RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

HOCHSCHULE HEILBRONN
TECHNIK WIRTSCHAFT INFORMATIK

BACHELOR-THESIS

System zur Erinnerung an Medikamenteneinnahmen für die Apple Watch mit Blick auf Gebrauchstauglichkeit und Akzeptanz der Plattform

Autor	Lukas Schmidt
Studiengang	Medizinische Informatik Universtität Heidelberg / Hochschule Heilbronn
Matrikelnummer	182706
Abgabe	31. März 2016
Referent	Prof. Dr.-Ing. Gerrit Meixner
Korreferent	Prof. Dr. Martin Haag

Zusammenfassung

Im fortschreitenden Alter fällt es Menschen schwer, regelmäßig Medikamente zum richtigen Zeitpunkt einzunehmen. Oft wird dies durch eine große Anzahl verschiedener Medikamente noch erschwert, die über den Tag eingenommen werden müssen.

In Zusammenarbeit mit dem Praktikum Informationssysteme/Telemedizinische Anwendungen an der Hochschule Heilbronn wird eine Smartwatch-Anwendung für die Apple Watch entwickelt. Diese basiert stark auf dezenten Benachrichtigungen am Handgelenk. Auch eine native Anwendung, die der Nutzer aktiv bedienen kann, wird erstellt.

Die Anwendung ist aufgrund fehlender Prototypen-Werkzeuge für die Apple Watch nativ in Swift realisiert. Hierbei wird auf Swift als relativ neue Programmiersprache eingegangen.

Die Evaluierung wird an stationären Patienten im Alter von 70-85 vorgenommen. Die Aussage der Befragung ergibt, dass sich eine Uhr als Medium sehr gut eignet, da sie etwas Vertrautes ausstrahlt. Die touchscreen-basierte Steuerung fällt aufgrund reduzierter sensomotorischer Fähigkeiten der Probanden negativ auf. Während die Benachrichtigung mit einhergehender Vibration sehr gut aufgenommen wird, ist die Interaktion mit der Uhr schwerfällig. Die Patienten haben Probleme eine native Anwendung zu starten. In Zukunft könnten diese Probleme mit Hilfe von Accessibility Funktionen der Plattform gelöst werden. Weiter bietet die Uhr interessante Anwendungsmöglichkeiten Menschen im Alltag zu unterstützen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

Git Git [5] ist eine verteilte Versionskontrolle. 5

Mode View Controller Architektur Muster zum Entwickeln von Software. Hierbei wird der Quellcode in die Aufgabengebiete Model(Daten), View(Bildschirminhalt), Controller (Verwaltet Model und View) eingeteilt.. 26

PITA Ein Praktikum im Rahmen des Bachelor Studienganges an der Hochschule Heilbronn zur Entwicklung von telemedizinischen Informationssystemen. i, 1, 3, 5, 42

Remote Server Ein Computer, der über das Internet Anfragen verarbeitet und beantwortet. 29

SDK Eine Schnittstelle zum System, mit dem Drittanbieter Software für die Zielplattform des SDKs entwickeln können. 17

1. Einleitung

1.1. Motivation

Durch steigende Anzahl an Medikamenten, die Patienten über den Tag nehmen müssen, kann dies zu einer großen mentalen Aufgabe für den Patienten werden. Durch Hilfsmittel wie Medikationspläne oder nach Zeit vorsortierte Medikamente, kann die Planung der Einnahme unterstützt werden.

Zur Erleichterung der Patienten soll der Medikationsplan nun per Smartwatch einsehbar gemacht werden und somit dem Patienten eine Interaktion mit dem Plan ermöglichen. Daraus bieten sich auch Vorteile für den behandelnden Arzt, der Einblick in die Einnahmegewohnheiten seines Patienten bekommt. Die initialen Ideen stammen vom PITA an der HS-Heilbronn. An der HS-Heilbronn wird eine Komponente für Ärzte entwickelt, die es ihnen ermöglicht, die Medikationen der Patienten zu pflegen, zu überwachen und auszuwerten.

1.2. Zielsetzung

Die Ziele der Arbeit sind wie folgt.

Es wird ein interaktiver Prototyp entwickelt, welcher auf der Apple Watch ausgeführt werden kann. Dieser Prototyp soll dann mit geeigneten Probanden, die zur passenden Zielgruppe gehören, evaluiert werden. Die Ergebnisse der Evaluierung sollen im Rahmen der technischen Möglichkeiten umgesetzt werden. Zum Schluss soll noch die Machbarkeit überprüft werden, die Anwendung an das Backend-System vom PITA anzuschließen.

1.3. Aufbau der Arbeit

In Kapitel 2 wird auf das Forschungsumfeld der Arbeit Bezug genommen. Weiter werden technologische Grundlagen beschrieben wie Apple‘s Swift Programmiersprache und die Technologie von Wearables. Im dritten Kapitel werden Anforderungen aufgeführt, die teilweise aus dem PITA-Praktikum stammen. Weiter werden Anforderungen geschildert,

die sich aus den Möglichkeiten der Apple Watch ergeben. Abschnitt 4 betrachtet die Kernpunkte der Implementierung. Weiter wird hier der Prototyp beschrieben. In Kapitel 5 wird die Planung und Durchführung der Evaluierung dargestellt. Die Ergebnisse der Evaluierung an der Zielgruppe sind hier ebenfalls zu finden. In der Diskussion in Kapitel 6 wird aufgezeigt, welche Probleme während der Arbeit entstanden sind. Es wird veranschaulicht, wie der Prototyp im Gesamtbild einzuordnen ist und es wird ein Ausblick gegeben, welche Ziele mit dem System weiter verfolgt werden können.

2. Stand der Wissenschaft

Während das Forschungsfeld der „Medication Adherence“- das Einhalten des Medikamentenplans - schon jahrzehntelang erforscht wurde, ist diese Arbeit Teil eines noch sehr jungen Forschungsgebietes. Smartwatches existieren noch nicht lange im Consumer Bereich. Im folgenden ist eine Zusammenfassung der relevanten Forschungen zu finden. Apple‘s Programmiersprache Swift sowie die Technologie der Wearables wird beschrieben.

2.1. Arbeiten im Forschungsumfeld

Leider gibt es kaum Arbeiten, welche sich mit dem Thema Smartwatch und „Medication Adherence“beschäftigen. Dies ist auf das noch junge Forschungsfeld zurückzuführen. Sailer‘s Artikel [22] bietet einen Einstieg für diese Arbeit. Hier wurde auch die Thematik für das PITA abgeleitet, dessen Erkenntnisse hier fortgeführt werden.

Weiter gibt es sehr spannende Forschungen im Bereich der Smartwatchanwendung, die in fortschreitender Entwicklung auch im Bereich „Medication Adherence“vorstellbar sind. Mit Ambient Assisted Living, also der technischen Unterstützung älterer oder eingeschränkter Personen im Haushalt, beschäftigt sich die Arbeit „Non-obstructive Room-level Locating System in Home Environments Using Activity Fingerprints from Smartwatch“ [16]. Die von dieser Arbeit abgeleitete Lösung kann auch für rechtzeitige Medikamenteneinnahme genutzt werden. So könnte ein Medikationsalarm nur ausgelöst werden, wenn der Patient sich auch im richtigen Zimmer befindet. Durch den kürzeren Weg zum Medikament sinkt die Gefahr, auf der Suche nach dem Medikament, die Einnahme wieder zu vergessen. Die Arbeit von Laput gliedert sich ebenfalls in diesen Bereich ein und hat auch einen Kontextgewinn zur Folge [15]. Berührt der Träger des Smartwatch Prototyps einen Gegenstand, so erkennt die Uhr das elektromagnetische Feld des Gegenstandes. Durch maschinelles Lernen werden nun die Gegenstände mit ihrem elektromagnetischen Feld verknüpft. Nun kann die Uhr erkennen, welchen Gegenstand der Träger berührt. Dieser Kontextgewinn, welche Gegenstände der Träger der Uhr berührt, könnte Fehler bei Medikamenteneinnahmen verhindern, indem die Uhr die Medikamente erkennt, die der Patient berührt. Die Studie „Smartwatch in vivo“ [24] untersucht das Nutzungsverhalten von Smartwatch Nutzern. Hierzu trägt der Nutzer eine Schulterkamera, die die Interaktion mit der Uhr über drei Tage filmt. So zeigt die Studie auf, dass neben der Uhrzeit (mit ca. 50% Nutzungsdauer) die Notification mit 20% an Nutzungsdauer die häufigste

Interaktion ist. Anwendungen werden so gut wie nie genutzt. Auch die durchschnittliche Interaktionszeit von ca. 7s ist ein wichtiges Ergebnis.

Orientiert man sich an Arbeiten, deren Ziel die smartphonegestützte Medication Adherence war, findet man unter anderem die aktuelle nationale Umfrage [14] aus den USA, bei der 1604 Smartphone Nutzer zu ihrer Nutzung von Gesundheitsanwendungen befragt wurden. Fast 50% der Befragten gaben an, eine oder mehrere Gesundheitsanwendungen auf ihrem Smartphone installiert zu haben. Eine wichtige Aussage der Umfrage war, dass ein Großteil der Nutzer nicht bereit ist, für Gesundheitsanwendungen zu bezahlen.

3. Analyse und Entwurf

Im Kapitel drei werden Ergebnisse der Analyse und des Entwurfes dargestellt. Große Teile der Anforderungsanalyse stammen aus dem PITA. Die Hardware und Software der Apple Watch wird genauer beschrieben.

3.1. Entwicklungsmethodik

Da das Projekt nur durch eine Person durchgeführt wurde, kann man nicht von einer definierbaren Methodik sprechen. Es handelte sich um ein iteratives Vorgehen. Funktionen, die bei der Evaluation erarbeitet wurden, sind direkt in eine neue Version der Anforderungen integriert worden. Der Quellcode wurde mit Git versioniert verwaltet.

3.2. Anforderungsanalyse

Die Anforderungen wurden von einem Gespräch mit Monika Pobiruchin am Anfang vom PITA erhoben. Kernaussage dieses Gesprächs war, dass das System von alten Menschen genutzt werden soll, um Medikamente regelmäßiger einzunehmen. Dies beinhaltet eine geringe Auseinandersetzung mit der Technik des Systems. Die Uhr soll möglichst autonom sein und nicht zwingend an ein Smartphone gekoppelt sein.

3.3. Usecases

Von dem Gespräch mit Monika Pobiruchin wurde eine Persona für die Zielgruppe erstellt. Mit Hilfe der Persona wurden die Kern-Usecases abgeleitet. Die folgenden Usecases 1 bis 4 wurden im PITA erarbeitet und von dort übernommen.

