Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 340

# Wearable Notifications in Industrial Environments

Josip Ledić

| | |
|---|---|
| **Course of Study:** | Softwaretechnik |
| **Examiner:** | Jun.-Prof. Dr. Niels Henze |
| **Supervisor:** | Dr. Stefan Schneegaß,<br>Dipl.-Inf. Dominik Weber |
| **Commenced:** | May 31, 2016 |
| **Completed:** | November 30, 2016 |
| **CR-Classification:** | H.5.2 |

# Abstract

Wearables and especially smartwatches are nowadays used by many people as everyday personal companion devices. Smartwatches extend the design space of smartphones, making it possible to receive notifications in a more direct manner, and access important information at a glance. These aspects of today's smartwatches make them potentially useful not only for daily life but as assistant devices for industrial maintenance tasks. This thesis presents a concept and a prototypical implementation of such a wearable notification system for industrial maintenance tasks, based on Android Wear. The implemented prototype is evaluated in an industrial environment to find answers about the usability and user acceptance of this wrist-worn approach. Furthermore, it's potential of complementing or replacing existing maintenance task support systems is discussed and capabilities, as well as limitations of the prototype, are presented. Finally, alternative approaches using different wearable devices for this use case are discussed that may be evaluated in the future.

# Kurzfassung

Tragbare Computer, vor allem in Form von Smartwatches, finden heutzutage bei vielen Menschen Verwendung als persönliche Begleiter des Alltags. Smartwatches erweitern den Entwurfsraum von Smartphones, indem Benachrichtigungen in einer direkteren Art vermittelt und Informationen auf einen Blick einsehbar gemacht werden können. Diese Aspekte heutiger Smartwatches machen sie nicht nur im Alltag zu Helfern, sondern potentiell auch zu hilfreichen Begleitgeräten für industrielle Wartungsarbeiten. Diese Arbeit enthält ein Konzept sowie eine prototypische Implementierung eines solchen tragbaren Benachrichtigungssystems für industrielle Wartungsarbeiten, basierend auf Android Wear. Der implementierte Prototyp wird im industriellen Umfeld evaluiert, um herauszufinden, wie die Benutzbarkeit und die Benutzerakzeptanz eines solchen Systems ausfällt. Desweiteren wird diskutiert, welches Potential ein solches System hat, bestehende Assistenzsysteme für Wartungsarbeiten zu ergänzen oder zu erstetzen. Es werden Möglichkeiten und Grenzen eines solchen Systems präsentiert. Abschließend werden alternative Ansätze mit anderen tragbaren Geräten vorgestellt, die in der Zukunft evaluiert werden könnten.

# Contents

# List of Figures

# List of Tables

# List of Listings

# 1 Introduction

2015 was the year in which interest in wearables and especially smartwatches and smart bracelets peeked. According to Gartner's annual Hype Cycle report on emerging technologies of 2015 [1], the main reason for the hype around wearables reaching it's highest point, was that most of the products couldn't hold up to the high expectations of the consumers as far as battery life and feature richness were concerned. The general perception was, that wearables didn't offer many helpful use cases, other than receiving notifications without having to take one's phone out of their pocket. Such being the case, the wearable market has matured since then and is still gaining in width and depth, thus naturally improving itself. In fact, the smartwatch market alone is expected to grow from $1.3 billion in 2014 to $117 billion in 2020 [2]. Reasons for that are that expectations in wearables will decrease compared to their initial hype, while manufacturers will try to improve battery life as well user experience and feature richness, which will justify the price tag of those devices in the future. But not only wearables for the wrist are being developed. According to Gartner's Hype Cycle for Wearable Devices 2016 [3], categories like smart footwear, smart contact lenses, pet monitors or Electromyography Wearables [4] are currently on the rise. In the meantime smartwatches have evolved even further with most manufacturers already presenting their 2nd and 3rd generations of smartwatches this year. The devices have mostly become thinner and faster - not least because of improvements in wearable operating systems - more computational power and often with additional features like advanced smart home integration or water resistance. Modern smartwatches offer advanced RGB-LED screens, very similar to screens in current smartphones, enough computational power and sufficient battery life to display information to the user over the period of atleast one day. It is expected, that in the near future wearable devices will have autonomous power supply in terms of piezoelectric energy harvesting [Jun+15], or atleast rely on the power generation of smart clothes with solar panels as mentioned above, which means that battery life will become less an issue as time passes.

[1] http://www.gartner.com/newsroom/id/3114217
[2] http://www.smartwatchgroup.com/smartwatch-industry-report/
[3] https://www.gartner.com/doc/3382217/hype-cycle-wearable-devices-
[4] https://www.liveathos.com/blog/engineering/getting-to-know-athos-muscle-effort-68

Because of recent and expectable future advancements in the wearable and particularly the smartwatch industry and with how smartwatches extend the design space of smartphones, in terms of making information available at a glance while offering more direct modalities for notifications, it is reasonable to evaluate those gadgets in an industrial environment. Especially nowadays, where the roles of workers in the manufacturing industry are becoming more and more supervisory, while machines are taking over most of the the assembly tasks, smartwatches have the potential to serve the needs of this new supervisory role, where the display of real-time data is often desired. While consumers are traditionally satisfied with the placement of watches at the wrist and the accessibility in everyday life, it yet remains to be seen if industrial users agree and if smartwatches bear any potential to enhance existing industrial processes. To discover potential surplus value that smartwatches - and especially their ability to display critical information to the user in real-time - might add to existing industrial processes, this thesis focuses on one of the most common industrial use cases. The use case of managing maintenance tasks in industrial environments. The objective is to assess the usability and user acceptance of smartwatch-based wearable notification systems. To fulfill this objective I have implemented a prototype of such a wearable notification system based on Android Wear. In the following chapters I will firstly present related work and background knowledge before moving to the conceptual part of this thesis. I will demonstrate a concept, starting with initial sketches that explain the use case of this thesis, followed by user interface mock-ups of the smartwatch application. Following the conceptual part, I will present the architecture and the prototypical implementation of the wearable notification system in detail, ranging from used technologies over limitations of current wearable operating systems to specific implementation details. These chapters are followed by an evaluation chapter, where I present the design and the results of the initial evaluation of the implemented prototype. This is followed by a discussion and a conclusion of the results of this thesis. At the end I will be offering an outlook where I discuss alternative approaches for such industrial use cases like the use of on-body public displays instead of smartwatches.

# Outline

Here is the corresponding outline of this thesis:

**Chapter 2 – Background and Related Work:** This chapter presents related work and background knowledge to ensure a common level of understanding for the chapters ahead.

**Chapter 3 – Concept:** In this chapter the main use case of this thesis and the concept of the prototype are presented.

**Chapter 4 – Implementation:** This chapter focuses solely on the details of the prototypical implementation of the wearable notification system and the used technologies along with their benefits and downsides.

**Chapter 5 – Evaluation:** This chapter presents the initial evaluation of the prototype. It describes evaluation design and presents participants, procedures, and used data gathering techniques followed by a discussion based on the results of the evaluation.

**Chapter 6 – Conclusion and Future Work:** The last chapter concludes this work and offers an outlook to possible future work.

# 2 Background and Related Work

There is a large body of work in the area of Wearable Computing and mobile device interactions, remarkably ranging back to the 1970s. This chapter concentrates on presenting background knowledge about Mobile and Wearable Computing, as well as work on current and prospective data in and output possibilities of wearable devices, especially of smartwatches. A dedicated subsection on related work about supporting workers with ubiquitous technology completes this chapter.

## 2.1 Background

This section sets the theoretical basis for the upcoming chapters. It offers definitions of Mobile and Wearable Computing and explains the motivation for developing wearable devices in the first place.

### 2.1.1 Mobile and Wearable Computing

It is not uncommon to find these two domains mixed up. Therefore, this section tries to explain what both terms stand for, where the differences are and how smartwatches should be classified.

#### Mobile Computing

Mobile Computing stands for the practise of human-computer interaction, where the computer is expected to be transported during usage [MPK14]. Zahorjan (1994) defined Mobile Computing as "taking a computer and all necessary files and software out into the field" [ZF94].

In general, we can see ’Mobile Computing’ as a very loose, summarizing hypernym for categories like mobile communication, mobile software and mobile hardware.

One of the fields of Mobile Computing that still seeks for improvement, is efficient energy management. The goal is to provide high portability, mobility, feature richness, wireless networking capability and performance for mobile devices, while trying to maximize battery life through both software and hardware improvements. One of the more recent approaches that emerged through the rise of Cloud Computing, was the idea to offload computation to the cloud whenever battery life could be saved. In 2010 Kumar and Lu did some energy analysis for computation offloading from mobile devices. They found that as long as network bandwidth is large enough on a mobile device, the cost of offloading computational tasks to the cloud can break even at some point, and thus enable saving battery life, as long as the minimum bandwidth required for efficient offloading, is exceeded by the system [KL10]. The importance of saving battery life plays a significant role in this work. Therefore I will recapture this thought in a later chapter.

### Mobile devices

According to Manilal Patel et al. there are three different classes of mobile computing devices [MPK14]:

**Portable lightweight computers** that include a fully fledged keyboard and often are hosts to software that can be parameterized like laptops, notebooks, convertibles, etc.

**Mobile phones** featuring a limited key set, primarily intended for, but not restricted to, mobile communication, such as traditional cell phones, smartphones, tablet computers, etc.

**Wearable computers** mostly limited to a small set of functional keys, often incorporations of software agents, as watches, wristbands, necklaces, key-less implants, etc.

In other words, mobile devices are all kinds of portable computational devices which are hosts to some kind of software, more times than not having a display screen and a keyboard (either physical or virtual on a touchscreen) and typically small enough to be handheld. Today, mobile devices are used in multiple ways. They are used for entertainment purposes, communication purposes, as personal assistants, as educational devices and serve as important tools for companies in those same categories. Companies are using mobile devices to reduce costs and to raise the productivity and attainability of their employees in general. One of the main advantages of mobile devices is that information, services, emails etc., can be accessed from any location, as long as there is a connection to the internet. Therefore it is not uncommon for companies to use mobile devices out in the field, e.g., to display latest design iterations of a new product

to their clients on said devices or, e.g., to provide a mobile assistance device for their field employees doing maintenance, delivery or investigative tasks.

### Wearable Computing

According to what we learned so far, Wearable Computing is most often presented as a subcategory of Mobile Computing. Sometimes it is also seen as a logical successor to Mobile Computing. Therefore it is important to explain what it stands for in detail, and why it deserves it's own domain. A very popular definition of Wearable Computing comes from one of the pioneers in this field, Steve Mann:

> "Wearable computing is the study or practice of inventing, designing, building or using miniature body-borne computational and sensory devices. Wearable computers may be worn under, over or in clothing or may also be themselves clothes" [Man96].

Wearable Computing in it's modern form, thus describing truly untethered wearable computers, was firstly introduced by the Canadian researcher Steve Mann in the early 1990s. Wearable Computing itself, thus including tethered wearable computers that were bound to stationary workstations can be traced back as far as 1968, where Ivan Sutherland described the first head-mounted display [Sut68].


**Figure 2.1:** Mann carrying an early untethered wearable, from [Man97].

Figure 2.1 shows one of the first tetherless wearable computers, worn by the inventor himself. The device already had a color stereo head-mounted display with two cameras. All the communications equipment was carried around the waist, while antennas and transmitters were mounted on the back of the head-mount to balance the weight [Man97].

### Wearable Computers

According to the definition of Steve Mann, a wearable computer does not have to be more than a computational sensory device. In fact, it is not required for it to have a display by definition. Many of the earlier mentioned devices like wristbands, necklaces or other body-borne devices like smart clothes often don't have a display. There are even wearables without a single physical or digital key at all, like fully integrated chips inside the soles of smart footwear. This definition also allows for head-mounted VR displays to be categorized as wearables, as long as all the required computational power can be worn, e.g., by carrying a PC inside a backpack.

*So what does a device need to have, to be categorized as a 'wearable'?*

As long as it is a body-borne device which either measures user input, collects some kind of data through it's sensors or receives data from another device and is able to either communicate the collected data to another device or display it on a built-in screen, using it's computational power, it is seen as a 'wearable' or more precisely a wearable computer or wearable device.

### Is a smartwatch a mobile or a wearable device?

The case of the smartwatch is kind of special. The intuititve reaction is to categorize it as both mobile and wearable device. This is somewhat reasonable, due to a smartwatch more often than not having multiple sensors, like GPS, accelerometers or gyroscopes, as well as a display screen and enough computational power to display data on itself and to communicate it's data to other devices via Bluetooth or WiFi. The only real problematic thing with categorizing a smartwatch as a classical mobile device, is the absence of a keyboard for user input. In the most prominent cases of Apple's WatchOS or Google's current version of Android Wear, not even a miniature keyboard is provided, as manufacturers traditionally didn't classify smartwatches as standalone devices but as companion devices, being always-dependant on a host device like a smartphone. Nonetheless, there are some initiatives by Google to make Android based smartwatches more independent from their hosts, i.e., smartphones, which I will be discussing in the following section.

## 2.2 Related Work

This section presents related work that served as inspiration, as well as foundation and guidance for my own work. It consists of a part on interaction with mobile devices

and smartwatches in general, presenting multiple ways of both, data in and output. It is followed by a more designated section about assisting workers through ubiquitous technology. Some of the ideas and findings presented in this section were adopted by myself during the implementation of my prototype.

### 2.2.1 Interaction with Smartwatches

In this subsection I will present current and prospective ways of interacting with mobile devices, some of them trying to extend the design space of human-wearable interaction. Since I used a smartwatch for my prototype there will be a focus on smartwatches in this section. Firstly, I am going to show different ways of data input, followed by ways of data output on mobile devices and smartwatches.

#### Input

According to a report from 2013, 90.5% of all users use their mobile device to send and receive text messages. 77.8% use it to write emails, and 65.3% use it to access social networks [Nie13]. Furthermore "interaction with mobile devices is still mainly limited to visual output and tactile finger-based input." [Sch+16].

One of the most common input ways, is classical text input through a physical or on-screen keyboard.

**Text input** Having their roots in early cell phones and pagers, in terms of clunky physical alphanumerical keyboards, text input methods have evolved ever since. In contrary to popular belief, pressing keys in a traditional manner isn't the only way one can input textual data into mobile devices anymore.

In fact, the rise of smartphones as well as advancements in mobile operating systems and software, have created multiple input methods that all lead to actual text being displayed on our device. Figure 2.2 shows the five most common text input methods - not neccessarily in the right order - being physical Qwerty, virtual on-screen Qwerty, tracing (aka. 'swyping'), handwriting through a finger or stylus pen and speech to text.

While classical text input through Qwerty keyboards is the most prefered and most performant form of input among untrained users on smartphones according to Smith et al. [SC15], smartwatches do not offer enough screen resolution or even physical space to host a 108-key Qwerty style keyboard. As a result, previous approaches of bringing Qwerty style soft keyboards to the smartwatch simply couldn't provide frustration-free text input for it's users. To address this issue, science, industry and wearable

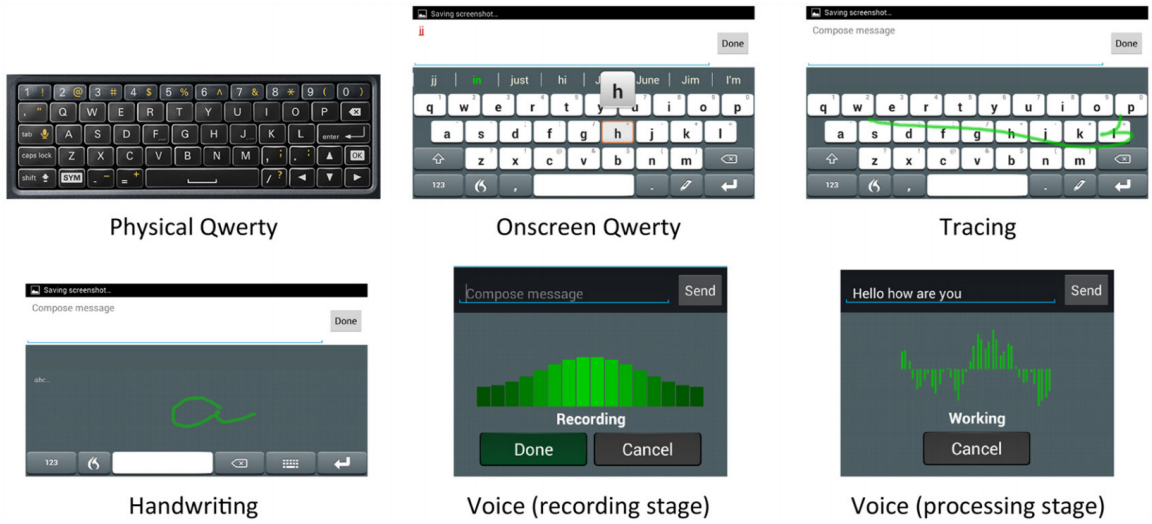



**Figure 2.2:** This figure from [SC15] shows the five most common text input methods in human-mobile interaction.

developers tried to come up with alternative keyboard layouts and text input methods for smartwatches.


