HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

INSTITUT
FÜR
SOFTWARE

Smart Navi Watch

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2015

Autoren: Konstantin Kayed, Mirco Strässle
Betreuer: Prof. Stefan Keller, GeometaLab HSR

Abstract

Smartwatches avancieren zu einem nützlichen Begleiter für Personen mit Smartphones. Besonders hilfreich könnten sie - dank ihrer hohen Verfügbarkeit am Handgelenk - bei der Fussgängernavigation sein. Momentan fehlt es jedoch noch an sinnvollen Applikationen.

In dieser Arbeit wurde geprüft, wie die frei verfügbaren Kartendaten von OpenStreetMap in einer GNSS-Navigationsanwendung für Smartwatches mit Android Wear zur Anwendung kommen können und welche Funktionen für einen Fussgänger sinnvoll sind.

Es wurden mehrere Umsetzungsmöglichkeiten geprüft. Durchgesetzt hat sich eine Plugin-Lösung zur bekannten OpenSource Navigationsapplikation OsmAnd für Android Smartphones, welche OpenStreetMap-Kartendaten verwendet. Das in dieser Arbeit entwickelte OsmAnd-Plugin kommuniziert mit der ebenfalls selbst entwickelten SmartNaviWatch-Applikation auf der Smartwatch über die standardisierte Bluetooth Schnittstelle.

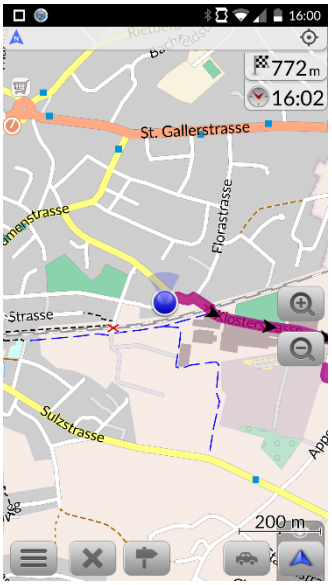
Der Funktionsumfang der SmartNaviWatch Applikation umfasst 1. das Anzeigen der aktuellen Position und 2. von wichtigen Informationen während einer laufenden Navigation auf dem Smartphone (Richtungsanweisungen, Kartenausschnitt). Die Navigationsinformationen werden automatisch mit fortschreitender Navigation auf der Smartwatch aktualisiert und der Benutzer wird mittels Vibration darauf aufmerksam gemacht.

In verschiedenen Praxistests mit Benutzern hat sich der Mehrwert einer Navigationshilfe am Handgelenk, wie sie die SmartNaviWatch Applikation darstellt, bestätigt. Das frühe Entwicklungsstadium von Smartwatches und speziell von Android Wear verhindert jedoch noch die Massentauglichkeit. Es ist dies eine der ersten Applikationen dieser Art für Android Wear bzw. allgemein für Smartwatches.

Weitere Informationen: http://kkade.github.io/SmartNaviWatch/

Management Summary

Problemstellung



Navigationssysteme und Mobiltelefone sind Bestandteil des heutigen Alltags geworden. Sie erleichtern die Navigation und ermöglichen uns einfachen Zugang zu Kartendaten. Doch als Fussgänger stört es eigentlich, wenn das Smartphone ständig in der Hand gehalten werden muss, um sein Ziel zu erreichen.

Der Nutzer muss das Display immer wieder betrachten um festzustellen, ob er noch auf der richtigen Route ist und wohin er nun laufen muss. Um dennoch die Hände frei zu haben, wird das Handy häufig in der Hosentasche oder in einem Rucksack verstaut. Somit resultiert das Navigieren entweder in einem pausenlosen Halten des Smartphones oder in einem ständigen Hervorholen und Wegstecken des Handys.

Mit dem Aufkommen von Smartwatches bietet sich hier eine neue Möglichkeit zur Navigationsunterstützung für den Benutzer. Obwohl fast jede Plattform eine Kartenapplikation von Haus aus besitzt, ist deren Nutzung nur mit herstellereigenen Navigationssystemen und proprietären Kartendaten möglich. Zudem zeigen die meisten Apps nur textbasierte Infos zum aktuellen Navigationsschritt an.

Vorgehensweise

Erster der Arbeit war die Ausarbeitung der Anforderungen an eine Navigationsapplikation, welche die Problemstellungen elegant und dezent lösen kann. Zu diesen Anforderungen zählt das Bestimmen des aktuellen Standortes, die Routennavigation und optional auch die Anzeige einer Karte zur besseren Orientierung.

Nach der Definition der Anforderungen wurden im Zuge eines Lösungskonzeptes mehrere Varianten ausgearbeitet und danach evaluiert. Zu den Kriterien zählten Wartbarkeit, Kosten, Nutzerfreundlichkeit, Funktionsumfang, Erweiterbarkeit, Risiko und Projektaufwand. Aufgrund dieser Kriterien fiel die Entscheidung auf eine Plugin-Lösung für die Navigationsapplikation OsmAnd, welche mit OpenStreetMap Karten arbeitet.

Die ausgearbeitete Variante wurde dann mittels eines agilen Prozess in ein fertiges Softwareprodukt überführt. Das fertige Produkt besteht aus einer Mobilapplikation mit zugehöriger Benutzeroberfläche auf der Smartwatch. Zudem wurde ein Plugin für OsmAnd erstellt, welches mit der Mobilapplikation kommuniziert, um Nachrichten an die Watch zu senden.

Nach der Implementierung der Hauptfunktionen wurde die App einem Praxistest unterzogen. Dieser umfasste sowohl die Nutzbarkeit der Smartwatch an sich, als auch die tatsächlichen Einsatzmöglichkeiten der neuen Navigationsapplikation. Mit den Praxistests konnten die Entscheidungen für die User Experience auf einer guten Grundlage gefällt werden. Der Agile Entwicklungsprozess ermöglichte eine schnelle Iterationsgeschwindigkeit und stellte damit einen effizienten Feedback-Loop zur Verfügung.

Lösungskonzept

Variante 1: Vollständige Eigenentwicklung

Diese Variante umfasst eine eigenständige Applikation auf dem Mobiltelefon. In der App werden sowohl Routenberechnung und Kartenspeicherung für die Offlinenavigation, als auch Onlineabfragen zur Navigation durchgeführt.

Diese Anwendung kommuniziert mit der, ebenfalls im Rahmen dieser Arbeit entwickelten, Smart-Watch Anwendung auf Basis von Android Wear. Die Wear Applikation wird dabei nur als erweitertes Display eingesetzt.

Variante 2: Plugin für bestehende Applikation

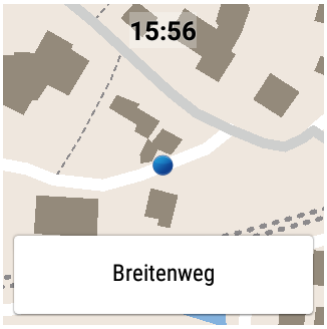
Diese Variante beinhaltet ein Plugin für eine bereits bestehende Navigationslösung. Dieses sendet dann die Navigationskommandos an die selbst entwickelte Wear-App. Die App auf der Smartwatch erhält ein öffentliches API um die Anbindung erleichtern. Die auf Kriterien gestützte Suche nach einem Hostsystem für das Plugin führte zu OsmAnd. OsmAnd ist eine Karten- und Navigationsapplikation unter Android, die mit Kartendaten von OpenStreetMap arbeitet.

Entscheidung

Die Entscheidung fiel auf die Realisierung eines Plugins für OsmAnd. Damit kann der Benutzer eine bereits bekannte Navigationsapplikation mit einer Fülle von Funktionen benutzen und erhält alle Vorteile der SmartNaviWatch.

Arbeitsergebnisse

Es wurde ein Plugin in OsmAnd realisiert. Das Plugin verbindet sich mit dem Navigationssystem und erhält von diesem alle wichtigen Ereignisse (z.B. Richtungswechsel oder Neuberechnung der Route) mitgeteilt. Daraufhin wird eine Meldung an die Smartwatch App ausgelöst und diese Zeigt dem Benutzer eine Meldung mit den entsprechenden Navigationshinweisen an.



Über die App auf der Smartwatch kann der Benutzer jederzeit seine Position abfragen. Dabei werden ihm die aktuelle Position, ein Kartenausschnitt und eine Bezeichnung (Strassenname oder Gebäudename) für seine Lokation ausgegeben. Falls der Benutzer sich momentan im Navigationsmodus befindet, wird ihm zusätzlich auch noch der Navigations-status angezeigt.

Erhält die Smartwatch App eine Meldung für einen Richtungswechsel, wird dieser dem Benutzer mittels Vibrationsalarm mitgeteilt. Danach kann der Benutzer auf dem Display der Watch diverse Informationen wie die Art des Richtungswechsels oder die Restzeit für die Route sehen. Alle Informationen werden auf einem Kartenausschnitt angezeigt. Damit erleichtert SmartNaviWatch die Navigation deutlich.

Testergebnisse

In einem ersten Test wurde die Art und Funktion eines Vibrationsalarmes untersucht. Konkret ging es um die Fragestellung, ob ein bestimmter Code (1x Vibrieren = links, 2x Vibrieren = rechts) die Navigation erleichtern kann. Hier konnten die Tests klar zeigen, dass dieses Vibrationsschema seine Grenzen hat (z.B. zum Linksabbiegen mehrere Strassen verfügbar sind) und daher nicht in dieser Form umgesetzt werden sollte. Es wurde aufgrund weiterer Tests entschieden, einen einfachen Alarm einzurichten, welcher dem Benutzer nur signalisiert, dass ein Ereignis eingetreten ist.

Neben den Tests zur Vibrationsalarmierung wurde getestet, ob die dargestellten Informationen für die Navigation ausreichend sind. Angaben zu Restzeit, der aktuellen Position und dem nächsten Navigationsschritt empfanden die Testpersonen als informativ und praktisch um die Navigation durchzuführen. In dichter bebauten Gebieten gab es jedoch teilweise Schwierigkeiten.

Sobald der Testperson aber die, als optionales Ziel definierte, grafische Karte auf der Uhr angezeigt wurde, konnten alle Testpersonen das Ziel ohne Irrwege erreichen. Aufgrund dieser Erkenntnis wurde zusätzliche Entwicklungszeit in die Kartendarstellung investiert um den Nutzen der App weiter zu steigern.

Ausblick

Die Hinzunahme einer Smartwatch mit SmartNaviWatch kann also die umständliche Navigation mit dem Mobiltelefon deutlich verbessern. Doch trotz der erfolgreichen Tests gibt es noch einiges Verbesserungspotenzial. So könnte man dem Benutzer verschiedene Einstellungsmöglichkeiten geben, sodass er sich die App an seine Nutzung anpassen kann. Zudem sehen wir noch Potenzial bei der Kartendarstellung. Diese Zeigt momentan eine kleine Menge von Objekten an, würde aber mit der Erweiterung um Strassennamen und Grasflächen deutlich an Nutzen gewinnen.

SmartNaviWatch läuft momentan nur als Plugin in OsmAnd, allerdings wäre es denkbar, dies auf weitere Navigations-Apps auszuweiten. Damit könnten noch mehr Nutzer in den Genuss der bequemen Navigationsmöglichkeit kommen.

Mit dem ausweiten der Nutzerbasis stellt sich auch die Frage nach zusätzlicher Funktionalität. Dabei wären zum Beispiel eine Scrolling- und Zoomfunktion für die Kartenansicht denkbar. Auch könnte das Erfassen von Markern und Notizen einen Mehrwert für die Benutzer bedeuten.

In Zukunft werden Spracheingaben immer wichtiger werden. Daher sehen wir auch für SmartNaviWatch die Erweiterungsmöglichkeit zur Sprachsteuerung. Damit könnte die Navigation komplett hands-free erfolgen und noch leichter von der Hand gehen.

Aufgabenstellung

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

Smart Navi Watch - Eine Smartwatch als dezenter Navigationshelfer

- Studienarbeit, FS 2015, Konstantin Kayed und Mirco Strässle
- Betreuer: Prof. Stefan Keller, GeometaLab HSR
- Partner: - (Community)

Hintergrund und Problemstellung

Smartwatches sind im Trend, doch fehlt es noch an sinnvollen Anwendungen - diese Mobile App-Kombination für Android Gear und Smartphone hat das Potential für eine nützliche App...

Navigationssysteme und Routenplaner sind Bestandteil des heutigen Alltags geworden v.a. beim Autofahren. Doch unterwegs zu Fuss (oder auf dem Velo) stört es eigentlich, wenn man das Smartphone ständig hochhalten muss, um ans Ziel zu gelangen.

Viel eleganter wäre es, wenn man durch Vibrationen oder einen kurzen Blick auf die Uhr ans Ziel geführt würde. Eine Smartwatch könnte so eine Hilfestellung sein, ohne abzulenken oder überhaupt aufzufallen (sog. „Internet der Dinge" und „Ubiquitous Comuting"). Zusätzlich könnte mittels Long-Press der aktuelle Standort angezeigt werden (Strassenname, Kartenausschnitt).

Ziel

Es soll ein Android Wear App realisiert werden, das 1. über Vibrationen sowie 2. über Routinganweisungen (Text ev. Kartenausschnitt) den Weg weist. Dies v.a. für die "Moto 360" von Motorola und auf Basis bestehender Routing-Webservices. Zusätzlich soll mind. konzeptionell angestrebt werden, dass möglichst viele Smartwartches auf Basis Android Wear unterstützt werden.

Da die meisten Smartwatches (wie auch Moto 360) keine Internetverbindungen haben, muss zusätzlich ein Smartphone App entwickelt werden, das diese Aufgabe übernimmt. Betreffend Routenplaner bietet sich der Online Routenplaner Routing OSM an, der ev. noch für Mobile Phones (und ev. Offline-Nutzung) optimiert werden muss.

Folgende Teilaufgaben stellen sich bei dieser Arbeit:

1. Realisierung einer Smartwatch App für "Moto 360" (Android Wear) sowie einer Smartphone App die miteinander kommunizieren
2. Evaluation, ob Eigenentwicklung oder Plugin für die Smartphone Routing App OsmAnd
3. Realisierung des Smartphone Routing einerseits möglichst so, dass viele Smartwatches unterstützt werden und andererseits für den Offline-Betrieb
4. Härtetest der Smartwatch im Alltag: Teil der Arbei soll sein, die Tauglichkeit der Smartwatch App zu testen.

Lieferobjekte

- Hinweis: Die Definition der genauen Aufgabenstellung (Requirements Engineering) ist ein erster Teil der Arbeit.
- Android Wear und Mobile App als Demo mit Teletext.
- Android Wear App für Smartwatch "Moto 360" (ev. weitere Smartwatches).
- Android Mobile App für ein Smartphone

Betreuer: Prof. Stefan Keller 15.2.2015 Stef Keller

Eigenständigkeitserklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben,
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Ort, Datum: Rapperswil, 29. Mai 2015

Konstantin Kayed Mirco Strässle

Inhaltsverzeichnis

I TECHNISCHER BERICHT

I.1 Einführung

I.1.1 Vision

Navigationssysteme und Routenplaner sind Bestandteil des heutigen Alltags geworden v.a. beim Autofahren. Doch unterwegs zu Fuss (oder auf dem Velo) stört es eigentlich, wenn man das Smartphone ständig hochhalten muss, um ans Ziel zu gelangen. Viel eleganter wäre es, wenn man durch Vibrationen oder einen kurzen Blick auf die Uhr ans Ziel geführt und den aktuellen Standort erfahren würde. Eine Smartwatch könnte somit eine Hilfestellung sein, ohne abzulenken oder überhaupt aufzufallen. Diese Lücke soll mit einer Android Wear Navigations-App basierend auf OpenStreetMap geschlossen werden.

I.1.2 Ziele

Es soll eine Android Wear App realisiert werden, welche über Vibrationen sowie über Routinganweisungen (Text ev. Kartenausschnitt) den Weg weist. Dies v.a. für die "Moto 360" Smartwatch von Motorola und auf Basis von OpenStreetMap und bestehender Routing-Services. Zusätzlich soll mind. konzeptionell angestrebt werden, dass möglichst viele Smartwatches auf Basis Android Wear unterstützt werden. Da die meisten Smartwatches (wie auch die Moto 360) keine Internetverbindungen haben, muss zusätzlich eine Smartphone App entwickelt werden, die diese Aufgabe übernimmt. Folgende Teilaufgaben stellen sich bei dieser Arbeit:

- Realisierung einer Smartwatch App für "Moto 360" (Android Wear) sowie einer Smartphone App die miteinander kommunizieren
- Evaluation, ob Eigenentwicklung oder Plugin für die Smartphone Routing App OsmAnd
- Realisierung des Smartphone Routing einerseits möglichst so, dass viele Smartwatches unterstützt werden und andererseits für den Offline-Betrieb
- Härtetest der Smartwatch im Alltag: Teil der Arbeit soll sein, die Tauglichkeit der Smartwatch App zu testen: Dazu gehört ein Tauglichkeitstest (z.B. in der Stadt Zürich, in einem Dorf und beim Wandern) sowie inkl. Abschalten (zu Hause/bei der Arbeit/Studium/im Kino).