Usecase 1	**Der Patient wird an ein Medikament erinnert**
Primärer Akteur	Patient

Beschreibung	Ein Erinnerungs Pop-Up erscheint auf dem Display und es wird ein Signal/eine Vibration ausgelöst. Das Pop-Up zeigt ein Abbild des Medikaments, dessen Namen und die Uhrzeit, zu der es eingenommen werden soll.
Vorbedingung	Es wurde ein Medikationsplan aus der DB auf die Uhr geladen.
Ablauf	• Das Pop-Up mit der Erinnerung erscheint auf dem Smartwatch-Display. Die Erinnerung beinhaltet Informationen zur Uhrzeit, Menge und Art der Medikation • Der Patient bestätigt, dass er das Medikament genommen hat • Gerät bestätigt visuell, dass der Patient das Medikament als genommen markiert hat
Alternativablauf	• Das Pop-Up mit der Erinnerung erscheint auf dem Smartwatch-Display • Der Patient wählt Option zum Verschieben der Medikation aus • Auf einem zusätzlichen Dialog kann er aus einer Auswahl eine Zeitdauer wählen, um die die Medikation verschoben wird
Alternativablauf 2	• Das Pop-Up mit der Erinnerung erscheint auf dem Smartwatch-Display • Der Patient reagiert nicht auf die Erinnerung • Die Erinnerung wird alle x Minuten wiederholt, solange der Patient nicht reagiert. • Das Fehlen einer Reaktion des Patientens innerhalb einer Zeit von x Minuten wird vermerkt
Ergebnis	Der Patient hat die Einnahme des Medikaments bestätigt und dieses auch eingenommen
Alternativergebnis 1	Der Patient hat die Erinnerung an die Medikamenteneinnahme verschoben
Alternativergebnis 2	Der Patient hat die Erinnerung an das Medikament ausgeschaltet

Usecase 2	**Der Patient wird an mehrere Medikamente erinnert**
Primärer Akteur	Patient
Beschreibung	Ein Erinnerungs Pop-Up erscheint auf dem Display und es wird ein Signal/eine Vibration ausgelöst. Das Pop-Up zeigt eine Liste der Medikamente, die eingenommen werden müssen.
Vorbedingung	Es wurde ein Medikationsplan aus der DB auf die Uhr geladen.
Ablauf	• Das Pop-Up mit der Erinnerung erscheint auf dem Smartwatch-Display. Die Erinnerung beinhaltet Informationen zur Uhrzeit, Menge und Art der Medikationen • Der Patient bestätigt, dass er die Medikamente alle genommen hat • Gerät bestätigt visuell, dass der Patient die Medikamente als genommen markiert hat
Alternativablauf	• Das Pop-Up mit der Erinnerung erscheint auf dem Smartwatch-Display • Der Patient wählt Option zum Verschieben der Medikationen aus • Auf einem zusätzlichem Dialog kann er aus einer Auswahl eine Zeitdauer wählen, um das die Medikationen verschoben werden
Alternativablauf 2	• Das Pop-Up mit der Erinnerung erscheint auf dem Smartwatch-Display • Der Patient wählt ein Medikament von der Liste aus • Patient befindet sich nun im Usecase 1

Alternativablauf 3	• Das Pop-Up mit der Erinnerung erscheint auf dem Smartwatch-Display • Der Patient reagiert nicht auf die Erinnerung • Die Erinnerung wird alle x Minuten wiederholt, solange der Patient nicht reagiert. • Das Fehlen einer Reaktion des Patientens innerhalb einer Zeit von x Minuten wird vermerkt
Ergebnis	Der Patient hat die Einnahme der Medikamente bestätigt und diese auch eingenommen
Alternativergebnis 1	Der Patient hat die Erinnerung an die Medikamenteneinnahme verschoben
Alternativergebnis 2	Der Patient hat die Erinnerung an das Medikament ausgeschaltet
Alternativergebnis 3	Der Patient hat bestimmte Medikamente ausgewählt und mit dem weiterführenden Usecase 1 bearbeitet

Usecase 3	**Einzelne Einnahmebestätigung zurücknehmen**
Primärer Akteur	Patient
Vorbedingung	Es wurde ein Medikationsplan aus der DB auf die Uhr geladen. Eine Medikation wurde als genommen markiert
Ablauf	• Das System zeigt das genommene Medikament an • Der Benutzer drückt auf den Button mit der Aufschrift “Rücknahme” • Das System wechselt zur Darstellung eines einzelnen Medikaments, beschrieben im Use-Case 1
Ergebnis	Die Einnahmebestätigung ist zurückgenommen. Die Erinnerung ist erneut zu bestätigen oder zu verschieben.

Usecase 4	**Mehrere Einnahmebestätigungen zurücknehmen**
Primärer Akteur	Patient
Vorbedingung	Es wurde ein Medikationsplan aus der DB auf die Uhr geladen. Mehrere Medikationen, welche zur gleichen Zeit genommen wurden, wurde als genommen markiert
Ablauf	• Das System zeigt die genommenen Medikamente an • Der Benutzer drückt auf den Button mit der Aufschrift “Rücknahme” • Das System wechselt zur Darstellung mehrerer Medikamente, beschrieben im UseCase 2
Ergebnis	Die Einnahmebestätigung ist zurückgenommen. Die Erinnerung ist erneut zu bestätigen oder zu verschieben.

Diese vier Usecases sind auch Grundlage für diese Arbeit. Jedoch werden nur die Usecases umgesetzt, welche auch in der Evaluation geprüft werden können.

3.4. Apple‘s Programmiersprache Swift

Swift wurde im Juni 2014 von Apple vorgestellt und genießt seitdem steigendes Interesse. Im Juni 2015 wurde Version 2.0 veröffentlicht [2]. Mit Version 2.0 wurde ebenfalls der Plan vorgestellt, Swift Open Source zu machen und somit auch anderen Plattformen die Entwicklung mit Swift zu ermöglichen [2]. Diesen Plan setzte Apple Ende 2015 in die Tat um. Swift ist nun völlig Open Source. Es können Vorschläge für neue Sprachfunktionen gemacht werden. Auch Apple‘s Entwicklungsteam diskutiert seine Pläne für die Sprache öffentlich [13].

3.4.1. Objective-C und Swift

In den frühen 80er Jahren entwickelte Brad Cox die Sprache Objective-C [3]. Weiter führt Dalrymple aus, dass die Sprache die Vorteile einer schnellen C-Sprache mit den Vorteilen der objektorientierten Sprache SmallTalk verbinden sollte. Die Firma NextSTEP nutzte

Objectiv-C und als NEXTStep von Apple aufgekauft wurde, integrierte Apple Objectiv-C und ermöglichte Mac-Entwicklern die Nutzung.

Als Apple nun 2008 seine iOS Plattform öffnete und Entwickler eigene Anwendungen für das System schreiben konnten, bekam Objective-C neue Aufmerksamkeit. Viele Entwickler sahen Objective-C als ein Überbleibsel alter Zeiten und waren der Sprache gegenüber negativ eingestellt. Apple stand nun unter Zugzwang um seine Plattform, mit der große wirtschaftliche Interessen verbunden sind, für Entwickler attraktiv zu halten [27]. Da jedoch alle highlevel APIs in Objectiv-C vorhanden sind, ist es technisch nicht möglich, auf eine bekannte Sprache wie Java für iOS und Mac Entwicklung umzusteigen. Man entschied sich für eine Neuentwicklung mit Hinblick auf neue Programmierparadigmen und auf sehr gute Kompatibilität zu alten Objectiv-C APIs [27].

3.4.2. Überblick der Neuerung

Swift bietet gegenüber Objectiv-C viele Neuerungen. Diese Neuerungen sind bekannte Konzepte, die schon in anderen Programmiersprachen Verwendung finden. Die meisten dieser Verbesserungen unterstützen den Entwickler früh, Fehler im Quellcode zu entdecken. Im folgenden sind die wichtigsten Neuerungen beschrieben.

Type Inference

Bei Type Inference erkennt der Compiler, welcher Typ in eine Variable instanziiert wurde. Es ist nicht nötig, für die Variable eine Typendefinition anzugeben [2]. Hierdurch wird der Quellcode leichter lesbar.

Listing 3.1: Beispiel zu Type Inference in Swift label

```
//Variable with type Definition
let medication: Medication = Medication()

//Variable with infered type
let medication = Medication()
```

Closures - Functions as First Class Types

Während bei strikt objektorientierten Sprachen nur Objekte und primitive Datentypen existieren, gibt es Sprachen, bei denen Funktionen als Typen existieren. Diese Funktionen können auch als Referenz in eine Variable gespeichert werden. Folgende Code Beispiele sollen dies veranschaulichen. Wir nutzen hierfür einen asynchronen Netzwerk-Request [2].

Listing 3.2: Java Beispiel zu First Class Objects

```
public interface NetworkRequestResponseHandler
    public handleNetworkResponse(Error error, Response response);
}

public interface NetworkRequest
    public performNetworkRequest(NetworkRequestResponseHandler
       handler);
}

public class ExampleClass implements NetworkRequestResponseHandler {
    public handleNetworkResponse(Error error, Response response) {
        //Use Response
    }

    public static main(String[] args) {
       NetworkRequestImpl().performNetworkRequest(this)
    }
}
```

In Beispiel 3.2 wird Java als Repräsentant für eine strikt objektorientierte Sprache verwendet. Hier wird ein Interface definiert, welches der Nutzer des Netzwerk-Requests implementieren muss, um den Request zu empfangen. Beim Aufrufen des Requests muss nun der Aufrufer als Referenz übergeben werden, damit bei Abschließen des Requests der Aufrufer benachrichtigt werden kann (`handleNetworkResponse`).

Listing 3.3: Swift Beispiel zu First Class Objects

```
protocol NetworkRequest {
    func performNetworkRequest(handleNetworkRequest:(Error, Response)
       ->())
}

class ExampleClass {
    func main() {
        //Store a function in a variable
        let handleRequest = {(error, response) in
            //Handle Response
        }
        NetworkRequestImpl.performNetworkRequest(handleRequest)
    }
}
```

Im Codebeispiel 3.3 benötigt es kein Interface für den Nutzer des Netzwerk-Requests. Es ist nun in Swift möglich, eine Funktion zu definieren und diese gleichzeitig in einer lokalen Variablen zu speichern. Nun kann diese Funktion als Referenz zum Netzwerk-Request übergeben werden. Die Funktion wird aufgerufen, wenn der Netzwerk-Request beendet ist.

Generics

Generics sind schon längere Zeit Teil moderner Programmiersprachen. Sie unterstützen den Entwickler, um Fehler zur Compilezeit zu entdecken. Ein sehr gutes Beispiel hierfür sind Collections (Arrays, Listen, Set, etc). Ein Collection Typ wie Array muss nicht für jeden Typ, den er hält, neu implementiert werden.

Listing 3.4: Swift Beispiel zu Generic Array Collection

```
class Medication: NSObject {
    //...
}
//Could only store type Medication
class CustomMedicationArray {
    //...
    func add(medication: Medication) {
        //...
    }

    func objectAtIndex(index: Int) -> Medication {

    }
}
//Could only store any type which inherits from NSObject
//loses all type information
class CustomArray {
    //...
    func add(medication: NSObject) {
        //...
    }

    func objectAtIndex(index: Int) -> NSObject {
        //...
        //return object
    }
}
//Could store any type
//Does not lose type information
```

```
class GenericArray<T> {
    //...
    func add(medication: T) {
        //...
    }

    func objectAtIndex(index: Int) -> T {
        //...
        //return object
    }
}
let genericArray = GenericArray<Medication>()
```

Im Codebeispiel 3.4 werden die Vorteile von Generics aufgezeigt. Der erste Versuch, einen Array zu implementieren (`CustomMedicationArray`), zeigt, dass dieser Array eine sehr gute Typen Deklarierung enthält. Der Compiler kann den Entwickler also maximal unterstützen, jedoch ist diese Implementierung minimal wiederverwendbar. Der zweite Ansatz (`CustomArray`) gibt als Typ-Einschränkung den globalen Supertyp an. Dies führt zu einer maximalen Wiederverwendbarkeit, da jede Klasse von diesem globalen Supertyp erbt (funktioniert nur theoretisch, da in Swift kein Zwang besteht, von einem globalen Supertyp zu erben). Die letzte Implementierung (`GenericArray`) nutzt nun Generics. So wird bei der Definition der Klasse ein generischer Typ `T` eingeführt. Dieser Typ ist eine Art Platzhalter für einen konkreten Typ, der später vom Array gehalten wird. So muss bei der Initialisierung der generischen Klasse die Typ-Information für `T` übergeben werden (siehe letze Zeile in 3.4).