**(a)** Eight total keys, to solve the fat-fingers issue.

**(b)** Each character has it's own gesture angle.

**Figure 2.3:** These figures from www.touchone.net show how single characters are accessed through different gestures.

One interesting keyboard layout approach for smartwatches that is endorsed by the likes of Yahoo Taiwan and CeBit Australia is called TouchOne [1] and comes from Rugang Yao.

[1] http://www.touchone.net/

Yao reinvented text input on smartwatches, by aligning the keyboard on the edges of the screen, as shown in figure 2.3 (a). This results in a total of 8 keys, where every key holds up to 4 characters, similar to old T9 keyboards. The characters are then selected through a gesture as shown in figure 2.3 (b). It is maybe worth noting that this is not the only approach by Android developers to create such an input method, as there are many more 'quasi-T9' predictive text input methods available in the Google Play store right now.

As mentioned before, not only independent developers are trying to alleviate the problem of text input on smartwatches, but also the two of the largest wearable operating system providers, Apple and Google. Apple introduced a handwriting feature called 'scribble' with their latest Watch OS 3.0 release. Google, having already done that in the first version of their wearable operating system 'Android Wear 1.0', is now bringing a fully featured, prediction enabled Qwerty keyboard to it's smartwatches, as well as an alternative text input method through predefined text messages, called 'smart replies', as shown in figure 2.4.


**Figure 2.4:** This figure from Google shows the built-in text input methods of Wear 2.0

The text input method that worked best across both young and old untrained smartphone users, was speech to text, as assessed by Smith et al. [SC15]. Presumably this is the best way of text input on a wearable device without a keyboard including smartwatches. The main argument being the lack of an alternative good typing method on said devices.

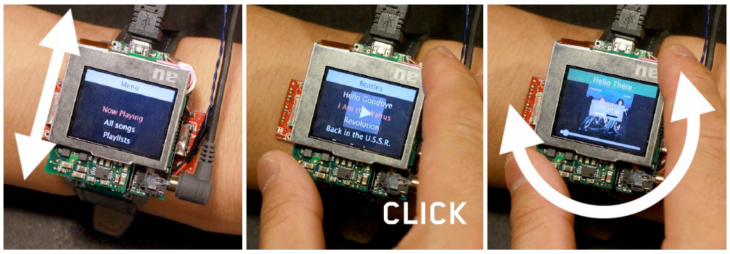



**Figure 2.5:** Navigating through the music application via panning, twisting to adjust the volume, from [XLH14].

An alternative approach of expanding the input capabilities of smartwatches featuring 2D panning and twisting of the screen itself along with binary tilt and click functionality was introduced by Xiao et al., in 2014. They designed a new modality for smartwatch-interaction, pictured in figure 2.5, and tested their prototypical implementation in several sample applications. The tested application scenarios included the input of gestures through clicking and panning, navigating through a music application (as pictured in Figure 2.5) or playing Doom. Their conclusion was that this approach was inexpensive and can coexist with current input methods like physical keys, speech to text or on-screen keyboards. One of the drawbacks was that it required the placement of additional parts inside the watch, and that battery life may be influenced in a negative way by such a system [XLH14].

### Output

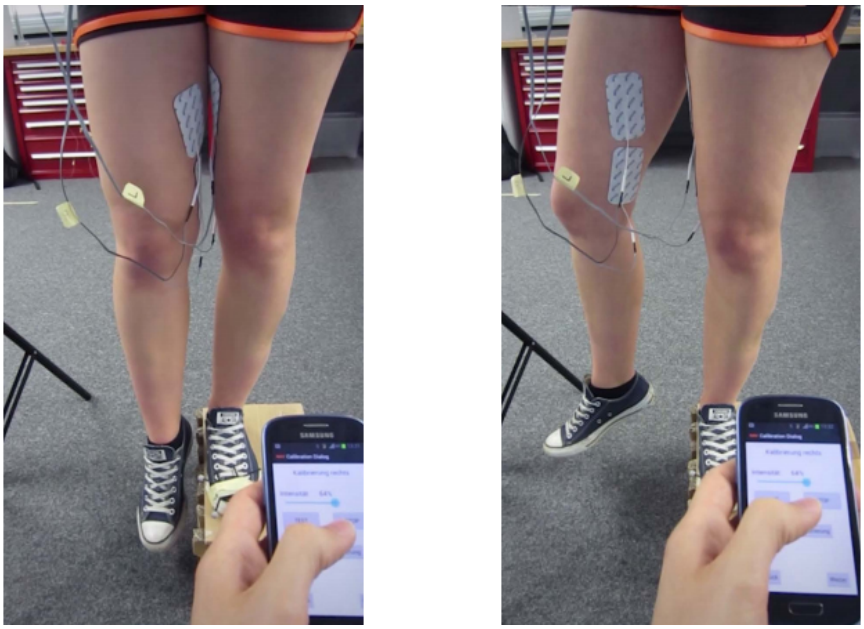
**Notifications** One of the most common data output methods on smartphones that is also seeking attention of the user, are notifications. This does not change very much when it comes to smartwatches. In fact, due to the restricted nature of smartwatches, notifications are a fundamental part of almost every use case current smartwatches have. Calendar apps, news apps, alarm apps, text messaging apps, system warnings like low battery warnings and even reminders of certain fitness apps to take a walk every now and then all make use of notifications on smartwatches, to get the attention of the user and to make sure that the displayed data is perceived accordingly.

Similarly how notifications on smartwatches are presented in a condensed manner at the top of the screen, both WatchOS and Android Wear smartwatches display notifications at the top or bottom of the screen and they can be resized to their original size through a gesture in both systems. The presentation of important data on the wrist in terms of notifications, is generally perceived well and is rated as the most important feature by smartwatch users. The issue being that not all notifications are equally important. Many apps are flooding the screen with notifications similar to spam emails. Another problem is that notifications have a disruptive nature not only in desktop but also in mobile environments, according to Shirazi et al.. Nonetheless users appreciated being notified about important events, even though some notifications display useless and annoying data. They also found that users usually click on a notification in less than 30 seconds, prefering the vastly prevalently occuring messaging notifications, while important notifications might not necessarily get immediate reaction by the user, if not very interesting or drowned by a number of other notifications [SS+14].

Independently from the disruptive nature of mobile notifications, it is no secret that interruptions of any type - not only through machines - reduce our performance by taking our focus off of the current task, as well as time off the clock while we're

recovering from an interruption. We then often have to catch the last thought we had, right before an interruption occured, costing us even more time and reducing our productivity in general [Bru+13]. As far as smartwatch notifications are concerned, users can see the type of the notification at a glance, without having to take out their smartphone out of their pocket following a vibro-tactile stimulation. So it may possibly be less disruptive to receive notifications on a smartwatch rather than on the smartphone, but due to smartwatches often presenting notification content in an abbreviated form, users might for example be motivated to read the whole text message on their smartphones, which would of course make things worse than they were with smartphone-only notifications. Nonetheless, there are industrial applications, where the danger of important notifications not reaching the targeted person may be a lot higher than the side effects of their disruptive nature. Especially when imminent reaction of personnel is imperative. To address the issue of unnoticed notifications, Schneegass and Rzayev proposed so called *Embodied Notifications* [SR16].

**Embodied Notifications** are a new way of gaining the user's attention, by using the body of the wearer as feedback channel. The difference is the user doesn't have to take a look at his smartphone or smartwatch, but implicitly understands what the notification wants to communicate. They are especially helpful when important notifications would else remain unnoticed when drowned by an ever increasing total number of unprioritized notifications [SR16]. Schneegass and Rzayev propose the use of Electrical Muscle Stimulation (EMS) as addition to existing ways of notifying. They state that the advantage to more traditional notification modalities like visual, vibro-tactile and auditory stimulation is that the user much more likely notices the notification due to their embodied nature. Earlier work by Pfeiffer et al., indicated that users liked EMS over other haptic stimuli like vibration [Pfe+14]. According to Schneegass and Rzayev, this opens "myriads of application scenarios" [SR16]. They also talk about combining embodied notifications with EMS for navigation, thus in some way steering the user in the right direction. Figure 2.6 demonstrates a possible way of such an EMS actuation. This idea might be helpful when navigating employees around gigantic factory buildings, where construction workers often have to change their workplaces and where it is easy to get lost, even for experienced staff, like in the famous case of Boeing's Everett Factory. Especially when fork lifts and similar vehicles that might be a potential threat for pedestrians use the same paths, traditional ways of navigation (i.e., pointing arrows on smart device screens), may take too much attention form the user and thus cause accidents. For this specific scenario it might even be interesting to combine different haptic feedback methods like EMS and vibration [Pfe+14], e.g., by using vibration for warnings about the surroundings and EMS for the actual navigation.



**(a)** Before the EMS actuation. **(b)** After the EMS actuation.

**Figure 2.6:** EMS actuation steering the user's leg in a desired direction, from [SR16].

When considering industrial applications of a wearable notification system, all of the findings presented above, play a key role during the design process of such a system. Depending on the application, companies presumably would want their employees to be notified about important things right away in some situations, but at the same time they wouldn't want their employees to get interrupted very often in other situations. The correct approach has to be somewhere in between, as previously suggested by Shirazi et al. [SS+14].

### Close-to-body Wearables

Wearable Computing has a much broader design space than Mobile Computing, because there is no limitation to handheld devices [Sch+16]. While the interaction possibilities of mobile devices - including smartwatches - are limited in terms of both, data in and output, close-to-body wearables offer new possibilities of interaction, for example by doing tasks implicitly and without the need of active user-interaction. Examples of wearables with a wider design space are head-mounted displays like Google Glass or the before mentioned haptic navigation example with EMS [Sch+16]. Head-mounted displays are private displays compared to mobile and smartwatch displays. They can be used for displaying secret data, thus providing advanced privacy or even be used for

navigation, where the users wouldn't have to take their eyes off their surroundings, thus providing advanced safety, as pictured in figure 2.7.


**Figure 2.7:** Navigation with Google Glass, keeping user's eyes close to the road, from Google.

This offers many application scenarios for both private and industrial usage. Another argument for head-mounted displays and close-to-body wearables in general, is that users can use their hands freely. This is an important requirement for many industrial use cases including the main use case of this work. Industrial maintenance or assembly workers often mustn't carry any kind of jewelry or watches, in particular for safety reasons. This is where head-mounted displays or EMS patches come in very handy. This directly leads us to the next part, where we discuss related work on supporting workers with ubiquitous technology.

### 2.2.2 Supporting workers with Ubiquitous Technology

There is a clear trend towards automatizing repetitive production tasks in production facilities. Companies are trying to be more productive and offer wider product lines at the same time. The overall need for assembly workers is declining with time, but at the same time machines are not ready yet to do all the different assembly tasks that humans do today. Removing human workers completely would make it impossible to ensure a modular and variant production. Therefore researchers have been thinking about ways of enhancing the performance of human workers ever since Mobile Computing became a thing. There has been a big body of work regarding the question of supporting workers with ubiquitous technology, both through stationary and through portable systems. In this subsection I am going to present a small subset of the work in this field that influenced my own work.

### Stationary Systems

There have been many research projects about industrial assistive systems using Augmented Reality. One of the most prominent use cases is delivering work instructions to workers to enhance their precision and workspeed, for example at manual assembly stations in factories. Navab mentioned in 2004 that there is a need for a killer app for *industrial augmented reality*. He discussed the application of a head-mounted AR assistive system in the following areas of the industrial space: design, commissioning, manufacturing, quality control, training, monitoring and control, and service and maintenance [Nav04]. He found that a killer AR application would provide a better solution in all of those areas, compared to old processes, as long as they met some important criteria. According to him, such systems should always be reliable, thus provide robust solutions and reproducible outcomes. Furthermore every assistive application should be as user friendly as possible and scalable beyond the level of simple prototypes [Nav04].

So called Context-Aware Assistive Systems (CAAS) make use of advancements in motion recognition which enables them recognizing the movement and actions of the user in real-time, thus enabling CAAS to react and give feedback accordingly [KFS]. This creates many new uses for such assistive systems that go far beyond instructing users during industrial assembly or maintenance tasks [KFS]. One use that CAAS allow are real-time error feedback at manual assembly workplaces [Fun+16]. Funk et al. evaluated different error feedback methods under this aspect and found that auditory feedback is often perceived as privacy-intrusive by the participants while haptic and visual feedback was rated similarly well [Fun+16]. This may be similarly perceived by smartwatch users receiving incoming maintenance tasks.

**Figure 2.8:** Head-mounted display showing 3D assembly instructions (left), projection-based AR system showing 3D-in-2D assembly instructions (right), from [FKS16]

In more recent work by Funk et al., they approached this idea and compared the use of head-mounted AR displays vs. in-situ projection for such Context-Aware Assistive Systems [FKS16]. They did this by letting their experimentees build different LEGO figures at assembly stations, where different LEGO parts were held in a multitude of

small containers. The participants were instructed through the head-mounted AR display and through the projection-based AR system accordingly. The assistive systems provided the participants with spatial picking information needed to assembly the figures, as well as a 3D representations of the assembly steps in close proximity to the workspace, as shown in figure 2.8. They found that users did less mistakes following the projection-based approach compared to the head-mounted system. They also assembled the figures slightly quicker using the projection-based approach. Funk et. al, explain these results with the limited viewing angle of current head-mounted displays and them not being very robust under bright light conditions of around 500 lux, usually found in real-life factories.

### Portable Systems

**Previous work with smartwatches** Aehnelt and Urban already did some work on smartwatch assistance during industrial assembly tasks. They defined a theoretical model for a situation-aware worker guidance system, [AU14]. Their system consists of three parts: *smartwatches*, *mobile* and *stationary displays*. It provides assisting functionality in terms of *activity recognition*, thus monitoring the worker's activities and progress, *awareness display*, thus displaying incoming new work tasks, *information assistance*, thus displaying instructions and related manufacturing data, and lastly *interaction* with the system through the smartwatch, allowing simple commands and feedback [AU14].

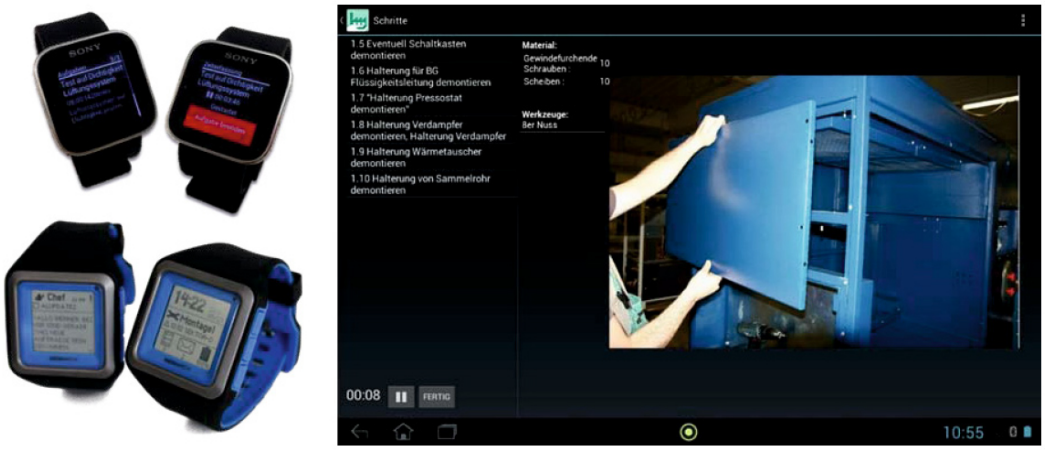



**Figure 2.9:** Plant@Hand application on smartwatches (left) and on display (right), from [AU14].

Their system was implemented by adding different smartwatch functionality to the preexisting manufacturing support system Plant@Hand [2] from Fraunhofer Institute. Their system used both mobile and stationary displays to show information that couldn't be displayed in full on the limited screens of smartwatches. Only important subsets of the total data were displayed on the smartwatches. In fact, only situation-dependent information was shown on the smartwatch, as pictured in Figure 2.9. Although their work suggests that there might be a positive effect on work performance using their system, they left the evaluation part of their system to future work.

