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 characteristic is the usage of large pin boards with cards where the team does the whole project planning. In this paper we present *aWall*, a responsive web application framework that supports the collaborative teamwork using large multi-touch wall systems and tablets for interacting with the system as a team and as an individual. Combining these two interaction methods present unique challenges with respect to the UI and the ow 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

Responsive design; agile; second sprint planning; collaborative work; multi-touch; wall; tablets; web components

INTRODUCTION

Agile promises that a project is finished on time, delivers high quality software, well constructed and maintainable code and that agile teams make the users happy [17]. The core of agile is a set of practices and methodologies around self-organizing teams relying on face-to-face communication [4]. According to recent study in Switzerland, 70% of all IT companies practice agile software development and 83% of all IT professionals [12]. Agile software development projects are split into short iterations usually lasting between 2-4 weeks. In Scrum these iterations are called sprints. 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. In the second sprint planning (SP2) meeting, the team then splits the selected user stories 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 traditional and most often used way to this, is to write the user stories and tasks on cards and attach them on a large pin or magnetic board. The setup on the board is called a *taskboard*. This board is typically divided into three columns (ToDO, in progrees, done) which indicate the progress of the individual tasks. This allows the team to easily show the state and progress of the current sprint. While these physical card wall offers many advantages with respect of ease of use and flexibility, it also has major disadvantages compared to digital solutions. However studies that current desktop based digital solutions can hardly fulfill the requirements concerning agile collaboration [10] and [13] .

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 [10] and [13] shows, they must provide the natural interaction and ease of use as physical boards. In [13] we present a concept for a large multi-touch wall based agile collaboration platform. The agile software development process has comprises different kind of meetings, which can all be executed 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 the multiple people directly and independently interact with the artifacts on the wall and enter new data or modify existing data during the meeting more comfortable via the tablet device; still everybody immediately seeing the changes made. In the collaborative SP2 meeting, where everyone helps in creating tasks for the next iteration,

tablets can be used to create the tasks in an interconnected web application.

The responsive web design (RWD) 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. A term often heard in the context of RWD is mobile-first. 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. Since we already developed the UI for the large multi-touch wall, our challenge was to find a suitable design for the tablet without sacrificing functionality.

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.

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 glace and is used in the daily stand-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 [16], 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. [11] introduces a task assignment system for tabletop computers that consists of two components: the tabletop application and an Android companion application. The companion application shows changes made on the tabletop in real-time and offers some limited interaction possibilities.

In [6], the authors identified and evaluated seven heuristics for managing informative workspaces in agile environments. In the follow-up paper [7], informative workspace patterns are generated and validated.

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 is impractical. 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 [14], 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 [18] developed an example implementation of a responsive web application that adjusts the information displayed according to the user's distance to the display.

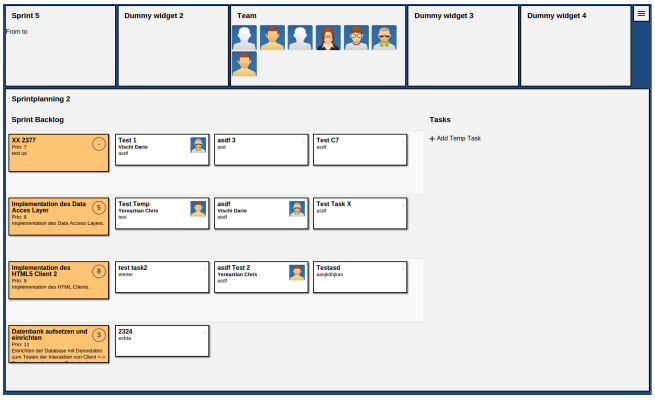


Figure 1: UI layout of Awall on a device larger than a tablet but smaller than the wall.

In [3], 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

The UI of Awall in Figure 1 is built around workspaces that are configured with different widgets to be most useful for a particular meeting in the agile process. The layout consists of the info-view, the workspace menu and the main widget.