I.1.3 Rahmenbedingungen

Die realisierten Softwareprodukte sollen auf einer Kombination aus einem Android Smartphone zusammen mit einer Android Wear Smartwatch laufen. Dafür wurden folgende Testgeräte zur Verfügung gestellt:

- Motorola Moto 360
- Sony Smartwatch 3
- Asus ZenWatch
- Samsung Galaxy

I.1.4 Aufbau der Arbeit

Die Arbeit startet mit einer Analyse des Standes der Technik. Aufbauend darauf wurde dieser mit einem Prototypen verifiziert. Danach werden aufbauend auf den Defiziten mehrere Lösungsvarianten ausgearbeitet und bewertet. Die gewählte Variante wird dann nach agilen Softwaremethoden realisiert. Ein Praxistest ermittelt dann die Tauglichkeit der Navigationsapplikation.

Der detaillierte Projektplan und die Meilensteine sind im Kapitel II.6.3 beschrieben.

I.2 Stand der Technik

I.2.1 Bestehende Lösungsansätze und Normen

Fast jede Plattform besitzt eine Kartenapplikation von Haus aus, es gibt jedoch noch sehr wenige Third-Party Applikation in dieser Richtung. In diesem Kapitel sind die wichtigsten Applikationen kurz beschrieben. Folgende Eigenschaften treffen auf alle vorgestellten Applikationen zu:

» Zeigt textbasierte Infos zum aktuellen Navigationsschritt an
» Macht Benutzer durch Vibration auf den nächsten Navigationsschritt aufmerksam
» Navigation kann vom Smartphone gestartet werden

I.2.1.1 Google Maps auf Android Wear

Da Google Maps auf jedem Android Smartphone vorinstalliert ist, ist es auch direkt auf jeder Android Wear Smartwatch verfügbar. Die Funktionen sind jedoch stark eingeschränkt und beschränken sich vor allem auf das stark vereinfachte Anzeigen von Navigationsinfos. Das Anzeigen der aktuellen Position oder einer Karte ist nicht möglich.

Das starten einer Navigation kann entweder auf dem Smartphone erfolgen oder auf der Uhr mittels Google Sprachkommando.

Abbildung 1: Google Maps Navigationsanzeige

I.2.1.2 Apple Maps auf der Apple Watch

Apple hat die eigene Maps Applikation stark ins System integriert. So kann z.B. eine Navigation direkt aus einer Email gestartet werden, die eine Adresse enthält. Es ist möglich, sich die aktuelle Position anzeigen zu lassen und mit dem Drehrad auf der Seite der Apple Watch kann in der angezeigten Karte gezoomt werden. Bisher ist die Apple Watch die einzige Smartwatch mit interaktiven Karten.

Eine neue Navigation kann auf dem Smartphone oder per Sprachkommando auf der Smartwatch gestartet werden. Infos zum nächsten Navigationsschritt wird in reiner Textform angezeigt.

Abbildung 2: Apple Maps Navigationsanzeige

I.2.1.3 PebbleNav auf der Pebble Watch

Für die Pebble Watch gibt e seine spezielle Kartenapplikation, die auf den Schwarz-Weiss Bildschirm der Pebble Watch optimiert ist. Die App funktioniert jedoch nur in Verbindung mit iOS, nicht mit Android. Das Anzeigen der aktuellen Position ist nicht möglich, dafür werden Navigationsinfos mit einem Karten-ausschnitt ergänzt der die Route anzeigt. Das Starten der Navigation erfolgt immer über das Smartphone.

Abbildung 3: PebbleNav Navigationsanzeige

I.2.2 Prototyp Applikation Teletext

Als Prototyp und zur Einarbeitung in Android und die API zu Android Wear wurde eine App erstellt, die auf dem Handy die Topstories von www.teletext.ch herunterlädt und diese auf der Smartwatch anzeigt.

Mit diesem Prototyp wurden die wichtigsten Aspekte für die Umsetzung der eigentlichen Arbeit geprüft. Dazu gehören:

» Mit der Handy-App externe Daten laden und weiterbearbeiten
» Befehle vom Handy an die Smartwatch senden
» Komplexe Daten/Objekte vom Handy an die Smartwatch senden
» Daten auf der Smartwatch anzeigen
» Deployment der Handy- und Smartwatch-App auf physikalische Geräte
» Debugging der Handy- und Smartwatch-App mit der Entwicklungsumgebung

I.2.2.1 Architektur

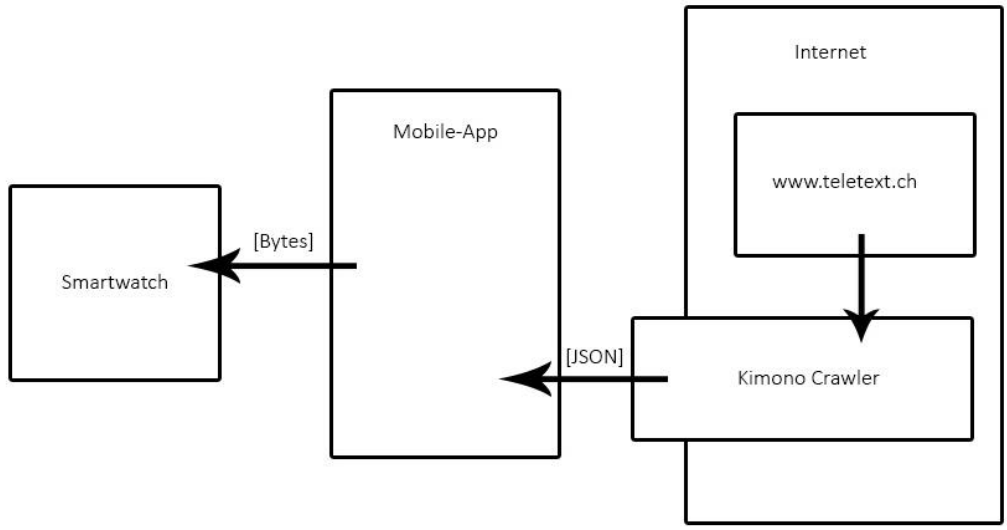


Abbildung 4: Architektur Teletext Prototyp

teletext.ch	Webseite mit den benötigten Daten im HTML-Markup. (http://www.teletext.ch/de/Webtext)
Kimono Crawler	Online-Crawler Dienst der die Daten aus der Teletext-Webseite ausliest und als JSON der App zur Verfügung stellt. (https://www.kimonolabs.com/)
Mobile-App	Holt JSON-Daten vom Crawler und deserialisiert sie zu einem Objekt. Anschliessend werden die Objekte in ein Byte-Array serialisiert und an die Smartwatch gesendet (Watch-API unterstützt zur Übermittlung nur Strings oder Byte-Arrays).
Smartwatch	Nimmt das Byte-Array entgegen, deserialisiert es und stellt die Daten für den User dar.

I.2.2.2 Installation

Smartwatches mit Android Wear sind nicht in der Lage eigenständig Apps zu installieren. Sie erhalten die Apps automatisch vom Smartphone mit dem sie verbunden sind. Aus diesem Grund musste die Wear-App des Prototyps mit der Smartphone-App zusammen paketiert werden in ein einziges APK-File. Dadurch kann der Prototyp in bekannter Art auf dem Smartphone installiert werden und steht dann automatisch auf jeder Uhr die mit dem Smartphone verbunden wird zur Verfügung.

Details zum Paketierungsverfahren sind auf der Android Entwicklerseite zu finden:
https://developer.android.com/training/wearables/apps/packaging.html

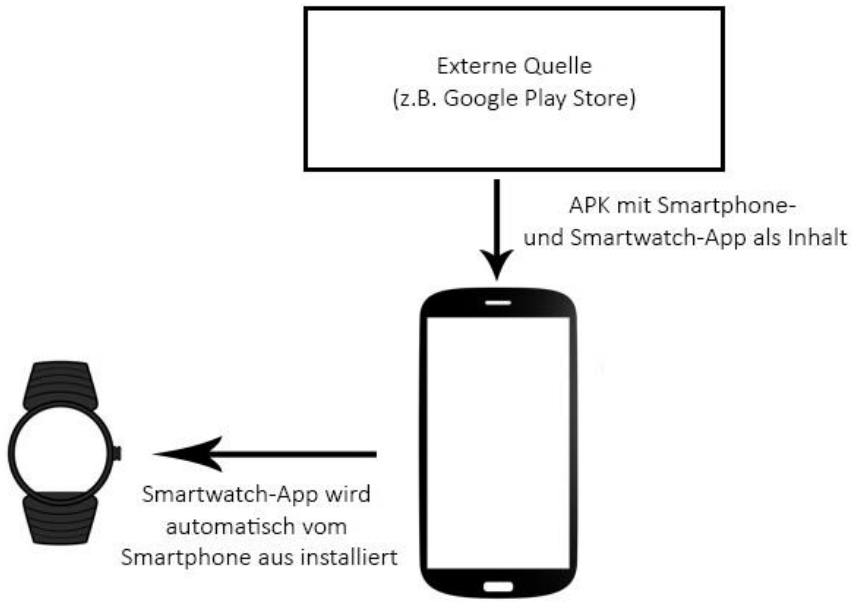


Abbildung 5: Installationsablauf Smartwatch App

I.2.3 Defizite

Von den bestehenden Smartwatch Applikationen zur Navigation, ist Google Maps für Android Wear am stärksten limitiert. Sie kann keine echten Kartenausschnitte anzeigen und auch den aktuellen Standort nicht abfragen.

Apple Maps auf der Apple Watch dagegen besitzt bereits einen sehr grossen Funktionsumfang, der durch die starke Systemintegration auch kaum von Third-Party Applikationen erreicht werden kann. Der Funktionsumfang von Apple Maps dürfte aktuelle das Ziel aller Kartenapplikationen auf Smartwatches sein. Mit Blick in die Zukunft sollte darauf hingearbeitet werden, die Kartenapplikationen auf den Smartwatches unabhängiger vom Smartphone zu machen.

Die umgesetzte Prototyp Applikation hat zudem gezeigt, dass die wichtigsten Anforderungen an die Arbeit umgesetzt werden können. Eine Einschränkung musste jedoch gemacht werden, im Use Case „Position bestimmen“, aufgrund der Erkenntnis, dass eine Android Wear Applikation keine beliebige Third-Party Applikation auf dem Smartphone starten kann.

I.3 Evaluation

I.3.1 Varianten

I.3.1.1 Eigenständige Applikation für Routing und Watch

Es wird eine eigenständige Applikation auf dem Mobiltelefon entwickelt. In der App werden sowohl Routenberechnung und Kartenspeicherung für die Offlinenavigation, als auch Onlineabfragen zur Navigation durchgeführt.

Die App kommuniziert mit der Smart-Watch Anwendung basierend auf Android Wear. Die Wear-App funktioniert dabei nur als erweitertes Display, wird aber ebenfalls im Rahmen dieses Projektes entwickelt.

I.3.1.2 Plugin in bestehendem Navigationssystem und Eigenständige App für die Watch

Es wird ein Plugin für eine bereits bestehende Navigationslösung entwickelt. Dieses sendet dann die Navigationskommandos an die selbst entwickelte Wear-App. Die App auf der Smartwatch erhält ein öffentliches API um die Anbindung erleichtern.

Bewertung	**Nutiteq libs**	**OsmAnd**	**Vespucci**
Dokumentationsqualität	+ +	-	-
Routing-Support	- -	+ +	- -
Offline verfügbare Karten	+	+	+
Individuelle Kartendarstellung möglich	+ +	+	-
Plugin System / Developer API vorhanden	+	+ +	-
Bearbeitung der Kartendaten möglich	+	- -	+ +
Routing "Turn by Turn"	- -	+ +	- -
Funktionalität für POI	+	+ +	+
Beispielcode vorhanden	+ +	- -	-
Fussgängernavigation möglich	-	+	-
Fazit	**5**	**8**	**-5**

Aufgrund der folgenden Evaluationsergebnisse wurde OsmAnd als Plugin-Host für diese Variante ausgewählt.

I.3.2 Bewertungskriterien

I.3.2.1 Wartbarkeit

Es soll eingeschätzt werden, welcher Aufwand in Zukunft nötig ist um die Applikation zu erweitern oder anzupassen.

I.3.2.2 Kosten

Bewertet werden die Kosten der Anschaffung von Lizenzen oder Entwicklungswerkzeugen.

I.3.2.3 Nutzerfreundlichkeit

Beschreibt, wie nutzerfreundlich die Variante ist. Dies betrifft in diesem Fall eher die eigentliche Handhabung, als die konkrete Usability des Programmes.

I.3.2.4 Funktionsumfang

Hierzu soll ermittelt werden, in welchem Mass der geforderte Funktionsumfang für die Lösung unterstützt wird.

I.3.2.5 Erweiterbarkeit

Dieser Punkt betrachtet den Umfang der Funktionen, welche dem Benutzer in Zukunft zur Verfügung stehen könnten.

I.3.2.6 Risiko

Es wird versucht einzuschätzen, wie hoch die Risiken für das Projekt sind, wenn die entsprechende Variante ausgeführt wird. Risiken, welche allgemein für ein Projekt gelten (z.B. Krankheiten o.Ä.) werden hier nicht betrachtet.

I.3.2.7 Projektaufwand

Es wird der Aufwand für die Realisierung der entsprechenden Projektvariante bewertet. Dazu zählen Konzept-, Analyse- und Entwicklungsaufwände. Ausserdem sollen die Testaufwände berücksichtigt werden.

I.3.3 Auswertung

Kriterium	Eigenständige Applikation für Routing und Watch	Plugin in bestehendem Navigationssystem und eigenständige App für die Watch
Wartbarkeit	Hoher Wartungsaufwand, da alle Komponenten des Systems eine Eigenentwicklung und somit Wartung benötigen.	Geringerer Wartungsaufwand, alle Navigations- und Kartenfunktionen werden vom Hostsystem übernommen.
Kosten	Keine Kosten	Keine Kosten
Nutzerfreundlichkeit	Der Benutzer muss eine neue Applikation erlernen.	Der User kann die bereits bestehenden Navigationsfunktionen nutzen. Diese sind bewährt und bekannt.
Funktionsumfang	Grundsätzlich beliebig, da es sich um eine Eigenentwicklung handelt. Alle Funktionen müssen aber implementiert werden.	Vollständig gemäss Anforderungen. Es sind aber bereits diverse Funktionen vom Host implementiert.
Erweiterbarkeit	Gut. Die Applikation wird allerdings für die momentan bekannten Anforderungen entwickelt, daher könnten diverse Refactorings notwendig sein.	Sehr gut. Die bereits bestehenden Funktionen im Plugin-Host decken ein breites Funktionsspektrum ab, welches über ein API zugänglich ist. Diese sind daher effizienter in den Funktionsumfang der Watch-App integrierbar (es muss nur ein UI realisiert werden). Zu beachten ist allerdings, dass zukünftige Funktionen, deren Machbarkeit und Qualität davon abhängen, wie das Host-System sich weiterentwickelt.
Risiko	Hoch. Es handelt sich um eine komplette Eigenentwicklung (keine Risikoverteilung möglich) mit hohem Entwicklungs-aufwand. Daher ist ebenfalls mit hohen Risiken zu rechnen.	Mittel-Hoch. Die Dokumentation kann unvollständig sein, und es können Bugs im Host-System vorhanden sein. Es handelt sich aber um bereits langfristig erprobte und genutzte Software, daher gehen wir eher von mittlerem Risiko aus.
Projektaufwand	Sehr hoch. Alle Funktionen müssen konzipiert, implementiert und getestet werden.	Mittel. Es kann auf eine Vielzahl von bereits implementierten und getesteten Funktionen zurückgegriffen werden.

