jennnifer Greene. 2014. *Learning Agile*. O'Reilly Media.

23. Ryan Sukale, Olesia Koval, and Stephen Voida. 2014. The Proxemic Web: Designing for Proxemic Interactions with Responsive Web Design. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication (UbiComp '14 Adjunct)*. ACM, New York, NY, USA, 171–174. DOI: http://dx.doi.org/10.1145/2638728.2638768

24. Luke Wroblewski. 2011. *Mobile First*. A Book Apart.