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. This ensures that the widgets always have enough space. The widgets that do not fit can be accessed by the workspace menu, where each widget can be enabled or disabled.

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 and to save or reset the current workspace configuration.

The main widget is the largest part of the UI and is the main working area. For the daily stand-up meeting, it displays the taskboard. For each meeting of the agile process there is a specialized main widget that shows exactly what is needed and offers tailored interaction methods. Each main widget is responsible for the responsive behavior if its content to show the appropriate amount of information depending on the available space. Since we have a very large screen for the wall and a smaller screen for the tablets, we have two layouts for the SP2 main widget. The layout for the large screen shows all the available information at once. That specifically means all user stories including their already created and assigned tasks plus all tasks that were just created but have not been assigned to a user story yet. For tablet-sized displays, the small-screen layout consists of two views. The first view

shows all user stories in a grid. When the user clicks respectively selects a user story, the second view is shown. In the detail view, the selected user story with all its tasks and the list of unassigned tasks is displayed.

The widgets in Figure 1 are in a docked position but can be undocked by dragging them out of the info-view. 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 different layouts for different sizes when being resized. In the docked position in the info-view the widgets show the default layout on larger screens. On smaller screens like on a tablet, only the title of the widget is shown. The default size with the same functionality as on larger screens is shown when it is undocked. Besides the title-only and the default layout, a widget can offer an additional full layout and an edit mode. The full layout 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 layout 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 layout and one that docks the widget in the info-view.

Even though the screen of tablet is not as small as a smartphone, bringing an UI designed for a large wall to such a device presents some challenges. An important aspect was to retain the same functionality on the tablet as on the wall.

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 layout. Thus, we created the title-only layout for the widgets. That way the widgets are still in the same place and visible, but have to be undocked to use. When the widget is undocked it enlarges to the default size and it can be used like on the wall. By thinking about the widgets and how to use them, we also extended the functionality with the full layout and the edit mode. In our collaborative scenario of the SP2 meeting, the wall and the tablet serve different needs. While the wall is used for overview in the collaborative discussion, the tablet is used by an individual to create the task for the currently discussed user story. That means that the layout and the amount of information displayed in the main widget can differ. So for our scenario, the full list of user stories is not important for the individual while creating a task on the tablet. Thus, we have split the widget into two views, an overview (Figure 2a) and a detail view (Figure 2b), as mentioned above.

ARCHITECTURE

The whole system consists of four components:

- Awall web application: The Awall application is a single-page application (SPA) and is built using HTML, Javascript and CSS.
- A web server: A web server delivers the Awall web application files to clients.
- Application server: An application server that provides a REST API for Awall to retrieve the data about the sprints, user stories, tasks and team members. It also provides a WebSocket endpoint for real-time updates.
- A data backend: The data used by the REST API is provided by a backend. Currently, the system uses a JIRA backend. JIRA is a bug, issue and project management software used by many software development teams around the world.

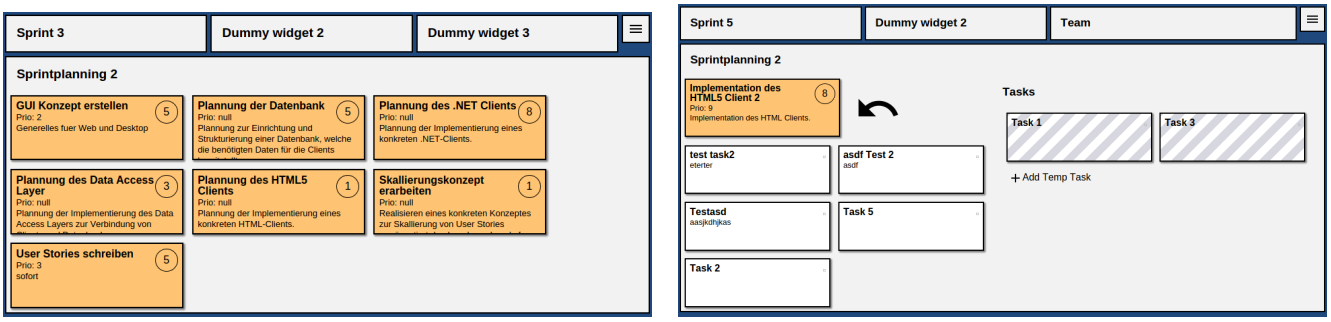
Awall is a SPA, that means that the website is not rendered by an application server and then sent to the client but is rendered completely on the client. It also means that the website does not reload, even when another page of the website is loaded. The resources like images and scripts are loaded dynamically when needed and added to the application at runtime. The information about sprints, user stories and tasks is retrieved from the application server over the REST API. The data is delivered in a light-weight format called JSON.