I.3.4 Ergebnis

Aufgrund der durchgeführten Evaluation wurde Variante 2 für die Realisierung freigegeben. Nachfolgend sind die wichtigsten Gründe nochmals aufgeführt:

» Funktionsumfang kann vollständig umgesetzt werden
» Nutzer muss kein neues Interface lernen
» Wartungsaufwand wird kleiner eingeschätzt
» Geringerer Projektaufwand mit kleinerem Risiko

Folgende Nachteile nehmen wir dafür in Kauf:

» Risiken bezüglich Dokumentationsqualität des Host-Systems
» Qualitätsrisiken im Host-System (aktive Community, daher dürften Bugs nicht lange bestehen bleiben)
» Einschränkungen bei Erweiterbarkeit sofern es Zielkonflikte gibt

I.4 Umsetzungskonzept

Auf der Smartwatch sollen auf einen Blick die wichtigsten Navigationsinfos, die Uhrzeit sowie der aktuelle Standort klar ersichtlich sein.

Es wurden zwei Konzepte ausgearbeitet, die sich dadurch unterscheiden, dass die eine Variante Informationen grafisch über einen Kartenausschnitt liefert, die andere jedoch rein textbasiert Informationen anzeigt. Erstere Variante wird bevorzugt, hängt aber von den technischen Einschränkungen von OsmAnd ab.

I.4.1.1 Navigation starten

Die Navigation wird auf dem Smartphone in der OsmAnd App wie gewohnt gestartet. Das SmartNaviWatch Plugin in OsmAnd muss aktiv sein.

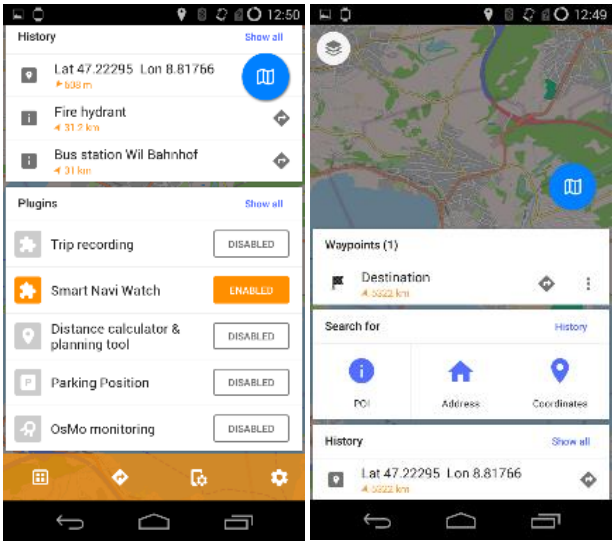


Abbildung 6: OsmAnd UI

I.4.1.2 Navigationsinfos anzeigen

Läuft auf dem Smartphone eine Navigation, wird automatisch die SmartNaviWatch App auf der Smartwatch gestartet und die aktuellen Navigationsinfos angezeigt. Der Navigationsfortschritt ist anhand einer Zeitanzeige ersichtlich, die die verbleibende Zeit zum Ziel angibt.

Mit Kartenunterstützung:

Abbildung 7: Konzept - SmartNaviWatch UI während Navigation mit Karte

Ohne Kartenunterstützung:

Abbildung 8: Konzept - SmartNaviWatch UI während Navigation ohne Karte

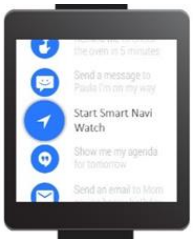
I.4.1.3 Navigationsupdate für nächsten Schritt

Steht eine Abzweigung, o.ä. unmittelbar bevor, wird der Benutzer durch Vibrieren der Smartwatch darauf aufmerksam gemacht und die Smartwatch zeigt die aktualisierten Navigationsinfos an.

Weicht der Benutzer von der geplanten Route ab, wird er durch 2-faches Vibrieren der Uhr gewarnt.

I.4.2 Positionsbestimmung

I.4.2.1 Smartwatch App starten



Die SmartNaviWatch kann direkt auf der Smartwatch gestartet werden.

Abbildung 9: Konzept - Direktes starten von SmartNaviWatch

I.4.2.2 Positionsanzeige

Die aktuelle Position wird angezeigt (Position wird über Rückfrage an Smartphone bestimmt). Als aktueller Standort wird, falls vorhanden, die aktuelle Strasse angezeigt und/oder ein relevanter Ort in der Nähe (z.B. Bahnhof).

Mit Kartenunterstützung:

Abbildung 10: Konzept - SmartNaviWatch UI bei Positionsbestimmung mit Karte

Ohne Kartenunterstützung:

Abbildung 11: Konzept - SmartNaviWatch UI bei Positionsbestimmung ohne Karte

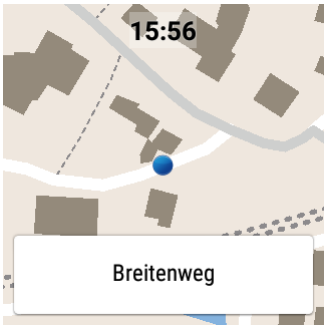
I.5 Resultate

I.5.1 Realisierte Softwareprodukte

I.5.1.1 Watch UI

Zur Navigationsunterstützung für den Benutzer wurde eine zugleich simple und effektive App für Smartwatches mit Android Wear implementiert. Diese App muss nicht separat installiert werden, sondern wird zusammen mit der Mobile-App installiert. Voraussetzung für die Nutzung der Smartwatch App ist somit einerseits die Installation der Mobilapplikation und andererseits das Navigationssystem OsmAnd mit dem entsprechenden Plugin. Nachfolgend werden die Features dieser App vorgestellt.

I.5.1.2 Anzeige der Aktuellen Position



Startet der Benutzer die App manuell direkt auf der Smartwatch, wird eine Positionsanfrage an das Navigationssystem gesendet. Danach erhält der Benutzer seine aktuelle Position dargestellt.

Abbildung 12: SmartNaviWatch UI bei Positionsbestimmung mit Karte

I.5.1.3 Vibrationsalarm für Hands Free Navigation

Um dem Fussgänger eine Navigation ohne ständiges halten eines Mobiltelefons zu ermöglichen, wurde die Smartwatch als bequem zu tragendes Display hinzugefügt. Damit man nicht ständig auf das Display schauen muss, um die nächste Abzweigung zu verpassen, wurde eine Vibrationsfunktion hinzugefügt.

Sobald der Nutzer in den Bereich einer Richtungsänderung kommt, wird diese ihm per Vibrationsalarm angekündigt. Aufgrund der Testergebnisse mit verschiedenen Usern wurde darauf verzichtet, die Vibrationsalarme für Links- und Rechtsabbiegen unterschiedlich zu gestalten.

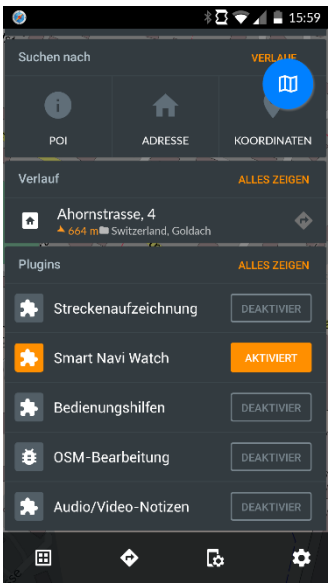
Abbildung 13: SmartNaviWatch UI bei der Navigation mit Karte

I.5.1.4 Mobile Proxy App

Für das Deployment und die Kommunikation zwischen Uhr und Handy muss eine App auf dem Mobiltelefon installiert werden. Diese App hat für den User keine weitere Funktion und muss nur einmalig installiert werden. Sie zeigt dem User eine kurze Hilfsnachricht an.

Abbildung 14: SmartNaviWatch UI auf dem Smartphone

I.5.1.5 OsmAnd Plugin



Das Plugin in OsmAnd ermöglicht dem Nutzer die Anbindung der Watch App an die bewährten und bekannten Kartendaten und Navigationsfunktionen von OsmAnd. Es benachrichtigt die App automatisch und sendet die Kartenansicht an die Android Wear App. Zudem beantwortet es die manuelle Positionsanfrage durch den Benutzer.

Der grosse Vorteil des Plugins in OsmAnd liegt auf der Hand: Der Nutzer kann einerseits ein vollständiges Navigationssystem mit Offlinekarten und POI Funktionen verwenden und hat andererseits alle Vorteile einer Watch als Navigationshelfer zur Verfügung.

Abbildung 15: Aktivierung des OsmAnd-Plugins

I.5.2 Ergebnisse aus Tests

I.5.2.1 Erkenntnisse Smartwatches allgemein

Aufgrund der stark eingeschränkten Funktionalität der Android Smartwatches bieten sie dem Benutzer über das einfache Verwalten von Smartphone Notifications hinaus kaum einen Nutzen. Ebenfalls ist es für Third-Party Applikationen schwierig essentielle Funktionen der Smartphone Apps auf den Smartwatches anzubieten, die über das Anzeigen von Informationen hinausgehen.

Die schnell fortschreitende Weiterentwicklung der Smartwatches und des Betriebssystems Android Wear deuten jedoch darauf hin, dass sich diese Möglichkeiten in naher Zukunft stark verbessern werden. Ein Ausblick über zukünftige, mögliche Weiterentwicklungen in Bezug auf die App SmartNaviWatch gibt es im Kapitel I.5.4.

I.5.2.2 SmartNaviWatch mit OsmAnd

I.5.2.2.1 Vibrationsfeedback

Fragestellung Ist eine Navigation nur über Vibrationsfeedback mit Vibrationsschemas wie z.B. 1 x Vibrieren für rechts abbiegen und 2 x Vibrieren für links abbiegen möglich?

Testergebnis Die Idee mit Vibrationsschemas Informationen an den Benutzer zu übertragen wurde einstimmig von allen Testpersonen verworfen. Dabei gab es folgende Kritikpunkte:

- Schema merken ist mühsam
- Notifications von anderen Apps können verwirren
- Über Vibration zu wenige Informationen verfügbar falls mehrere Möglichkeiten z.B. zum links abbiegen Verfügbar sind

Als eindeutig nützlicher wurde die Variante empfunden, in welcher der Benutzer durch einfaches Vibrieren auf aktualisierte Navigationsinfos aufmerksam gemacht wird und sich dann mit einem kurzen Blick auf die Smartwatch informieren kann.

I.5.2.2.2 Positionsbestimmung

Fragestellung Lässt sich der aktuelle Standort schnell und einfach nur über Interaktion mit der Smartwatch ermitteln?

Testergebnis Aufgrund technischer Einschränkungen von Smartwatches, kann die aktuelle Position nur über Rückfrage mit dem Smartphone bestimmt werden. Die daraus resultierende Wartezeit wurde von den Testpersonen als unproblematisch eingestuft.
Die technische Limitierung von Android, das eine Smartwatch App keine anderen Apps auf dem Smartphone starten kann, erzwingt jedoch eine Interaktion mit dem Smartphone. OsmAnd muss auf dem Smartphone gestartet worden sein und zumindest im Hintergrund ausgeführt werden.
Diese Tatsache wurde als störend empfunden, kann jedoch momentan nicht umgangen werden.

I.5.2.2.3 Navigation

Fragestellung Kann die Navigation nach dem Start nur mit Hilfe der Informationen, die die Smartwatch liefert, beendet werden?

Testergebnis Die Angaben Restzeit, aktuelle Position und nächster Navigationsschritt empfanden alle Testpersonen als ausreichend um die Navigation durchzuführen. In dichter bebauten Gebieten gab es jedoch teilweise Schwierigkeiten.
Mit den Zusatzinformationen einer grafischen Karte traten jedoch auch diese Schwierigkeiten nicht mehr auf und die Testpersonen absolvierten die Strecken ohne Hilfe des Smartphones.

I.5.3 Zielerreichung

Realisierung einer Smartwatch App für "Moto 360" (Android Wear) sowie einer Smartphone App die miteinander kommunizieren

Zu Beginn des Projektes konnte die Funktionalität mittels einem Prototypen erzielt werden. Aufgrund der Evaluationsergebnisse wurde aber keine vollständige Navigationsapplikation erstellt, sondern ein Plugin für eine bereits bestehende App. Die Anforderungen und die Erstellung der Wear App blieben dadurch aber unberührt. Die Smartphone App wird nur noch als Meldungsverteiler genutzt. Dennoch kann dieses Ziel als erreicht betrachtet werden, da dieselbe Funktionalität für den Benutzer besteht.

Evaluation, ob Eigenentwicklung oder Plugin für die Smartphone Routing App OsmAnd

Die Evaluation wurde zu Projektbeginn erfolgreich durchgeführt. Das Ergebnis war die Entscheidung für die Realisierung eines Plugins. Dieses Ziel war für den Projektverlauf entscheiden und wurde daher bereits früh erreicht.

Realisierung des Smartphone Routing für viele Smartwatches und Offline-Betrieb

Durch die Realisierung der App basieren auf Android und Android Wear können diverse Smartphones und Smartwatches unterstützt werden. Mit der Entscheidung für eine Nutzung von OsmAnd als Navigationssystem mit Plugin sind Funktionen wie Offlinenavigation und Kartendownload bereits enthalten. Dieses Ziel konnte also durch eine gezielte Wahl von Technologien und Architektur-entscheidungen erreicht werden.

Härtetest der Smartwatch im Alltag

Der Alltagstest wurde als Praxistest mit mehreren unbeteiligten Benutzern durchgeführt. Die Testergebnisse sind im Kapitel II.4.4 beschrieben. Der Alltagstest hat uns nützliche Inputs zur Gestaltung des UI / UX geliefert und damit zur Erfüllung dieses Ziels beigetragen.

I.5.4 Ausblick

I.5.4.1 Verbesserungspotenzial

Momentan bietet die Anwendung dem Benutzer keinerlei Einstellungsmöglichkeiten. Eine deutliche Verbesserung könnte also erzielt werden, indem man dem Benutzer diverse Einstellungen anbietet. So könnte zum Beispiel die Dauer und Stärke der Vibrationsalarme eingestellt werden. Weiter würde es sich auch anbieten, dem Benutzer die Alarmierungsdistanz zur Einstellung anzubieten.

Neben den Einstellungsmöglichkeiten könnte die Anzeige der Karte deutlich verbessert werden. Aktuell werden nur Strassen, Eisenbahnlinien, Fusswege, Gebäude sowie Gewässer dargestellt. Dies könnte noch deutlich erweitert werden. Dabei sollte allerdings die Karte nicht mit zusätzlichen Objekten überladen werden, sondern durch präzisere Angaben wie z.B. Strassennamen oder Geländenutzungen (Wald, Wiesen) lesbarer gemacht werden. Damit könnte die Anzeige näher an die Darstellung der Navigationsapplikation gebracht werden. Zudem würde die Karte dann eine bessere Orientierung ermöglichen, da die Karte den Gegebenheiten vor Ort genauer Rechnung tragen kann.

I.5.4.2 Erweiterungsmöglichkeiten

Die Anwendung läuft momentan nur als Plugin in OsmAnd. Es wäre allerdings durchaus denkbar, dies auf weitere Navigations-Apps auszuweiten. Es könnte sowohl für Applikationen mit Plugin-Architektur, als auch für andere Apps umgesetzt werden.

Neben dem Einbezug weiterer Apps könnte auch die Funktionalität der bestehenden Lösung ausgebaut werden. So könnte zum Beispiel die Kartenanzeige um eine Scrollingfunktion, zur Betrachtung grösserer Bereiche, erweitert werden. Neben dem Verschieben des Kartenausschnitts könnte auch das Vergrössern / Verkleinern einen weiteren Nutzen bringen. So könnten dem Benutzer zum Beispiel genauere Informationen zu Positionen von Restaurants in der Nähe bekommen.

Die Speicherung von Notizen könnte eine weitere nützliche Ergänzung sein. Mittels einer einfachen Geste könnte der Benutzer so zur aktuellen Position einen Marker speichern. Weiter könnte dieser Marker mit einer Positionsbezogenen Notiz versehen werden.

Um die Nutzung der App noch komfortabler zu machen, bestünde die Möglichkeit, Spracheingaben für die App zu ermöglichen. Diese Möglichkeit ist allerdings etwas weiter entfernt als andere Möglichkeiten, denn die Qualität der Spracherkennung auf aktuellen Smartwatches ist noch nicht auf einem stabilen Niveau ist.