[2] http://www.igd.fraunhofer.de/Institut/Abteilungen/IDE/Projekte/PlantHand

# 3 Concept

The idea of using wearables and especially smartwatches as assistive devices for workers in industrial environments will be evaluated by building a prototype of an assistive wearable notification system. The main use case of the prototype for this thesis is the use case of assisting maintenance employees throughout their workday. The reasoning behind such a prototype is that it may provide additional value for both employees and employers when compared to traditional maintenance work flows. Currently, maintenance tasks often get reported in a centralized manner, e.g., to a central terminal at the maintenance staff office. The drawbacks are that the more important issues may remain unnoticed if all of the maintenance staff is busy when the issue gets reported. By notifying maintenance employees of new maintenance tasks as they arrive and giving them the possibility to report maintenance task related data back to the company's business software right at their wrists, it is believed that existing stationary systems can be enhanced or even superseded by such wearable systems.

Before I built this prototype, I did preceding conceptual work, where I determined all the functional requirements that such a system has to meet, how it can be realized and which techniques and technologies can be used for a trouble-free implementation.

## 3.1 Context of use

Maintaining industrial production plants is a difficult task that requires both the maintenance staff and the maintenance work flows themselves to follow certain concepts [ZHU15]. It is common that the maintenance staff as well as all necessary tools and equipment for ensuring a fault-free operation of the plant are located closely to the actual production plant. Depending on the actual task, the work conditions that maintenance workers find themselves in, may vary a lot. The range of these factors is wide. Everything from varying light conditions, rough terrain, over noise, vibration, dirt or even lubricating, coloring or harmful substances may influence the maintenance staff during work [Wit08]. In addition to that, maintenance workers often work alone for longer periods of time without contact to other workers [ZHU15]. Considering all of that and the fact that the maintenance staff often wears protective equipment that

exceeds the equipment normal machine workers wear, such as temperature shielding gear, helmets, eye protection or heavy gloves, the idea of placing such an assistive device at the wrist in terms of a smartwatch seems to be a reasonable choice compared to more sophisticated devices like smartphones or tablet PCs which require more advanced means of input compared to smartwatches.

### 3.1.1 Requirements for wearable industrial assistive systems

The maintenance workflow when using such a wearable assistive system is characterized by a constant alternation between actual maintanance work on the machine and interaction with the wearable itself and therefore with the underlying IT system [ZHU15]. Therefore it is important to minimize interaction times with the wearable by simplifying the user interface while still providing all important data and interfaces, a maintenance worker needs to begin, pause and complete his task. Furthermore such a system should never patronize the maintenance worker, as maintenance tasks often include spontaneous shifts of focus to optimize activities on-site [ZHU15]. In other words, such a system should always be flexible and adaptive enough to let the maintenance worker do whatever needs to be done to finish the task in a safe, fast and satisfactory manner, without imposing rigid and counterproductive sub-tasks onto the worker. This means that a maintenance worker shall be able to pause his current task in favor of an incoming more important one or even be able to start multiple similar tasks at once, to make sure the collected work data matches the parallel nature of performing these tasks on-site.

### 3.1.2 Technological challenges

Given the context of use in the previous section, it is clear that the described environment bears many dangers that must be considered when designing such a prototype. A very common danger would be that a maintenance worker could catch onto something with the wristband of the smartwatch, threatening the safety of the wearer's arm. Therefore it is important that the smartwatch can easily be detached from a worker's arm in case of emergency, thus it mustn't have an unnecessarily sophisticated closing mechanism or atleast have a predetermined breaking point. As machine outages bear high economic risks and timely remedial maintenance is of highest priority, both the software and the hardware in use, have to be robust, consistent and reliant. A requirement that falls into this category and that such a smartwatch based system shall meet, is that the battery of the smartwatches used for the prototype at least survive the period of a common stint in the industrial maintenance industry, which approximates around eight to ten hours. Depending on the chosen technologies for the prototype, such a

system requires a reliant and robust mobile networking infrastructure in the majority of cases, to ensure actuality and cohesiveness of in and outgoing data. The most common options current smartwatches and smartwatch operation systems use are Bluetooth and WiFi. For both of these networking technologies difficulties may arise when maintenance workers are surrounded by radio wave absorbent solid objects like big steel machines or thick concrete walls. Because of that, possible data inconsistencies, caused by network outages have to be handled accordingly. To achieve that and to maximize battery life of the smartwatches, one possibility is to hold a global state for each maintenance task on a server and to synchronize the latest state to all clients (smartwatches) as soon as a connection is reestablished. In other words, it is reasonable to make the smartwatches behave like 'thin clients' in this case.

## 3.2 Initial Sketches

To find out all possible requirements the prototype has to meet and to understand the process of maintenance tasks in general, as well as what in- and outgoing communication is associated with maintenance tasks, I created some hand drawn sketches of the aforementioned use case, which can be seen in figures 3.1 and 3.2.

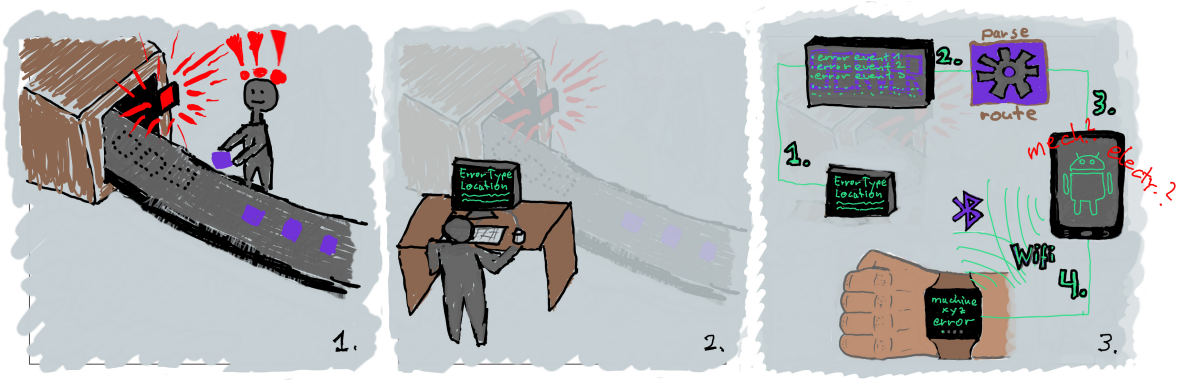



**Figure 3.1:** Hand drawn sketches showing the prototype in use during an incoming maintenance task.

Figure 3.1 contains the first three steps of handling an incoming maintenance task. The first picture shows a machine worker during a machine failure scenario. In this case, the machine worker notices that the machine stopped creating parts and tries to assess what might have caused the issue. In picture two, the same machine worker then goes on to a stationary workplace, where he creates a machine failure ticket, specifying the details of the failure as precisely as possible (e.g., the mechanical or electrical nature of a failure) along with additional useful information like the location of the broken machine and

which cost center has to be invoiced. Picture three shows how such a ticket might be transferred to the smartwatches of the maintenance workers. Firstly the ticket is created inside the company's business software right after the machine worker issued it. From there this data is made available to an external server component (e.g., in terms of XML, JSON or CSV files). As current smartwatches do not bear the potential of connecting to the outer world by themselves - which I will be discussing in a later chapter in more detail - an external server component has to parse this data accordingly, send it to a smartphone, which then routes the data to all the paired smartwatches or to a subset of all the paired smartwatches in some cases. As this data is received by the smartwatch, a push notification is created instantly, only displaying a small subset of the whole data. The idea is to display only the most telling pieces of text about a certain maintenance ticket, such as the error ID, the error type and some initial descriptive keywords written by the machine worker who reported the failure in the first place. This ensures that the worker who receives this notification, sees the basic information about a ticket at a glance without having to navigate into the detailed view. This approach can be extended by mapping different task priorities to certain colors or vibration patterns, to further raise the awareness of the workers about the incoming tickets without the need of active interaction with the wearable device itself. Figure 3.2 , starting with picture 4, shows

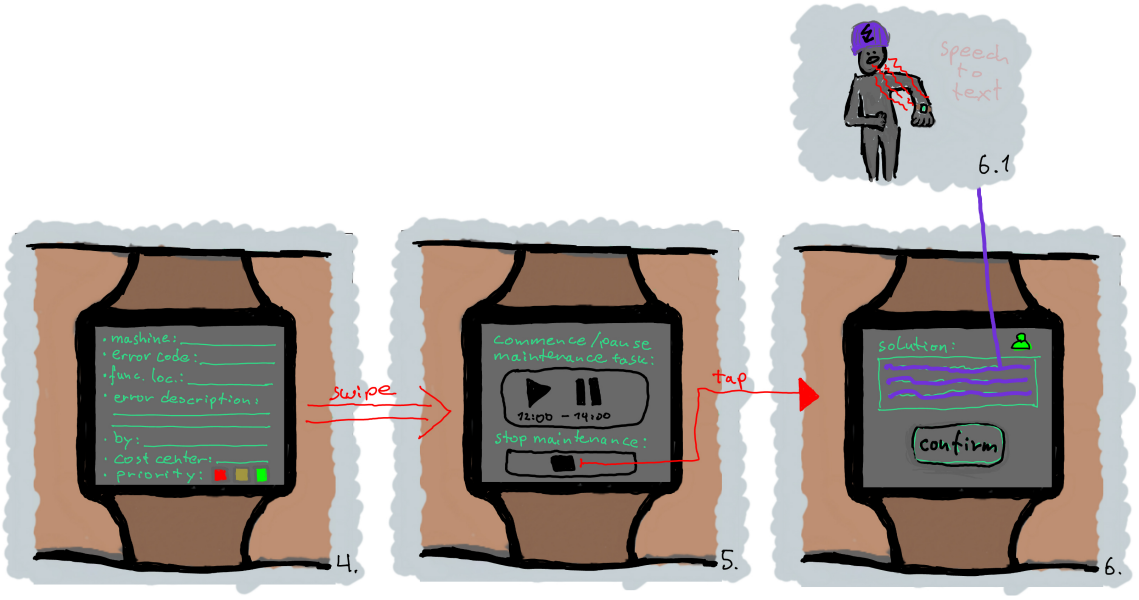



**Figure 3.2:** Hand drawn sketches showing the prototype in use during an incoming maintenance task.

how such a machine failure ticket may be represented in terms of a user interface on a smartwatch. The idea is to make all important information necessary to begin and complete a maintenance task, available to the maintenance staff and more importantly to present it in a simplistic manner.

As smartwatches have pretty limited display sizes, a reasonable way of displaying all of this data, is to use multiple screens as shown in these sketches or to use a so called ScrollView, which allows the user to move the focus of the view by scrolling vertically. I decided to use the latter approach in the final implementation. This approach enables the use of a single page view for each maintenance ticket, leaving out unnecessary navigation buttons and shortening interaction times. Swipe gestures are used to both navigate back to a global list view containing all tickets, and navigate forth to the detailed view of a single ticket. Since maintenance tasks often get started without getting completed right away (e.g., because of missing spare parts or a more important task requiring immediate action of a maintenance worker) there has to be a flexible way of recording the beginning, the duration, possible disruptions and the completion of a maintenance task. Picture five shows how a simple and minimalistic user interface for such a functionality may look like. It contains a button for beginning, pausing and completing maintenance tasks and displays the time when the task has been commenced, as well as the duration of the current task. Picture six shows a text field through which maintenance workers have the possibility to report back details that led to the successful completion of the task. Since text input is a difficult process on smartwatches, as discussed earlier in chapter 2, especially when we consider the bulky equipment maintenance workers often have to wear, a presumably more favorable but yet to be tested way of inputing text is 'speech to text'.

## 3.3 Software Architecture and Used Technologies

After having considered the context of use, the industrial setting and it's characteristics and after having created sketches of the prototype itself, it was time to choose a general architecture and specific technologies for the implementation of the prototype. Even though the basic principles discussed so far are completely independent of the used technologies, some major architectural decisions were driven by the possibilities and limitations of the chosen technologies.

### 3.3.1 Architecture

I implemented an architecture, consisting of three components: a *smartwatch component* (wear application), a *smartphone component* (mobile application) and a *server component*. Figure 3.3 shows the basic architecture of the prototype. It consists of one server instance, one mobile instance and a multitude of wear instances. The server component is the interface to any external data source that may deliver maintenance tickets, like the backend of a company's business software. The main task of the server component

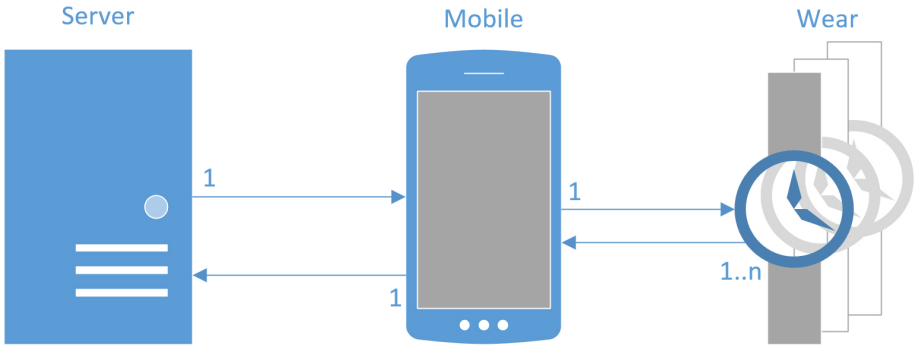



**Figure 3.3:** Basic architecture of the prototype of this thesis

is the input and output of maintenance related data. In this specific case, the server component watches a remote directory for any changes and parses every incoming maintenance ticket file, as soon as it's made available by the external source, before sending it to the mobile component. The server component is also responsible to write all relevant data which gets collected during the process of a maintenance task, into a new file, making it available to the external system. When the data of a single maintenance ticket is received by the mobile component, it takes this data and routes it to the corresponding smartwatches. The mobile component serves only as a messaging middleware. It contains no business logic, apart from holding information about it's paired smartwatches and the location of the server.

### 3.3.2 Technology Choices for the Wear Component

As far as the choice of the operating system of the smartwatches is concerned, I chose the *Google Android Wear* operating system for the client side of my prototype. There were many reasons that led to this choice, the first one being that *Android Wear* is very similar to the classic mobile operating system *Google Android*. Applications for both Android and Android Wear are created with nearly the same tools and frameworks. Another reason was that Android is holding the majority of the mobile operating system market share, due to multiple smartwatch manufacturers using this operating system. There is a good chance that this will also be the case with Android Wear in the future, as it is the only smartwatch operating system that is used by a high number of different manufacturers. Additionaly, due to it's highly documented and open source nature, in contrary to other potential choices like *Apple's Watch OS* or *Pebble's OS* it seemed to be the right underlying technology for the implementation of the prototype of this thesis.

In fact, Android Wear is nothing more than a modification of the standard Android OS, specialized for small screen use, promising quick and precise display of data by extending

traditional ways of displaying notifications and user interaction controls. Using Google's words "Android Wear extends the Android platform to a new generation of devices, with a user experience that's designed specifically for wearables" [1].