Awall uses Web Components [5], a set of four standards currently being produced as a W3C specification. Those four specifications offer the following features that can be used together or individually:

- Custom Elements: Define and use new HTML elements. That ranges from simple format-elements to complex elements with functionality like for example the HTML5 video element.
- HTML Imports: Include and reuse HTML documents. Allows to define custom elements in a separate file, import it and then use those custom elements. Similar to how CSS and Javascript files can be imported.
- Templates: Declare inert Document Object Mode (DOM)-trees that can be reused to instantiate custom elements.
- Shadow DOM: Encapsulate functionality and style in DOM-trees. CSS definitions are by default global no matter where they are defined. By encapsulating the style definitions in a Shadow DOM, they become local and do not affect any other DOM elements.

The application is composed of nested custom HTML elements and does not follow the more traditional MVC pattern. This approach goes more in the direction of component-based software engineering that emphasizes the separation of concerns. The custom elements can be defined in separate HTML documents and be reused in multiple web applications by importing the document. This also means that the elements can be loosely-coupled and independent. Even the access to the REST API is encapsulated in an element. To use the REST API, another element only needs to instantiate the appropriate custom element by inserting the corresponding HTML tags into its own HTML.

The workspaces that make up the web application are configured in JSON files. Apart from a name and the description for the workspace, a path, a layout and widgets must be defined. The path is the fragment identifier '#' of the URL including variables that are required by the given workspace.



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

Figure 2: The two views of the split up SP2 user interface for smaller displays.

For example ”/project/:projectId/sprint/:sprintId/sp2” is the path for the SP2 worksapce. ’:projectId’ and ’:sprintId’ are the variables required for that workspace and the complete URL for this workspace is ”http://example.com/#/project/ATBD/sprint/10204/sp2”. A workspace must define which layout to use. It defines how the widgets are arranged and how many widgets besides the main widget are shown. It also defines whether additional visual elements like the workspace menu are visible. Currently there are two layouts, more can be added if desired. One is for simple selection pages like selecting the sprint that only has a main widget. The other is the workspace layout as seen in Figure 1 with the widgets panel at the top and the main widget. The widgets property defines which widgets occupy which position in the layout. Each layout must at least have a main widget. The others are configured with a positional attribute that defines the order in the top panel, a visibility attribute and a width attribute.

The web application loads all workspace configuration files first, then determines which workspace to load by examining fragment identifier of the URL. The requested workspace is then loaded by importing the HTML file that defines the layout and all the defined widget’s HTML files. Some custom elements are used in multiple widgets and are thus defined in their own HTML document. For example the custom element that represents a task, called taskcard, is defined in taskcard.html and imported by the widgets that use it. Those elements are imported directly by using the HTML Imports specification. Once the layout and the widgets have been loaded, the widgets are instantiated and inserted into the layout according to their configuration attributes like position and width. The layout is then set as the body of the HTML DOM-tree.