I.5.5 Persönliche Berichte

I.5.5.1 Erfahrungen aus der Verwendung von OsmAnd

Die Verwendung von OsmAnd als Plugin-Host bringt einige technische Einschränkungen mit sich:

» Die API von OsmAnd kann nur angesprochen werden, wenn OsmAnd bereits läuft. Dies ist auf eine Einschränkung der Startup-Activity von OsmAnd zurück zu führen.
» Die Berechnung und Aktualisierung der Route wird vom Navigationssystem durchgeführt, daher hat ein Plugin keinen Einfluss auf die Aktualisierungsrate der Positionsdaten o.Ä. Dies führte in unserem Fall dazu, dass wir die gelieferten Positionsdaten in der Frequenz reduzieren mussten.
» Die Implementierung der Navigation unterstützt keine direkten Rückschlüsse auf die bereits zurückgelegte Strecke

Neben technischen Einschränkungen gibt es weitere Punkte, welche bei der Realisierung aufgefallen sind:

» Die Plugin-API ist nicht dokumentiert, dies führt zu einem erhöhten Zeitaufwand in der Entwicklungsphase
» Da OsmAnd vollständig OpenSource ist, kann man Fehler leichter Nachvollziehen
» Das Einbinden eines Plugins benötigt Änderungen am Core um das Plugin zu laden (dies gilt für jedes Plugin, nicht nur in unserem Fall)
» Die Qualität der Implementierung des Core API lässt zu wünschen übrig. Methodennamen ergeben häufig keinen Sinn oder haben Überladungen, welche die zusätzlichen Parameter nicht nutzen.

I.5.5.2 Erfahrungen aus dem Projektverlauf

Neben den gesammelten Erfahrungen aus der Verwendung von OsmAnd, konnten wir auch einige Erfahrungen in Sachen Projektabwicklung sammeln. So war z.B. die Führung der Sitzungsprotokolle auf www.slack.com sehr hilfreich, denn so konnten alle Teammitglieder jederzeit das aktuelle Protokoll einsehen.

Im Gegensatz dazu erwies sich die Verwaltung aller aktuellen Aufgaben mittels GitHub Issues als nicht praktikabel. Das Erfassen erlaubte nur eine lose Struktur und somit war es schwierig eine strukturierte Übersicht vom Arbeitsvorrat zu erhalten. Wir nutzten daher zusätzlich noch diverse Tabellen um die Zeitschätzung und -planung zu verwalten.

Die Vorgehensweise im Projekt erwies sich als zielführend, allerdings würden wir in den nächsten Projekten bereits früher mit einer agilen Arbeitsweise beginnen. Denn gerade bei Innovationsprojekten sind Anforderungen oft starken Änderungen unterworfen, sei es aus technischen oder anderen Gründen.

I.5.6 Dank

Danken möchten wir in erster Linie unserem Betreuer, Herr Prof. Stefan Keller, für seine ausgiebige Unterstützung. Durch ein konstruktives Kommunikationsklima unterstützte er uns tatkräftig und brachte seine Ideen und sein Feedback ein. Durch das zur Verfügung stellen von Smartwatches und anderen Testgeräten gab er uns die Möglichkeit dieses spannende Thema zu bearbeiten. Auch bedanken wir uns bei den Testpersonen Robert Vogt, Fabio Strässle, Carmen Storz, Raphael Schläpfer, Daniela Strässle, Alexander Kayed, Angela Kayed und Luca Aquino für die Unterstützung bei den Alltagstests mit den Smartwatches und dem Navigationssystem. Sie lieferten wertvolles Feedback und eine Entscheidungsgrundlage zur Gestaltung des Vibrationsalarms und der Benutzeroberfläche.

II PROJEKTDOKUMENTATION

II.1 Vision

II.1.1 Navigation mit Smart Watches

Navigationssysteme und Routenplaner sind aus unseren modernen Motorfahrzeugen kaum noch wegzudenken. Sie weisen uns den Weg zu unbekannten Orten und steigern unsere Mobilität.

Anders sieht es allerdings bei der Navigation als Fussgänger aus. Die Navigation läuft meist über das Mobiltelefon. Dieses ist sperrig und damit bei der Fortbewegung zu Fuss - sei das beim Tragen einer Tasche, oder in den kalten Wintermonaten - öfters mal im Weg.

Abhilfe könnten hier die sehr im Trend liegenden Smart Watches schaffen. Sie sind ein ständiger Begleiter am Handgelenk. Damit hat man beide Hände frei und kann bei Bedarf auf die Smart Watch schauen.

II.1.2 Lösungsansatz

Es wird eine Applikation entwickelt, welche es dem Benutzer möglich macht, mit minimaler Handy-Interaktion zu Navigieren. Der Benutzer soll nur zu Beginn der Navigation die Route auf dem Handy berechnen müssen, danach wird er von seiner Smart Watch ans Ziel geleitet.

II.2 Anforderungsspezifikation

II.2.1 Anforderungen an die Arbeit

Um Unterwegs nicht auf das Display des Mobiltelefons angewiesen zu sein, könnte eine Smartwatch somit eine Hilfestellung sein, ohne abzulenken. Es soll eine Navigationsapplikation entwickelt werden, welche Mobiltelefon und Smartwatch kombiniert um eine dezente Navigationsmöglichkeit zu bieten. Neben der Navigationsfunktion soll diese Applikation auch den aktuellen Standort des Nutzers bestimmen können.

Es ist keine Anforderung, dass die Uhr eigenständig navigieren kann.

II.2.2 Use Cases

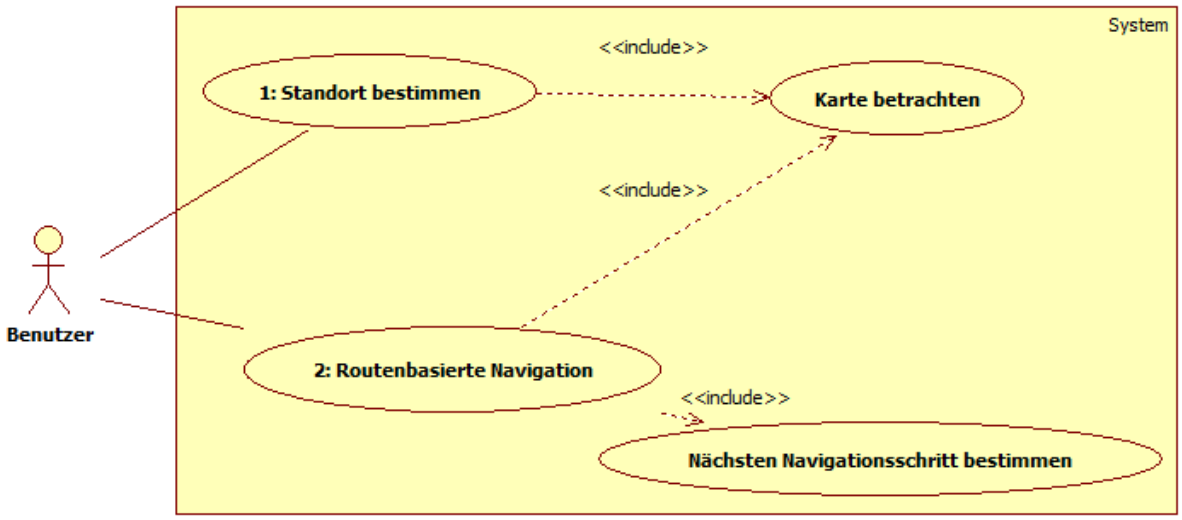


Abbildung 16: Übersicht Use Cases

II.2.2.1 Bestimmung des aktuellen Standorts

Als Benutzer möchte ich meinen aktuellen Standort bestimmen, um z.B. meinen Freunden mitzuteilen, wo ich bin.

UseCase	**1: Standortbestimmung**
Actor	Navigationsnutzer
Precondition	App auf Mobiltelefon ist gestartet und Mobiltelefon hat eine Lokalisierungsmöglichkeit (z.B. ein GNSS-Service wie GPS oder WLAN) und eine Internetverbindung (ausser Karten sind Offline verfügbar).
Postcondition	User kennt seinen Standort
Main Path	1: Benutzer startet die App auf der Uhr 2: System bestimmt Position über GPS 3: System zeigt dem Benutzer seinen Standort (evtl. mit Karte)
Alternatives	3.1: Wenn der Benutzer sich im Navigationsmodus befindet, werden zusätzlich der nächste Navigationsschritt und die Route angezeigt.

II.2.2.2 Navigation vom aktuellen Standort zu einem Ziel

Als Reisender möchte ich vom aktuellen Standort (z.B. Hotel) zu Fuss oder per Fahrrad an ein Ziel navigieren. Dabei möchte ich nicht ständig auf mein Display schauen, sondern die Szenerie betrachten. Ich möchte sehen wo ich bin, in welche Richtung ich mich bewegen muss und wie weit ich noch vom Ziel entfernt bin.

Use Case	**2: Routenbasierte Navigation**
Actor	Navigationsnutzer
Precondition	App auf Mobiltelefon und Uhr sind gestartet und Mobiltelefon hat eine Lokalisierungsmöglichkeit (z.B. ein GNSS-Service wie GPS oder WLAN) und eine Internetverbindung (ausser Karten sind Offline verfügbar).
Postcondition	User erreicht das gewünschte Ziel mit dem gewünschten Verkehrsmittel
Main Path	1: System bestimmt Position des Nutzers via GPS und verwendet diese als Startpunkt 2: Benutzer wählt Verkehrsmittel 3: Benutzer startet Suchanfrage nach Zielort 4: System ermittelt mögliche Zielorte 5: Benutzer wählt den gewünschten Zielort 6: System berechnet die Route 7: System zeigt nächsten Schritt in der Navigation 7a: User läuft/fährt zur berechneten Zwischenstation 8: Wenn Ziel noch nicht erreicht, dann weiter bei 6 10: Benutzer hat sein Ziel erreicht
Siehe auch	Use Case 1: Standortbestimmung, Alternative 3.1

II.2.2.3 Betrachtung einer Route zwischen zwei Punkten

Als Reisender möchte ich am Vorabend meines Ausflugs eine beliebige Route betrachten um zu sehen, ob die Strecke zu Fuss oder mit dem Fahrrad zu schaffen ist.

Use Case	**Routenanalyse**
Actor	Navigationsuser
Precondition	App auf Mobiltelefon ist gestartet und Mobiltelefon hat eine Lokalisierungsmöglichkeit (z.B. ein GNSS-Service wie GPS oder WLAN) und eine Internetverbindung (ausser Karten sind Offline verfügbar).
Postcondition	User sieht die errechnete Route
Main Path	1: Benutzer wählt Verkehrsmittel 2: Benutzer startet Suchanfrage nach Startort 2: System ermittelt mögliche Startorte 3: Benutzer wählt gewünschten Startort 4: Benutzer startet Suchanfrage nach Zielort 5: System ermittelt mögliche Zielorte 6: Benutzer wählt den gewünschten Zielort 7: System berechnet die Route 8: System zeigt die gesamte Route mit ihren Zwischenschritten an

II.2.3 Sequenzdiagramme

II.2.3.1 Starten der Watch App

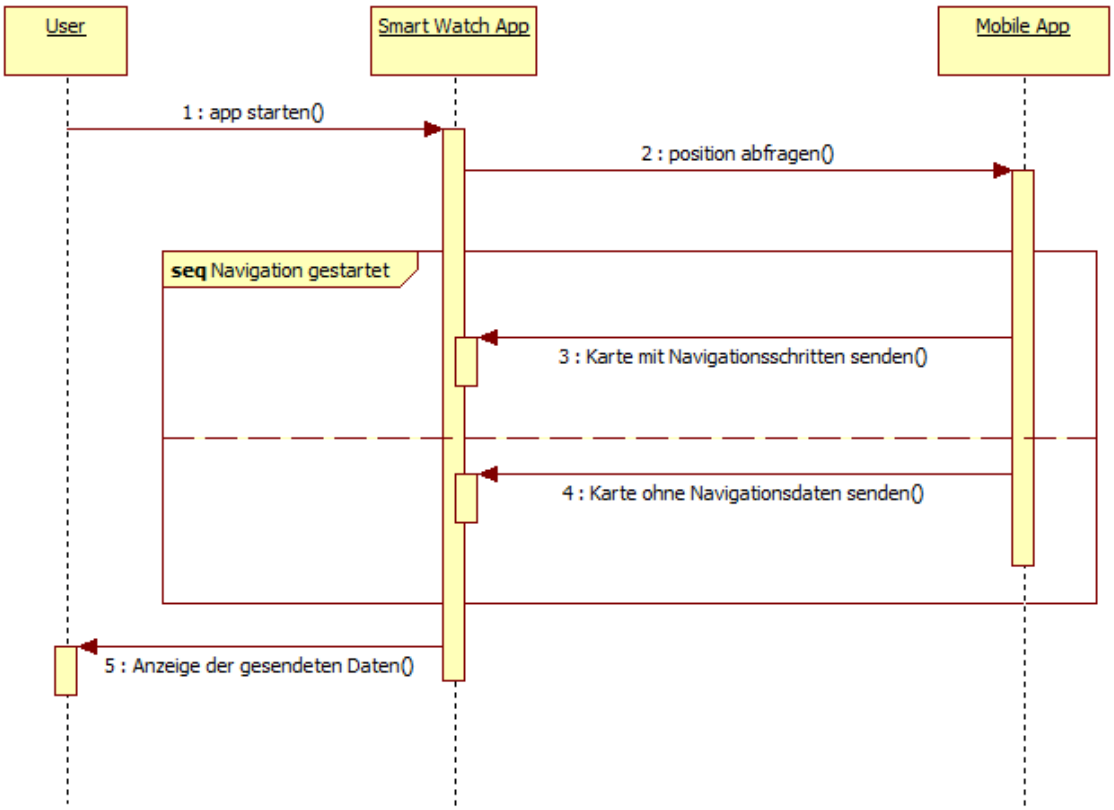


Abbildung 17: Sequenz – Starten der Watch App

II.2.3.2 Benachrichtigung bei Routenschritt

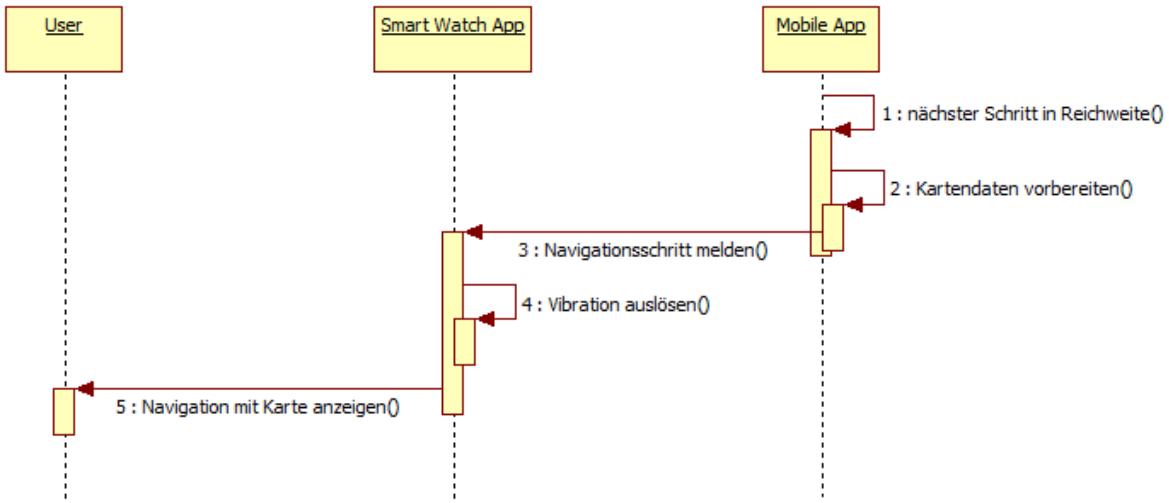


Abbildung 18: Sequenz - Benachrichtigung bei Routenschritt

II.2.4 Nicht funktionale Anforderungen

» Die Navigation soll nicht vom Verkehrsgeschehen ablenken. Der Benutzer soll eine Möglichkeit haben, zu navigieren ohne lange auf das Display zu sehen oder viele Interaktionsschritte zu machen.
» Die Smartwatch- und die Mobilapplikation sollen rund eine Stunde im Dauerbetrieb navigieren können, ohne den Akku der Geräte zu leeren.
» Die Smartwatch-Applikation soll auf der Moto 360, der Asus Zen Watch sowie der Sony Smartwatch 3 laufen.

II.3 Architektur und Design

II.3.1 Varianten

II.3.1.1 Standalone-Applikation Mobile und Standalone-Applikation Wear

Es wird eine eigenständige Applikation auf dem Mobiltelefon entwickelt. In der App werden sowohl Routenberechnung und Kartenspeicherung für die Offlinenavigation, als auch Onlineabfragen zur Navigation durchgeführt.

Die App kommuniziert mit der Smart-Watch Anwendung basierend auf Android Wear. Die Wear-App funktioniert dabei nur als erweitertes Display, wird aber ebenfalls im Rahmen dieses Projektes entwickelt.