Das oben beschriebene Beispiel dient nur zur Verdeutlichung des Konzeptes. Swift bietet eine Reihe an Collection Types, darunter auch eine generische Array Implementierung.

Protocol Extension

Protocol Extension helfen dem Entwickler Abstraktionen, die über ein Interface (in Swift Protocol) definiert werden, zu implementieren und dadurch Duplizierungen zu minimieren. So kann es sein, dass eine Methode eines Interfaces in jeder Klasse, die das Interface implementiert, gleich umgesetzt wird. Dies führt zu einer Code-Duplizierung.

Listing 3.5: Swift Beispiel zu Generic Array Collection

```
protocol List {
    typealias T
    var count: Int { get }
    func objectAtIndex(index: Int) -> T

    var last: T? { get }
```

```
    var first: T? { get }
}

extension List {
    var last: T {
        return objectAtIndex(count - 1)
    }

    var first: T {
        return objectAtIndex(0)
    }
}

class ArrayList<Element>: List {
    typealias T = Element
    var count: Int = 0

    func objectAtIndex(index: Int) -> T {
        //some Implementation
    }
}
```

Im Codebeispiel 3.5 wird ein List Interface definiert. Die Extension zu diesem Interface ermöglicht die Implementierung von `last` und `first`. Diese Implementierung teilen sich nun alle Klassen, die List implementieren.

Optional Types

Optional Types eröffnen neue Möglichkeiten beim Modellieren von Datenmodellen und Erstellen von APIs. Es ist so möglich, Argumente in einer Methode als „0ptional“zu kennzeichnen und somit dem Nutzer der Methode zu erlauben, eine null-Referenz zu übergeben. Ist kein Optional-Type gekennzeichnet, so verbietet der Compiler eine null-Referenz Übergabe [2]. Im folgenden Beispiel wird eine Medikation in Java ohne Optional Types und in Swift mit Optional Types modelliert.

Listing 3.6: Java Beispiel mit fehlenden Optional Types

```
public class Medication {
    Date creationDate = new Date();
    Date executionDate;
}

Medication medication = new Medication()
medication.creationDate // null?
```

```
medication.executionDate // null?
```

Im Codebeispiel 3.6 ist zu erkennen, dass zur Compilezeit keine Aussage über den Zustand der Instanz-Variablen getroffen werden kann. Der Entwickler muss also aus dem logischen Kontext erkennen, welche Variablen eine null-Referenz enthalten könnten. Dies kann zu Laufzeitfehlern führen.

Listing 3.7: Swift Beispiel mit Optional Types

```
class Medication {
    var creationDate = Date()
    var executionDate: Date?
}

let medication = Medication()

medication.creationDate // compiler promises to be not null
medication.executionDate // could be null

medication.creationDate = nil // compile error
```

In 3.7 sind die Instanzvariablen nun mit Optional Types modelliert. Nun kann zur Compilezeit zugesichert werden, welche Variablen eine null-Referenz enthalten können und welche Variablen sicher mit einem Wert belegt sind. Dies führt zu weniger Fehlern während der Laufzeit.

Auch APIs können mit „Optionals“modelliert werden. So darf ein Parameter nicht null sein, wenn er nicht als Optional definiert wurde. Dies macht eine API strikter und führt ebenfalls zu weniger Laufzeitfehlern, da Fehler schon zur Compilezeit erkannt werden.

3.5. Wearables

Übersetzt man Wearables ins Deutsche, bedeutet es „Tragbares“oder „Anziehbares“im Sinne von einem Kleidungsstück tragen. Wenn man nun den Begriff Wearables mit Kleidung assoziiert, liegt man nicht falsch. Anstatt Computer auf dem Schreibtisch oder in der Hosentasche zu haben, trägt man sie am Körper wie Kleidungs- oder Schmuckstücke [4]. Die Grenzen zwischen Kleidung und Computer verschmelzen und es ist manchmal nicht klar, was man schon als Wearable bezeichnen kann oder auch nicht.

Wearables sind meist mit Sensoren ausgestattet, die Daten aus der Körperregion sammeln, an der sie getragen werden [26]. Diese Daten zeigen immer nur einen Teilausschnitt. Durch das Tragen von mehreren Geräten am Körper verteilt, können mehr Daten gesammelt werden. So ist es möglich, einen noch genaueren Überblick über den Zustand des Körpers zu erhalten [26].

Erst mit der Miniaturisierung der Computertechnik und Sensoren war es möglich, Computer mit integriertem Akku in Größe einer Streichholzschachtel herzustellen. Während Swatch 1995/1996 eine Uhr vorstellte, die als Skipass funktionierte, stellte Apple 18 Jahre später die Apple Watch vor, die über einen Mikroprozessor, WLAN und eine Vielzahl an Sensoren verfügt (mehr zur Apple Watch in 3.6)

Doch nicht nur Uhren zählen zu den Wearables. Google präsentierte mit der Google Glass eine Datenbrille, die über eine Kamera, ein Heads „Prismatic head-mount„Display, Sprachsteuerung sowie Internetverbindung und weitere Sensoren verfügt [17]. Auch ein Ring, der mit Hilfe eines Sensors den Puls misst, existiert als Prototyp [26].

Während im Smartphone-Markt noch ein Focus auf Leistung und Funktion der Geräte gelegte wurde, darf man beim Wearables-Markt den Faktor der Ästhetik nicht vergessen. Hier wird ein Bereich betreten, der starke Einflüsse von Mode aufzeigt. Anwender eines Gerätes achten also nicht mehr nur auf die Funktion, sondern auch auf Form, Farbe und Lifestyle, den das Produkt verkörpert. Apple bietet unter dem Namen „Watch Edition “eine Apple Watch aus echtem Gold an, deren Preis über 10.000 Euro beträgt und sich an den Luxusmarkt richtet. Auch TAGHeuer, eine Firma, die sich auf luxuriöse und modische Uhren spezialisiert hat, betritt nun auch den Markt der Wearables [25].

3.6. Apple Watch

Eine Andorderung, welche sich aus dem Kontext dieser Arbeit entnehmen lässt, ist die Nutzung der Apple Watch als Zielplattform. Die Apple Watch wurde im September 2014 vorgestellt und startete im April 2015 mit dem Verkauf.

3.6.1. Hardware

Die Apple Watch existiert in zwei Versionen. Eine Uhr mit 38mm (272x340) und eine mit 42mm (312x390) großem Display. Die Uhr bietet einen 8GB großen internen Speicher. Mit einer Akkulaufzeit von 18h unter durchschnittlicher Nutzung hält die Uhr einen Tag durch [20].

3.6.2. Abhängigkeit zum iPhone

Die Apple Watch wurde als Erweiterung zum iPhone entwickelt. Und so sind auch viele integrale Funktionen nur vom iPhone aus steuerbar. Die Uhr kann ohne ein iPhone nicht durch den Setup-Prozess geleitet werden. Auch native Anwendungen (siehe 3.6.3) können nur über das iPhone installiert werden. Sind diese Schritte getan, also die Uhr betriebsbereit und Anwendungen installiert, kann die Uhr teilweise auch autonom agieren. So kann sie auch ohne iPhone, über WLAN, mit dem Internet kommunizieren.

3.6.3. Software

Mit Erscheinen der Uhr wurde auch das Betriebssystem in Version 1 ausgeliefert und dazu das Software Development Kit namens WatchKit. Dies erlaubte es Anwendungen zu entwickeln, welche auf dem verbundenen iPhone ausgeführt wurden. Die Folge war schlechter Performance der Anwendungen und zu völliger Abhängigkeit zum iPhone.

Im Juni 2015 veröffentlichte Apple die erste Vorabversion von watchOS 2.0, welches später im September 2015 für Endnutzer bereit gestellt wurde. WatchOS bietet mehr Unabhängigkeit für Anwendungen, da diese direkt auf der Uhr ausgeführt werden. Für Anwendungsentwickler gibt es vier Arten, Informationen auf der Uhr darzustellen. Es handelt sich um native Anwendungen (Apps), Glances, Complications und Actionable Notifications [9].

Native Anwendungen sind fest installiert auf der Uhr. Sie können unabhängig auf der Uhr gestartet werden. In einer nativen Anwendung lassen sich komplexere Programme realisieren, da der Nutzer durch viele Möglichkeiten zur Eingabe und Interaktion (3.6.6) hat. Installiert werden die Apps vom iPhone aus. Eine Watch-App benötigt immer eine iPhone App, die jeweils auf dem iPhone installiert ist.

Complications sind kleine Interface Elemente, die sich auf dem Zifferblatt der Uhr platzieren lassen. So können mit einem Blick auf die Uhrzeit auch Informationen aus der App abgelesen werden.

Ein Glance ist ein Interface, auf dem die wichtigsten Informationen einer App übersichtlich dargestellt werden. Der Nutzer soll mit einem Blick die Informationen erkennen. Es ist keine Interaktion mit einem Glance möglich. Tippt der Nutzer auf einen Glance, öffnet sich die zugehörige App. Glances sind optional zu einer App zu entwickeln.

Notifications, oder auch auf deutsch Benachrichtigungen, informieren den Benutzer über Ereignisse. Diese Ereignisse können entweder zeitlich geplant sein, vom Betreten einer Ortskoordinate ausgelöst werden, oder von einem Server auf das Gerät gepusht (Server sendet ein Ereignis, wie z.B. eine Nachricht) werden. Notifications können Aktionen beinhalten. So kann eine Bestätigung einer Notification direkt geschehen, ohne dafür die dazugehörige Anwendung zu starten. Dies nennt man eine Actionable Notification [9].

Es gibt zwei verschiedene Arten von Notifications, die beide die gleiche Funktion haben, jedoch mit unterschiedlichem Informationsgehalt angereichert werden. Zum Einen die Standard Notification. Diese Notification wird vom iPhone gespiegelt. Sie bietet nicht mehr Informationen gegenüber der iPhone Notification. Es ist jedoch möglich, mit einer nativen Anwendung auch eine optimale Darstellung der Notification zu entwickeln. Diese Darstellung kann detaillierte Informationen beinhalten wie Bilder, Karten oder eine genauere Beschreibung der Information [10].

3.6.4. Schnittstellen

Bluetooth 4.0 und Wi-Fi 802.11b/g sind die Netzwerkschnittstellen der Apple Watch. Dazu kommt noch ein NFC Chip, der jedoch nicht über eine API nutzbar ist und vorerst nur für Apple Pay, dem Apple eigenen Bezahldienst, vorgesehen ist [21].