The REST API alone only offers communication by means of request and response. To make the application more dynamic, the application server also offers a WebSocket [8] endpoint. WebSocket is full-duplex communication protocol designed to be implemented by web browsers and web servers. It allows both participants (client and server) to send asynchronous messages for real-time communication. Once the client has established the connection, it is persistent until closed explicitly. WebSocket allows the web application not only to recognize changes made by itself, but also changes made by other instances of the web application on different devices no matter where they are. This works because all instances of the web application establish a WebSocket connection to the application server when started. If, for example a task has been modified, the task is saved by the application server in the backend and then the information is propagated to all clients over the WebSocket connection. When the web application receives an update, it checks whether the current workspace uses this task. If that is the case, the task is updated in-place without reloading the page. Has a task been created for a user story, it will pop-up on all connected devices in real-time.

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

- 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 is not the same 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 [15]. 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 DOM element. Let’s assume we have a simple element with a title in 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.

IMPLEMENTATION

The implemented Awall web application uses the Polymer [2], 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 a 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.

The different layouts of the widgets are implemented using custom elements. All widgets must place their content in the custom element tag <awall-widget>. This gives them features like resizability and drag&drop. For the title-only mode to work, the widget must use the element <awall-widget-title>to define its title. The default content is also contained in the element<awall-widget-size-default>. To fully utilize the resize functionality and support the full layout of a widget, the content for this layout must be placed in the element <awall-widget-size-full>. The edit mode also has its own element: <awall-widget-editable>.

All elements except title-only are enclosed in an element whose visibility is controlled by a Media Query. As long as the browsers viewport height is smaller than a defined breakpoint, only the title of the widget is shown and the height of the widget is only as high as the title. During resizing, the width and height of the widget are compared to the breakpoints. Once both breakpoints have been exceeded, the default layout is is hidden and the full layout is displayed. The edit mode is controlled by the user by means of a button in the widget.

DISCUSSION AND FUTURE WORK

The framework we developed with Awall show, that current web technologies and emerging new web standards make it possible to build a responsive web application for different sized devices. The fact that it is an SPA and uses asynchronous messages for updates over WebSockets allows the application to run on multiple devices with the displayed data always up-to-date without the need to reload the website. The framework is also designed to be easily extendable with additional workspaces, layouts and widgets. Through its design using loosely-coupled components or in the web context custom elements, there are no dependencies between the widgets.

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 movements of a task for instance are mirrored on other walls.

REFERENCES

1. Taye Adeyemi. 2015. interact.js. http://interactjs.io. (2015). Accessed: 2015-05-30.
2. Polymer Authors. 2015. Polymer. https://www.polymer-project.org. (2015). Accessed: 2015-05-30.
3. 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
4. Alistair Cockburn. 2001. *Agile Software Development*. Addison-Wesley Professional.
5. WebComponents.org contributors. 2015. WebComponents.org – a place to discuss and evolve web component best-practices. http://webcomponents.org. (2015). Accessed: 2015-05-27.
6. R. de Melo Oliveira and A. Goldman. 2011. How to Build an Informative Workspace? An Experience Using Data Collection and Feedback. In *Agile Conference (AGILE), 2011*. 143–146. DOI: http://dx.doi.org/10.1109/AGILE.2011.33
7. Renan de Melo Oliveira, Alfredo Goldman, and Claudia O. Melo. 2013. Designing and Managing Agile Informative Workspaces: Discovering and Exploring Patterns. In *System Sciences (HICSS), 2013 46th Hawaii International Conference on*. 4790–4799. DOI: http://dx.doi.org/10.1109/HICSS.2013.171
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. Stevenson Gossage. 2014. Understanding the Digital Card Wall for Agile Software Development. (May 2014). Master thesis.
11. 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
12. 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.
13. Roger Burkhard Carmen Zahn Magdalena Mateescu, Martin Kropp. 2015. aWall Designing Socio-Cognitive Tools for agile team collaboration with large Multi-Touch Wall Systems. (October 2015).
14. 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

15. 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.

16. 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

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

18. 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