II.3.1.2 Plugin in bestehende Navigations- und Standalone-Applikation Wear mit Public Interface

Anstelle einer Eigenentwicklung für die Mobile-App wird ein Plugin für eine bereits bestehende Navigationslösung entwickelt. Dieses sendet dann die Navigationskommandos an die eigenständige Wear-App. Die App auf der Smartwatch erhält ein öffentliches API um anderen Entwicklern die Anbindung der Watch-App zu erleichtern. Als Hostanwendung für das Plugin werden die folgenden Apps/Libraries analysiert und evaluiert: „Nutiteq libs", "Vespucci" und "OsmAnd".

II.3.2 Bewertung der Varianten

Komplette Eigenentwicklung		**Plugin in bestehende Applikation**	
Vorteile	**Nachteile**	**Vorteile**	**Nachteile**
» Volle Kontrolle über Entwicklung und Funktions-weise » Optimierungen sind einfacher möglich	» Hoher Aufwand » Höheres Risiko	» Entwicklungs-aufwand » Funktionsumfang der Hostapplikation » bessere Wart-barkeit » Wiederverwend-barkeit der API	» Abhängigkeit » Externer Code » Qualität

Aufgrund des besseren Funktionsumfangs der Hostapplikation und des geringeren Entwicklungs-aufwandes (Risikominimierung) haben wir uns für die Realisierung der Applikation als Plugin in eine bestehende Navigationsapplikation entschieden.

II.3.3 Evaluation der Plugin Host Applikation

Es wurden "Nutiteq libs", "Vespucci" und "OsmAnd" auf die Eignung für eine Realisierung geprüft. Nachfolgend sind die Analyseergebnisse aufgelistet.

II.3.3.1 Nutiteq

Kategorie	Ergebnis
Typ	Library für Kartendaten
Funktionsumfang	» Map API für Android » Unterstützung für 2.5 / 3D Objekte auf Karten » Offline Mapping » Unterstützung für individuelle Mapprojektionen » Kein Plugin-System
Dokumentation	API, Beispielapplikationen, Anleitungen
Routing-Support	Nicht vorhanden
Implikationen	» Routing muss selbst implementiert werden » Es muss eine eigene App geschrieben werden, kein Plugin im eigentlichen Sinne möglich

II.3.3.2 Vespucci

Kategorie	Ergebnis
Typ	Editor für OSM Kartendaten
Funktionsumfang	Alle Funktionen eines Offline-Karteneditors, kein direkter Plugin-Support.
Dokumentation	API, Beispielapplikationen, Anleitungen
Routing-Support	Nicht vorhanden
Implikationen	» Routing muss selbst implementiert werden

II.3.3.3 OsmAnd

Kategorie	Ergebnis
Typ	Vollständige GPS-Navigations- und Kartenapplikation für Android
Funktionsumfang	» Navigation offline/online » Turn-by-turn (foot, bike, car, …) » Anzeige von Karten und POI » Integration von Wikipedia für zusätzliche Infos zu POIs » Fully-Fledged Plugin System
Dokumentation	Wenig Dokumentation zu Plugin-Entwicklung
Routing-Support	Vollständig, siehe Funktionsumfang.
Implikationen	» Pluginentwicklung ist nicht umfangreich Dokumentiert, aber es gibt bereits Plugins, deren Source-Code frei verfügbar ist.

II.3.4 Entscheidung

Es wird ein Plugin für OsmAnd realisiert. Dieses erfüllt alle fachlichen Anforderungen und bringt Vorteile in Wartbarkeit und Stabilität. Zudem bietet OsmAnd eine breite Nutzerbasis. Die mit der nicht vorhandenen Dokumentation verbundenen Risiken können abgefedert werden, indem einerseits der Source Code frei verfügbar ist, andererseits auch Beispiel-Plugins zur Verfügung stehen.

II.3.5 Architekturübersicht

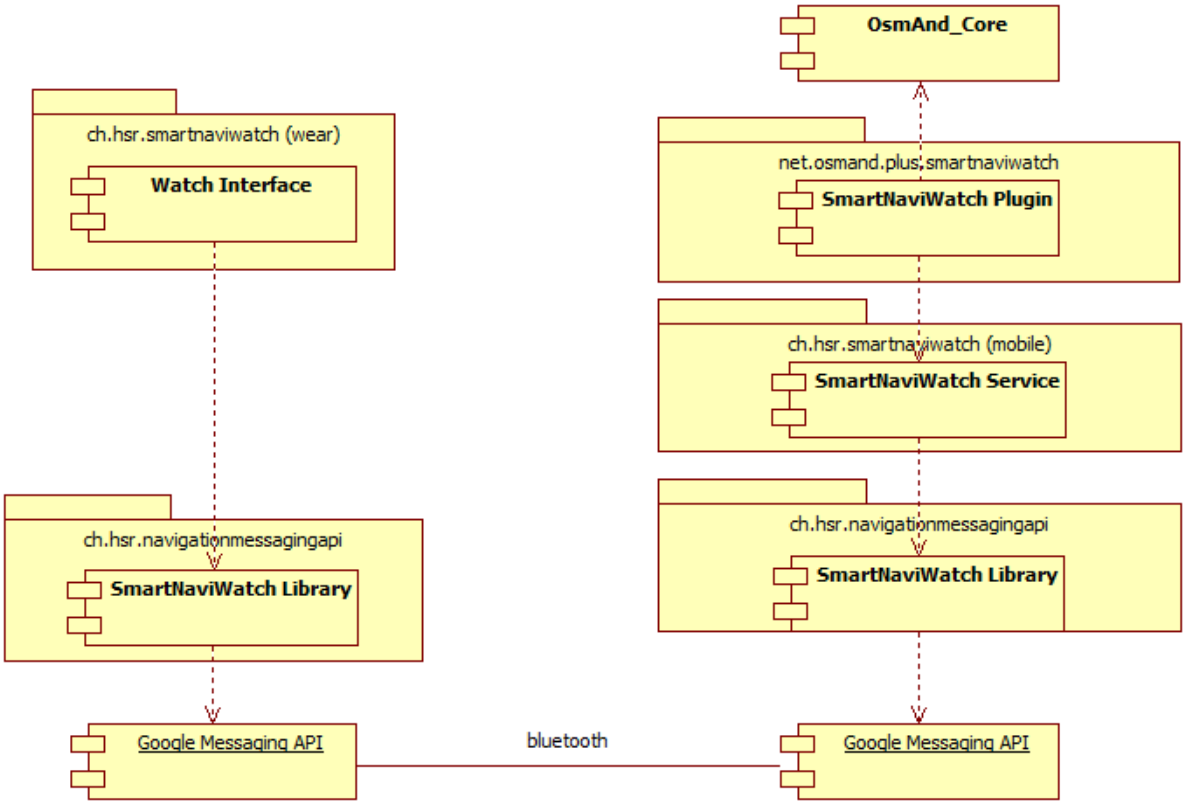


Abbildung 19: Architekturübersicht Gesamtarbeit

Die Architektur sieht drei verschiedene, separat deploybare Komponenten vor:

ch.hsr.smartnaviwatch (mobile): Basierend auf der Google MessageApi werden Funktionen zum Versenden und Empfangen von Nachrichten zur Navigationsunterstützung angeboten.

ch.hsr.smartnaviwatch (wear): Diese Komponente enthält die Benutzeroberfläche auf der Uhr. Sie empfängt die Nachrichten von der Uhr.

net.osmand.plus.smartnaviwatch: Das Plugin läuft innerhalb von OsmAnd und kann von der Uhr nach der aktuellen Position gefragt werden, oder liefert Navigationsschritte als Meldung an die Uhr.

II.3.6 Domainmodell

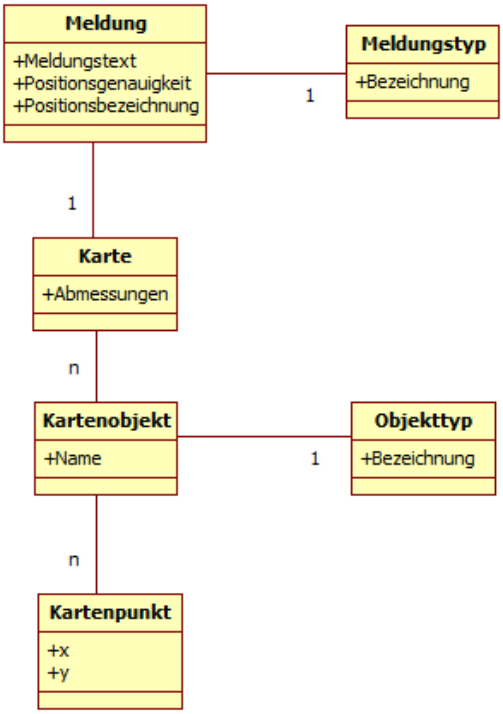


Abbildung 20: Domainmodel Navigation

II.3.6.1 Wichtige Entitäten

Entität	Funktion
Meldung	Eine Navigationsmeldung wird dem Benutzer immer dann angezeigt, wenn er die Position anfordert, oder eine Richtungsänderung ausführen soll.
Karte	Die Karte enthält neben allen darzustellenden Objekten auch die Abmessungen und die Position des Benutzers. Die zu navigierende Route wird als Kartenobjekt modelliert.
Kartenobjekt	Dies sind Objekte wie Strassen, Gebäude, Eisenbahnlinien und Weitere.

II.3.7 Messaging Architektur

II.3.7.1 Message Flow und Systemgrenzen

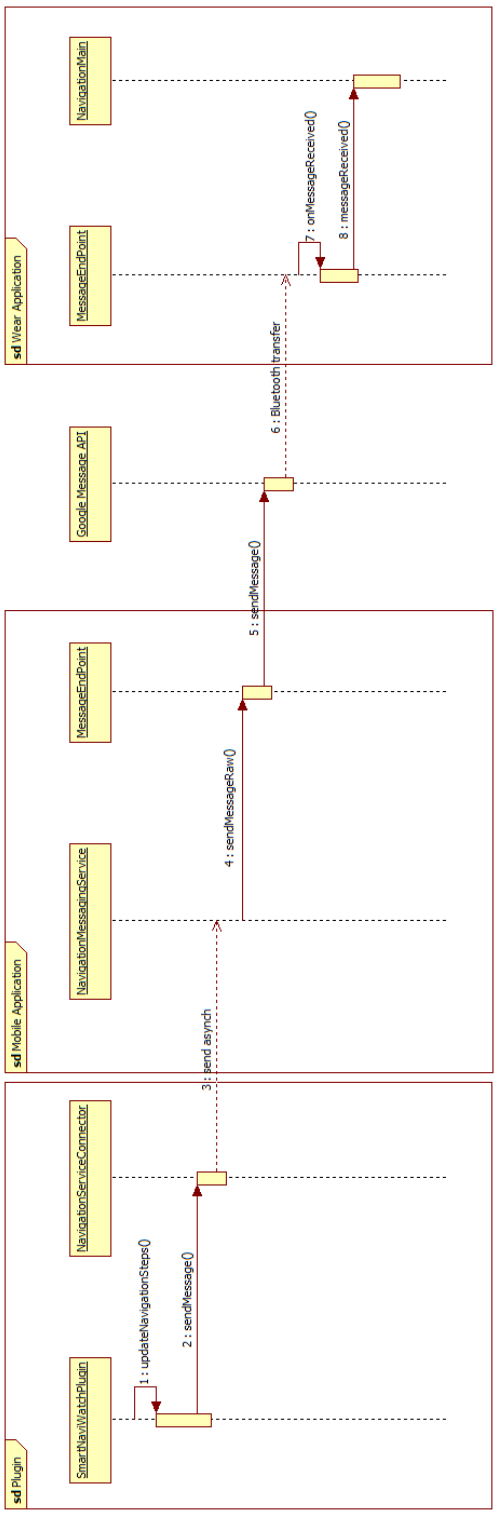


Abbildung 21: Message Flow im Gesamtsystem

II.4 Implementation und Test

II.4.1 Wichtige Klassen

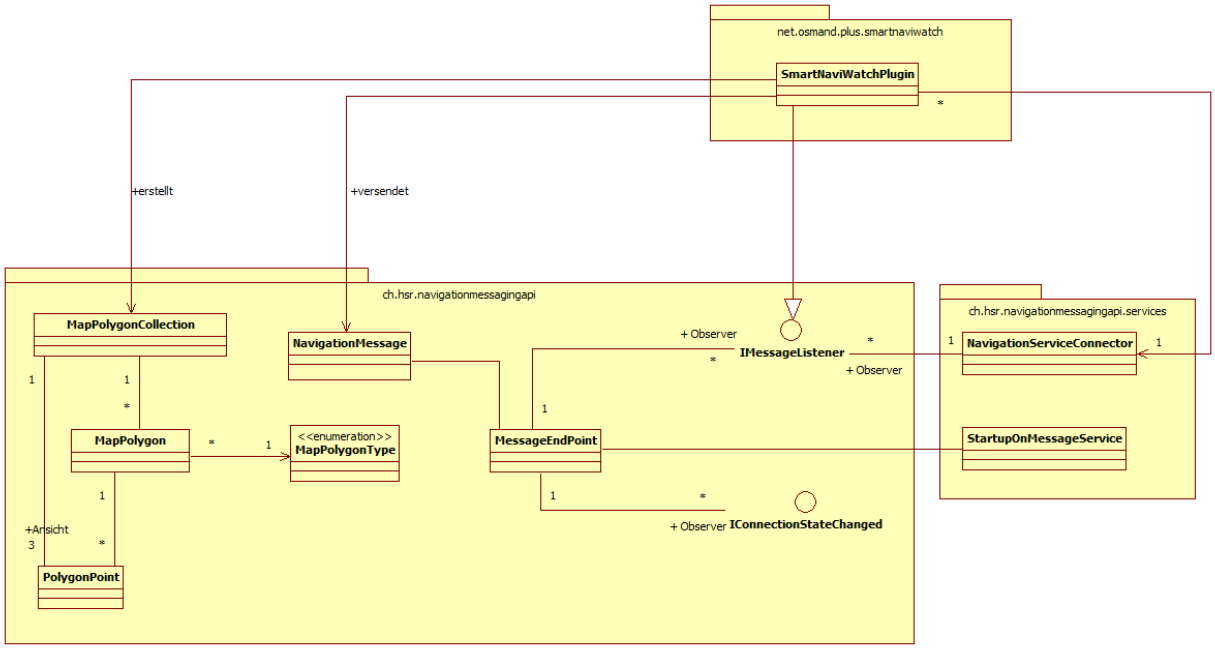


Abbildung 22: Klassendiagramm der wichtigsten Klassen

Klasse	Funktion	Referenzen
SmartNaviWatchPlugin	» Empfängt Ereignisse von OsmAnd und leitet diese als Meldung weiter. » Ermittelt die aktuelle Position und einen Kartenausschnitt und leitet diesen als Meldung weiter.	» Kartendaten (MapPolygons) » NavigationMessage » IMessageListener » NavigationServiceConnector
MessagingEndpoint	» Stellt einen Endpoint des Messagingsystems dar und gibt Auskunft über Verbindungsereignisse	» - Google Message Api
MapPolygonCollection, MapPolygon, PolygonPoint	» Kartendaten dargestellt als Ansammlung von Polygonen bestehend aus Punkten.	
NavigationMessage	» Meldung, welche von der Uhr ans Plugin gesendet wird, und umgekehrt. » Die Daten werden innerhalb der Meldung in einer HashTable gehalten und binär serialisiert.	

II.4.2 Wichtige Pattern

Messages: Für den Daten- und Informationsaustausch zwischen den einzelnen Applikationsteilen werden Meldungen asynchron ausgetauscht. Damit lassen sich die auftretenden Ereignisse gut abbilden, da sie nicht in einer strengen synchronen Abfolge sind.

Observer Pattern: Beim Empfangen von Meldungen kommt das Observer-Pattern zum Einsatz. Jedes Objekt, welches Meldungen empfangen möchte, meldet sich beim MessageEndPoint oder beim Service Connector als Observer an.

(Remote) Proxy: Da unter Android die Bluetooth-Kommunikation nur unter Applikationen mit demselben Signierungszertifikat möglich ist, muss die Kommunikation auf dem Handy über einen Proxy (Mobile App) umgeleitet werden. Der NavigationServiceConnector und der NavigationMessagingService bilden die beiden Schnittstellen zum Proxy in den unterschiedlichen Teilsystemen.