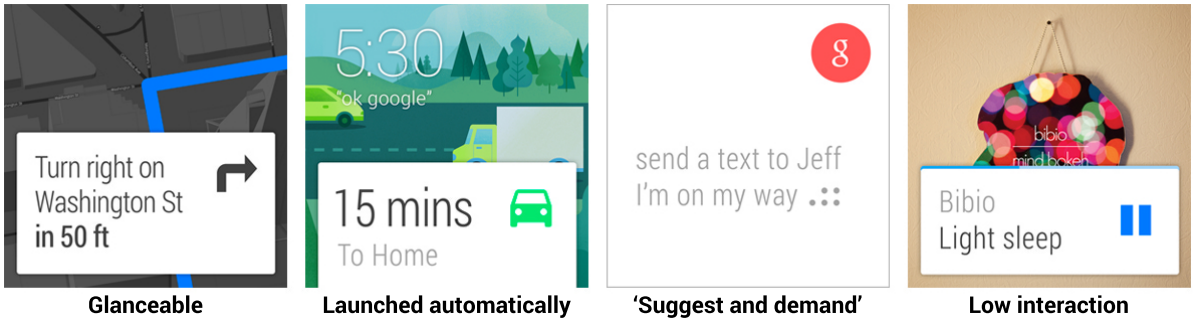



**Figure 3.4:** Creative Vision for Android Wear, from android.com

The main design principles that Google suggests for Android Wear applications are that applications should require little interaction with the user that they should only display the most important information in a 'glanceable and actionable' manner and that there shall be a high context sensitivity within every application. More precisely, Google named four main experiences that come with Android Wear as part of their 'Creative Vision' for the operating system. Figure 3.4 shows examples of applications that implement these four main principles.

It was designed to resemble a personal assistant more than to just become another mobile operating system for small screens. In contrary to traditional mobile operating systems, displayed information should be graspable in a split second, similarly how we see the time on traditional wrist watches. The first picture shows an example of a navigation application which tells the user what to do in the most simple manner, leaving out unnecessary details of the map and concentrating only on the next step of navigation. In addition to that, Android Wear can launch apps automatically, in contrary to classic mobile operating systems where users have to click on an app to launch it, as it is aware of the user's context. In picture two there is an example of such an automatically launched application, where Android Wear understands that the user typically goes home at 5:30pm, and therefore informs the user about the traffic situation in a very specific way. The interactions with and interruptions by the smartwatch shall be minimized as much as possible, which means that applications shall only require user input when absolutely necessary. Picture three shows how Android Wear can suggest and predict the content of a text message the user wants to send, while demanding as little user input as possible. The last picture shows the typical user interface of an audio

[1] https://developer.android.com/wear/index.html

playback application on Android Wear, which only has a single button to pause and resume the current song, demanding the least amount of interaction from the user.

I tried to follow these principles as much as possible to optimize the user experience of my prototype when I designed the user interface of my wear application. But of course, some of these principles like strictly displaying information in a glanceable way, had to be broken to some degree. The use case of industrial maintenance tasks requires longer user interactions with the smartwatch by nature. The reason for that being that the displayed data is always a form of briefing for the maintenance worker, and every kind of briefing or instruction takes some time that just cannot be reduced to a split of a second. To minimize the time maintenance workers interact with the device regardless of the given setting, my application makes use of the special Android Wear style push notifications which only show the most relevant data whenever a newly created maintenance ticket is received by the smartwatch.

Shortly after I started implementing the prototype for this thesis, Google announced the developer preview version of the second generation of it's Android Wear OS called *Android Wear OS 2.0* [2]. The second generation of the operating system comes with substantial changes, which I will not discuss in detail, as my work is based on the first version of Android Wear. But an important change that might have influenced the architecture of my software differently is that the new version allows smartwatches to have outgoing network connections in terms of real TCP/IP connections, thus more or less turning Android Wear based smartwatches into standalone devices. This is not the case with the initial version of Android Wear. Initially, Google classified smartwatches purely as companion devices, thus disabling every outgoing network connection on the smartwatches and forcing smartwatches to rely on their host device, thus a smartphone, whenever information is exchanged. This design principle was the major reason for me to use a three-component architecture for my prototype. In the future it may be possible to remove the mobile component, as there would be no need for a messaging middleware, if the smartwatches could communicate with a server application autonomously.

### 3.3.3 Technology Choices for the Mobile Component

Having already decided to use Android Wear based smartwatches, the choice of classic Android for the mobile component was an easy one to make. The current version 6.0 of Android, along with the latest versions of *Google Play Services* and the *Android Wear* applications for Android smartphones bring all of the infrastructure that is needed to

[2] https://developer.android.com/wear/preview/index.html

pair smartwatches to a smartphone and to enable communication between the two operating systems.

### 3.3.4 Technology Choices for the Server Component

To create a server application that is capable of watching certain directories for changes, read and write from and into files, respond to http-requests in a simple and reliable way and parse different data types and formats I decided to use node.js due to it's amazing flexibility and event-driven nature that matches the needs of this use case perfectly.

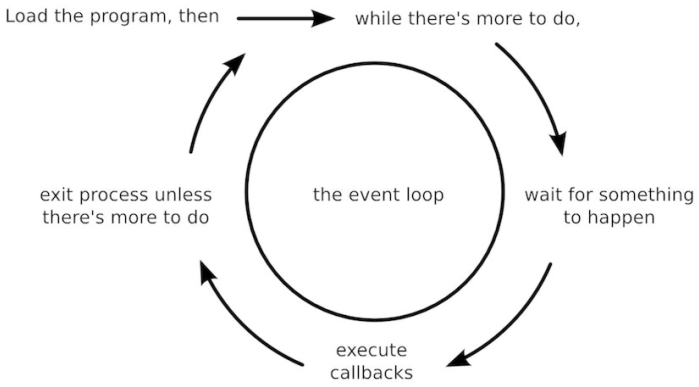



**Figure 3.5:** The node.js event loop, from strongloop.com

Node.js is an open-source cross-platform JavaScript runtime environment, mostly used for various backend and server side applications. It's event-driven nature allows asynchronous I/O operations, thus allowing applications to run continuously without blocking the main thread. In fact, node.js uses an *event loop* instead of classic processes or threads as shown in figure 3.5. It registers itself within the underlying operating system and gets notified when certain events occur. Every method that the node.js application is waiting for executes callbacks when it's done. This asynchronous approach enables scalability, even though it is a single-threaded approach. The pre-installed module manager *npm* allows easy installation of external modules and handles missing dependencies automatically. There is a large developer community in the node.js universe. Therefore there are many preexisting solutions in terms of external librares that I used, especially for the more essential things like setting up a http connection or parsing a file. I will become more specific about the used packages in the next chapter.

## 3.4 User Interface Mockups

After having decided which set of technologies to use for the implementation of my prototype, I created some initial mockups of a possible user interface of the wear application, using Google's publicly available set of Android graphics while considering the chronological order of events in the aforementioned use case of industrial maintenance tasks. The final application didn't necessarily adopt this UI design entirely, as you will see in the next chapter. Nonetheless this step was essential as many shortcomings of the application design were discovered and avoided a priori by dedicating enough work to these mockups at the right time. While I didn't pay attention to using suitable text and element colors during the sketching phase, I made sure to simplify the way the data is presented as much as possible during the mock-up phase while also maximizing the contrast and brightness of the user interface, as suggested by Fortmann et al. [For+14]

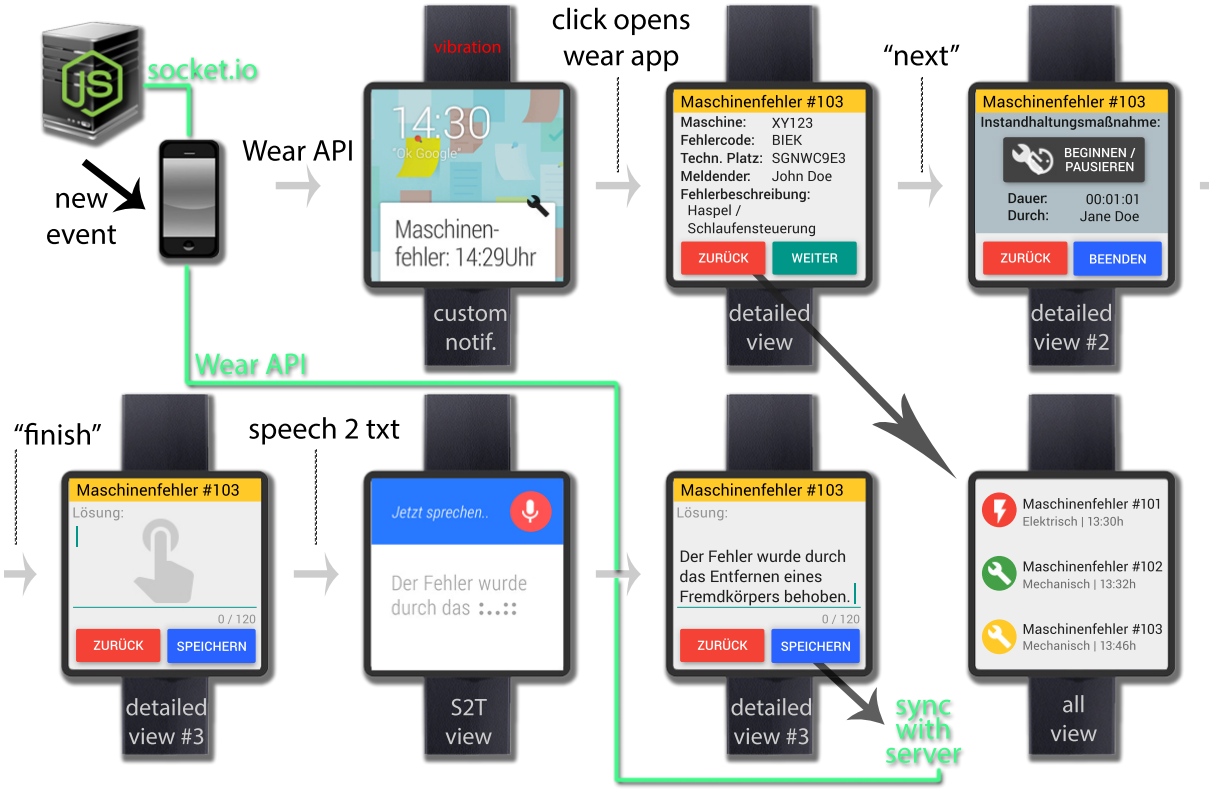



**Figure 3.6:** Initial mockups of the wear application user interface.

The UI mockups which resemble a succession of events and user inputs can be seen in figure 3.6. The small arrows indicate the chronology of events and data transfers that occur when going over an incoming maintenance ticket notification. The green connection symbolically stands for the data transfer from the smartwatch to the server

that occurs whenever a user updates the status of a maintenance ticket and eventually after the user finishes a maintenance task. Whenever a maintenance ticket gets created and it's data enters the system, the node.js server application notices this change of data, parses it in some way, and sends it to the mobile component, e.g., via HTTP socket or in this case using the *socket.io* [3] environment, which I will describe in more detail in the next chapter. The mobile application is a pure messaging middleware as stated earlier, thus it only routes the data from the server to the connected smartwatches using the integrated Android Wear API which was solely created for this purpose. After a smartwatch receives this data it creates a notification, displaying some important data about this maintenance ticket. Then, after the user clicks onto this notification, a detailed table view, containing all relevant data about this issue get's displayed. From there the user has the ability to *start or pause* and to *complete* a maintenance task using the provided buttons. Clicking into the edit text field opens the speech to text assistant, which allows users to specify the steps that led to the solution of the machine failure. Completing a maintenance ticket removes it from the local storage of the smartwatch after transfering all relevant data of this ticket to the server. Pressing the back button or using the respective Android Wear gesture when inside the detailed view of a maintenance ticket opens the list view containing all current maintenance tickets in a more condensed and overseeable manner, as shown in the most right hand side mockup of the lower row.

[3] http://socket.io/

# 4 Implementation

This chapter presents the prototypical implementation of the described three-component prototype in detail. There will be specific sections for each of the three components, explaining design decisions and difficulties that occurred during the implementation. Note that this is not the only possibility of realizing such a system, but that it was chosen specifically due to various factors, mentioned in the previous chapter. First of all I am going to present an overview of all the development tools that I used, followed by a more specific description of the architecture, before explaining each of the three separate applications and it's modules in more detail. I am not dedicating a special chapter to a description of the concepts of the Android OS or certain peculiarities of the used technologies and programming languages. I will rather occasionally add descriptions of certain concepts or patterns inside this chapter, especially where it's helpful for a deeper understanding. All of the three prototype components were implemented by myself with the assistance of and guidance by my supervisors Stefan Schneegass and Dominik Weber.

## 4.1 Development and Testing Tools

This section presents all the tools that I used for the implementation and testing of the prototype, in terms of hardware, operating systems, IDEs (integrated development environments), programming languages and frameworks. Specific third party libraries and packages that were used to simplify the development process and to extend the functionality of the applications will be mentioned in the specific sections about the particular applications. For a better oversight, please take a look at table 4.1. The boldly printed devices were used during the actual test phase. The others were primarily used for debugging.

### 4.1.1 Android Studio

The wear and mobile applications both were developed using the latest stable version of Google's Android Studio 2.0. Android Studio is an open-source multi-platform IDE

**Table 4.1:** Tools & hardware that was used for the development and test of the prototype

| | **Wear Application** | **Mobile Application** | **Server Application** |
|---|---|---|---|
| Developed on: | Macbook@MacOS 12<br>PC@Windows 10 | Macbook@MacOS 12<br>PC@Windows 10 | PC@Windows 10<br>Laptop@Windows 7 |
| Tested on: | **Samsung Gear Live**<br>**Moto 360** | 1+ One@Android 7.0<br>**Nexus5X@Android 6.0** | **Laptop@Windows 7**<br>Raspberry Pi@Raspbian |
| IDE: | Android Studio 2.0 | Android Studio 2.0 | WebStorm 2016<br>Notepad++ |
| Language: | Java 8 | Java 8 | JavaScript |
| Framework: | Android Wear (API 23) | Android 6.0 (API 23) | Node.js 6.x |

especially designed for Android based development. Google officially recommends Android Studio for all Android based development. It is based on the IntelliJ IDEA environment for Java by JetBrains and it is available under the Apache Licence 2.0. Android Studio offers advanced project management tools as well as support for building Android Wear applications, thus it automatically creates subpackages inside the project structure for the mobile and wear application respectively. Because of these factors and because of me being familiar with other JetBeans products, this was an easy choice to make.

### 4.1.2 JetBrains WebStorm

The node.js server application was mainly developed using the WebStorm IDE 2016 from JetBrains. WebStorm is a multi-platform IDE for JavaScript and TypeScript development. It can be used for web development in general, as it supports many other web languages as well. Since I was already familiar with JetBrains' other IDEs and since I own an education licence for this product, I decided to use it for this thesis.

## 4.2 Communication between the different Components

In the concept chapter I described the three-component architecture briefly, mainly to explain the thought process behind my technology choices. Here I will explain the architecture in more detail and especially the way the different components communicate with each other. A main requirement was that the whole prototype should work without

an active internet connection, as it is not uncommon for industrial companies to disable outgoing internet connections or cloud services at this application level, due to obvious security reasons. This requirement was the deciding factor to not use any cloud based communication services like Google Cloud Messaging or Firebase [1], and instead to create a reliable way of data consistency and communication even in cases of brief intranet outages.