3.6.5. Sensoren

Die Apple Watch besitzt einen Beschleunigungssensor und Gyroskop, welche genaue Bewegungsdaten liefern. Ebenso ist ein optischer Herzschlagsensor verbaut, der an der Unterseite der Uhr auf der Haut anliegt. Ein Mikrofon, welches für Spracheingabe genutzt werden kann, ist auch vorhanden.

3.6.6. Eingabe Interfaces

Neben eines normalen Touchscreens führte Apple in der Uhr auch eine Eingabeart namens Force Touch ein. Diese Technologie erlaubt der Uhr zu erkennen, wie fest der Nutzer auf das Display drückt. Dies ermöglicht eine neue Art der Eingabe. Besonders bei einer kleinen Eingabefläche, wie die der Apple Watch, ist eine neue Interaktionsdimension interessant, da es eine differenziertere Interaktion erlaubt. Leider ist eine mögliche Interaktion mit Force Touch optisch nicht zu erkennen, was eine klare Nutzerführung schwer macht.

Von analogen Uhren ist die Krone, das Rad an der Seite einer Uhr, bekannt. An ihr kann die Uhr eingestellt oder aufgezogen werden. An der Apple Watch ist die Krone auch zu finden. Apple nennt sie „Digital Crown“, also digitale Krone. Sie befindet sich ebenfalls an der Seite der Uhr. Die Krone ist drehbar und lässt sich als Druckknopf nutzen. Sie dient zum Scrollen von Inhalten sowie zum präzisem Auswählen von Elementen aus einer Auswahlliste. Durch das Nutzen der Krone wird der Bildschirm nicht durch einen Finger verdeckt, was bei einem kleinen Display von großem Vorteil ist.

3.6.7. Armband

Apple bietet sechs verschiedene Armbänder für die Apple Watch an (Stand Nov. 2015). Zusätzlich ist es möglich, Armbänder von Drittherstellern zu erwerben.

Das in der günstigsten Version mitgelieferte Armband (Sportarmband) verfügt über einen sehr komplizierten Verschlussmechanismus und ist deswegen weniger für Menschen geeignet, die über schwache sensomotorische Fähigkeiten verfügen. Es gibt auch Armbänder, die einen magnetischen Verschluss bieten und deswegen leichter zu handhaben sind. Diese Armbänder sind jedoch drei mal so teuer wie das Sportarmband.

An der Verbindung zwischen Armband und Uhr ist ein nicht weiter spezifizierter Wartungsport verbaut. Dieser Anschluss könnte in Zukunft Armbänder ermöglichen, die Informationen aus dem Armband an die Uhr zu übermitteln.

3.6.8. Prototyping

Zum Erstellen des frühen Prototyps wurden Papier-Prototypen erstellt. Durch die Nutzung dieses Werkzeuges konnte schnell und iterativ gearbeitet werden. Da an dieser Arbeit nur eine Person gearbeitet hat, bietet sich diese Methode an. Sie ist schnell zu erlernen, setzt keine Vorkenntnisse voraus und erzielt schnell Ergebnisse.

4. Umsetzung

Die in Kapitel 3 aufgeführten Analyse-Ergebnisse wurden für die erste Iteration des Prototyps umgesetzt. Es handelt sich hierbei um einen funktionsfähigen Prototypen, der nativ auf der Apple Watch ausführbar ist. Im folgenden werden Schritte der Umsetzung genauer beschrieben. Hierbei wird auf die Benutzeroberflächenerstellung sowie auf die Verbindung zwischen Uhr und iPhone eingegangen.

4.1. Werkzeuge

Für die Entwicklung wurde Apples Entwicklungsumgebung Xcode in Version 7.2.1 mit den SDKs iOS 9.2 und watchOS 2.1 verwendet. Zum Verwalten des Source-Codes wurde Git [5] eingesetzt. Als Git-Remoteserver wurde Github [6] verwendet. Die graphische Repräsentation der Medikamente sind keine statischen Bilder. Jede Art von Medikament hat eine Vorlage, die dynamisch mit jeglicher Farbe eingefärbt werden kann. Zum Erstellen dieser Vorlagen wurde PaintCode in Version 2.4 [19] genutzt. Zu beachten ist, das die Entwicklung von iOS und watchOS Anwendungen nur auf einem Mac möglich ist. Für diese Arbeit wurde Mac OS X 10.11.3 verwendet.

4.2. Benutzeroberfläche

Xcode bietet für visuelle Erstellung von Benutzeroberflächen einen eigenen integrierten Editor namens InterfaceBuilder. Hiermit können graphische Elemente per DragAndDrop zu einer Benutzeroberfläche zusammengestellt werden (Abbildung 4.1). Ebenfalls per DragAndDrop werden diese Interface-Elemete mit dem Quellcode verbunden (Abbildung 4.2).

Die Auswahl an Interface-Elementen ist sehr begrenzt und bietet kaum Möglichkeiten eigene Interface-Elemenente zu erstellen. Trotz dieser Begrenztheit lassen sich komplexere Anwendungen bauen.

Abbildung 4.1.: Interface Elemente zum Erstellen von Benutzeroberflächen

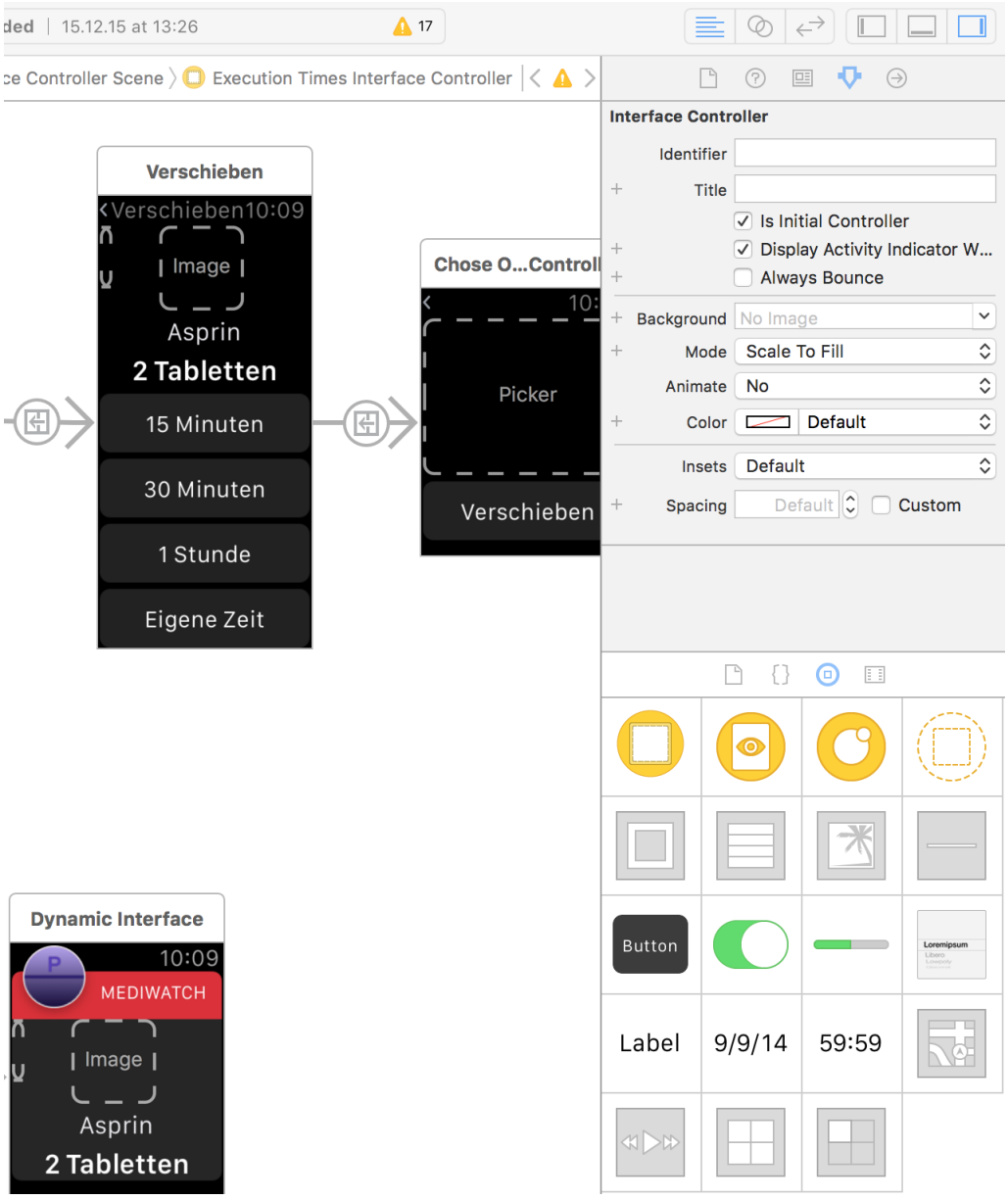
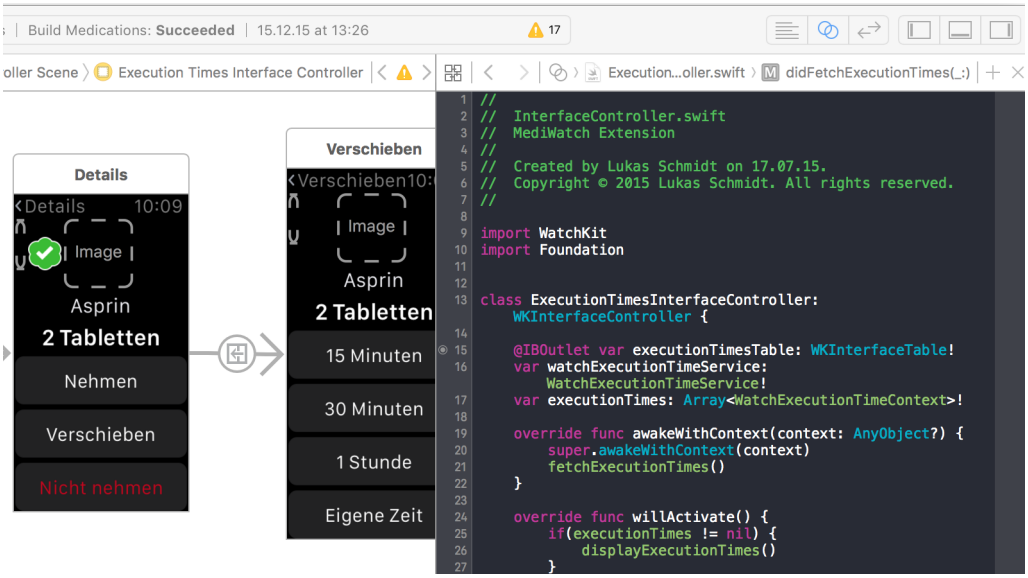


Abbildung 4.2.: Interface Elemente mit Quellcode verknüpfen



4.3. Datenhalten - Persistence

Da generell das iPhone das primäre Objekt der Nutzung ist, bietet sich die Datenhaltung auf dem iPhone an. Apple bietet mit CoreData [13] eine Framework zur Persistierung von Objekt-Graphen. Das Framework, welches auf SQLite aufsetzt, bietet eine gute Abstraktion einer Datenbank und ist deswegen leicht und schnell zu integrieren. So bringt CoreData einen graphischen Editor mit, der es erlaubt, Datenschemata zu erstellen und diese in Klassen zu transformieren. Daten, die in CoreData gespeichert wurden, lassen sich mittels gezielter Datenbankanfragen abfragen. So muss keine eigene Logik zum Suchen und Sortieren implementiert werden. Des Weiteren bietet CoreData Schnittstellen zur Benachrichtigung bei Änderung in den Daten. Durch langjähriges Bestehen des CoreData Frameworks lässt sich eine stabile Nutzung garantieren. CoreData kann auch auf der AppleWatch genutzt werden. Dupliziert man nun die Datenhaltung vom iPhone auf die Uhr, müssen diese beiden Datensätze immer konsistent gehalten werden. Dieser Abgleich bei Änderung einer Datenhaltung ist aufwendig, zu implementieren. Wird dieser Abgleich jedoch realisiert, ist die Uhr nochmal ein Stück unabhängiger.