II.4.3 Automatische Testverfahren

II.4.3.1 Testabdeckung

Da dieses Projekt eine Vielzahl von out-of-project Schnittstellen und Benutzeroberflächen umfasst, können nur die Core-Klassen der Transferbibliothek mit Unit-Tests abgedeckt werden.

II.4.3.2 Tests

testEndPointMessageEvents: Testet, ob Observer korrekt benachrichtigt werden, wenn Nachrichten empfangen werden. Dabei wird die Nachrichtenserialisierung ebenfalls überprüft.

testEndPointConnectionFailureEvents: Überprüft, ob bei negativen Verbindungs-Ereignissen eine korrekte Mitteilung an Observer erfolgt. Positive Verbindungsereignisse können nicht getestet werden, da deren Code nur im Google API ausgeführt wird.

testEmptyMessage: Testet ob eine leere Meldung korrekt erzeugt werden kann.

testPolygonPoint: Testet ob die Funktionalität des PolygonPoints korrekt funktioniert.

testPolygonBounds: Überprüft, ob die Berechnung der Bounding Rectangles in MapPolygon korrekt rechnet.

testMapPolygonCollection: Testet, ob die PolygonCollection die Sortierung der Objekte korrekt beibehält und das Hinzufügen funktioniert.

II.4.4 Manuelle Testverfahren

Die Whitebox-Tests wurden von zwei Entwicklern unabhängig oder in Zusammenarbeit wiederholt (mehrfach pro Iteration) durchgeführt.

Die Blackbox-Tests wurden von 8 unbeteiligten Personen unabhängig voneinander durchgeführt.

II.4.4.1 Simulationstest Entwickler (Whitebox)

Ziel:
- » Sicherstellen das kürzlich implementiertes Feature funktioniert
- » Debugging der App

Vorgehen:
1. App auf die Entwicklungsgeräte deployen
2. Mit Hilfe der App ‚FakeGPS' auf dem Smartphone GPS Signal an verschiedenen Orten simulieren um gewünschtes Feature zu testen

II.4.4.2 Praxistest Entwickler (Whitebox)

Ziel:
- » Sicherstellen das sich alle Features wie geplant Verhalten
- » Edge Cases bestimmen, testen und App Verhalten kontrollieren

Vorgehen:
1. App auf verschiedenen Geräten installieren (2 quadratische Smartwatches, 1 runde Smartwatch und 2 unterschiedliche Smartphones)
2. Testrouten / -orte gemäss Testzielen bestimmen (Getestete Regionen St. Gallen, Goldach, Wil, Rapperswil)
3. Zu Fuss Routen ablaufen und App Verhalten überprüfen

II.4.4.3 Praxistest User (Blackbox)

Ziel:
- » Prüfen ob Features gemäss User Erwartungen funktionieren
- » Testen der UI / UX Verständlichkeit der App

Vorgehen:
1. App auf verschiedenen Geräten installieren (2 quadratische Smartwatches, 1 runde Smartwatch und 2 unterschiedliche Smartphones)
2. Geräte dem User zur Verfügung stellen und ihn instruieren
3. Beobachten, wie der User selbstständig eine Navigation durchführt
4. User Feedback einholen und mit gezielten Fragen vervollständigen

II.5 Weiterentwicklung

II.5.1 Möglichkeiten

II.5.1.1 Ausweitung auf weitere Navigations-Applikationen

Die Anwendung funktioniert momentan nur als Plugin in OsmAnd. Es wäre allerdings durchaus denkbar, dies auf weitere Navigations-Apps auszuweiten. Dies ist sowohl für Applikationen mit Plugin-Architektur, als auch für andere Apps denkbar.

II.5.1.2 Scrolling / Zoomen der Karte

Die Karte zeigt momentan einen statischen Ausschnitt. Dieser reicht zwar für die Turn-Boy-Turn Navigation völlig aus, dennoch wäre eine Scrollingfunktion denkbar und nützlich. Dabei swiped der Benutzer mit dem Finger über den Bildschirm um zu scrollen. Der Benutzer könnte damit die Route in ihrer Gesamtheit auch auf der Uhr betrachten.

Eine Zoomfunktion würde es zudem ermöglichen, je nach Zoomstufe detailliertere Ansichten anzuzeigen. Damit könnten bei nahen Ansichten auch POIs wie Bankomaten o.Ä. angezeigt werden. Als Interaktionsmöglichkeit für diese Funktion wäre Pinching denkbar.

II.5.1.3 Erfassen von Markern

Mittels einer einfachen Geste könnte der Benutzer zur aktuellen Position einen Marker speichern. Weiter könnte dieser Marker mit einer Positionsbezogenen Notiz versehen werden.

II.5.1.4 Steuerung über Sprachbefehle

Die Android-Watches unterstützen Spracherkennung. Über Befehle wie "Ok Google, take a note" können Anwendungen gestartet mit entsprechenden Aktivitäten gestartet werden, oder laufende Anwendungen Anweisungen erhalten. Damit wäre eine berührungslose Bedienung möglich.

II.5.2 Vorgehen

II.5.2.1 Ausweitung auf weitere Navigations-Applikationen

Die zu erweiternde Applikation oder das darin integrierte Plugin benötigt einen Verweis auf die Transfer-Library. Dies kann entweder als vorkompilierte JAR-Datei, oder aber durch das Einbinden des offenen Sourcecodes erfolgen.

Danach kann die Anwendung oder das Plugin die definierten Meldungs- und Datentypen nutzen, um die Watch-App fernzusteuern (z.B. Anzeigen einer Richtungsänderung mit Karte). Eine Anpassung an der Mobile App von Smart Navi Watch oder der Watch App dazu sind nicht nötig.

II.5.2.2 Scrolling / Zoomen der Karte

Für die nachfolgende Beschreibung der notwendigen Änderungen wird davon ausgegangen, dass das OsmAnd-Plugin erweitert / angepasst wird.

In einem ersten Schritt müsste das Versenden der Karte vom Senden der restlichen Daten getrennt werden. Dies erfordert allerdings nur Anpassungen am Plugin und an der Benutzeroberfläche auf der

Uhr. Die Transferlibrary verwendet generische Key-Value-Pairs für die Datenübertragung und ist daher nicht von Anpassungen betroffen.

Danach kann die Benutzeroberfläche auf der Watch um die entsprechenden Input-Möglichkeiten erweitert werden.

II.5.2.3 Erfassen von Markern

Diese Anpassung erfordert eine Anpassung an dem Watch UI, damit der User seine Marker erfassen oder annotieren kann.

Zudem müsste das Transfer API so erweitert werden, dass die Marker zwischen Uhr und Handy ausgetauscht werden können. Dazu definiert man einen neuen Message-Typ und eine neue, serialisierbare Transferklasse mit primitiven Properties.

Danach ist es dann notwendig, auf der Watch die neuen Meldungstypen zu empfangen und mit dem Renderer darzustellen. Optional könnte auch noch das Plugin um ein Map-Overlay angepasst werden, dann kann man die erfassten Marker auch auf dem Mobiltelefon bearbeiten. Eine Anpassung des Plugins zur Speicherung der Marker ist aber auf jeden Fall notwendig.

II.5.2.4 Steuerung über Sprachbefehle

Damit der Benutzer die Applikation per Sprachkommando erreichen kann, muss die Watch-App so angepasst werden, dass die gewünschten Sprachbefehle im Manifest der Anwendung vorhanden sind.

Danach können die, bereits durch Android Wear interpretierten Kommandos, als Strings via die vorhandenen Meldungsklassen an das Handy gesendet werden.

Für eine erweiterte Nutzung, z.B. das Angeben von Zielorten oder Adressen, könnte die Hostanwendung (Navigationssystem) in ihrem Manifest weitere Sprachoptionen anbieten. Über das Plugin könnten dann wieder wie gewohnt Meldungen zur Steuerung der Benutzeroberfläche auf der Uhr gesendet werden.

II.6 Projektmanagement

II.6.1 Team

Die Arbeit wurde als 2er Team bestehen aus Konstantin Kayed und Mirco Strässle durchgeführt. Betreut hat die ganze Arbeit Prof. Stefan Keller. Es waren keine externen Industriepartner beteiligt.

II.6.1.1 Verantwortlichkeiten Entwicklung

	Wear App	**Mobile App**	**Transfer Library**	**OsmAnd**	**Prototyp**
Konstantin Kayed		X	X	X	
Mirco Strässle	X	X			X

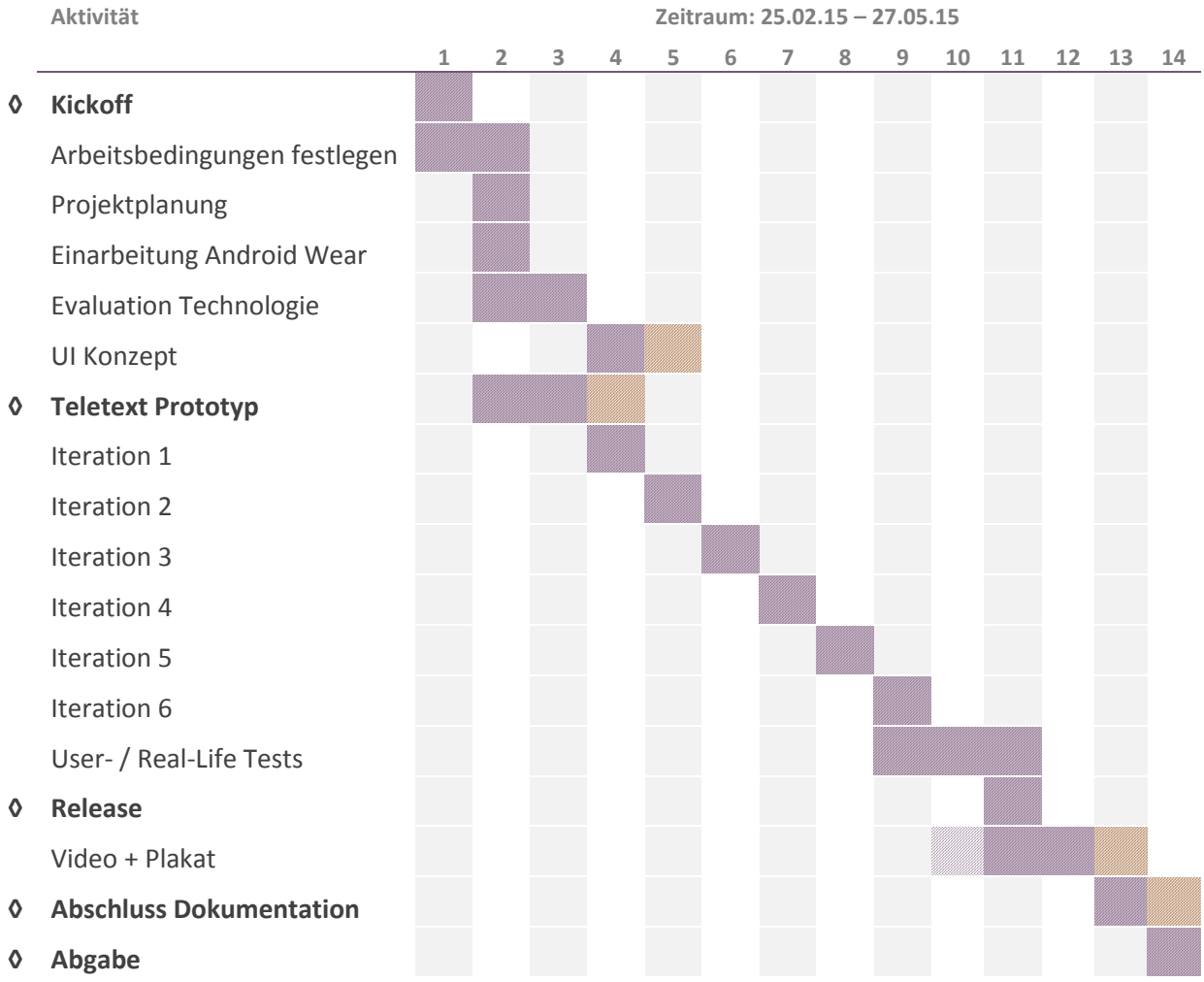
II.6.2 Aufwandschätzung

In der Projektplanung wurde mit folgenden Planwerten für die groben Kategorien gearbeitet:

Kategorie	Geschätzt in h	Benötigt in h	Abweichung in h
Projektmanagement	52,00	44,00	-8,00
Konzeption	29,00	42,80	13,80
Watch: Analyse / Entwicklung	58,00	56,00	-2,00
Mobile: Analyse / Entwicklung	34,00	28,00	-6,00
Plugin: Analyse / Entwicklung	63,00	78,00	15,00
Transfer: Analyse / Entwicklung	65,00	68,00	3,00
Erstellung Multimedia	52,00	46,00	-6,00
Erstellung Dokumentation	28,00	31,00	3,00
Testing	15,00	21,00	6,00
Total	**396,00**	**414,80**	**18,80**

Die detaillierte Aufwandschätzung zu den einzelnen Tasks ist im Anhang zu finden(Kapitel II.9.4)

II.6.3 Projektplan



Die Entwicklungsphase des Projekts wurde in einwöchige Iterationen unterteilt. Am Ende einer Iteration wurde jeweils der Inhalt der nächsten Iteration festgelegt. Durch diese Vorgehensweise konnte schnell auf geänderte oder neue Anforderungen reagiert werden und es war sichergestellt, das durch die kleinen Arbeitspakete am Ende der Iteration immer ein funktionstüchtiger Stand vorhanden zu Demozwecken war.

II.6.3.1 Meilensteine

II.6.3.1.1 Kickoff

Am Kickoff wurden die Aufgabenstellung, die Rahmenbedingungen sowie diverse organisatorische Punkte geklärt und definiert. Dieser Meilenstein bezeichnet den offiziellen Start der Arbeit.

Resultate:

» Aufgabenstellung
» Rahmenbedingungen
» Vorgehen bei Statussitzungen
» Kommunikationskanäle

II.6.3.1.2 Teletext Prototyp

Dieser Meilenstein bezeichnet den Abschluss der konzeptionellen Phase der Arbeit. Zu diesem Zeitpunkt sind alle Technologien und die Entwicklungsumgebung definiert und durch eine Prototyp Applikation geprüft. Das weitere Vorgehen der Arbeit ist geplant.

Resultate:

» Abgeschlossene Evaluation
» Projektplan
» Benötigtes Technologie Know-How
» UI Konzept
» Demofähige Teletext Prototyp Applikation

II.6.3.1.3 Release

Mit dem Release ist die gesamte Entwicklungsphase inkl. Entwickler- und Usertests abgeschlossen. Alle geplanten Features wurden über den Verlauf der Iterationen eingebaut und ggf. aufgrund von Testergebnissen verbessert.

Resultate:

» Installationsbereite Applikationen
» Testresultate
» Installationsanleitung
» API Beschreibung

II.6.3.1.4 Abschluss Dokumentation

Der Dokumentationsabschluss bezeichnet das Ende des inhaltlichen Teils der Arbeit.

Resultate:

» Dokumentation
» Poster
» Video
» Kurzzusammenfassung
» Abgabe-CD

II.6.3.1.5 Abgabe

Abgabe der Arbeit an die verantwortlichen Stellen und somit endgültiger Abschluss der Arbeit.

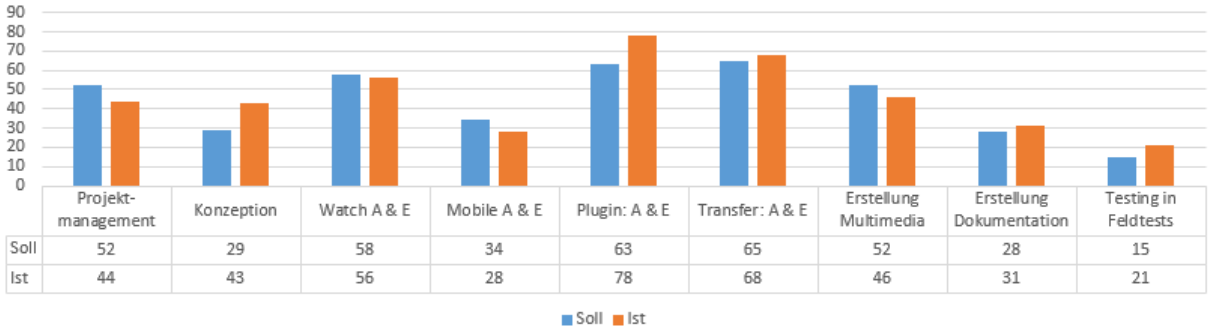
II.6.4 Risiken

Mit der unten dargestellten Risikotabelle wurden die projektspezifischen Risiken ständig neu beurteilt und beobachtet. Dies mit dem Ziel auf Problemfälle vorbereitet zu sein und schnell reagieren zu können. Allgemeine Projektrisiken wie Krankheit eines Beteiligten, o.ä. wurde nicht beachtet, da für dieses Projekt keine Marktbedingungen herrschten und solche Probleme administrativ mit der HSR geklärt werden müssten.