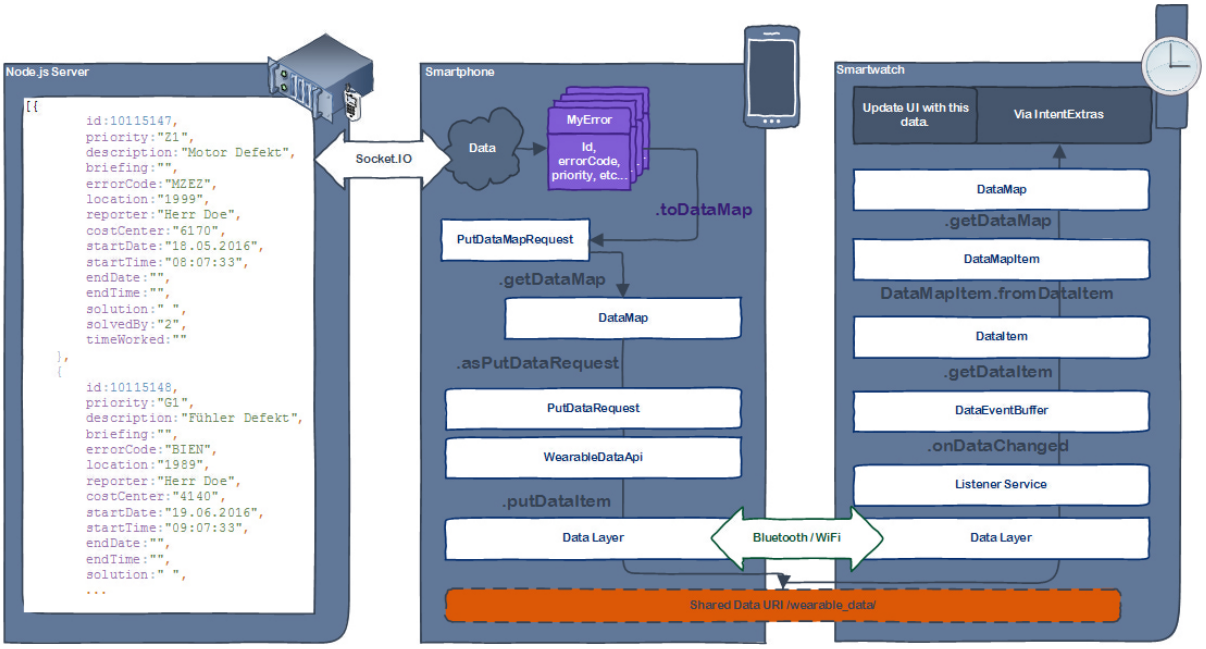



**Figure 4.1:** Bidirectional communication architecture of the implemented prototype.

Figure 4.1 shows the architecture of the bidirectional communication between the node.js server and client applications on the smartwatches through the mobile messaging middleware. I will be referring to this figure multiple times inside this section, to explain all the communication processes that occur in detail.

### 4.2.1 Communication between Server and Mobile

For the communication between the node.js server application and the mobile component of the Android application I used a HTTP / WebSocket connection by implementing the *socket.io* framework, on both the node.js as well as the Android side.

[1] https://firebase.google.com/

### Socket.io

Socket.io is a realtime application framework, written in JavaScript that enables bidirectional communication between a server and clients using the WebSocket protocol. It extends the functionality of the WebSocket protocol by adding different ways of addressing specific sockets when broadcasting messages, supporting asynchronous I/O operations and storing client related data on connection. Similarly to node.js itself it is event-driven, therefore it fits well into event based paradigm of node.js server side development. The *socket.io project* [2] on github also offers an Android based version that is identical to the original JavaScript library apart from little differences due to the Java vs. JavaScript syntax. These factors in addition to many well documented socket.io sample implementations and recommendations by other developers made this decision an easy one to make.

**Server side** The inclusion and initialization of the *socket.io-server* inside my *node.js express* application was realized as shown in Listing 1 from line 1 through line 4.

```
1  // instantiate socket.io-server listening for HTTP requests at the server's ip
2  var app = require('express')();
3  var http = require('http').Server(app);
4  var io = require('socket.io')(http);
5  ⋮   ⋮   ⋮
6  // new 'socket' object for each new connection
7  io.sockets.on('connection', function (socket) {
8      // listening for message event called 'message'
9      socket.on('message', function (data) {
10         // specified callback function
11         ⋮   ⋮   ⋮
12     });
13 });
```

**Listing 1:** Socket.io server side code structure, **file:** *index.js*

Whenever a new client connects - in this case this happens only once when the mobile application connects to the server - a new '*socket*' object is passed to the specific listeners, which invoke callback functions whenever the specified message events occur. In Listing 1 you can see the callback function for the message event called '*message*' starting from

[2] https://github.com/socketio/socket.io

line 9. Now whenever a client emits a message that uses the name 'message' inside it's header, the code inside the respective callback function gets executed. I specified a total of eight different callback functions that handle different events like a newly connected or disconnected client, an updated or completed maintenance ticket or a smartwatch requesting an update of the global state.

**Client side** On the client side - in this case, inside the MainActivity of the mobile application - one has to create a new 'socket' object after importing the *socket.io-android* package from their github project site. Listing 2 shows the client side code that I wrote to connect to my node.js server. Firstly i initialized a new '*socket*' object of the imported class *Socket* (line 2), then I specified the server's ip address and port on which the HTTP socket is listening, in this case I used the default port *3000* (line 8). I chose to connect to the server during the onCreate() method of the main activity (line 12), because in our use case, the mobile component didn't have any active function for the user and didn't have to be portable.

```
1  // Client side socket object
2  private Socket socket;
3  ⋮   ⋮   ⋮
4      @Override
5      public void onCreate(Bundle savedInstanceState) {
6          try {
7              // server ip and port
8              socket = IO.socket("http://10.***.***.***:3000");
9          } catch (URISyntaxException e) {
10             throw new RuntimeException(e);
11         }
12         socket.connect();
13         ⋮   ⋮   ⋮
14     }
```

**Listing 2:** Socket.io client side code, **file:** *mobile/MainActivity.java*

Therefore it was no problem that the application had to be *'always on'* for the HTTP connection to be active. A more reliant solution would be to create a *Service* that lives on in the background of the Android process stack, even if the application itself get's closed. You will see an example of such a *Service* when I discuss the details of the wear component in a following section.

To establish a bidirectional communication between the server and mobile applications, I also had to specify listener methods on the client side, similar to the ones used on the

server as shown in Listing 1. In this case, those listener methods receive the data from the server application and redirect it to the paired smartwatches.

### 4.2.2 Communication between Mobile and Wearable

Currently in Android, there is only one way of establishing a communication between a wearable - in our case a smartwatch - and it's host device, a smartphone. Google created the so called Wearable Data Layer API [3] for this matter. It comes as a part of Google Play services and provides three different types of sendable objects:

**DataItem** All DataItem objects hold some data and get synced automatically between the wearable and mobile devices whenever data is changed. To guarantee data synchronization at every interaction with the API, one can add an unique attribute to each call, for example a current timestamp.

**Message** Instances of the MessageApi class can send lightweight messages up to a payload size of 100kb. To enable a bidirectional communication both the mobile and the wear application need to implement the MessageApi as well as respective Message Listeners to receive and identify incoming messages. In contrary to DataItems, this type of communication is not recommended for transporting actual data, but for remote procedure calls (RPC) or one-way requests.

**Asset** The Asset class adds some functionality compared to classic DataItems as it enables sending binary data. Each *asset* relies on a DataItem for communication, in other words, asset objects such as images or audio files can only be synced between the wearable and mobile devices when attached to a DataItem. The API automatically takes care of marshalling (unmarshalling) the files into reasonably sized blobs of data.

For the communication between the wear and mobile applications I only used the first two types of communication as I never really needed the added functionality of the Asset class. But first of all I had to make sure that there are always active instances of the GoogleApiClient class on both the mobile and wear side, before starting any kind of data exchange. This is a prerequisite for all of the mentioned types of communication between wearables and smartphones, as the GoogleApiClient objects manage all the (re)connection tasks between the different devices implicitly.

So the wear application uses an instance of the MessageAPI class to send one-way requests to the smartphone, e.g., requesting the latest state of the globally managed

[3] https://developer.android.com/training/wearables/data-layer/index.html

maintenance ticket data. The header of such a message specifies it's cause. In other words, the API enables sending messages to uniquely named virtual paths to distinguish between different types of messages.

To make sure that no message gets stuck or lost due to a missing active instance of the GoogleApiClient, I wrote a *sendMessage(path, text)* method that checks for an active instance of the GoogleApiClient, reconnects it if it does exist but it's connection is just off or creates a new instance of it, if it's missing completely, before sending any message at all. I reused this method for most of the messaging that occurred inside the MainActivities of the wear and the mobile applications as well as inside the NotificationActivity, DetailActivity and DataLayerListenerService of the wear application. Listing 3 shows this method in detail.

```
1  /**
2    * Used to send messages between wearable <-> mobile. Param 'text' is optional.
3    * @param path
4    * @param text
5    */
6  private void sendMessage(final String path, final String text) {
7      // New thread for each message
8      new Thread(new Runnable() {
9          @Override
10         public void run() {
11             // preliminary GoogleApiClient checks
12             if (mGoogleApiClient != null && !(mGoogleApiClient.isConnected() or
                ↪  mGoogleApiClient.isConnecting())) {
13                 mGoogleApiClient.connect();
14             }
15             // All connected nodes (NodeApi)
16             NodeApi.GetConnectedNodesResult nodes =
                ↪  Wearable.NodeApi.getConnectedNodes(mGoogleApiClient).await();
17             for (Node node : nodes.getNodes()) {
18                 MessageApi.SendMessageResult result =
                    ↪  Wearable.MessageApi.sendMessage(mGoogleApiClient, node.getId(),
                    ↪  path, text.getBytes()).await();
19                 if (!result.getStatus().isSuccess()) {
20                     return;
21                 }
22             }
23         }
24     }).start();
25 }
```

**Listing 3:** The often reused *sendMessage(String path, String text)* method.

Starting from line 8 it is noticable that every sent message creates a new thread. This is necessary because of the asynchronous nature of these messages, which do get delivered in the right order (by respecting a FIFO order et cetera) not necessarily instantly in all cases, but with a small delay.

The NodeApi enables fetching all connected devices, which in this case returns the node of the mobile device. Then a message is sent with the previously provided parameters path and text while also awaiting the result of the sending request, to be able to handle possible delivery errors accordingly (line 18 - 25). It has to be said that a lot of this code is available openly as part of the official documentations of the Android and Android Wear projects. I mostly modified preexisting and recommended pieces of code so that they match my own requirements and solve use case specific difficulties.

For example, the path that I used to signalize the wear application's request for the latest data was: `"/wear_to_mobile_update_data_query/"`. So the mobile application, which implements a MessageAPI listener at this certain path, notices an incoming message and invokes a method, which in my case just emits another message to the server via the previously described socket.io framework. The only thing that the mobile application adds to this message before redirecting it to the server, is the node ID of the smartwatch that requested the data update, so that the different smartwatches are known by the server application.

When the server responds to such a request, it takes all relevant maintenance ticket data and sends it to the mobile application, again, by using the socket.io framework. Then, the mobile application receives this data and handles it, as pictured in Figure 4.1 in the center and right hand side rectangles. Firstly, the data gets transformed in a way that I will describe in more detail in the following section. Then for each maintenance ticket a so called DataMap object is created. The DataMap class is a map that implements the DataItem logic described earlier. To guarantee a synchronization for 100% of the sent messages, I always added a current timestamp to the DataMap, as data changes weren't recognized correctly during early debugging sessions. Secondly, one has to create a request of transporting this item to the DataLayer so that it can be synced to the wearable by the Wearable Data Layer API. To distinguish between different types of data, I used a unique shared virtual path like `"mobile_to_wearable_data"`, similarly to the MessageAPI case described before. The wearable implements the DataListener interface, as part of a WearableListenerService which itself is a part of the Wearable Data Layer API. This interface enables listening for data changes, transports the data from the DataLayer into the application layer, and let's you specify what to do with the data, inside the onDataChanged() method. In this case, I used the data to create notifications and to update the user interface accordingly, which I will describe in more detail when I present the details of the wear application at the end of this chapter.

## 4.3 Representations and Transformation of Data

There are different representations of the same data inside the three components. Figure 4.1, on the left hand side, shows how the data that represents incoming maintenance tickets, is kept inside of an array of JSON objects in the implemented node.js server application. All relevant maintenance tickets are held inside server memory in this manner, after they've been parsed from an external source, like for example, an existing business software backend. For the small industrial study, I implemented a csv-to-json parser, which watches a remote directory for incoming csv files, parses them and appends the maintenance ticket to the existing array of JSON objects. Such a *maintenance ticket object* can consist of different attributes like an unique ***id***, a ***description*** of the failure, information about the ***error type***, the ***location*** of the afflicted machine or the ***cost center*** that a certain machine belongs to. In addition to those descriptive attributes, in our case each maintenance ticket is enriched with state relevant attributes in terms of binary bits, like if a certain ticket was ***delivered*** to a certain smartwatch or if a ***notification*** was created for it or if the user has ***seen*** it et cetera. All These state attributes are initialized as **false** and then get set to ***true***, whenever a smartwatch sends feedback to the server via the previously described ways of communication.

The server is able to send single JSON objects, representing a single maintenance ticket as well as the whole array of all active maintenance tickets, depending on the query type sent from the smartwatch. For example, when the server sends the whole array of JSON objects to the mobile application, after a smartwatch requested it, the received array is iterated upon by the mobile application and for each element of the JSON array a DataItem is created and sent to the DataLayer for the smartwatch to receive it in the next step. When these DataItems are received by the wear application, each attribute of the JSON object is used to either update the user interface or to decide if a notification has to be created. No data is stored permanently on the smartwatch. As soon as the application dies, all local data is lost. Therefore a request for the latest set of data gets issued whenever the wear application is started and also whenever a connection between wearable and mobile is (re)established.

## 4.4 Wear application

The wear application is the central component of the implemented prototype as it is the actual object under test for this thesis. I will describe it in more detail in this section. The wear application is the only part of the prototype that the end user interacts with, therefore it is the only component with a real graphical user interface. It consists of a total of seven classes which are shown in an UML class diagram attached at the

end of this thesis. The largest four classes are the MainActivity, the DetailActivity, the NotificationActivity and the DataLayerListenerService, which I will all describe in detail in the respective subsections in this chapter while also providing descriptive screenshots of the user interface where possible. The smaller classes are the SplashActivity, the ListAdapter and the ListItemTemplate class.

**SplashActivity** The SplashActivity was mainly created to briefly show the logo of the application when the application gets launched to avoid letting the user stare at a blank screen. Apart from its aesthetic function there is nothing noteworthy about it.

**ListAdapter** The ListAdapter class that I used is a brief modification of the standard example of a list adapter from the Android documentation. It's purpose is to deliver the data from the model to the view, thus in this case to populate the ListView with the maintenance ticket data.

**ListItemTemplate** The name of this class says everything about it. It is a template that each list item implements. It is a slightly modified version of the two-row ListItem template from the official Android documentation, with the difference of a color coded priority icon on the left hand side that I use to display the priority of a maintenance ticket inside my application.

### 4.4.1 Notifications

A wear notification only gets created when a certain maintenance ticket arrives at a smartwatch for the first time via the previously described socket.io and Wearable Data Layer API communication channels. In other words, when the *'notified'* attribute of a ticket's JSON object is set to *false*, which is the case initially when the server issues a new ticket, the smartwatch vibrates and the wear app issues a *local* notification as it receives the ticket data. A local notification is a notification that's neither created nor displayed on the smartphone but rather locally on the respective smartwatches. It is important to state that this is not the state of the art when it comes to Android Wear notifications. Best practice would be to use the Google endorsed method of notifying devices through their Google Cloud Messaging (GCM) service [4].

[4] https://developers.google.com/cloud-messaging/gcm

**Google Cloud Messaging (GCM)**

GCM is a cloud service for Android developers by Google that enables and simplifies the communication between a server and it's Android clients. It requires a registration at Google's developers page and a generation of an unique GCM-API key, which has to be specified inside the server application as well as the Android application that is supposed to use the service. GCM receives messages sent by one's server application and either directly sends them to the client device(s) via HTTP - which is not much different from the socket.io method I used - or notifies the respective device(s) about the existence of a new message addressed to them. The main benefit of the second method is that a device doesn't have to listen for incoming messages the whole time. It rather gets 'woken up' from a more battery life friendly state by the GCM service. This approach is highly recommended if battery life is important and the security risk of using cloud based solutions is moderate.

### 4.4.2 The NotificationActivity

Every notification is an instance of the NotificationActivity which can be seen in Figure 4.2. Picture 1 shows an incoming notification in it's condensed version. The only information the user gets from looking at this screen is the gravity or the priority of the task. In this case the orange electricity icon indicates that it is some kind of electrical issue with medium priority if one assumes that red symbolizes highest priority and green symbolizes low priority tickets.