Im konkreten Fall wurde keine doppelte Datenhaltung genutzt. Das iPhone ist also die zentrale Datenquelle. Die Apple Watch kann als Client betrachtet werden, der die Daten des iPhones präsentiert. Zur Realisierung einer Client-Server Verbindung zwischen Uhr und iPhone wurde das WatchConnectivity Framework genutzt, welches in 4.4 beschrieben wird. Daten müssen vor dem Versenden an die Uhr jedoch immer in ein serialisierbares Format übertragen werden. CoreData Objekte können nicht direkt übertragen werden.

4.4. WatchConnectivity

Wichtig für die Kommunikation zwischen Uhr und iPhone ist das WatchConnectivity Framework [12]. Hierbei ist im Listing 4.1 zu sehen, wie genau eine Verbindung aufgebaut werden kann. Es wird auch demonstriert, wie ein Datenaustausch realisiert werden kann. Wichtig ist, dass diese Verbindung zum richtigen Zeitpunkt im Application-Lifecycle aufgebaut wird, da es sonst zu Datenverlusten kommen kann. Wird die Verbindung zu früh erstellt, also wenn noch kein Handler zur Weiterverarbeitung gesetzt wurde, so laufen die Daten ins Leere und sind verloren.

Listing 4.1: Nutzung des WatchConnectivity Framework

```
class WatchExecutionTimeService: NSObject, WCSessionManagerDelegate {
    private let fetchExecutionTimesEventName = "fetchExecutionTimes"
    let namespace = "WatchExecutionTimeService"
    private let getTodayExecutionTimeEventName = "-
        get_today_executionTime"
    private let updateEventName = "-update_execution_time"

    let sessionManager: WCSessionManagerProtocol

    var didUpdateExecutionTime: (([String: NSObject])->())?
    var fetchExecutionTimesFunction: (()-> Array<
        ExecutionTimeProtocol>)?

    init(sessionManager: WCSessionManagerProtocol) {
        self.sessionManager = sessionManager
        super.init()
        self.sessionManager.registerDelegate(self)
    }
  /.../
    func fetchExecutionTimesOfToday(callback:(Array<
        WatchExecutionTimeContext>)->()) {
        let eventName = namespace + getTodayExecutionTimeEventName
        session.sendMessage([eventName: ""], replyHandler: {data in
            guard let contexts = self.executionTimesContextsFromDict(
                data) else { return }
            dispatch_async(dispatch_get_main_queue(), {
                callback(contexts)
            })
            }, errorHandler: onError)
    }

    func session(session: WCSession, didReceiveMessage message: [
```

```
        String : AnyObject]) {
        recieveData(message)
    }

    func session(session: WCSession, didReceiveMessage message: [
        String : AnyObject], replyHandler: ([String : AnyObject]) ->
        Void) {
        if let fetchExecutionTimesFunction =
            fetchExecutionTimesFunction {
            let dataTranform = fetchExecutionTimesFunction().map({(obj
                ) in
                return obj.codingData
            })
            replyHandler([fetchExecutionTimesEventName: dataTranform])
        }
    }

    func recieveData(message:[String: AnyObject]) {
        if let userInfo = message[namespace + updateEventName] as? [
            String: NSObject] {
            didUpdateExecutionTime?(userInfo)
        }
    }

    func session(session: WCSession, didReceiveUserInfo userInfo: [
        String : AnyObject]) {
        recieveData(userInfo)
    }
}
```

Mit der `sendMessage:` Methode können Nachrichten und Datenpakete gesendet werden, welche sich durch eine ID identifizieren lassen. Eine direkte Antwort auf eine solche Nachricht ist mittels eines Replay-Handlers möglich. So kann eine Art Request/Response Datentransfer realisiert werden. Zu beachten ist jedoch, dass die Kommunikation relativ langsam ist. Sie sollte nur verwendet werden, um kleine Informationen zu senden.

Werden erst beim Watch-App Start benötigte Daten vom iPhone angefordert, kann dies zu einer großen Verzögerung führen. Daher sollten Informationen, die auf der Uhr angezeigt werden, nicht erst zum Zeitpunkt der Darstellung angefordert werden. Gibt es eine Datenänderung auf dem iPhone, die relevante Daten für die Uhr enthält, so sollten diese mit der Methode `transferUserInfo:` an die Uhr gesendet werden. Mit dem Aufrufen dieser Methode wird die übergebene Information nicht direkt zur Uhr geschickt. Das System sendet die Daten zu einem optimalen Zeitpunkt (starke Verbindung zur Uhr, mögliche WLAN Verbindung) an die Uhr. Wenn nun die Watch-App startet, sind die Daten bereit

und können direkt dargestellt werden.

Zum Übertragen von größeren Daten steht die Methode `transferFile:` zur Verfügung. Die Methode verhält sich equivalent zu `transferUserInfo:`, wird jedoch in der Realisierung nicht verwendet.

4.5. Modellierung des Programmablaufes

Apple empfiehlt für iOS und auch watchOS die Nutzung von Mode View Controller als Grundlage zur Entwicklung von Anwendungen. So existiert in watchOS die Klasse `WKInterfaceController` als Supertyp für jeden ViewController. Jeder ViewController ist standardmäßig für einen logischen Bildschirm zuständig. So wird hier als Beispiel in Listing 4.2 der Controller zur Darstellung von Medikamentendetails beschrieben.

Die mit `IBOutlet` annotierten Variablen sind Referenzen für View Elemente. So hält `drugNameLabel` ein Label zur Darstellung des Medikamentennamens. Methoden, die mit `IBAction` annotiert sind, sind Aktionen, die bei der Interaktion mit dem Interface vom Nutzer ausgelöst werden. `onTakeMedication` wird ausgelöst, wenn der Nutzer den Button zur Bestätigung der Einnahme klickt. Der Prefix `IB` steht bei den Annotationen für InterfaceBuilder, welcher zum Erstellen der View-Elemente genutzt wird (4.2).

Zum Wechseln zwischen ViewControllern, also z.B. von der Darstellung einer Liste zu einer Detailansicht, führt Apple den Begriff Segue ein. Ein Segue beschreibt einen Übergang zwischen zwei ViewControllern. Es gibt zwei Typen von Segues: Push und Modal. Ein modaler Segue fährt vom unteren Rand des Bildschirms über den vorherigen ViewController. Diese Interaktion fordert den Nutzer auf, eine Eingabe oder Interaktion zu tätigen. Ist diese Interaktion erfolgt, verschwindet der ViewController wieder nach unten. Der Push Segue eignet sich für mehrdimensionale Interaktionen. Eine ideale Anwendung dafür ist, Informationen im Detail anzuzeigen. Der Segue verdeckt von der rechten Seite den aktuellen ViewController. Diese Interaktion vermittelt dem Nutzer eine Kontinuität. So kann dieser Segue sehr gut hintereinander genutzt werden, um immer tiefer in eine Informationsstruktur vorzudringen. Der Segue lässt sich durch einen Button am oberen linken Rand wieder verlassen. Auch eine Wischgeste, die dem Nutzer erlaubt den ViewController wegzuschieben, ist vorhanden. Der Push Segue ist den meisten Nutzern auch von iOS bekannt.

Durch die geringe Anzahl an Segues kann eine sehr konsistente Interaktion auf der ganzen Plattform garantiert werden. Dies erleichtert dem Nutzer die Navigation.

Die überschriebene Methode `awakeWithContext:` ist eine Lifecycle Methode des ViewControllers. Sie wird immer nach dem Erstellen des ViewControllers ausgeführt. Sie hilft, Daten von einem ViewController zum nächsten zu geben. Die Daten werden vom vorhergehenden ViewController bereit gestellt. Dieser überschreibt `contextForSegue WithIdentifier:` und kann nun Daten abhängig vom Identifier übergeben. So sind diese

zwei Methoden der einzige Kontaktpunkt von ViewControllern. Dies erhöht extrem die Kapselung der Komponenten.

Listing 4.2: ViewController zur Darstellung von Medikamentendetails

```
//
// MedicationDetailsController.swift
// Medications
//
// Created by Lukas Schmidt on 23.10.15.
// Copyright 2015 Lukas Schmidt. All rights reserved.
//

import WatchKit

class MedicationDetailsController: WKInterfaceController,
   ExecutionTimesDisplayDetailsProtocol {
    IBOutlet var drugImage: WKInterfaceImage!
    IBOutlet var amountDrugLabel: WKInterfaceLabel!
    IBOutlet var drugNameLabel: WKInterfaceLabel!
    IBOutlet var checkTakenImage: WKInterfaceImage!
    IBOutlet var delayLabel: WKInterfaceLabel!
    IBOutlet var takeMedicationButton: WKInterfaceButton!
    IBOutlet var delayMedicationButton: WKInterfaceButton!

    var executionTimeContext: WatchExecutionTimeContext!
    var executionTime: ExecutionTimeProtocol {
        return executionTimeContext.executionTime
    }

    override func awakeWithContext(context: AnyObject?) {
        guard let context = context as? WatchExecutionTimeContext
           else { return }
        executionTimeContext = context
        displayExecutimeDetails(context.executionTime)
    }

    override func willActivate() {
        updateUI()
    }

    IBAction func onTakeMedication() {
        executionTime.executionDate = executionTime.
           hasTakenMedication ? nil : NSDate()
        let executionTimeService = executionTimeContext.
```

```
            executionTimeService
        executionTimeService.updateExecutionTime(executionTime)
        updateUI()
        animateTakenIcon()
    }

    func updateUI() {
        checkTakenImage.setHidden(!executionTime.hasTakenMedication)
        takeMedicationButton.setTitle(executionTime.
            hasTakenMedication ? "Nicht genommen" : "Nehmen")
        delayMedicationButton.setEnabled(!executionTime.
            hasTakenMedication)
        displayExecutimeDetails(executionTime)
    }

    func animateTakenIcon() {
        self.checkTakenImage.setWidth(5)
        animateWithDuration(0.3, animations: {
            self.checkTakenImage.setWidth(26)
        })
    }

    IBAction func notTakenMedication() {
    }

    override func contextForSegueWithIdentifier(segueIdentifier:
        String) -> AnyObject? {
        return executionTimeContext
    }
}
```

4.6. Picker - Auswahl

Mit `WKInterfacePicker` bietet Apple eine Standard View-Componente zum Auswählen von Elementen aus einer Liste. Der Nutzer kann so mit der Digital-Crown die Elemente selektieren. Diese leicht zu integrierende View-Komponente ist einfach zu bedienen und hebt das Nutzungserlebnis von anderen Uhren ab. Mit dieser Komponente wird die Auswahl realisiert, mit der der Nutzer die Zeit zum Verschieben des Medikamentes wählt.