Risiko	Wahrscheinlichkeit	Auswirkung	Bewertung
Plugin Schnittstelle von OsmAnd ist fehlerhaft	2: Andere Plugins wurden bereits erfolgreich integriert	10: Geplante Features könnten verunmöglicht werden	**12**
Kartenanzeige kann nicht an Smartwatch übertragen werden	6: Diverse Stellen können Probleme verursachen, z.B. Performance der Smartwatch	7: Nutzen der App wird stark verringert	**13**
Weiterentwicklung von OsmAnd wird eingestellt	1: OsmAnd basiert auf einer Community und verbreitet	1: Arbeit kann mit bestehender Version abgeschlossen werden	**2**
Entwicklungsaufwand wird durch unterschiedliche Smartwatches erhöt	2: Gemeinsames Android Wear Betriebsystem bildet gemeinsame Abstraktion für Applikationen	5: Arbeit ist die erhaltenen 3 Smartwatches limitiert	**7**
Google Schnittstelle zwischen Smartphone und Smartwatch zu unzuverlässig	3: Google nutzt dieselbe Schnittstelle für ihre Dienste.	6: Weniger Vertrauen in die Applikation durch Benutzer. Mehr Entwicklungsaufwand für Verbesserungen.	**9**

II.7 Projektmonitoring

II.7.1 Soll-Ist-Zeitvergleich



	Projekt-management	Konzeption	Watch A & E	Mobile A & E	Plugin: A & E	Transfer: A & E	Erstellung Multimedia	Erstellung Dokumentation	Testing in Feldtests
Soll	52	29	58	34	63	65	52	28	15
Ist	44	43	56	28	78	68	46	31	21

Abbildung 23: Soll-Ist-Zeitvergleich über das gesamte Projekt

II.7.1.1 Bewertung und Begründung

» Die eingesparte Zeit im Bereich Projektplanung lässt sich auf die gute Dokumentation während der Projektphase zurückführen, es wurde deutlich weniger Zeit für die Auswertungen benötigt, als geplant.

» Durch zusätzliche Aufwendungen in der Konzeptionsphase konnten die Projektrisiken früh abgebaut werden.

» Die Entwicklung der Android Wear App konnte in der geschätzten Zeit fertiggestellt werden.

» Für die Entwicklung der Mobile-App wurde weniger Zeit benötigt, da diese kaum Funktionen besitzt (Verlagerung auf Plugin).

» Die Pluginentwicklung dauerte aufgrund einer Funktionsverlagerung von der Mobile-App ins Plugin und der Hinzunahme des optionalen Features "Anzeigen der Karte" länger als geplant.

» Die gesamte Zeitschätzung weist eine Abweichung von rund 4.8% auf. Dies ist für ein solches Projekt durchaus im Rahmen.

II.7.2 Codestatistiken

II.7.2.1 Mobile

Metrik	Wert
Lines Of Code	104
Lines Of Comments	1517
Number of Packages	1
Number of Classes	3
Number of Methods	11
Durchschnittliche Cyclomatic Complexity	1.55

II.7.2.2 Transfer

Metrik	Wert
Lines Of Code	586
Lines Of Comments	167
Number of Packages	2
Number of Classes	16
Number of Methods	78
Durchschnittliche Cyclomatic Complexity	1.45

II.7.2.3 Plugin

Metrik	Wert
Lines Of Code	422
Lines Of Comments	128
Number of Packages	1
Number of Classes	3
Number of Methods	23
Durchschnittliche Cyclomatic Complexity	2.65

II.7.2.4 Wear

Metrik	Wert
Lines Of Code	389
Lines Of Comments	299
Number of Packages	1
Number of Classes	9
Number of Methods	18
Durchschnittliche Cyclomatic Complexity	5.08

II.7.2.5 Bewertung auffälliger Statistiken

- Die hohe Anzahl Kommentare im Projekt 'mobile' ist auf eine Fehlinterpretation des Plugins zurückzuführen. Es wurde offensichtlich die deklarative Sprache zur UI-Definition ebenfalls als Kommentar aufgefasst.
- Dieselbe Feststellung lässt sich auch im Projekt 'wear' machen. Die UI-Deklaration wurde als Kommentar interpretiert.
- Die überdurchschnittliche Komplexität des Projekts 'wear' kann man auf das Programmiermodell der Android UI-Sprache zurückführen. Sie ist grösstenteil asynchron und erfordert daher die häufige Definition von anonymen Interfaces. Dies beinflusst die Statistik negativ.

II.7.3 CI-Statistiken

Nachfolgend sind alle Builds der Continous Integration aufgelistet. Damit ergibt sich ein gutes Bild des Entwicklungsverlaufs. Die CI wurde in diesem Projekt über Travis CI umgesetzt.

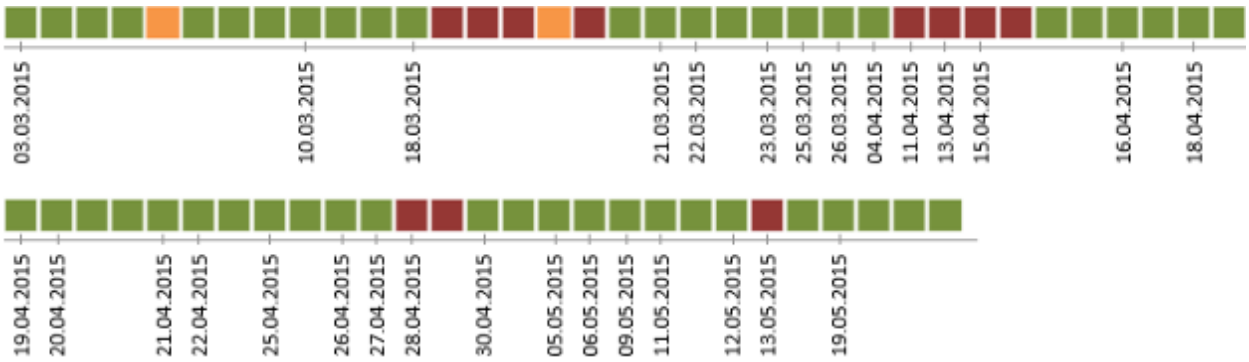


Abbildung 24: CI-Statistik über das gesamte Projekt

Die häufigen Buildfehler am 18. März haben mit der Umstellung der Entwicklungsumgebung vom Prototyp auf die Umgebung für den restlichen Projektverlauf zu tun. Dabei wurden verschiedene leere Repositories und Projekte angelegt. Dies führte zu Buildfehlern, da diese keine Klassen oder sonstigen Source Code enthielten.

Die Fehler vom 11. - 15. April haben mit einem Ausfall des Travis-Stores für die Android Pakete zu tun. Darauf hatten wir keinen Einfluss, das Problem wurde aber seitens Travis schnell behoben. Dasselbe geschah am 28. April nochmals, diesmal war es offensichtlich ein ungeplanter Patch der die Travis-Umgebung offline gehen liess.

II.8 Softwaredokumentation

II.8.1 Installation

Um die Installation zu ermöglichen muss sichergestellt sein, dass das Zielgerät die Installation von Apps erlaubt, die nicht aus dem App Store kommen. Dazu muss im Android Einstellungsmenü folgende Einstellung vorgenommen werden:

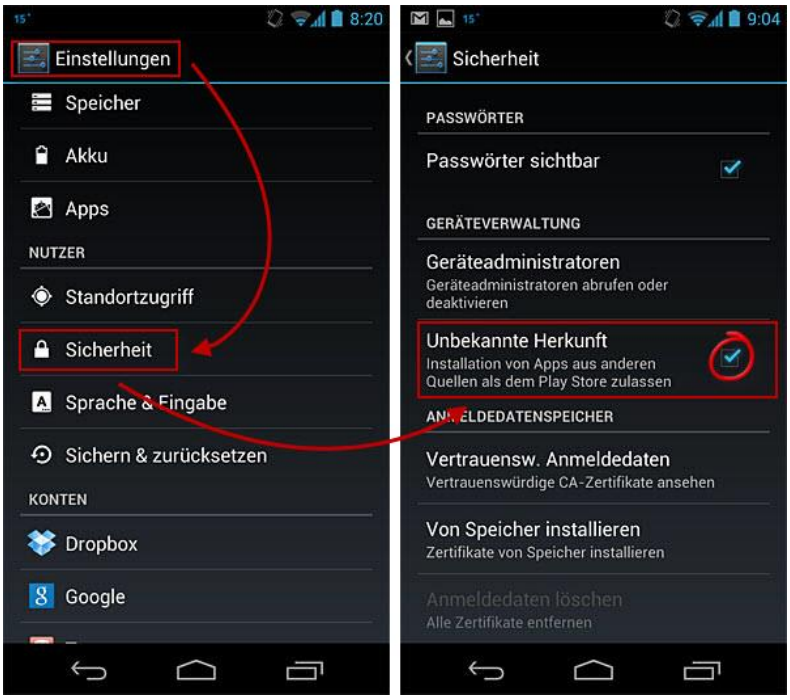
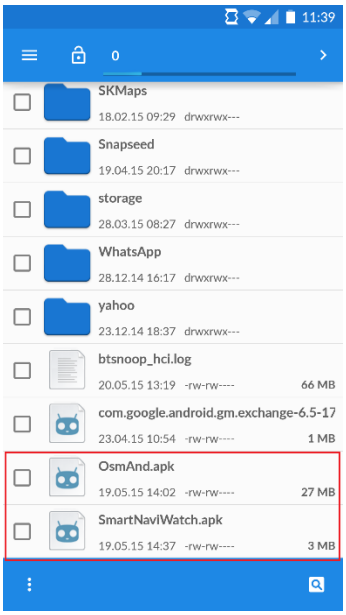


Abbildung 25: Android Sicherheitseinstellung für die Installation

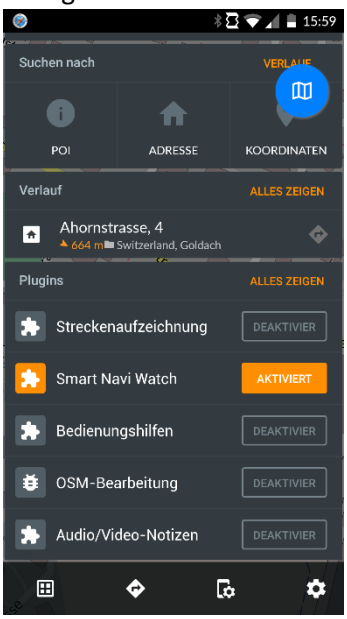
Folgende Schritte sind für die Installation und Inbetriebnahme der App notwendig:

1. „OsmAnd.apk“ und „SmartNaviWatch.apk“ von der Abgabe CD oder von http://kkade.github.io/SmartNaviWatch/ auf das Smartphone kopieren.

2. Auf dem Smartphone den Android Dateimanager starten. Die beiden APKs sind dort ersichtlich und können durch Antippen installiert werden.



3. Den Anweisungen auf dem Bildschirm folgen, bis die Installation der beiden APKs abgeschlossen ist.
4. Die gerade installierte OsmAnd App starten und das SmartNaviWatch-Plugin aktivieren:



5. Um Sicherzustellen, dass alle Services im Hintergrund sauber laufen, sollte das Smartphone einmal neu gestartet werden.
6. Sind beide APKs installiert, kann eine Smartwatch via Bluetooth mit dem Smartphone verbunden werden. Die SmartNaviWatch App installiert sich selbständig auf der Smartwatch und ist nach ein paar Minuten im App Menü der Smartwatch ersichtlich.
7. War die Installation erfolgreich, wird nun beim Start einer Navigation in OsmAnd automatisch die SmartNaviWatch App auf der Smartwatch gestartet.

II.8.2 API Tutorial

II.8.2.1 Overview

There are two main usage scenarios for the SmartNaviWatch library. This section tries to outline those scenarios and give a brief overview of the architecture you will be working with.

Using the provided Wear App: This is most useful when you try to quickly provide your own application with the features that SmartNaviWatch has. The communication stack in this instance looks like the following:

```
SmartNaviWatch Wear Application <-> Bluetooth <-> SmartNaviWatch Mobile App <->  Your Application
```

The only part you need to worry about is your own application. You install the Mobile App (comes with the Wear App) and implement the connection to it. Then you send messages with navigation instructions and the library will take care of displaying them to the user.

Using only the messaging layer: Use this approach if you want to create your own shiny user interface for both mobile and watch applications. The communication stack will look like this:

```
Your Wear Application <-> EndPoint <-> Bluetooth <-> EndPoint <-> Your Mobile Application
```

In this scenario you take full control over both ends of the user experience. The API is only used to send and receive messages.

Warning: Please note that events occur asynchronously, in another thread. Changing UI elements as a result of an event requires you to make sure that () is called in order to run your code in the proper thread.

II.8.2.2 Setting up the development environment

» Download the source code for SmartNaviWatch here
» Include the source code or a reference to a compiled version into your project
» If you want to use the premade Watch App, install the SmartNaviWatch APK on your phone

II.8.2.3 Basic Messaging

Before you can start to send or receive messages, you need to understand the basic concept of all important messaging components.

NavigationMessage This is the main class for containing messages. Messages consist of a message type and a payload object. The payload object is required to be marked with the Serializable interface.

When working with the default APK (sending messages via NavigationServiceConnector) you should use a HashTable<String, Object> as payload. The keys correspond to the MessageDataKeys enumeration.

Custom keys will be ignored by the default APK, but can come in handy in other scenarios where you build both ends of the connection. See "Sending Data" for some examples of creating messages with useful content.

Warning: This protocol does not ensure reliable messaging. If you need to make sure messages are sent and received you need to take care of this yourself by adding another layer on top.

MessageEndPoint EndPoints are used to receive or send messages. If you are working with the SmartNaviWatch APK you don't need to create your own EndPoint since you can send messages via the service connector.

The creation of a custom watch and mobile application based on our library requires you to use EndPoints on both sides of the connection. Also note that both your mobile application as well as your wear application need to be from the same APK (and therefore have the same application id and certificate) to get packages passed through.

NavigationServiceConnector Working with the provided APK, you send and receive messages via this connector from within your mobile application. This is to ensure that the Bluetooth connection is accepted by the underlying messaging layer based on the application id.

II.8.2.4 Sending messages from a mobile application

» Create and store an instance of NavigationServiceConnector:

```
private NavigationServiceConnector messageService;
...
private NavigationServiceConnector getServiceConnector() {
    if (messageService == null)
    {
        messageService = new NavigationServiceConnector(appContext);
    }
    return messageService;
}
```

» Create a message:

```
NavigationMessage msg = NavigationMessage.create(messageType, payload);
```

» Send your message:

```
getServiceConnector().sendMessage(msg);
```

II.8.2.5 Receivingmessages in a mobile application

» Create and store an instance of NavigationServiceConnector (see "Sending messages from a mobile application"

» Subscribe to the provided events by implementing IMessageListener

```
public class Example implements IMessageListener {
...
    private void initListener() {
        getServiceConnector().addMessageListener(this);
    }
 ...
    @Override
    public void messageReceived(NavigationMessage message) {
        // Your message handling here
    }
}
```

II.8.2.6 Sending / receiving messages without the default APK

» Create and store an instance of EndPoint (you can use the same EndPoint for sending and receiving)

```
private MessageEndPoint endPoint;
...
endPoint = new MessageEndPoint(getApplicationContext());
...
```

» Subscribe to the provided events

```
public class Example implements IMessageListener {
 ...
    private void initListener() {
        endPoint.addMessageListener(this);
    }
 ...
    @Override
    public void messageReceived(final NavigationMessage message) {
        // Your message handling here
    }
}
```

» Create a message

```
NavigationMessage msg = NavigationMessage.create(messageType, payload);
```

» Send your message

```
endPoint.sendMessage(msg);
```

II.8.2.7 Sending Data

When sending data between your own applications, you are free to use any payload objects, as long as they implement the Serializable interface and can be serialized by the default Java serializer. The same goes for the message type, choose whatever fits your needs. Just make sure the type passed is a path like this "sample/path/for/reference".

However, sending data to the provided Wear App requires you to use a more strict data format.

The type of the message should be one of the constants defined in MessageTypes. The content of the messages is made of a HashTable<String, Object>. The following table shows which keys should be set and what their expected values are:

Key	Expected Value

MessageDataKeys.TurnType	String:
	`"C"    -> straight on`
	`"TL"   -> turn left`
	`"TSLL" -> turn left slightly`
	`"TSHL" -> turn left hard`
	`"TR"   -> turn right`
	`"TSLR" -> turn right lightly`
	`"TSHR" -> turn right hard`
	`"KL"   -> turn left slightly`
	`"KR"   -> turn right slightly`
	`"TU"   -> U-turn`
	`"TRU"  -> U-turn`
	`"OFF"  -> off route warning`
	`"RNDB" -> Roundabout`

MessageDataKeys.RoutingDescription	String: Description of what the user should do next.
MessageDataKeys.RouteLeftTime	Integer: Estimated time left for the user to reach the destination.
MessageDataKeys.MapPolygonData	MapPolygonCollection: Polygons to be rendered on the map background.
MessageDataKeys.LocationName	String: Name of the location the user is currently at.
MessageDataKeys.LocationAccuracy	Float: Accuracy of the location in meters.

II.9 Anhang

II.9.1 Abbildungsverzeichnis

II.9.2 Literaturverzeichnis

descary.com. (20. 05 2015). Von http://descary.com/wp-content/uploads/2009/2014/08/google-maps-android-wear.jpg abgerufen

Github OsmAnd. (05. 03 2015). Von https://github.com/osmandapp/Osmand abgerufen

mashable.com. (20. 05 2015). Von http://rack.1.mshcdn.com/media/ZgkyMDE0LzA5LzA5LzdlL21hcHMuMDhjMTYuanBnCnAJdGh1bWIJMTIwMHg5NjAwPg/ed26c05e/24c/maps.jpg abgerufen

Nutiteq. (10. 03 2015). Von https://developer.nutiteq.com/ abgerufen

OpenStreetMap Wiki. (01. 03 2015). Von https://wiki.openstreetmap.org/wiki/Main_Page abgerufen

PC Mag. (20. 05 2015). Von http://www4.pcmag.com/media/images/416331-pebbgps.png?thumb=y abgerufen

II.9.3 Abkürzungsverzeichnis

Abkürzung	Bedeutung
GNSS	Das Global Navigation Satellite System bezeichnet die verschiedenen Ortungssysteme wie GPS, GLONASS, Galileo oder Beidou.
APK	Ist die Abkürzung für „Android Application Package“. Dabei handelt es sich um Dateien zu Verteilung und Installation von Software aufAndroid.

II.9.4 Detaillierte Aufwandschätzung

Task	Geschätzt	Benötigt	Abweichung
Projektmanagement	**52,00**	**44,00**	**-8,00**
Erstellung Projektplanung Grob	10,00	6	-4,00
Erstellung Projektplanung Detail / Laufend	14,00	15	1,00
Sitzungen / Besprechungen	20,00	18	-2,00
Nachbearbeitung	8,00	5	-3,00
Konzeption	**29,00**	**42,80**	**13,80**
Konzeption UX	10,00	15	5,00
Konzeption Film	5,00	8	3,00
Konzeption Architektur	6,00	9,6	3,60
Evaluation Varianten	5,00	6	1,00
Icon Erstellen	3,00	4,2	1,20
Watch: Analyse / Entwicklung	**58,00**	**56,00**	**-2,00**
Analyse Renderingmöglichkeiten für Karte	8	11	3,00
Implementierung Rendering Karte	12	14	2,00
Analyse Darstellungsmöglichkeit Fortschrittsbalken	6	4	-2,00
Implementierung Fortschrittsbalken	8	11	3,00
Entfernen Fortschrittsbalken	5	2	-3,00
Analyse Meldungsempfang auf Watch	6	4	-2,00
Implementierung Meldungsempfang auf Watch	7	7	0,00
Layout zwischen Square und Round abgleichen	6	3	-3,00
Mobile: Analyse / Entwicklung	**34**	**28**	**-6**
Analyse Service für Proxy	10	11	1,00
Implementierung Service für Proxy	8	5	-3,00
Meldungsverarbeitung auf Mobile	6	4	-2,00
Implementierung Main Activity	4	2	-2,00
Definitionen im Manifest korrigieren	0	2	2,00
Icons integrieren (W, M, P)	6	4	-2,00
Plugin: Analyse / Entwicklung	**63**	**78**	**15**
Analyse API-Funktionen	4	5	1,00
Analyse Auslesen von Kartendaten	10	20	10,00
Implementierung Routing Events	12	11	-1,00
Implementierung Routing Daten	6	2	-4,00

Implementierung Routing Alarm	4	8	4,00
Implementierung Auslesen von Kartendaten	20	25	5,00
Implementierung Fortschrittsbalken	5	6	1,00
Fortschrittsbalken wieder ausbauen	2	1	-1,00
Transfer: Analyse / Entwicklung	**65**	**68**	**3**
Analyse Google Message API	8	12	4,00
Analyse Service für Proxy (Adapter)	9	6	-3,00
Prototyping Meldungsdesign	10	12	2,00
Implementierung EndPoint / Service Proxy	20	24	4,00
Implementierung Meldungen	12	8	-4,00
Implementierung Kartendaten	6	6	0,00
Erstellung Multimedia	**52**	**46**	**-6**
Erstellung Poster	8	10	2,00
Erstellung Film	28	24	-4,00
Erstellungen Grafiken	16	12	-4,00
Erstellung Dokumentation	**28**	**31**	**3**
Dokumentation fortlaufend	20	22	2,00
Dokumentation Abschluss	8	9	1,00
Testing in Feldtests	**15**	**21**	**6**
Testing	15	21	6,00
Total	**396,00**	**414,80**	**18,80**

II.9.5 Sitzungsprotokolle aus Slack.com

Sitzungszusammenfassung 25.02.2015

Erreichtes aus Vorwoche

- Entwicklungsumgebung mit Debugtools ist eingerichtet
  - Sowohl via Emulator, als auch mit Devices
- Kleines Testprogramm zur Demonstration der Kommunikation zwischen Uhr und Handy
- Einarbeitung in Android und Android Wear
- Einarbeitung in Online-Routing APIs (OSRM/YOURS)
- Informationen über weitere Smartwatch zu Testzwecken gesammelt
  - https://www.digitec.ch/de/s1/product/asus-zenwatch-braun-smartwatch-3483106
  - https://www.digitec.ch/de/s1/product/sony-smartwatch-3-swr50-schwarz-smartwatch-2754992

Offene Punkte

Traktanden

- Definition des Arbeitsumfangs
- Formalen Ablauf der Arbeit definieren
- Entscheid für zweite Smartwatch
- Aktueller Stand der Entwicklungsumgebung

Beschlüsse

- Das Projektteam bekommt ein Galaxy Nexus. Herr Keller klärt ab, ob ein weiteres Phone zur Verfügung steht.
- Die Anforderungen werden ergänzt durch "Positionsbestimmung - wo bin ich?"
- Als optionales Zusatzziel könnte eine Teletextanzeige als Demo realisiert werden
- Herr Keller sendet uns eine Checkliste mit Tipps zur Dokumentation
- Pro Sitzung ist folgendes Protokoll auf Slack gepostet:
  - Traktanden
  - Erreichtes aus Vorwoche
  - Offene Punkte, Fragen und Problemstellungen
  - Beschlüsse (Nach der Sitzung festgehalten)
- In der nächsten Phase wird die Anforderungsanalyse durchgeführt
- Das Projektteam ergänzt das Wiki mit den Links auf das Repository (https://github.com/kkade/SmartNaviWatch) und lädt Herrn Keller ein
- Das Projektteam prüft die Erstellung einer Landingpage mit github.io
- Es werden continuous integration Tools eingesetzt (Travis CI)

March 2nd, 2015

Sitzungsvorschau 03.03.2015
Erreichtes aus Vorwoche:
- Galaxy Nexus wurde überprüft, es ist momentan noch eine zu alte Androidversion installiert
- Dokumentation wurde als Share für das Projektteam eingerichtet
- Anforderungen als UseCases beschrieben, bereit für Besprechung
- Hr. Keller ist bei allen Repositories / Kommunikationsmitteln eingeladen
- Landingpage ist testhalber vorhanden
- Die Bestellung für die weiteren Watches ist erfolgt (Asus Zen + Sony SWR50)
- OsmAnd Plugin-Möglichkeiten analysiert
-Automatische Builds mit travis ci sind eingerichtet

Offene Punkte:
- Soll ein OsmAnd-Plugin weiter verfolgt werden?
Traktanden:
- Besprechung der ausgearbeiteten Anforderungen
- Aktueller Stand der Entwicklungsumgebung
- Nächste Phase: User Interface Concepts / Technical Concepts
Beschlüsse:
- SmartWatches Sony/Asus erhalten
- Team passt Link auf Travis im Wiki an
- Eigenständige App auf Uhr als nicht-Anforderung im Dokument aufnehmen
- Nach Release der Apple-Watch noch ein Vergleichskapitel in Dokumentation
- Entscheid: OsmAnd-Plugin wird weiterverfolgt
- Evalutaionsonsvorgehen zu OsmAnd-Entscheid in Doku beschreiben
- Einarbeitungskapitel in Doku mit Teletext
- Öffentliche API für Uhr-App erstellen und dokumentieren
- Google Support Group einschreiben

March 10th, 2015

Sitzungsprotokoll 11.03.2015 11:00
Erreichtes:
- Handy ist auf aktueller Android Version
- Abklärungen zu OsmAnd-Plugin Funktionalität
- Prototyp mit Teletext
- Dokumentation erweitert
Offene Punkte:
- Format der Zeiterfassung
Traktanden:
- Besprechung der erreichten Punkte
- Kurze Architekturübersicht & Dokumentationsvorgehen

- Nächste Schritte: UI Konzepte, Entwicklungsumgebung / Umsetzung OsmAnd-Plugin

Beschlüsse:

- In 'Stand der Technik' werden Smartwatches und News bis zum 11.03.2015 beachtet (MWC und Apple Watch Vorstellung enthalten)
- Ebenfalls bei der Analyse des Technikstandes wird die "NeoMappingApp Reloaded" betrachtet.
- Es soll stärker darauf eingegangen werden, welche Komponenten entwickelt werden. (Eigenentwicklung kennzeichnen)
- Die Beschreibung des Prototyps soll die Installation auf der SmartWatch ebenfalls erklären.
- Zeiterfassung erfolgt wöchentlich / Auflistung mit Tasks pro Woche
- UnitTesting wird für alle Komponenten ausser der View und der Bluetoot-Schnittstelle eingesetzt.
- Trackpoints (also das markieren der aktuellen Position wird als additional Feature aufgenommen

---

March 17th, 2015

Erreichtes:

- Beschreibung Installation des Prototypen in Dokumentation
- Einarbeitung OsmAnd-Pluginsystem
- Erste Integration der App in OsmAnd

Offene Punkte:

- Projektplanung besprechen

Traktanden:

- Vorstellung der erreichten Punkte
- Besprechen der offenen Punkte
- Nächste Schritte: Abschluss Konzeptphase (UI Konzept), Start erste Entwicklungsphase

Beschlüsse:

- Nächste Sitzung 01.04.15 11:00 Uhr
- Projektplanung mit iterativem Entwicklungsvorgehen i.O.
- In Dokumentation auf Probleme mit Fussgängernavigation eingehen

---

March 31st, 2015

Erreichtes:

- UI Konzept
- Architekturschichten vollständig abgebildet
- Kommunikation Smartphone / Smartwatch in beide Richtungen implementiert
- Aufwecken der Smartwatch aus Ruhezustand & Vibration implementiert

Offene Punkte:
- Planung Entwicklungsphase 2

Traktanden:
- Vorstellung der erreichten Punkte mit Demo
- Besprechung der offenen Punkte

Beschlüsse:
- Weiterarbeit in nächster Entwicklungsphase gemäss Backlog auf GitHub
- Klare Systemgrenzen in Architekturübersicht einzeichnen
- Forschrittsbalken im UI Konzept anpassen (unten noch oben füllen) für bessere Verständlichkeit

---

April 12th, 2015

Sitzungsprotokoll 13.04.2015 11:00

Erreichtes:
- UI-Konzept aktualisiert
- Informationen zu Apple-Watch gesammelt
- Beginn mit UI für Watch-App
- Meldungen werden vom Plugin versendet, wenn der User nahe am nächsten Wegpunkt ist
- Meldungen enthalten nun alle wichtigen Daten zum nächsten Navigationsschritt
- Analyse zu den Möglichkeiten zum Auslesen/Anzeigen der Map

Offene Punkte:
-

Traktanden:
- Besprechung der erreichten Analyseergebnisse
- Besprechung der erreichten Entwicklungsergebnisse
- Nächste Schritte: User Interface, "Wo bin ich?"-Funktion, Fine-Tuning der Plugin-Logik, Testing in der Praxis

Protokoll:
- Besprechung über das Testing von GPS-Features (aus Zeitgründen macht es keinen Sinn, hier eine eigene App zur Simulation von GPS-Koordinaten zu entwickeln)
- Pakete werden im Archtekturdiagram analog dem noch zu erstellenden Paketdiagramm eingezeichnet
- Erfahrungen und Limitierungen zu OsmAnd und dem Pluginsystem
- Nächster Termin (29.04.2015 11:00)

April 28th, 2015

Sitzungsprotokoll 29.04.2015 11:00

Erreichtes:
- Entwicklungsziele (Features) erreicht, mit Ausnahme der Fortschrittsanzeige

- Detailiertes Klassendiagram für Transfer Library
- Beschreibung von entdeckten Einschränkungen durch OsmAnd
- UI-Konzept an Spezialist gesendet (kein Feedback erhalten stand 28.04)

Offene Punkte:

- "Wo bin ich?"-Funktion nur eingeschränkt umsetzbar

Traktanden:

- Demo der App
- Besprechung offene Punkte

Protokoll:

- Weblink zu Repository mit plugins (net.osmand.plus.smartnaviwatch) https://github.com/kkade/Osmand/tree/master/OsmAnd/src/net/osmand/plus/smartnaviwatch
- Positionsmarkierung soll auf Karte im Watch-Display angezeigt werden
- Es wird ein Logo entworfen
- Demonstration der Applikation wurde durchgeführt
- Diskussion zum Fortschrittsbalken:
    - Fortschritt auf dem Weg zur Route kann nicht angezeigt werden (OsmAnd rechnet Route häufig neu)
    - Evtl. könnte die Distanz zum nächsten Schritt angezeigt werden -> Testen im Praktischen Einsatz
    - Falls keine nützliche Information angezeigt werden kann, wird der Fortschrittsbalken nicht angezeigt

---

May 12th, 2015

Sitzungsprotokoll 13.05.2015 14:00

Erreichtes:

- Verbesserungen der App nach Tests
- Umsetzung der Punkte aus der letzten Sitzung und aus dem Feedback von Herr Hunkeler
- Weiterarbeit an Dokument / Konzeption Film
- Logo eingebaut

Offene Punkte:

- Inhalt Film definieren
- Festlegung Vibrationsfunktionalität

Traktanden:

- Demo der App
- Besprechung offene Punkte

Protokoll:

- Zeitangaben sollen auch auf unruhigeren Kartenabschnitten gut lesbar sein
- Einfaches Vibrationssignal bleibt bestehen
- Wiki-Seite wird nachgeführt

- Deployment wird entweder via Github oder HSR-Wiki gelöst
- Intro und Outro für das Video werden vom HSR Wiki übernommen
- Das Video zeigt die Funktionen des Arbeitsergebnisses
- Nächster Termin 20.05.2015 10:00 - 11:00

May 19th, 2015

Sitzungsprotokoll 20.05.2015 10:00
Erreichtes:
- Konzeption für Video abgeschlossen
- Erste Aufnahmen und Grafiken für Video fertig
- UI Anpassunge an Wear-App gemäss letzter Sitzung
- Release Version 1 mit APKs
- Github Landing-Page mit Link auf Release

Offene Punkte:
- Besprechung Abgabevorgehen / genauer Inhalt

Traktanden:
- Besprechung offene Punkte
- Kurzbesprechung Storyboard Video

Protokoll:
- Landing Page wird auf Englisch formuliert und enthält API-Tutorial
- Landing Page wird auf Github in ReadMe erwähnt
- Landing Page soll Link auf Geometa Lab enthalten
- Besprechung des Videoinhalts: Technische Erklärungen bleiben dem Leser des Dokuments vorbehalten
- Repository wird auf public gestellt
- Geometa Lab wird in Endscene des Videos erwähnt

II.9.6 Abgabe CD