The freely specifiable vibration patterns of notifications can be used to inform the wearer about the significance of an incoming notification without the user having to take a look at the smartwatch at all. For example one could use intrusive patterns if the notification is of highest priority and on the other have leave out vibration completely if the incoming ticket doesn't contain an urgent assignment but rather a long term recommendation or optional tasks. Similarly vibration patterns could be used to encode location related data or even for navigation as described by Herzog et al., in a US patent [HOC15].

The second graphic of Figure 4.2 shows the notification in it's expanded state. In this form, the description of the error is displayed in addition to the error priority and error id, and the user can decide if he or she wants to dismiss it by using a swipe-down gesture or to take a closer look at it by applying a swipe-left gesture. This will bring up a confirmation button (3) which on-click creates an intent for creating an instance of the DetailActivity of this specific maintenance ticket.


**Figure 4.2:** This is how the wearable notifications look like.

### Intents

Intents are a key concept of the Android operating system. They can be described as envelopes which get sent from and to Activities or Services to start instances of the receiving Activity or Service. This metaphorical envelope doesn't necessarily need to have a recipient specified and can even travel across the borders of applications. It can even travel cross-device in the case of Android Wear, meaning that a mobile application can fire an Intent that gets answered on a smartwatch. Such an Intent - still think of an envelope - can contain data as well, making it possible for Activities to hand data to other Activities. If no recipient is specified, the user gets asked which application shall respond to the request.

In this case, the notification already contains all the data we need, namely the whole JSON object of this specific maintenance ticket, making it possible to add the contents of this JSON object to the Intent via so called 'IntentExtras'. So when the button inside a notification gets clicked, we get to see the DetailActivity of this maintenance ticket showing all of this data, as pictured in the third graphic of Figure 4.2.

A demonstration of how a notification and the ticket-specific Intents are created is shown in Listing 4. First a Intent object is created in line 8. To ensure that by killing the activity that get's launched by an Intent - in this case a specific DetailActivity - doesn't bring us back to the OS launcher but rather to our MainActivity, the code displayed from line 10 through line 12 is needed. What this does is that the stack builder object will contain a virtual back-stack for the launched DetailActivity, which allows specifying the parent stack and the Intent that it was meant for. On the next lines the different IntentExtras are added to the intent. The code between the lines 23 and 33 shows how the details of a notification can be set. I am setting the correct icon according to priority and type of the machine failure, the text content of the notification in terms of failure description

as well as the vibration pattern and the corresponding PendingIntent object. Lastly the notification gets issued in line 38, where providing an id for every notification makes it recoverable and updatable for the future.

### 4.4.3 The MainActivity

The MainActivity is the main class of the wear application and thus the starting point of code execution whenever it gets launched. The main purpose of the MainActivity is to manage the application lifecycle via specific methods that are typical for any kind of Android application like *onCreate()*, *onDestroy()*, *onResume()*, *onPause()* et cetera. In this case the MainActivity's user interface is a ListView that holds all active maintenance tickets and presents them in a condensed manner, as pictured in Figure 4.3.


**Figure 4.3:** This is how the MainActivity (1) looks like. Clicking on an item (2) opens it's DetailActivity (3).

The ListAdapter that feeds the maintenance ticket data into the user interface is an extension of the standard two-row ListAdapter included in Android. I extended the Adapter with an ImageView element on the left hand side of each list element's layout that can hold different icons. In this case the icons use different colors and symbols to specify the priority and importance of a maintenance ticket as well as it's specialism.

Figure 4.4 shows the different variants of the ticket icon used in this case. Green icons indicate non-urgent long-term recommendations while red icons indicate emergencies and maintenance tickets of highest priority. Icons with the wrench symbol indicate that the machine failure is of mechanical nature and the lightning symbol indicates that a failure is of electrical nature. This component of code is very modular, therefore it is possible to add more classes of failures and colors with very little effort.

```java
/**
  *  Creating a notification for a specific maintenance ticket,
  *  given a JSON representation of it.
  *  @param obj
  */
private void createNotification(JSONObject obj) {
    // Intent pointing to DetailActivity
    Intent notificationIntent = new Intent(getBaseContext(), DetailActivity.class);
    // Preliminary process stack management
    TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
    stackBuilder.addParentStack(DetailActivity.class);
    stackBuilder.addNextIntent(notificationIntent);
    try {
        // Put Extras to Intent
        notificationIntent.putExtra("id", obj.getInt("id"));
        notificationIntent.putExtra("priority", obj.getString("priority"));
        notificationIntent.putExtra("description", obj.getString("description"));
        ⋮   ⋮   ⋮ ... and many more ...

        PendingIntent resultPendingIntent =
         ↪  stackBuilder.getPendingIntent(0,PendingIntent.FLAG_UPDATE_CURRENT);
        // Specify content of the notification as well as vibration pattern, ringtone and
         ↪  status LED pattern (not relevant for current smartwatches)
        NotificationCompat.Builder mBuilder =
                new NotificationCompat.Builder(this)
                        .setSmallIcon(iconFinder(errorCode, priority))
                        .setContentTitle("Neuer Fehler: "+obj.getInt("id"))
                        .setContentText(">: "+obj.getString("description"))
                        .setVibrate(new long[] {500, 500})
                        .setStyle(new NotificationCompat.BigTextStyle())
                        .setAutoCancel(true)
                        .setLights(Color.BLUE, 500, 500)
                        .setSound(RgtnMngr.getDefaultUri(RgtnMngr.TYPE_NOTIFICATION))
                        .setContentIntent(resultPendingIntent);

        NotificationManager mNotificationManager =
                (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
        // issue the notification to the operating SystemService
        mNotificationManager.notify(obj.getInt("id"), mBuilder.build());
            --> method that reports "notified = true" to the server goes here <--
    } catch (JSONException e) {e.printStackTrace();}
}
```

**Listing 4:** The ***createNotification()*** method, **file:** *DataLayerListenerService.java*.

**Figure 4.4:** All the variations of the ticket icons of this application.

Clicking on a list element opens it's DetailActivity via an Intent, very similarly to the described code beneath the *open* button of a notification.

### 4.4.4 The DetailActivity

The DetailActivity consists of a sticky color-coded topbar and a ScrollView containing the user interface elements in a vertically aligned manner. This design decision was influenced by the presentation of Android Wear 2.0 at the Google I/O 2016 conference, where the Android development team introduced Material Design for wearable devices [5]. Of course I wasn't able to use Android Wear 2.0 for the prototype of this thesis because of it being presented so late relative to the start date of this thesis, but the re-engineered application design patterns of Wear 2.0 came just in time, for me to apply some of the new concepts within my own user interface design.

A big issue that was introduced with the release of Android Wear was that having both vertical and horizontal scrolling enabled inside an application could make traversing it very confusing [6]. Therefore, the new recommended approach of simplifying wearable application design, is to use one dimensional scrolling only, in terms of scrollable vertical layouts, as explained in Figure 4.5. To simplify the user interface that I initially designed and to reduce user interaction times and confusion, I decided to go for this kind of scrollable vertical layout as pictured in figures 4.6 and 4.7 instead of having the user navigate through multiple screens, as planned earlier during the concept phase.

Screen 1 of Figure 4.6 shows the top most part of the DetailActivity of an error with the id 10116861. Depending on the importance of the ticket, which is gained from the priority attribute of a ticket during the onCreate() method of the DetailActivity - in this case M1 - it is possible to color the topbar respectively. In this case the maintenance

[5] Android Wear 2.0 presentation video - https://www.youtube.com/watch?v=LtD7eJp2ILo
[6] https://www.google.com/design/spec-wear/system-overview/vertical-layouts.html

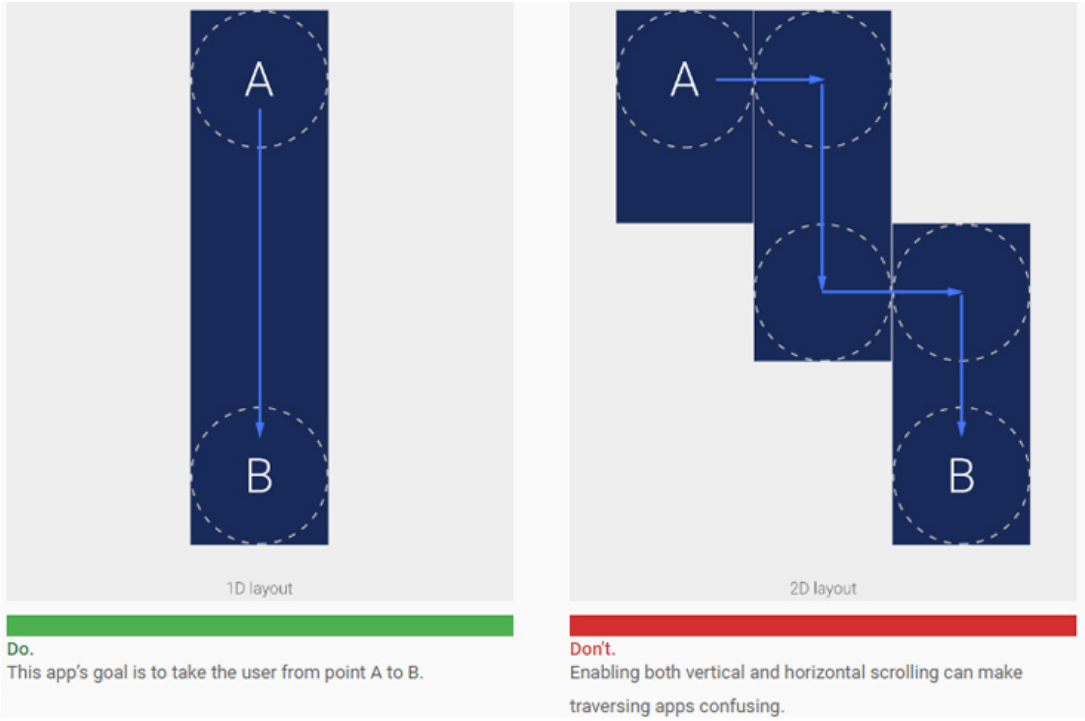



**Figure 4.5:** The new *Vertical layouts* pattern of Wear 2.0, from [**verticallayoutsgoogle**]

ticket is an urgent one, so the topbar is red. Of course it is possible to specify different colors with different meanings, but nonetheless it is important to state that I chose only a little set of colors for my application and all of the colors I used are part of the official material design color palette for developers by Google [7], again to keep the design simple, overseeable and clean.

The topbar is followed by a TableView where each row element contains some information about the maintenance task. Starting from the first row it contains the ticket priority, a four-digit code that classifies the failure, the location of the respective machine, the cost center that it belongs to, the date and time of creation of this ticket as well as which employee created this ticket in the first place. The second row item also specifies which maintenance staff is responsible for it, in this case the *'E'* in *'BIEJ'* stands for *electrical*. Under the TableView there is a text field which contains an explanation or description the reporting worker wrote when creating the ticket. Scrolling further down reveals the begin / pause maintenance button. This is pictured in the middle and right hand side screens of 4.7. This button (a) changes it's state and caption from *begin* to *pause* when pressed. The underlying begin-function creates a timestamp, sends it to the server and begins counting the duration of the maintenance task. The reason for creating a

[7] https://material.google.com/style/color.html

timestamp locally on the smartwatch rather than using the timestamp of the message that goes to the server is that the working times are always consistent like this, even if there is a brief delay or loss of connection somewhere between the server and the smartwatch. The pause button (b) switches back to the begin button when pressed. The underlying pause-function creates another timestamp, sends it to the server with the attribute *paused* = *true* and stops the chronometer that was counting the duration of the maintenance task.


**Figure 4.6:** First part of the DetailActivity GUI layout.

Beneath the button there is a drop down list containing predefined answers so the employees can report what exactly led to the solution of the maintenance task. The drop down list can be seen in the left hand screen of Figure 4.7. There is also an older version of the wear application which still uses Google's speech to text engine. Unfortunately there is no possibility to use it without a constant connection to the speech recognition services in the cloud.

### Predefined Answers as main input method

I felt that the least frustrating input method apart from speech to text was a set of predefined answers. Related work suggests that a keyboard is not the best idea on smartwatches so the input method of choice became this drop down list. The set of predefined answers can easily be edited. It would even be possible to offer multiple drop down lists where the first list could offer sentence starters, the middle lists could offer different actions, adjectives and adverbs and the final list may contain sentence closers that inform about the success of the maintenance mission. An example of such a constructed sentence would be:

```
EXAMINED <Machine> AT <Location> REPLACED OLD <Part> SUCCESSFULLY
```


**Figure 4.7:** Second part of the DetailActivity GUI layout.

In this case the user selected the predefined answer that states that the error could not be fixed (c). Pressing the end button below that closes this maintenance ticket, which means that it gets removed from the MainActivity's ListView after that. The work time that was measured before, gets transmitted to the server along with a *completed = true* flag, the selected predefined answer and a timestamp. In this case this was the only active maintenance ticket, therefore the ListView of the MainActivity is empty after this ticket was resolved, as shown in the right hand side screen of Figure 4.7. The interesting part of these transactions happens on the server side, which I will describe in a later section.

### 4.4.5 The DataLayerListenerService

The DataLayerListenerService is an extension of the WearableListenerService provided on the Android Wear documentation website [8]. As the name indicates, it extends the important Android concept of the *Service* class.

#### Android: *Service*

A Service in Android is similar to an Activity but it comes without an user interface and is not meant to interact with the user in any way. It is used whenever a longer-term operation is desired. The obvious advantage being that a Service can run in the background, even when an application get's paused or destroyed. An example of

[8] https://developers.google.com/android/reference/com/google/android/gms/wearable/WearableListenerService

a Service is the playback of audio files, which doesn't abort when the application's MainActivity gets destroyed but rather keeps running in the background. Services can be registered at OS-level inside the *manifest.xml* specification file of an application, therefore they can be flagged to get started at boot without the user having to open the application that uses this Service.

In this case, the WearableListenerService is a minimal working example of a background service that listens for incoming messages and DataItems coming from the mobile application. It already implements most of the APIs I used, like the MessageAPI and the Wearable DataAPI in terms of an instance of the MessageListener and the DataListener. Thus, it enables handling incoming data and messages through it's event-driven methods onDataChanged() and onMessage(). My DataLayerListenerService extends this basic functionality with the ability to launch notifications - as described in the Notifications section - and to inform the MainActivity about incoming data.

Initially, my prototype didn't use a Service but the MainActivity itself to handle incoming data. The importance of maximizing battery life then led to substituting the old code with this Service.

## 4.5 Mobile application

In contrary to the previously demonstrated wear application the mobile application doesn't have any kind of sophisticated user interface, due to the passive role of the smartphone inside the prototype. The mobile application is not intended for user interaction, therefore I used it's screen to display data that was helpful during debugging sessions. The layout of the mobile application's MainActivity consists of a sole element, namely a simple ListView that displays the same errors as the smartwatch. When clicking on one of the list elements a short Toast gets displayed, containing the raw JSON String of the clicked maintenance ticket. This proved to be very helpful, when I wanted to check if a certain flag or field got updated properly in the JSON representation of the data. Apart from this, there is no other feature that the user interface offers. The main purpose of the mobile applications only sole class is the onward transfer of every message and JSON object or JSON array that gets sent from and to the smartwatch by and to the server. This messaging middleware functionality is created by nesting the event-driven methods of the Wearable Data API within the event-driven methods of socket.io (and the other way around for the other direction). To get a deeper understanding of the concept take a look at Figure 4.8. A socket.io message listener **contains** the method that sends the MessageAPI message to the smartwatch as part of it's callback method and vice versa.