4.7. Notification Management

Notifications für die Medikationen werden mit der Notification API registriert [10]. Auch kann das Verhalten der Notifications angepasst werden. So können vordefinierte Aktionen wie „Einnehmen“oder „Verschieben“in die Notifications konfiguriert werden.

Bei der Notification handelt es sich um eine UILocalNotification [10]. Die Art von Notification benötigt keinen Remote Server, sondern wird vom verbundenen iPhone verwaltet. Zum jetzigen Zeitpunkt ist es nicht möglich, UILocalNotification von der Uhr zu erstellen. Dies würde die Uhr im Falle der Mediwatch-Anwendung noch unabhängiger machen. Sind die Notifications einmal auf dem iPhone registriert, werden sie zum Ausführungszeitpunkt auf der Uhr angezeigt. Dazu ist keine Verbindung zum iPhone mehr nötig.

4.8. Anwendung

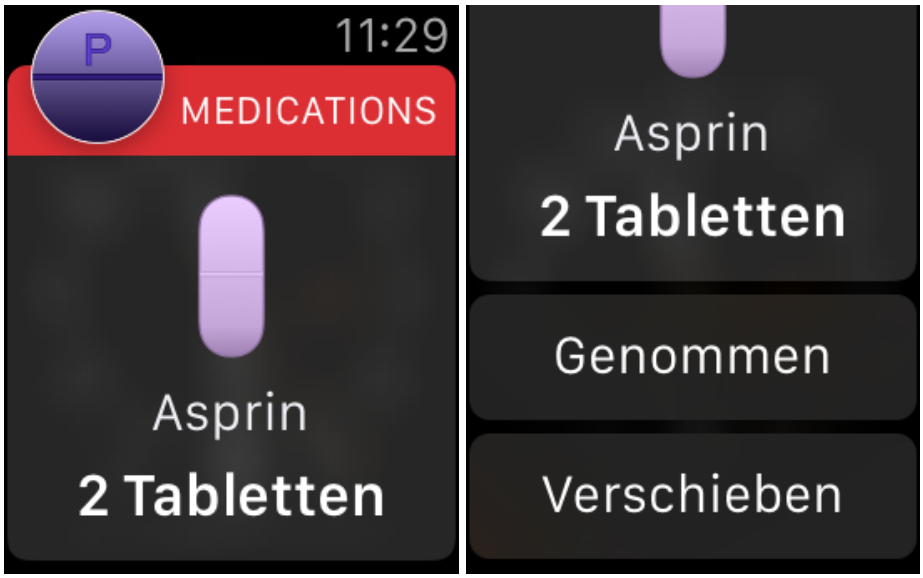
Die Anwendung besteht aus zwei Komponenten auf der Apple Watch. Eine native Notification und eine auf der Uhr installierte und ausgeführte Anwendung (siehe 3.6.3). In Abbildung 4.6 finden sich Bilder der Anwendung auf der Uhr. Dies sind Screenshots, die mit Hilfe einer Bildbearbeitungssoftware eingefügt wurden. Diese Bilder können ein Gefühl für die echte Anwendung auf der Uhr vermitteln.

4.8.1. Notification

Die Notification bringt die Uhr zum Zeitpunkt der geplanten Medikamenteneinnahme zum Vibrieren und lässt optional einen Ton erklingen. Dies führt dazu, dass der Anwender auf die Uhr blickt und an die bevorstehende Medikamenteneinnahme erinnert wird. Die Notification wurde daraufhin optimiert, sodass eine große graphische Repräsentation des Medikaments angezeigt wird. Dem Nutzer wird so die Wiedererkennung des Medikaments erleichtert (siehe 4.3). Neben dem Medikamentennamen wird auch die Dosierung des Medikaments dargestellt. Mit nur einer Aktion wird dem Nutzer ermöglicht, die Medikamenteneinnahme zu bestätigen oder sie zu verschieben. Die Notification ist darauf ausgelegt, eine sehr schnelle Interaktion zu ermöglichen, damit der Nutzer nicht länger als 10 Sekunden mit der Uhr interagieren muss (siehe 4.3). Diese Zeit führt sich auf die Studie „Smartwatch in vivo„ [24] zurück.

Diese Erweiterung der Notification mit dem Bild des Medikaments wird jedoch vom System geregelt. Dauert das Rendern der Notification zu lange, stellt das System eine reine Textdarstellung der Notification dar. Dies führt zu einem gewaltigen Informationsverlust. Als Anwendungsentwickler hat man nur die Option, seine Notification mit bestmöglicher Geschwindigkeit zu rendern, um eine Textdarstellung zu vermeiden. Das System bietet jedoch keine Garantie für eine erweiterte Darstellung der Notification. Dies kann im Extremfall zu Verwirrungen des Nutzers führen.

Abbildung 4.3.: Notification für ein Medikament



Die Notification sind ungeeignet, wenn es darum geht, mehrere Datenpunkte gleichzeitig darzustellen. Wenn also mehrere Medikamente in einer Notification angezeigt werden, so verliert man die Übersicht. In der konkreten Realisierung der Arbeit wurde deswegen vorerst nur ein Medikament pro Notification dargestellt.

4.8.2. Native Anwendung

Neben der Notification gibt es noch eine native App. Diese läuft auf der Uhr und stellt eine Liste der Medikamente des aktuellen Tages dar. Die App muss aktiv vom Nutzer gestartet werden. Der Nutzer muss also auf die Digital Crown drücken und dann die App aus der Übersicht wählen. In Abbildung 4.4 wird ein Medikament aus der Liste ausgewählt. Nachdem man ein Medikament ausgewählt hat, bekommt man in einer Detailansicht das Medikament zu sehen. Nun kann das Medikament als genommen oder nicht genommen markiert werden. Ein visuelles Feedback (grüner Haken) zur Einnahme zeigt dem Nutzer seine Aktionen an. Von der Detailansicht aus ist es auch möglich, ein Medikament zur späteren Einnahme zu markieren. Hier hat der Anwender die Möglichkeit, aus vordefinierten Zeiten auszuwählen oder eine eigene Zeit zu bestimmen. Um die gewählte Zeitdauer wird die Benachrichtigung für das Medikament verschoben (siehe Abbildung 4.5).

Abbildung 4.4.: Native Anwendung: Interface zum Wählen eines Medikaments (oben links). Medikament als genommen bestätigen (oben rechts). Medikament ist als genommen markiert (unten links). Medikament ist in der Übersicht als genommen markiert(unten rechts)

Abbildung 4.5.: Native Anwendung: Interface zum Verschieben eines Medikaments (oben links). Zeitdauer für das Verschieben wählen (oben rechts). Eigene Zeitdauer auswählen (unten links). Medikament ist in der Übersicht als verschoben markiert (unten rechts)

Abbildung 4.6.: Native Anwendung - dargestellt auf realen Apple Watches

4.9. Quellcode

Das ausführbare Projekt lässt sich von Github unter der URL [23] auschecken. So kann es kompiliert und getestet werden. Die hierzu benötigten Werkzeuge sind in 4.1 zu finden.

5. Evaluation

In Kapitel 5 werden die erarbeiteten Ergebnisse mit Probanden der Zielgruppe evaluiert. Hier werden Stärken und Schwächen des erstellten Konzepts sichtbar.

5.1. Evaluation

Die Evaluierung des Prototyps soll ein quantifiziertes Ergebnis liefern, anhand dessen Schwachstellen aufgezeigt werden sollen. Diese Schwachstellen werden mit der nächsten Iteration der Software ausgebessert.

5.1.1. Zielgruppe der Evaluation

Bei der Zielgruppe für die Evaluation handelt es sich um Menschen, deren Alltag von Medikamenteneinnahmen geprägt ist. Es sollten Menschen mit wenig oder keinen Vorkenntnissen mit touchscreen-basierten Geräten sein. Patienten fortgeschrittenen Alters, die sich stationär im Krankenhaus aufhalten, eignen sich gut. Diese nehmen in der Regel Medikamente und haben viel freie Zeit während des Aufenthaltes im Krankenhaus für eine Befragung.

5.1.2. Vorgehen

Im ersten Schritt wird dem Probanden die Apple Watch gezeigt und die dahinterliegende Technologie erklärt, um Neugier bei dem Patienten zu wecken. Nun können die Patienten die Uhr selbstständig anlegen. Im zweiten Schritt wird eine Notification, die eine Vibration auslöst, gestartet. Die Patienten müssen die Vibration spüren und dann auf die Uhr schauen. Dort sollen sie den Bildschirminhalt wiedergeben. Haben sie diese Aufgabe erfolgreich gelöst, so müssen sie die Benachrichtigung bestätigen, also auf den Button mit der Aufschrift „Genommen “(siehe 4.3) tippen.

In Schritt Drei müssen die Patienten den App-Bildschirm durch Drücken auf die Digital Crown aufrufen, um dort die Mediwatch App zu öffnen. Wenn die App geöffnet ist, soll ein Medikament ausgewählt werden und dessen Einnahmezeitpunkt um eine definierte Zeitdauer verschoben werden. Ist der Einnahmezeitpunkt verschoben, so ist die Aktion beendet.

5.1.3. Auswertung

Für die Evaluation werden zwei Fragebögen genutzt. Einmal AttrakDiff [7], welcher die subjektive Wahrnehmung der Bedienbarkeit und das Aussehen des Prototyps erfragt. Es handelt sich um einen standardisierten Fragebogen und eine standardisierte Auswertungsmethode. Da dieser Fragebogen keine Antworten über Funktionen des Prototyps gibt, ist es nötig, einen zweiten Fragebogen zu erstellen. Dieser erfragt die Situation, also den Kontext, in dem sich der Anwender befindet und die daraus folgenden Ansprüche an den Prototyp. So sollen fehlende Funktionen oder Fehler in der Analyse aufgedeckt werden. Die Fragen sind im Anhang zu finden. Die Erfassung der Fragen wird mit Limesurvey [1] in Version 2.05 durchgeführt. Da sich Limesurvey selbst hosten lässt, bleiben die erfragten Daten auf einem sicheren Hochschulserver und werden nicht bei Drittanbietern gespeichert.

5.1.4. Methodik - Thinking Aloud

Die Thinking Aloud nach Nielsen [18] beschreibt eine Methode zur Feedback-Gewinnung von Software. Probanden werden aufgefordert, während der Nutzung der Software und deren Nutzungsschnittstelle ihre Gedanken laut auszusprechen. Hierfür sind passende Probanden zu finden, die jeweils eine vorgegebene Aufgabe mit der Software ausführen (siehe 5.1.2).

Die Vorteile dieser Methode liegen auf der Hand. Sie ist sehr günstig da, keinerlei Geräte verwendet werden. Auch muss sie nicht in einem Labor durchgeführt werden. Es ist sogar von Vorteil, die Anwendung im üblichen Benutzungskontext zu testen. Meist reicht schon eine kleine Anzahl an Probanden, um ein deutliches Feedback zu bekommen. Die Methodik erweist sich als sehr robust, da kaum Fehler bei der Durchführung gemacht werden können. Die Methode kann zu jedem Zeitpunkt im Entwicklungsprozess eingesetzt werden. So eignet sie sich sehr gut für agile Prozesse, bei denen Projektteile flexibel getestet werden sollen. Durch die niedrige Barriere zur Durchführung können auch Softwareentwickler oder Manager diesen Prozess beiwohnen und erhalten dadurch sehr schnell Feedback. Da dieses Feedback sehr persönlich ist wird es stärker von den Testern aufgenommen als Ergebnisse, die reine Zahlen auf Papier sind.

Da die meisten Menschen nicht gewohnt sind, Selbstgespräche zu führen, sollte darauf geachtet werden, den Probanden immer im Redefluss zu halten. Zudem kann es für den Durchführenden schwierig sein, zwischen unwichtigen und wichtigen Aussagen des Probanden zu unterscheiden. Der Durchführende muss darauf achten, dass er mit der Anweisung den Probanden nicht beeinflusst und so das Ergebnis verfälscht.

5.2. Durchführung der Evaluation

Die Befragung wurde im Kreiskrankenhaus Mosbach unter Aufsicht der leitenden Ärztin Frau Flohr durchgeführt. Die Patienten befanden sich stationär in der geriatrischen Rehabilitationsklinik. Es haben insgesamt elf Patienten an der Befragung teilgenommen. Davon waren sieben weiblich und vier männlich. Alle waren im Alter zwischen 70 und 85.

5.2.1. Befragung der Patienten

Die Befragung in der Zielgruppe verlief sehr schwierig und nicht wie geplant. Der erste Schritt der Befragung klappte bei fast allen Probanden sehr gut. So erkannten sie die Notification mit der einhergehenden Vibration. Der Bildschirminhalt wurde von fast allen Patienten als verständlich geschildert. Das darauf folgende Bestätigen einer Notification verlief zu großen Teilen schlecht. Die Patienten konnten die Buttons zum Bestätigen nicht treffen, da bei ihnen die Feinmotorik eingeschränkt war. Das Öffnen einer App ist aufgrund der reduzierten Feinmotorik ebenfalls nicht möglich. Die App-Icons sind zu klein und werden nicht getroffen. Das Öffnen der Anwendung wurde bei allen Patienten nicht erreicht.

5.2.2. Auswertung - Ergebnis

Der Fragebogen AttrakDiff ist in dieser Zielgruppe nicht praktikabel. Eine sehr genaue Unterscheidung zwischen den Begrifflichkeiten, die der AttrakDiff abfragt, ist für die Zielgruppe nicht möglich. Dies fällt auf, wenn man mit den Probanden spricht. Sie schweifen oft ab und man ist nicht in der Lage ihre Aussagen zu erfassen. Direkte Fragen, die den Alltag der Probanden betreffen, werden zuverlässig beantwortet. Schwierigkeiten, die die Patienten mit der Medikamenteneinnahme haben, werden nicht zugegeben bzw. heruntergespielt. Die Befragungen wurden deswegen auf eine Thinking Aloud Methode (5.1.4) umgestellt. Leider sind die Ergebnisse so nicht mehr quantifizierbar, jedoch sind aus den Aussagen der Patienten und den Beobachtungen gute Schlussfolgerungen möglich. Auch der zweite Fragebogen zu den Funktionen und deren Nutzungskontext wurde nicht verwendet. Hier wurde versucht, während des Gesprächs mehr Einblick in das Leben der Probanden zu erhalten. Probanden erzählen gerne über ihr Leben und suchen eher das persönliche Gespräch. Daraus geht hervor, dass die Patienten mit großer Übereinstimmung keinen Internetanschluss besitzen. Dies schließt entfernte Aktualisierungen des Medikationsplans und Einnahmebestätigungen des Patienten für den Arzt aus. Missgefallen äußern die weiblichen Patientinnen über die Größe und das Aussehen der Uhr. Eine Uhr muss dem Geschmack der Patientin entsprechen. Hierbei spielt die Größe wie auch das Aussehen der Uhr eine Rolle. Bei der Beschreibung von anderen modischen Farben der Uhr oder der Armbänder zeigen sie sich interessiert. Ein Großteil der

Patienten sieht jedoch ein, dass sie aufgrund ihrer Sehschwäche die große Uhr (42mm) benötigen. Die große Uhr wird im Gegenzug als zu groß beschrieben. Die meisten Patienten haben keine Probleme die Schrift zu lesen. Meist nehmen sie ihre Brille zur Hilfe, die im Krankenhaus natürlich immer griffbereit ist. Nur bei einem kleinen Teil kommt es vor, dass sie den Text auf dem Display auch mit Brille nicht lesen können. Das größte Problem ist jedoch die schon erwähnte Einschränkung der Feinmotorik. So können die Probanden die Uhr nicht wirklich aktiv bedienen, sondern reagieren nur auf die Vibration am Handgelenk. Dies führt zu einer Hilflosigkeit gegenüber der Technik der Uhr. Eine genaue Auflistung der Probleme findet sich in 5.1

5.2.3. Auswertung des Fragebogens zum Nutzungskontext

Der Fragebogen zum Nutzungskontext wurde im Laufe des Gesprächs oder im Nachgang des Thinking Alouds besprochen. Die Aussagen waren jedoch oft nicht erkennbar, also handelt es sich eher um eine Einschätzung, als um Antworten. Das Ergebnis ist in 5.1 graphisch dargestellt.

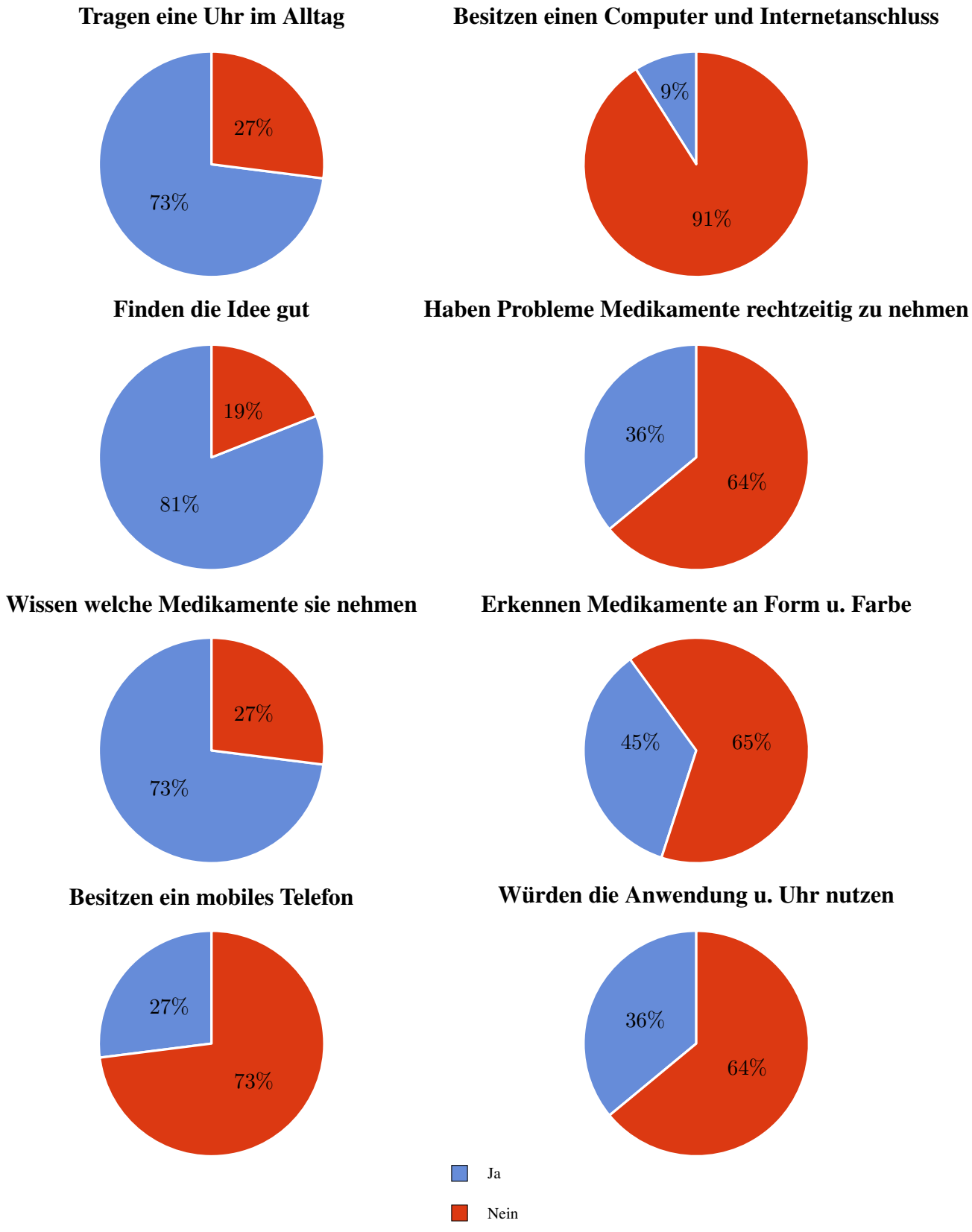
5.3. Probleme bei der Evaluation

Die Zielgruppe im Alter zwischen 70 und 85 erwies sich als schwierig. Eine bessere Vorbereitung auf diese Gruppe von Menschen wäre von Vorteil gewesen. Die Fragen, die geprüft werden, sollten auf eine minimale Anzahl reduziert werden. Es ist eher wichtig, eine persönliche Verbindung zum Befragten am Anfang des Gesprächs aufzubauen. Konfrontiert man sie gleich mit einer großen Anzahl an Fragen, verunsichert man sie. Auch sollte man hierbei geduldig sein und persönliche Themen aufgreifen. Die Befragten stehen anfangs einer Befragung eher skeptisch gegenüber. Diese Skepsis gilt es zu überwinden.

Tabelle 5.1.: Probleme und mögliche Lösungen des Smartwatch Prototyps

Problem	Auswirkungen	mögliche Lösung
fehlender Internetanschluss	keine Aktualisierungen des Medikamentenplans und keine Benachrichtigungen für den Arzt	Uhr oder Smartphone mit Mobilfunkverbindung ausstatten
reduzierte Feinmotorik der Probanden	Buttons werden nicht getroffen oder falsche Aktionen werden ausgelöst	große haptische Knöpfe in die Hardware integrieren, die gut drückbar sind
Vibration zu leicht	Vibration wird nicht gespürt und Medikament wird vergessen	Stärkere Vibration, lauten Ton dazu abspielen, starkes optisches Feedback (grelles Blinken)
Nur mit Brille lesbar	Benachrichtigung wird erkannt, bis jedoch die Brille im Haushalt gefunden, ist die Medikation schon vergessen oder die Uhr hat die Benachrichtigung zurückgestellt	Größere Schrift, die auch ohne Brille lesbar ist. Audiowiedergabe der Medikation
Armband schwer anlegbar	Uhr wird morgens nicht gerne angezogen, bleibt liegen und Patient bekommt keine Benachrichtigungen	Verzicht auf Standard Sport-Band und dafür Armbänder, die leicht anzuziehen sind, verwenden. Viele Patienten tragen dehnbare Bänder ohne Verschluss
Uhr ist unästhetisch	Patient trägt die Uhr nicht und wird so nicht an die Medikamente erinnert	Keine Standard Uhren kaufen, sondern den Patienten die Farbe und Form der Uhr und die Art des Armbandes aussuchen lassen

Abbildung 5.1.: Ergebnis des Fragebogens für den Nutzungskontext



6. Zusammenfassung und Ausblick

6.1. Zusammenfassung

Ziel der Arbeit war es, die Akzeptanz eines Systems zur Erinnerung an Medikamenteneinnahme zu testen. Als Methode wurde eine Patientenbefragung verwendet. Patienten stehen dem Tragen einer Uhr am Handgelenk positiv gegenüber. Ob sie jetzt digital oder analog ist, ist nebensächlich. Aus diesen Gründen ist die Uhr ein ideales Gerät zur Erinnerung von Medikamenten. Die Apple Watch mit ihrer touchscreen-basierten Navigation stellt für die getestete Zielgruppe eine große Herausforderung dar. Reduzierte feinmotorische Fähigkeiten erschweren den Patienten die Interaktion mit der Uhr. Auch Vibration und Geräuschwiedergabe sind zu schwach und können häufig nicht wahrgenommen werden. Das ästhetische Aussehen der Uhr ist nicht zu vernachlässigen, da eine Uhr oft als eine Art Schmuck getragen wird. So trägt die Farbe, Form und Größe zur Akzeptanz der Uhr bei. Findet der Patient die Uhr unschön/unmodisch, so wird er sie nicht tragen und schlimmstenfalls seine Medikamente vergessen.

Da Smartwatches noch keine große Verbreitung haben, sind auch die Werkzeuge zur Umsetzung von Anwendungen begrenzt. Es gibt keine Werkzeuge zur Erstellung interaktiver Prototypen, die auf der Uhr ausführbar sind. Ein Prototyp kann ausschließlich unter Verwendung von Code erstellt werden. So vermehrt sich der Aufwand für schnelles iteratives Vorgehen. Das watchOS SDK von Apple bietet eine relativ übersichtliche Schnittstelle. Die Schnittstellen ermöglichen es komplexere Anwendungen zu erstellen. Diese sind jedoch in manchen Anwendungsgebieten noch limitiert. Da es sich um die erste Geräte-Generation handelt, muss abgewartet werden mit welchen Leistungsverbesserungen weitere Generationen nachgerüstet werden, um die Uhr zu einem wirklich nützlichen Werkzeug zu machen. Das Potential zu einem guten Nutzen zeigt die erste Generation auf, jedoch ist sie an manchen Stellen (Geschwindigkeit, lange Wartezeiten) noch nicht ausgereift.

Apples neue Programmiersprache Swift, die zur Implementierung genutzt wurde, bietet Konzepte moderner Programmiersprachen. Diese Konzepte unterstützen den Entwickler meist schon zur Compilezeit. Dies führt zu einer frühen Fehlererkennung. Durch diese Spracheigenschaften, wie Optionals (3.4.2), die Sicherheit in der Modellierung geben, oder Closures und First Class Functions (3.4.2), die Konzepte funktionaler Programmierung bereitstellen, eignet sich Swift sehr gut als Lehrsprache. Dadurch, dass Swift unter einer Open Source Lizenz veröffentlicht wurde, kann man nun für mehrere Zielplattformen entwickeln.

6.2. Ausblicke

Mit den gewonnenen Ergebnissen kann dieses Projekt in unterschiedlichen Thematiken neu ausgerichtet werden.

Projektziele

Die Uhr als Hilfsmittel im Alltag zeigt gute Ansätze auf und ist in der Zukunft weiter zu verfolgen. Wichtig hierbei ist, dass die Uhr als solches bestehen bleibt und nicht durch einen großen unschönen Kasten am Handgelenk ersetzt wird, damit die Nutzer nicht die Lust am Tragen verlieren.

Zielgruppe

Die Zielgruppe sollte über mehr Erfahrung mit digitalen Geräten verfügen. Der Einsatz der Uhr bei Kindern und Jugendlichen im Alter ab 12 Jahren ist denkbar. Diese könnten bei ihrer Therapie unterstützt und durch die moderne Technik, mit der sie aufgewachsen sind, motiviert werden, Medikamente zeitgerecht einzunehmen. Auch Erwachsene, die den Umgang mit moderner Informationstechnologie gewohnt sind, jedoch Schwierigkeiten mit der regelmäßigen Einnahme von Medikamenten haben, würden sich als Zielgruppe eignen. Bei diesen Nutzern könnte man die Medikation mit mehr Informationen erweitern, um ihnen den Mehrwert und Grund der Medikation zu erläutern. Dies könnte ebenfalls zu einer erhöhten Motivation führen.

Technische Möglichkeiten

Nachdem die Defizite mit der Bedienbarkeit durch die Evaluation entdeckt wurden, ergab die Recherche, dass die watchOS Plattform über Funktionen der Accessibility verfügt [8]. Damit ist es möglich, Nutzungs-Erleichterungen für Menschen mit körperlichen Einschränkungen zu schaffen. So kann mit DynamicType, also einer dynamischen Schriftgröße in der App, die Lesbarkeit verbessert werden. Der Nutzer kann so seine eigene Schriftgröße wählen. Auch die Navigation zum Öffnen von Anwendungen wird mit Accessibility Einstellungen verbessert.

Eine Apple Watch Complication (siehe 3.6.3) wäre eine gute Erweiterung, um dem Nutzer ohne eine Interaktion zu zeigen, welche Medikamente zu nehmen sind. Sie würde auf der Uhr neben der Uhrzeit platziert werden. So erkennt der Nutzer, wenn er auf die Uhr sieht, nebenbei auch gleich den Stand der Medikamenteneinnahme.

Der Anschluss an das PITA Backend stellt keine große Herausforderung da. iOS und watchOS bieten gute Schnittstellen zur Integration des Systems. So kann die Uhr direkt

nach Bestätigung der Einnahme eine Nachricht an das System senden und somit den Arzt informieren.

Zum Ende dieser Arbeit kündigte Apple mit CareKit [11] ein interessantes Framework für den Einsatz mit Patienten an. Bei CareKit handelt es sich um ein OpenSource Framework, welches Entwicklern helfen soll, Gesundheitsdaten von Nutzer zu verarbeiten. Patienten sollen ihr Befinden und ihre Medikamente in solche Apps eintragen. Das Framework unterstützt dann die Weiterverarbeitung. Die Daten können auch sicher mit dem Arzt geteilt werden. Apple legt bei diesem Framework sehr viel Wert auf Datenschutz. Es ist jedoch abzuwarten, wie gut sich solch ein Framework für Anwendungen eignet.

Literaturverzeichnis

[1] Limesurvey. https://www.limesurvey.org.

[2] *The Swift Programming Language*. Apple, Inc., 2014.

[3] DALRYMPLE, M. Learn objective-c on the mac. 1–20.

[4] DVORAK, J. L. Moving wearables into the mainstream.

[5] GIT. Git. http://git-scm.com.

[6] GITHUB, I. Github. https://github.com.

[7] GMBH, U. I. D. Attrakdiff fragebogen. http://attrakdiff.de.

[8] INC., A. Accessibility - apple watch. https://www.apple.com/accessibility/watch/.

[9] INC., A. Developing for apple watch. https://developer.apple.com/library/ios/documentation/General/Conceptual/WatchKitProgrammingGuide.

[10] INC., A. Notification essentials. https://developer.apple.com/library/ios/documentation/General/Conceptual/WatchKitProgrammingGuide/BasicSupport.html.

[11] INC., A. Resarchkit and carekit. https://www.apple.com/researchkit/.

[12] INC., A. Sharing data on apple watch. https://developer.apple.com/library/ios/documentation/General/Conceptual/WatchKitProgrammingGuide/SharingData.html#//apple_ref/doc/uid/TP40014969-CH29-SW1.

[13] INC., A. Swift open source. https://swift.org.

[14] KREBS P, D. D. Health app use among us mobile phone owners: A national survey. *MIR mHealth uHealth* (2015).

[15] LAPUT, G., YANG, C., XIAO, R., SAMPLE, A., AND HARRISON, C. Em-sense: Touch recognition of uninstrumented, electrical and electromechanical objects. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (New York, NY, USA, 2015), UIST '15, ACM, pp. 157–166.

[16] LEE, S., KIM, Y., AHN, D., HA, R., LEE, K., AND CHA, H. Non-obstructive room-level locating system in home environments using activity fingerprints from smartwatch. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (New York, NY, USA, 2015), UbiComp '15, ACM, pp. 939–950.

[17] MUENSTERER, O. J., LACHER, M., ZOELLER, C., BRONSTEIN, M., AND KÜBLER, J. Google glass in pediatric surgery: An exploratory study. *In-*

ternational Journal of Surgery 12, 4 (2014), 281 – 289.

[18] NIELSEN, J. *Usability engineering*. Academic Pr., Boston, Mass. [u.a.], 1993.

[19] PIXELCUT. Paintcode. `http://www.paintcodeapp.com`.

[20] RICHES, G. Apple watch for developers.

[21] RITCHIE, R. Apple watch specs. `http://www.imore.com/apple-watch-specs`, 2015.

[22] SAILER, F., POBIRUCHIN, M., WIESNER, M., AND MEIXNER, G. An approach to improve medication adherence by smart watches, 2015.

[23] SCHMIDT, L. Repository des quellcode der arbeit zum ausführen in xcode. `https://github.com/lightsprint09/Medications-AppleWatch/tree/results`.

[24] STEFANIA PIZZA, BARRY BROWN, D. M. A. L., Ed. *Smartwatch in vivo* (2016).

[25] TAGHEUER. Tagheuer connected. `http://www.tagheuerconnected.com/product`.

[26] TENG, X.-F., ZHANG, Y.-T., POON, C., AND BONATO, P. Wearable medical systems for p-health. *Biomedical Engineering, IEEE Reviews in 1* (2008), 62–74.

[27] WELLS, G. The future of ios development: Evaluating the swift programming languag. Master's thesis, Claremont McKenna College, 2015.

A. Appendix

A.1. Fragebogen

1. Tragen Sie eine Uhr im Alltag?
   a) Ja b) Nein
2. Wären Sie bereit eine digitale Computeruhr zu tragen?
   a) Ja b) Nein
3. Finden Sie die Idee gut, von der Uhr an Ihre Medikamenteneinnahme erinnert zu werden?
   a) Ja b) Nein
4. Finden Sie die Idee gut, von der Uhr an ihre genauen Medikamente erinnert zu werden?
   a) Ja b) Nein
5. Wie viele Medikamente nehmen Sie täglich?
   a) Ein Medikament b) Mehr als drei c) Mehr als fünf d) Mehr als acht
6. Haben Sie Probleme, Ihre Medikamente rechtzeitig zu nehmen?
   a) Ja b) Nein
7. Wissen Sie welche Medikamente Sie nehmen?
   a) Ja b) Nein
8. Können Sie die Medikamente an Form und Farbe unterscheiden?
   a) Ja b) Nein
9. Haben Sie einen Internetzugang zu Hause?
   a) Ja b) Nein
10. Benutzen Sie einen Computer?
    a) Ja b) Nein
11. Benutzen Sie ein mobiles Telefon?

a) Ja b) Nein

12. Würden Sie die Erinnerung an die Medikamente mit der Uhr nutzen?

    a) Ja b) Nein

13. Welche Funktionen fehlen, die Sie als wichtig ersehen?

14. Gibt es Funktionen, die Sie für unwichtig halten?

Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum Unterschrift