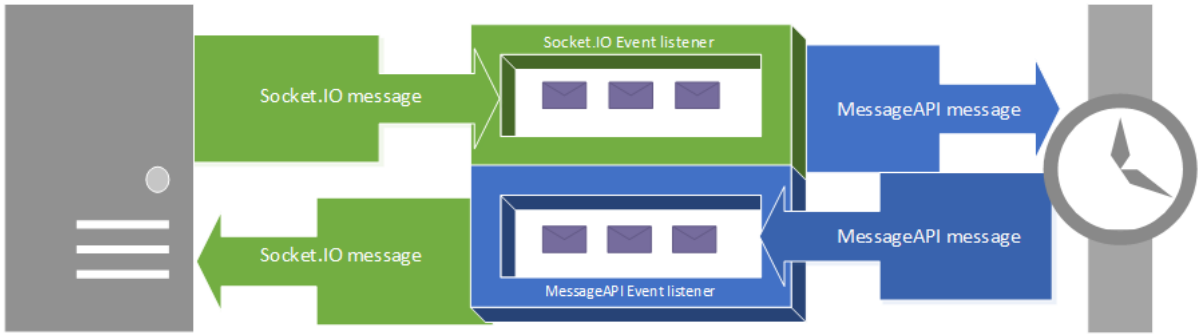



**Figure 4.8:** Nested communication methods enable reliable forwarding of messages and data.

## 4.6 Node.js Server application

The node.js server application of this prototype makes use of several external libraries and packages that simplify and enable communication and I/O operations. Each of these packages can be installed through the official *node package manager* (npm). The npm also takes care of the dependencies of dependencies automatically so that the developer doesn't have to manage dependencies at a lower level. The packages can be saved locally on the machine that is hosting the server application so that no active internet connection is needed to run it. This was the case during the test phase of this thesis, where the machine that hosted this program was cut off from the internet due to security reasons. I will describe some of the important packages that were used for the implementation, as well as their role within the server application:

**chokidar** The *chokidar* library calls itself "a neat wrapper around the node.js *fs* module". The included file system (fs) module of node.js enables watching directories and files for changes, doing I/O operations such as reading from and writing to a file but it lacked some functionality that I desired for my server application such as fault-free reporting of added, modified and deleted files within a watched directory on all operating systems. The fs module also cannot handle Windows paths very well. Chokidar fixes this issue by normalizing every path, independent of the underlying operating system.

**csvtojson** This library offers very diverse settings for converting csv files or strings into JSON objects. For example one can specify custom delimiters or if a csv file is headerless and the package then allows accessing the respective elements in a simple manner via 'field1.get(), field2.get(), etc..'. I used this package to integrate my prototype within the business software backend that was used during the test phase which produced separate headerless csv files for each maintenance ticket.

**json2csv** This library was used for the opposite direction and offered very similar functionality to the csvtojson package. Surprisingly both of them didn't offer complete support of the opposite direction of file format transformation.

**express** Express is a web framework for node.js applications requiring a minimal amount of setup. It provides a set of robust tools for HTTP servers, making it a popular choice for both web and hybrid applications. In this case, the express framework provided the basic HTTP functionality which the previously described socket.io framework needed as a dependency.

### 4.6.1 Event logging

Apart from messaging, the server logs every important event, like when each ticket got started, paused and completed into a separate file. Like this it is possible to measure the effects of such a wearable notification system compared to traditional systems. This functionality is achieved by overloading the classic JavaScript console.log() method [9].

When the server application receives a message or actual data from a smartwatch, which happens whenever the list of all maintenance tickets is requested or a maintenance ticket is updated, it updates the latest information into it's *runtime memory*, the previously mentioned array of JSON objects. To find a specific maintenance ticket inside runtime memory a method is iterating through each element of the JSON array and comparing the received id with the ids inside the JSON array, returning the object if there is a match.

### 4.6.2 Time data consistency logic

When a user clicks the begin button on a smartwatch, the server receives the timestamp of when the user pressed it shortly after that. It gets stored immediately as 'startTime' variable (analogue for 'startDate'). These timestamps are transported to the server in the *milliseconds* format inside a *long* variable. This long variable is packed into a JSON object along with some other important information that is necessary to ensure sound mathematics when combining the data of multiple sequences of *begin-pause-begin-...-* operations. To be precise each JSON object of this type contains the id of the smartwatch that sends it, the id of the respective maintenance ticket, a boolean value that says that the work has been commenced for this specific maintenance ticket in terms of a *started = true* flag and another flag that tells the server that this was a begin and not a pause

[9] https://goo.gl/QxX7va

operation (*paused* = *false*). Like this, the server application can distinguish between the identically looking timestamps of begin and pause messages. When a user completes a ticket, the server receives another timestamp which immediately gets stored inside the 'endTime' variable (analogue for 'endDate').

The interesting part is how the timeWorked variable is updated during each of these begin and pause events. The code from Listing 5 shows how the calculations are done. Passed time is added to the total work duration when a commenced ticket is halted (line 7) and subtracted when a paused ticket is commenced once again (line 10). Like this the working time is always consistently calculated on the server, as all messages are guaranteed to be delivered in FIFO order.

```
1  if (ticket.commenced == false && ticket.working == false) {
2      ticket.commenced = true;
3      ticket.timeWorked = 0;
4      ticket.startTime = jsonObject.time;
5      ticket.calcTime = jsonObject.time;
6      ticket.working = true;
7  } else if (ticket.commenced == true && ticket.working == false) {
8      ticket.working = true;
9      ticket.calcTime = jsonObject.time;
10 } else if (ticket.commenced == true && ticket.working == true) {
11     ticket.working = false;
12     ticket.timeWorked = ticket.timeWorked + jsonObject.time - ticket.calcTime;
13 }
```

**Listing 5:** Adding time passed between events when begun->paused, subtracting when paused->begun.

When the user completes a ticket, the callback method of the socket.io message listener that listens for the "*ticket completed*" event starts the process of grabbing the respective JSON object from runtime memory and writing it into a headerless csv file using this format:

```
<ID>|<StartDate>|<StartTime>|<EndDate>|<EndTime>|<TimeWorked>|<SolutionText>
```

# 5 Evaluation

To test the usability and user acceptance of the implemented prototype in a realistic setting I conducted an initial evaluation with real maintenance workers, at the manufacturing facilities of an industrial group. The goal was to find out if such wearable systems do have a positive effect on how maintenance tasks are done and managed today.

## 5.1 The test environment

The evaluation was conducted inside the maintenance staff office and lasted about three hours. The tests occurred at daylight coming through the multiple windows of the office, supplemented with artificial lighting coming from the ceiling. The participant was allowed to move freely inside his natural work environment for the duration of the tests. The smartwatch was mounted on the wrist of the participant for the duration of the tests. I used a Windows 7 laptop to run the node.js server application on, a Google Nexus 5X phone to run the mobile application on as well as a Samsung Gear Live, and a Motorola Moto 360 smartwatch to run the wear application on.

### 5.1.1 Participants

I handed a smartwatch to a single worker that was connected to the smartphone via bluetooth, while the smartphone was kept in bluetooth range of the smartwatch for the whole duration of the test. The participant was a 19-year-old male maintenance employee with the profession of an electrician. He uses an Android smartphone daily but never got in touch with a smartwatch before.

Besides this participant, I held conversations with multiple maintenance workers who did not participate in the actual test but were given the ability to wear the smartwatch and to give initial feedback about the look and feel of such a wearable system.

## 5.2 The testing procedure and course of events

The participant was told to ignore the present maintenance ticket reporting system that is a stationary and centralized workplace shared by all of the maintenance staff, and to trust the smartwatch system instead while not paying attention to the smartwatch itself.

I then monitored the present maintenance ticket reporting system closely and redirected incoming tickets to the smartwatch whenever they arrived. Since the tests were conducted during a very calm phase of the work day, I told the participant that we may supplement the system with artificial tickets if no real tickets get created but that every ticket has to be treated seriously.

I redirected the tickets to the smartwatch whenever the participant was concentrating on his work, to test how notifications are perceived and if they get noticed at all.

### 5.2.1 Questionnaire

After the test phase the participant was asked to fill out a questionnaire and also given the opportunity to give unstructured feedback in terms of spoken and written words. The questionnaire is attached in the appendix of this work.

### 5.2.2 Semi-structured interviews with the maintenance staff

Before the actual test phase I was able to speak to six maintenance employees in form of a semi-structured interview. The employees issued their ideas and concerns about a smartwatch based system. They felt that a smartwatch imposed serious safety risks in some use cases like when there is a possibility for the wristband to get stuck inside of a dangerous machine. Another concern was that they find themselves elbow deep in oily lubricants from time to time which could cause damage to the smartwatch itself or at least make the smartwatch touchscreen unusable when contaminated with oil. They thought that if that was the case they would be forced to take the smartwatch off before starting their repair task, which would disable the notification functionality on the one hand and raise frustration and interaction times due to always having to look after the smartwatch and remembering to put it back on after completing a task on the other hand. They clearly expressed that a smart*phone* would better serve their needs, as it can be carried in a pocket and substitute existing cordless phones as well. Nonetheless they mostly agreed that the presented wearable prototype is an improvement relative to the status quo. Most of the employees said that they could imagine working with such

a system, especially because it would eliminate the case when emergency repair tasks remain unnoticed if nobody is near the maintenance ticket terminal for a prolonged period of time.

## 5.3 Test results and findings

The maintenance employee that participated in our test scenario liked the smartwatch approach. In a trialogue with a company representative and myself the participant revealed that he thinks that such a system would highly benefit him and his colleagues, while also stating that he believes that the less tech savvy colleagues might dislike the smartwatch approach due to the small screens and small font size. After the test phase he was asked to fill out the questionnaire. The questionnaire uses 'Yes' or 'No' answers and a linear scale from 1 to 5 to express the level of agreement with a presented statement (and to rate certain aspects), where 1 stands for "strongly agree" (very good) and 5 stands for "strongly disagree" (very bad). The participant's answers of the questionnaire are attached at the end of this thesis as part of the appendix.

### 5.3.1 Analyzing response times

During the test phase, the server was logging every event that occurred. The participant knew that he was under surveillance for the duration of the test so the data is not representative. Apart from that, the sample size is way to small to make any kind of conclusions from it, but nonetheless I will explain how the measuring was done. The logfile consists of messages such as the following ones:

```
$ Notified: d2fd1b about new ticket: 10116867 successfully. Time: 11:53:10.
$ Watch d2fd1b has started work on ticket: 10116867. Time: 11:53:43.
$ Watch d2fd1b has halted work on ticket: 10116867. Time: 12:51:57.
$ Watch d2fd1b has completed ticket: 10116867. Time: 12:52:04.
$ Result csv: 10116867|20161104|115343|20161004|135157|005814|Behoben.
```

Like this it was possible to at least get an idea about the response times. Having analyzed the logfile and calculated the average response times, I found that it took the participant **10 seconds** on average **to open a notification**, **less than a second to notice** the vibration of a notification and **35 seconds** on average **to press the begin button** of a ticket.

### 5.3.2 Smartwatch hardware under industrial conditions


**Figure 5.1:** The two Samsung(1, 2) and Motorola (3, 4, 5) smartwatches during the tests.

To find out how commodity hardware in terms of consumer level smartwatches performs under industrial conditions, I did some tests with the two types of smartwatches that were available to us. Figure 5.1 contains five pictures which show the different form factors of the two smartwatches (1) vs. (4), their different closing mechanisms (2) vs. (3) as well as a touchscreen that was contaminated with machine oil (5) to test it's functionality under said conditions.

I tested the closing mechanisms of both the Samsung Gear Live and the Motorola Moto 360 smartwatches. As the Moto 360 uses the classic closing mechanism of traditional wrist watches as shown in picture 3 of 5.1, it wasn't possible to strip the smartwatch off of the wearer's wrist just by applying a pulling force to the wristband. In contrary to that, the Gear Live uses two pins which go into holes on the other side of the wristband (2). This closing mechanism opened when a moderate pulling force was applied to the wristband.

To seize one of the major concerns that were brought up during the debate with the maintenance staff, namely that they get into contact with machine lubricants quite often and that this might influence the user experience in a negative way, I took machine oil that is typically applied as a lubricant by the maintenance employees to reduce friction between machine components, and covered the touchscreens of both smartwatches with it. Surprisingly, the touchscreens were not influenced in any way by the lubricant as one could still navigate through the app normally. As both the Gear Live and the Moto 360 are compliant with the IP67 ingress protection standard, thus dust and water resistant up

to a depth of 1m, washing hands while wearing the smartwatch is no issue. Nonetheless waterdrops can influence the touchscreen interaction. As water conducts electricity it confuses the capacitance of the touchscreen and makes interaction difficult.

## 5.4 Discussion

The results and findings that were presented before are not representative for industrial smartwatch notification systems or wearable assistance systems in general as they are merely descriptive and presumably only replicable under the previously described conditions. Apart from that, the participant knew that he was surveyed for the duration of the test and therefore it is only natural that he started working on a ticket as soon as it arrived.

What I could observe though, is that the user interface layout of the wear application as well as the gesture based navigation through it generally pleased it's potential users. Presumably, tech savvy employees will support this kind of system more than those who don't use smartphones regularly, but I don't have any real results that substantiate this claim. As far as the wearing comfort is concerned no maintenance employee stated anything negative about it, but it yet remains to be seen if that changes when worn for a more realistic amount of time. Both employers and employees seemed to share a similar vision of how such a wearable notification system could benefit them.

The safety issue of wearing a wristband when performing repair tasks on potentially harmful machines has to be further evaluated. My current understanding is that there has to be some kind of sweet spot in terms of a predetermined breaking point on the wristband or closing mechanism of a smartwatch. But it seems that solving this issue will always will remain a tradeoff between the user being notifiable at all times and the cost of potentially damaging a machine permanently by dropping a solid item like a smartwatch into it, in case of emergency.

The amount of data that was displayed on the smartwatches was complete, in the sense of providing everything a maintenance worker needs to start and finish a task, on the wrist, without having to look up additional information at the stationary ticket terminal. I personally believe that solely the potential benefits from this finding are enough for companies to pursue wearable solutions.

The big issue that remains unanswered is the input of data on smartwatches. Predefined answers in terms of a drop down list are presumably not precise enough to satisfy the complexity and variance of repair solutions in this field. The consequence of this would be that some kind of stationary terminal would have to remain in place for the maintenance employees to be able to provide additional information to what led to

the solution of a task that they weren't able to provide using the input capabilities of a smartwatch. The input method that yet remains to be tested is the speech to text method. As mentioned in an earlier chapter, I did implement a speech-to-text version at first and while it worked surprisingly well during debugging sessions, it is no indication of how well it works inside of a manufacturing facility. The biggest potential issue with speech-to-text engines in industrial environments is that the noisy environment may make this form of input impossible or frustrating at least. As companies tend to avoid cloud solutions that are intended for end users, such as Google's speech analytics cloud service, which is a requirement for the Android speech-to-text engine, one might have to explore offline speech to text solutions at first.

## 5.5 Summary

This thesis has solidified the hypothesis that wearable notification systems can enhance or even replace existing stationary systems, even though no satisfactory proof could be acquired for this claim with the little amount of data collected during this initial evaluation. Nonetheless I believe that the potential benefits of such a wearable notification system outweigh the potential risks and costs a company has to consider before acquiring such a system and that the potential amount of improvement that such a system could provide justifies further research in this field.

# 6 Conclusion and Future Work

This chapter concludes this thesis. It contains a round up of the results and findings of this thesis and gives a lookout to possible future work in the field of industrial wearable notification systems and industrial wearable assistance applications in general.

## 6.1 Conclusion

In this thesis a wearable notification prototype was implemented for the common industrial use case of assisting maintenance employees at production facilities. In chapter 3 the concept of this prototype was described, starting with initial hand drawn sketches, which helped with the understanding of industrial maintenance tasks in general and usual work flows in this field. The hand drawn sketches were followed by more detailed user interface mock-ups and a choice of suitable technologies for the implementation. After that, first architectural decisions were made, based on the capabilities and restrictions of the chosen technologies. In chapter 4 the three different components of the prototype were described in detail. The different representations and transformations of data were discussed and the the different ways of communication between the prototype components were explained. As the wear component is the central object of investigation of this thesis, its user interface was described elaborately. In chapter 5 the initial evaluation of the prototype was presented followed by a discussion and a summary of the results. The results and findings are not representative of such systems, but nonetheless they indicate that wearable notification systems do have a potential of enhancing industrial maintenance tasks and existing processes. To what extent, is yet to be determined. The thesis is rounded up by the following part on future work, in which alternative and more in-depth approaches are discussed.

## 6.2 Future work

The functionality of the implemented prototype can be supplemented with different functions that may improve the benefits and acceptance of such a system. The imple-

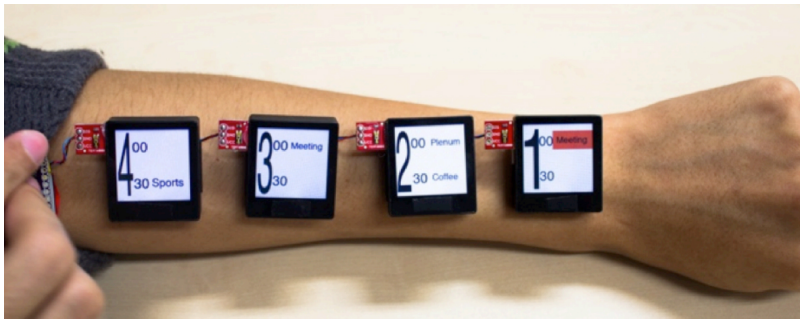



**Figure 6.1:** The display-enhanced forarm, consisting of a vector of interconnected screens, from [Olb+13].

mented prototype could be reduced to a two component system in the future as the upcoming version 2.0 of the Android Wear OS allows newer generation smartwatches to connect to the outside world without having to rely on a smartphone as an intermediary. This simplifies the process of reaching *specific* smartwatches drastically, as they will be directly addressable through the GCM service or HTTP based connections.

An example of such a function would be a routing mechanism that sends specific types of tickets to specific user groups. Such a routing mechanism could make use of the preexisting permissions and responsibilities structure of a company. If the 1st level employee feels that a certain ticket surpasses or doesn't match his qualifications or permissions he may redirect it to the next level or another user group by pressing a button in the wear application. This approach may drastically reduce the amount of communication between employees that is currently needed to manage these situations. One might even implement functions that resend important tickets to other employees automatically, if the initial recipient doesn't react to a notification in time.

As mentioned before, different smartwatch text input methods yet have to be evaluated in an industrial environment, including speech-to-text and on-screen keyboards, like the one that is going to be introduced in the next version of Android Wear. In my opinion, solving the problem of frustration free text input on smartwatches is one of the biggest steps towards making smartwatch applications more attractive to those who are yet undecided about using this technology as part of their business processes.

Even though the future of wearable applications in the industry looks promising, smartwatches may not be the best wearable to use, at least in our scenario, especially due to the limited screen size. Therefore it is up to future work to explore if different wearable approaches are more suitable for the use case of industrial maintenance tasks. Some promising wearable alternatives that might be more suitable but would still have to be evaluated under industrial circumstances are the display-enhanced forearm [Olb+13]. The advantage of the display-enhanced forearm, which is shown in figure 6.1 is that it

offers more screen area than a traditional smartwatch while also extending the design space. While a smartwatch can serve as a public display [PRJ15], it doesn't necessarily qualify as a public display in the use case of this thesis. This might be different with the augmented forearm approach. A display-enhanced forearm could be exposed to the public more noticeably than a smartwatch and thus provide room for different applications that benefit from a public screen.

Another possible option of extending the output of wearable devices are on-body displays [SOA16]. On-body displays would solve the battery life issue, as batteries could be sewed into the clothing. They would also remove the safety issues that come with wristbands. Nonetheless, suitable input methods would also have to be explored in the case of on-body displays in industrial environments.

# UML classes of the Wear Application

## Umfragebogen

Haben Sie schon einmal eine Smartwatch getragen?

- ○ Ja
- ○ Nein

Benutzen Sie regelmäßig ein Smartphone?

- ○ Ja
- ○ Nein

Die Smartwatch ist während der Arbeit angenehm zu tragen.

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Stimme sehr zu | ○ | ○ | ○ | ○ | ○ | Stimme gar nicht zu |

Die Smartwatch informiert mich besser über hereinkommende Aufträge, als das alte System.

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Stimme sehr zu | ○ | ○ | ○ | ○ | ○ | Stimme gar nicht zu |

Die Smartwatch lenkt mich von den eigentlichen Aufgaben ab.

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Stimme sehr zu | ○ | ○ | ○ | ○ | ○ | Stimme gar nicht zu |



## Umfragebogen

Ich nehme hereinkommende Aufträge (Vibration) gut wahr.

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Stimme sehr zu | ○ | ○ | ○ | ○ | ○ | Stimme gar nicht zu |

Die angezeigten Daten reichen aus, um den Auftrag ohne weiteres ausführen zu können.

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Stimme sehr zu | ○ | ○ | ○ | ○ | ○ | Stimme gar nicht zu |

Die Smartwatch ist generell viel zu klein, um darauf arbeiten zu können.

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Stimme sehr zu | ○ | ○ | ○ | ○ | ○ | Stimme gar nicht zu |

Die Schrift sollte größer sein.

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Stimme sehr zu | ○ | ○ | ○ | ○ | ○ | Stimme gar nicht zu |

Gibt es etwas spezielles was Ihnen gefallen hat?

Gibt es etwas spezielles was Sie gestört hat?

Haben Sie irgendwelche Verbesserungsvorschläge?

## Umfragebogen

Bewerten Sie folgende Eigenschaften der Smartwatch App

| | Schlecht | Nicht gut | Weder gut noch schlecht | Nicht Schlecht | Gut |
|---|---|---|---|---|---|
| Bedienbarkeit | ○ | ○ | ○ | ○ | ○ |
| Schnelligkeit / Flüssigkeit | ○ | ○ | ○ | ○ | ○ |
| Verständlichkeit | ○ | ○ | ○ | ○ | ○ |
| Layout / Aussehen | ○ | ○ | ○ | ○ | ○ |

3/3

# Questionnaire results

These are the answers of the 19-year old male maintenance worker that participated in the initial evaluation of the prototype:

```
Did you wear a smartwatch before? - No
Do you use an Android smartphone regularly? - Yes
The smartwatch feels comfortable when worn during work. - 1
A wearable informs me better about new tickets than the current system. - 1
The smartwatch perceivably takes my attention during work. - 4
I notice incoming notifications (vibration) well. - 2
The data provided on the smartwatch is enough to start and finish my work. - 1
The smartwatch is too small to be able to work on it. - 5
The font could be larger. - 5
Something special you liked? - The wearable approach in general.
Something special you disliked? - No.
Any recommendations? - The vibration could be stronger.
How do you rate the ease of use of the smartwatch app? - very good
How do you rate the performance/fluidity of the app? - very good
How do you rate the affordance of the smartwatch app? - good
How do you rate the user interface layout in terms of productivity? - good
```

# Bibliography


[AU14] M. Aehnelt, B. Urban. “Follow-Me: Smartwatch Assistance on the Shop Floor.” In: *HCI in Business: First International Conference, HCIB 2014, Held as Part of HCI International 2014, Heraklion, Crete, Greece, June 22-27, 2014. Proceedings*. Ed. by F. F.-H. Nah. Cham: Springer International Publishing, 2014, pp. 279–287. ISBN: 978-3-319-07293-7. DOI: 10.1007/978-3-319-07293-7_27. URL: http://dx.doi.org/10.1007/978-3-319-07293-7_27 (cit. on p. 29).

[Bru+13] D. P. Brumby, A. L. Cox, J. Back, S. J. J. Gould. “Recovering from an interruption: Investigating speed-accuracy trade-offs in task resumption behavior.” In: *Journal of Experimental Psychology: Applied* 19.2 (2013), pp. 95–107. DOI: 10.1037/a0032696. URL: http://dx.doi.org/10.1037/a0032696 (cit. on p. 25).

[FKS16] M. Funk, T. Kosch, A. Schmidt. “Interactive Worker Assistance: Comparing the Effects of In-situ Projection, Head-mounted Displays, Tablet, and Paper Instructions.” In: *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp ’16. Heidelberg, Germany: ACM, 2016, pp. 934–939. ISBN: 978-1-4503-4461-6. DOI: 10.1145/2971648.2971706. URL: http://doi.acm.org/10.1145/2971648.2971706 (cit. on p. 28).

[For+14] J. Fortmann, H. Müller, W. Heuten, S. Boll. “How to Present Information on Wrist-worn Point-light Displays.” In: *Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational*. NordiCHI ’14. Helsinki, Finland: ACM, 2014, pp. 955–958. ISBN: 978-1-4503-2542-4. DOI: 10.1145/2639189.2670249. URL: http://doi.acm.org/10.1145/2639189.2670249 (cit. on p. 40).

[Fun+16] M. Funk, J. Heusler, E. Akcay, K. Weiland, A. Schmidt. “Haptic, Auditory, or Visual? Towards Optimal Error Feedback at Manual Assembly Workplaces.” In: 2016 (cit. on p. 28).

[HOC15] S. Herzog, E. Ofek, J. Couckuyt. *Navigation instructions using low-bandwidth signaling*. US Patent 9,008,859. 2015. URL: https://www.google.com/patents/US9008859 (cit. on p. 53).

[Jun+15] W.-S. Jung et al. “Powerful curved piezoelectric generator for wearable applications.” In: *Elsevier Nano Energy* 13 (2015), pp. 174–181. DOI: http://dx.doi.org/10.1016/j.nanoen.2015.01.051. URL: http://dx.doi.org/10.1016/j.nanoen.2015.01.051 (cit. on p. 13).

[KFS] O. Korn, M. Funk, A. Schmidt. “Assistive Systems for the Workplace: Towards Context-Aware Assistance.” In: *Assistive Technologies for Physical and Cognitive Disabilities* (), pp. 121–133 (cit. on p. 28).

[KL10] K. Kumar, Y.-H. Lu. “Cloud computing for mobile users: Can offloading computation save energy?” In: *Computer* 43.4 (2010), pp. 51–56 (cit. on p. 18).

[MPK14] M. Manilal Patel, D. V. Kaushik. “Mobile Computing.” In: *IJISET* (2014) (cit. on pp. 17, 18).

[Man96] S. Mann. “Smart Clothing: The Shift to Wearable Computing.” In: *Commun. ACM* 39.8 (Aug. 1996), pp. 23–24. ISSN: 0001-0782. DOI: 10.1145/232014.232021. URL: http://doi.acm.org/10.1145/232014.232021 (cit. on p. 19).

[Man97] S. Mann. “Wearable computing: a first step toward personal imaging.” In: *Computer* 30.2 (1997), pp. 25–32. DOI: 10.1109/2.566147. URL: http://dx.doi.org/10.1109/2.566147 (cit. on p. 19).

[Nav04] N. Navab. “Developing killer apps for industrial augmented reality.” In: *IEEE Computer Graphics and Applications* 24.3 (2004), pp. 16–20. DOI: 10.1109/MCG.2004.1297006. URL: http://dx.doi.org/10.1109/MCG.2004.1297006 (cit. on p. 28).

[Nie13] Nielsen. *The Mobile Consumer: A global snapshot*. Tech. rep. The Nielsen Company, 2013. URL: http://www.nielsen.com/content/dam/corporate/uk/en/documents/Mobile-Consumer-Report-2013.pdf (cit. on p. 21).

[Olb+13] S. Olberding, K. P. Yeo, S. Nanayakkara, J. Steimle. “AugmentedForearm: Exploring the Design Space of a Display-enhanced Forearm.” In: *Proceedings of the 4th Augmented Human International Conference*. AH ’13. Stuttgart, Germany: ACM, 2013, pp. 9–12. ISBN: 978-1-4503-1904-1. DOI: 10.1145/2459236.2459239. URL: http://doi.acm.org/10.1145/2459236.2459239 (cit. on p. 72).

[PRJ15] J. Pearson, S. Robinson, M. Jones. “It’s About Time: Smartwatches As Public Displays.” In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. CHI ’15. Seoul, Republic of Korea: ACM, 2015, pp. 1257–1266. ISBN: 978-1-4503-3145-6. DOI: 10.1145/2702123.2702247. URL: http://doi.acm.org/10.1145/2702123.2702247 (cit. on p. 73).

[Pfe+14] M. Pfeiffer, S. Schneegass, F. Alt, M. Rohs. "Let Me Grab This: A Comparison of EMS and Vibration for Haptic Feedback in Free-hand Interaction." In: *Proceedings of the 5th Augmented Human International Conference*. AH '14. Kobe, Japan: ACM, 2014, 48:1–48:8. ISBN: 978-1-4503-2761-9. DOI: 10.1145/2582051.2582099. URL: http://doi.acm.org/10.1145/2582051.2582099 (cit. on p. 25).

[SC15] A. L. Smith, B. S. Chaparro. "Smartphone Text Input Method Performance, Usability, and Preference With Younger and Older Adults." In: *Human Factors: The Journal of the Human Factors and Ergonomics Society* 57.6 (2015), pp. 1015–1028. DOI: 10.1177/0018720815575644. URL: http://dx.doi.org/10.1177/0018720815575644 (cit. on pp. 21–23).

[SOA16] S. Schneegass, S. Ogando, F. Alt. "Using On-body Displays for Extending the Output of Wearable Devices." In: *Proceedings of the 5th ACM International Symposium on Pervasive Displays*. PerDis '16. Oulu, Finland: ACM, 2016, pp. 67–74. ISBN: 978-1-4503-4366-4. DOI: 10.1145/2914920.2915021. URL: http://doi.acm.org/10.1145/2914920.2915021 (cit. on p. 73).

[SR16] S. Schneegass, R. Rzayev. "Embodied Notifications: Implicit Notifications Through Electrical Muscle Stimulation." In: *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct*. MobileHCI '16. Florence, Italy: ACM, 2016, pp. 954–959. ISBN: 978-1-4503-4413-5. DOI: 10.1145/2957265.2962663. URL: http://doi.acm.org/10.1145/2957265.2962663 (cit. on pp. 25, 26).

[SS+14] A. Sahami Shirazi, N. Henze, T. Dingler, M. Pielot, D. Weber, A. Schmidt. "Large-scale Assessment of Mobile Notifications." In: *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*. CHI '14. Toronto, Ontario, Canada: ACM, 2014, pp. 3055–3064. ISBN: 978-1-4503-2473-1. DOI: 10.1145/2556288.2557189. URL: http://doi.acm.org/10.1145/2556288.2557189 (cit. on pp. 24, 26).

[Sch+16] S. Schneegass, T. Olsson, S. Mayer, K. van Laerhoven. "Mobile Interactions Augmented by Wearable Computing:" in: vol. 8. 4. IGI Global, 2016, pp. 104–114. DOI: 10.4018/IJMHCI.2016100106. URL: http://dx.doi.org/10.4018/IJMHCI.2016100106 (cit. on pp. 21, 26).

[Sut68] I. E. Sutherland. "A Head-mounted Three Dimensional Display." In: *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*. AFIPS '68 (Fall, part I). San Francisco, California: ACM, 1968, pp. 757–764. DOI: 10.1145/1476589.1476686. URL: http://doi.acm.org/10.1145/1476589.1476686 (cit. on p. 19).

[Wit08] C. Wittenberg. “Is multimedia always the solution for human-machine interfaces? - a case study in the service and maintenance domain.” In: *2008 15th International Conference on Systems, Signals and Image Processing*. 2008, pp. 393–396. DOI: 10.1109/IWSSIP.2008.4604449 (cit. on p. 31).

[XLH14] R. Xiao, G. Laput, C. Harrison. “Expanding the Input Expressivity of Smartwatches with Mechanical Pan, Twist, Tilt and Click.” In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’14. Toronto, Ontario, Canada: ACM, 2014, pp. 193–196. ISBN: 978-1-4503-2473-1. DOI: 10.1145/2556288.2557017. URL: http://doi.acm.org/10.1145/2556288.2557017 (cit. on pp. 23, 24).

[ZF94] J. Zahorjan, G. H. Forman. “The Challenges of Mobile Computing.” In: *Computer* 27.undefined (1994), pp. 38–47. ISSN: 0018-9162. DOI: doi.ieeecomputersociety.org/10.1109/2.274999 (cit. on p. 17).

[ZHU15] J. Ziegler, S. Heinze, L. Urbas. “The potential of smartwatches to support mobile industrial maintenance tasks.” In: *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*. 2015, pp. 1–7. DOI: 10.1109/ETFA.2015.7301479 (cit. on pp. 31, 32).

All links were last followed on November 29, 2016.

**Declaration**

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature