Bachelorarbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Software Engineering

“Evaluierung einer multimodal bedienbaren und personalisierten Kalenderanwendung auf Basis des mobilen Dialogsystems des DFKI”

Patrick Teich
Matrikelnummer: 4387900
Patrick.Teich@fu-berlin.de

Gutachter: Prof. Dr. Lutz Prechelt
Zweitgutachterin: Prof. Dr. Claudia Müller-Birn

Betreuerin: Julia Schenk

Berlin, 14.03.2013

Zusammenfassung

Diese Arbeit beschreibt eine praktische Anwendung von Methoden der Usability-Evaluation zur zielgerichteten Verbesserung einer Kalender-Applikation auf Basis eines durch das DFKI entwickelten Frameworks. Dieses Framework dient der Entwicklung multimodaler mobiler Anwendungen.
Es wird der entwickelte Prototyp der Applikation beschrieben, welcher ein relativ frühes Entwicklungsstadium der Applikation darstellt, sowie die Methoden zur Evaluation dieses Prototypen und die daraus resultierenden Verbesserungen. Die Evaluation ist in 2 Iterationen mit zusammen 8 Nutzern abgelaufen. Sie soll Rückschlüsse auf die Usability des Frameworks des DFKI erlauben, die ebenfalls gezeigt werden.
Im Rahmen dieser Arbeit wurden insbesondere 3 Methoden zumindest teilweise verwendet: Usability-Tests, Heuristische Evaluation und der SUS-Fragebogen. Die praktische Anwendung aller 3 Methoden wird gezeigt.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

14.03.2013

Patrick Teich

Inhaltsverzeichnis

1 Einführung **2**
1.1 Hintergrund . . . 2
1.1.1 das *Framework* im Überblick . . . 3
1.1.2 die Applikation im Überblick . . . 4
1.2 Stand vor der Evaluation durch Nutzer . . . 6
1.2.1 Server Verbindung . . . 6
1.2.2 Authentifizierung . . . 6
1.2.3 Erinnerungsverarbeitung . . . 7
1.2.4 Sprachsteuerung . . . 7
1.2.5 Personalisierung . . . 8
1.2.6 Probleme und Schwächen . . . 8

2 Evaluationsprozess **11**
2.1 Ziele . . . 11
2.2 Methoden . . . 11
2.3 Iterationen . . . 14
2.3.1 1. Iteration . . . 14
2.3.2 Anpassung . . . 18
2.3.3 2. Iteration . . . 25

3 Auswertung **30**
3.1 Ergebnisse der *SUS*-Befragung . . . 30
3.1.1 Iteration 1 . . . 30
3.1.2 Iteration 2 . . . 31
3.2 Verbesserungen, übrige Schwächen . . . 31
3.3 Barrieren durch das *Framework* . . . 32
3.4 Bewertung des *Framework*s . . . 33

4 Fazit **34**

5 Anhang **36**
5.1 Definitionen verwendeter Begriffe . . . 36
5.2 Anforderungen an die Applikation bei der Entwicklung . . . 37
5.3 Die Kalenderschnittstelle . . . 39
5.4 Die CalDAV-Anfragen des Prototypen . . . 40
5.4.1 addEvent . . . 40
5.4.2 findEvents . . . 40
5.4.3 findEventsByReminders . . . 41
5.4.4 deleteEvent . . . 41
5.5 Bilder des Prototypen . . . 42
5.5.1 Vor der Anpassung . . . 42
5.5.2 Das Datums-Widget . . . 44
5.5.3 Das Popup - vorher - nachher . . . 45

5.5.4 Die Sprachmemoaufnahme - vorher - nachher 46
5.5.5 Einige Rückmeldungen der Applikation nach der Anpassung 46
5.5.6 Die Tagesübersicht 48
5.6 Die Szenarien der Studie 49
5.7 Fragebogen (SUS) und dessen Bewertung 50
5.8 Interviewfragen 51
5.9 Veränderungen nach der ersten Iteration 52

Zur Form

Sprache Ich habe mich bemüht ein Gemisch englischer und deutscher Begriffe zu vermeiden. Jedoch sind viele Fachbegriffe in Englisch und deren deutsche Varianten erschweren für mich häufig das Verständnis, sodass ich aus praktischen Gründen die englischen bevorzugt habe, nicht zuletzt, weil sie auch geläufiger sind. Es sind jedoch meist beide angegeben.

Formatierung Besondere technische Begriffe, wie z.B. die eben behandelten englischen Fachbegriffe, sind in einem Definitionsverzeichnis im Anhang näher erläutert. Im Text sind Begriffe, die dort zu finden sind, *kursiv* hervorgehoben.

1 Einführung

Ursprung Ursprung dieser Arbeit war eine Anstellung beim Deutschen Forschungsinstitut für Künstliche Intelligenz (DFKI), während der ich ein *Framework* nutzen sollte, das das DFKI bis heute (März 2013) noch entwickelt. Dieses *Framework* dient der Entwicklung multimodaler mobiler Applikationen. Meine Aufgabe bestand darin einen Kalender damit zu implementieren, um nachzuweisen, dass das *Framework* auch real verwendet werden kann.

Ziel Diese Arbeit untersucht, ob das *Framework* die Entwicklung einer benutzbaren Applikation im Sinne der *Benutzbarkeit* (auch *Usability*) wie sie unter 5.1 definiert ist, unterstützt oder behindert. Dazu dient eine *Usability*-Studie am Beispiel der von mir entwickelten Applikation.
Die Studie basiert auf *Nutzertest*s. Die *Nutzertest*s dienen dabei dem Auffinden von Schwachstellen, dem Optimieren des Aufbaus der Benutzerschnittstelle und letztlich der Feststellung des Nutzens des *Framework*s. Hierbei bedeutet Nutzen eine Eigenschaft des *Framework*s den Entwickler bei der Entwicklung nutzbarer Applikationen zu unterstützen. Diese Art von *Nutzertest*s wird induktiv bezeichnet ([1]) und sollte möglichst früh im Entwicklungsprozess stattfinden: “Bei einfachen Prototypen fällt es den Personen hingegen leichter, auch grundsätzliche Kritik anzubringen und nicht nur auf der Design-Ebene zu argumentieren” ([1], bei dem Vergleich zu weit entwickelten Prototypen). In meinem Fall handelt es sich um einen Prototypen, der alle geforderten Funktionen auf sehr einfache Weise bereit stellt.

Aufbau Diese Arbeit besteht aus 2 inhaltlichen Teilen, der Auswertung und einem Anhang. Der erste Teil behandelt den Stand der Applikation, bevor sie durch *Nutzertest*s evaluiert wurde. Der zweite Teil die Evaluation und Anpassung der Applikation. Der Anhang enthält einmal alle Definitionen spezieller und technischer Begriffe, die ich verwendet habe. Weiterhin enthält er konkrete Informationen über die Anforderungen, die ich während der Entwicklungsphase der Applikation zu beachten hatte und technische Details der Applikation selbst. Als drittes enthält der Anhang Bilder der Applikation vor nach der Anpassung in Folge meiner *Nutzertest*s, die Szenarien, den Fragebogen und die Interview-Fragen, die ich für die *Nutzertest*s verwendet habe, sowie darauf folgend die genauen Anpassungen, die ich an der Applikation vorgenommen habe.

1.1 Hintergrund

Die Basis der entwickelten Applikation, die wiederum die Basis der Studie darstellt, ist das *Framework* des DFKI. Dieses befindet sich bis heute (März 2013) noch in Entwicklung und konnte nicht alle Funktionen bieten, die anfangs für

meine Applikation geplant waren. In den folgenden Abschnitten werden Probleme, die das *Framework* verursacht hat, detailliert aufgezeigt. Zum großen Teil sind diese Probleme darauf zurückzuführen, dass das *Framework* mitten in der Entwicklung ist und lange Zeit, die ich es benutzte, am Anfang stand.
Direkt am Anfang wurde mir vom DFKI vorgegeben, ich solle auf Plattformunabhängigkeit achten. Aus diesem Grund, habe ich mich zusammen mit meiner Betreuerin beim DFKI für *CalDAV* entschieden, das zur Kommunikation mit dem Google Calendar dient. Und auch dieser hat einige Probleme verursacht, die ich aufführen werde.
Die Evaluation der entwickelten Applikation soll damit auch Rückschlüsse auf die *Nutzbarkeit* des *Framework*s ermöglichen. Da ich allerdings nur diese eine Applikation damit gebaut habe und das *Framework* während der Entwicklung nicht in einem konsistenten Zustand war, werden diese Rückschlüsse auf die *Nutzbarkeit* auch nur eingeschränkt aussagekräftig sein.

1.1.1 das *Framework* im Überblick

Das *Framework* selbst basiert auf Apache Cordova[1], um plattformunabhängige Applikationen entwickeln zu können. Applikationen werden damit in Javascript und HTML entwickelt. Darüber hinaus bietet das *Framework* eine *MVC*-Struktur und Schnittstellen zu Sprachsteuerung, Dialogverwaltung, Unterstützung mehrerer Dialogsprachen und einigen Funktionen, die ich nicht verwendet habe.

Sprachsteuerung Die Sprachbefehle werden in Form einer kontextfreien Grammatik angegeben, d.h., zur Nutzung der Spracheingabe muss die Grammatik erweitert werden. Die Spracheingaben werden dann anhand der Grammatik in eine Objektstruktur konvertiert, die dann auf Applikationsebene weiter verarbeitet werden kann.

Dialogverwaltung Das *Framework* verwendet 2 Zustandsautomaten (einer sollte für die grafischen Dialoge, einer für die sprachlichen verwendet werden) auf SCXML-Basis (siehe [2]), um die Definition von Benutzereingaben zu ermöglichen. D.h., auf jede Eingabe wird ein Event erzeugt, das zu Zustandswechseln führen kann. So ist zum Beispiel definiert, dass die Applikation im Zustand “mmig_state” beginnt und mit Betätigen des Login-*Button*s wird ein Event erzeugt, das zum Wechsel in den “logged_in” Zustand führt. In diesem sind dann wiederum für alle vorhandenen Bedienelemente Zustandsübergänge definiert.

Dialogsprachen Textbausteine werden in einem Wörterbuch für jede Sprache abgelegt und können über eine Übersetzungsfunktion nach ihrem Schlüssel verwendet werden. Die Dialogsprache kann dann dynamisch geändert werden.

[1] http://cordova.apache.org/

1.1.2 die Applikation im Überblick

Die Basis vor der Entwicklung Bevor ich mit der Entwicklung meiner Applikation begonnen habe, hatte ich einen kleinen Zustandsautomaten, einen Login-Bildschirm, das Layout, einen einfachen CalendarController, ein CalendarModel, ein UserModel (Begriffe aus *MVC* bezüglich der zwei Datenelemente in der Applikation: Nutzer(User) und Kalender(Calendar)) und einen "Termin erstellen"-Bildschirm mit einfachem HTML-Formular. Damit war es zumindest möglich die Daten für einen Termin mit Titel, Start und Ende einzugeben. Jede weitere Verarbeitung fehlte. Das Design der Bedienelemente basiert auf jQuery-Mobile[2], einem Webdesign-*Framework*. Zudem verwendete diese Basisapplikation auch schon das *Datums-Widget*[3], das immer noch Teil meiner Applikation ist. Zuletzt war noch eine Funktion in der Benutzerschnittstelle implementiert, um die Dialogsprache dynamisch ändern zu können. Ich habe diese allerdings entfernt, weil sie für die *Nutzertest*s unerheblich war und zudem die Sprache der Spracheingabe nicht geändert hatte. (Im Übrigen gibt es noch keine Unterstützung für mehrsprachige Grammatiken, sodass diese Funktion mehr Probleme gemacht hätte.) Für die Spracheingabe hat diese Basisapplikation einen Mikrofon-*Button*, der vor jeder Eingabe mit Sprache betätigt werden muss. Durch Vibration wird angezeigt, dass die Spracherkennung aktiviert wurde, nach einer kurzen Sprechpause wird dann wieder durch Vibration angezeigt, dass die Erkennung beendet wurde. Auch diese Funktion habe ich nie geändert.

Vorgaben für die Entwicklung Meine Aufgabe war es eine Kalenderapplikation auf Basis dieses *Framework*s zu erstellen. Die Vorgaben seitens des DFKI beschränkten sich auf folgende Funktionen:

- Termine eintragen (mit Titel, Start, Ende)
- Suchen von Terminen nach Datum
- Erinnerung an Termine
- Verwendung eines Web-Kalenders
- Implementierung von Sprachbefehlen wie "Trage Termin am ... ein", "Zeige Termin am", "vor", "zurück"

Etwas genauere Informationen befinden sich im Anhang unter 5.2.

Die entstandenen Funktionen Aus den Vorgaben hat sich die Applikation wie folgt ergeben:

[2] siehe http://jquerymobile.com/
[3] zu finden unter http://dev.jtsage.com/jQM-DateBox2/

1. Es lassen sich Termine mit Titel, Startdatum, Enddatum, beides mit Uhrzeit, einem Intervall vor Start des Termins, in dem an diesen erinnert werden soll, einem Verweis auf eine Person aus dem Adressbuch und einem Verweis auf eine *Sprachmemo* erstellen
2. Es lassen sich Termine auf Basis des Datums auf einen Tag genau finden
3. Es ist möglich durch Termine zu blättern
4. Die *Sprachmemo* lässt sich innerhalb der Applikation anhören
5. Als Verweis auf eine Person steht neben dem Termin ein Bild dieser aus dem Adressbuch, falls vorhanden, sonst ihr Name
6. Für nahezu jede Funktion lässt sich auch die Spracheingabe verwenden
7. Falls an einen Termin erinnert werden soll, erscheint einmalig eine Erinnerung
8. Alle Termine, an die erinnert wurde und die noch nicht beendet sind, erscheinen in einer Übersicht

Für die Terminverwaltung wird der Google Calender genutzt, der über ein Modul, das ich entwickelt habe, mittels *CalDAV* angesprochen wird. Dieses Modul (der CalendarClient) unterstützt dazu das Erstellen, Finden und Löschen eines Termins, sowie das Suchen von Terminen anhand ihrer Erinnerungen. Der CalendarClient ließe sich eigentlich ohne Änderung auch mit anderen CalDav unterstützenden Kalendern verwenden. Die Erläuterung dieser Einschränkung findet sich unter (1.2.6). Bei speziellen APIs müsste nur ein neuer spezieller CalendarClient geschrieben werden. Technische Details sind unter 5.3 zu finden.
Insgesamt besteht die Applikation nun aus 4 Bildschirmen, 1 Controller, 1 Model, mehreren Datenstrukturen für die Terminverwaltung, 1 Helper, einem Parser, der die Rückgaben der Spracheingabe weiter abstrahiert, einer Grammatik, dem CalendarClient, 1 Zustandsautomaten für die Bildschirmübergänge und Funktionen auf der Benutzerschnittstelle und 1 Zustandsautomaten für die Sprachbefehle.
Bis dahin musste eine Reihe von Problemen gelöst werden. Und zwar die Kommunikation mit dem Google Calendar, die Darstellung für den Nutzer, das Einsetzen der Sprachbefehle, der Umgang mit Spracheingaben, die auf vorhergehende Befehle verweisen, die Verknüpfung der Termine mit zusätzlichen Daten (wie Verknüpfungen zu Personen) und ein Fehler im *Framework* bei der Verarbeitung von Spracheingaben.

1.2 Stand vor der Evaluation durch Nutzer

Vor Beginn der Evaluation habe ich einen Prototypen fertig gestellt, der alle Funktionalitäten, die evaluiert werden sollten, hatte. Nach dem Prinzip möglichst früher *Nutzertest*s (nach [1]) wäre es zwar noch früher möglich gewesen, allerdings ergaben die Voraussetzungen der Applikationsentwicklung (der Arbeitsstelle), dass ich als erstes einen Prototypen fertig stelle.
Entsprechend habe ich primär auch nur auf Funktionalitäten und nicht auf die Benutzerschnittstelle geachtet. Die Evaluation soll dann Probleme in der Benutzerschnittstelle aufzeigen und gegebenenfalls auch noch technische Probleme aufdecken, die mir während der Entwicklung nicht aufgefallen sind.
Im folgenden werden die einzelnen Schritte bis zum Prototypen kurz erläutert.

1.2.1 Server Verbindung

Der Prototyp kommuniziert über *CalDAV* (siehe [3]) mit dem Google Calendar Server, um dort die Termine zu speichern. Die Idee dahinter *CalDAV* zu benutzen war, dass es ein Standard ist, der für unterschiedliche Kalenderdienste im Internet genutzt werden kann und die Applikation damit weniger anfällig für äußere Veränderungen ist, indem der Kalenderdienst ohne Änderung an der Applikation geändert werden kann, sofern er *CalDAV* unterstützt. Alternativ müssten spezialisierte Module eingesetzt werden, die die jeweilige API eines Dienstes verwenden. Die Entscheidung für *CalDAV* kam aber eher aus Experimentierfreude, als aus einem Verlangen nach Unabhängigkeit.
Während der Entwicklung des Prototyps zeigten sich dann auch wenige Male die Nachteile der Entscheidung Google Calendar mit *CalDAV* zu verwenden. Denn Google hat das Protokoll einerseits nicht vollständig implementiert und andererseits mindestens einmal das Verhalten des Calendar Servers verändert, sodass der Prototyp überraschend nicht mehr funktionierte.
Anfangs bestand das Problem jedoch darin überhaupt die URL finden, um Schreibrechte zum Kalenderdienst zu bekommen, denn Googles Dokumentation ist im Bereich *CalDAV* völlig unzureichend. Inzwischen steht sie jedoch unter [4].
Die gesamte Kommunikation mit dem Server beschränkt sich dann nur noch auf einfache AJAX-Aufrufe mit HTTP Authentifizierung. Diese Aufrufe werden in der konkreten Implementierung der Kalenderschnittstelle (siehe im Anhang unter 5.3) für jeweils speziellen *CalDAV*-Anfragen verwendet (unter 5.4).

1.2.2 Authentifizierung

Der Prototyp authentifiziert sich gegenüber Google über “HTTP Basic Authentication” (nach [5]), bei welcher Nutzername und Passwort einfach in Klartext in Base64-Kodierung an den Server gesendet werden. Dafür unterstützt Google jedoch ausschließlich https-Verbindungen, um die Sicherheit zu gewährleisten

(siehe [6]). Auf der anderen Seite stehen Nutzername und Passwort jedoch noch hartkodiert im Quellcode. Aber es handelt sich auch nur um einen Prototypen. Andere Authentifizierungsmethoden werden von Google nicht unterstützt (siehe [4]), also auch vom Prototypen nicht.

1.2.3 Erinnerungsverarbeitung

Nachdem die Verbindung zum Server und eine funktionierende Kommunikation sichergestellt waren, sowie Termine von der Applikation eingetragen und gelesen werden konnten, musste die Applikation auch Erinnerungen unterstützen. Dazu bedient sie sich einer einfachen Technik: Sie fragt den Server periodisch nach neuen Erinnerungen an Termine ab (5.1) und sortiert sie in eine listenartige Struktur. *Polling* ist hier notwendig, da die *CalDAV*-Spezifikation keine Methoden vorsieht, mit denen der Server den Client benachrichtigen kann. Jedoch existieren Vorschläge *CalDAV* um solche Funktionalitäten zu erweitern [4]. Allerdings müsste die Applikation an den Kalender Server die Anforderung stellen solche Benachrichtigungen zu unterstützen, wenn sie auf *Polling* verzichten soll.
Nach der Abfrage wird für die nächste Erinnerung ein Javascript-Timeout registriert, das die Erinnerungsanzeige auslöst. Für diese Anzeige wird ein einfaches *Popup* geöffnet. Außerdem existiert noch eine Übersicht über alle Termine, an die erinnert wurde und die noch nicht vorbei sind.

1.2.4 Sprachsteuerung

Wie die Sprachsteuerung zu implementieren ist, gibt das *Framework* vor. Zum Auslösen der Spracheingabe existiert eine Funktion, die das Mikrofon aktiviert und nach erfolgter Eingabe wieder deaktiviert und dann die Erkennung startet. Wenn die Eingabe fertig bearbeitet ist, wird sie über eine Callback-Funktion als Text zurückgegeben und dann über den Grammatik-Parser weiter zu strukturierten Daten verarbeitet. Für den Grammatik-Parser muss eine Definition der Grammatik vorliegen. Die Grammatik ist kontextfrei. Ich möchte hier anmerken, dass natürliche Sprachen existieren, die nicht kontextfrei sind. Siehe dazu [7], in dem bewiesen wird, dass Schweizerdeutsch (Schwizerdütsch) nicht kontextfrei ist. Das bedeutet zwar nicht, dass es unmöglich ist unter dieser Voraussetzung eine praktikable Sprachsteuerung zu bauen, aber zeigt, dass es Einschränkungen gibt. Diese könnten relevant werden.
Die Verarbeitung der Spracheingabe geschieht im Moment auf einem Server eines speziellen Dienstleisters. Dies ließe sich aber zu jeder Zeit ändern.

[4] siehe http://svn.calendarserver.org/repository/calendarserver/CalendarServer/trunk/doc/Extensions/caldav-notifications.txt

1.2.5 Personalisierung

Als Besonderheit dieser Applikation gelten die Verknüpfungen, die den Terminen angefügt werden können. So können bisher *Sprachmemo*s und Personen aus dem Adressbuch mit einem Termin verbunden werden. Diese Verknüpfungen werden einfach in die Beschreibung des Termins ("DESCRIPTION" nach [3]) kodiert. Das bedeutet zwar, dass sie über andere Schnittstellen zum Kalender geändert werden könnten, stellt aber zumindest sicher, dass sie von allen Kalenderdiensten unterstützt werden. *Sprachmemo*s werden zudem auch nur auf dem Gerät selbst gespeichert, könnten aber natürlich auch bei einem weiteren Dienst über das Internet gespeichert werden. Personen werden im Moment nur anhand ihres Bilds oder ihres Namens aus dem Adressbuch angezeigt. Weitergehende Funktionen (wie z.B. das Öffnen des Kontakts im Adressbuch) hat das *Framework* noch nicht zugelassen.

Anforderungen an den Kalenderdienst sind Terminobjekte, die Start, Ende, Titel und Beschreibung und bisher mindestens eine Erinnerung pro Termin speichern. Alle diese Daten müssen auch abgerufen werden können.

1.2.6 Probleme und Schwächen

Als letzter Punkt in der Beschreibung des Prototypen folgt nun eine Liste von Problemen und Schwächen, die mir aufgefallen sind und die größtenteils in den *Nutzertest*s umgangen werden sollen. Dadurch vermeide ich, dass den Nutzern Dinge auffallen, die ich schon weiß und dass sie auf Probleme stoßen, die mir bekannt sind, anstatt unbekannte zu finden. Zudem handelt es sich häufig um technische Probleme bzw. Schwächen, die natürlich noch bis zur Marktreife zumindest zum größten Teil gelöst werden müssen, aber für die *Nutzertest*s nicht unbedingt relevant sind.

der Prototyp ist nicht vollständig mehrsprachig Mehrsprachigkeit wird zwar vom *Framework* für die Benutzerschnittstelle unterstützt, allerdings ließ sich die Sprache der Spracheingabe nicht dynamisch ändern, sodass ich nach einiger Zeit vorerst auf Mehrsprachigkeit verzichtet habe

bisher nur *CalDAV* über den Google Calendar getestet Da Google's *CalDAV*-Implementierung nicht vollständig ist, kann es sein, dass der Prototyp andere *CalDAV*-Implementierungen nicht unterstützt. Zudem sind alle Einstellung dazu noch hartkodiert.

CalendarClient liest XML Ausgaben von Server nur teilweise Die Rückgabe des Google Calendars wird von der Applikation nur soweit gelesen wie es notwendig ist, um alle Funktionalitäten zu implementieren. Sie achtet nicht darauf, welche Bezeichner die Namensräume haben und verlässt sich stattdessen darauf, dass der Google Server immer gleichartige

XML-Ausgaben macht. Also die Form der XML-Ausgabe nicht variiert. Dies hat auch bisher ausgereicht, würde jedoch bei Verwendung anderer *CalDAV*-Dienste zu Problemen führen können.

Kontaktverknüpfungen ohne besondere Funktion Wie erwähnt, war es noch nicht möglich ohne weiteres die Verknüpfungen mit Kontakten aus dem Adressbuch sinnvoll zu nutzen. Der Aufwand etwas nutzbares zu implementieren war zur Entwicklungszeit noch zu hoch.

Das *Framework* läuft nur auf Android Obwohl das *Framework* plattformunabhängig sein sollte, enthält es noch plattformspezifische Funktionen. Selbst Apache Cordova funktioniert nicht vollständig auf allen Plattformen. Diese Schwäche lässt sich durch mich allein nicht entfernen. Mit der Zeit sollte sie allerdings verschwinden.

lange Ladezeit Es dauert eine ganze Weile bis die Applikation geladen ist. Die Ursache hierfür liegt im *Framework* und wurde inzwischen entfernt.

die Applikation läuft nur langsam Dieses Problem hat meine gesamte Arbeit besonders behindert. Die Applikation führt ständig zu etwa 1-3% CPU-Last auf dem Gerät. Die Ursache hierfür liegt zumindest nicht in meiner Applikation, da das *Framework* auch ohne irgendeine Applikation dieses Phänomen aufweist. Solange dieses Problem besteht, kann mich zumindest meine Applikation nicht zufrieden stellen (wie auch viele Nutzer nicht).

keine weiteren Eingabemethoden implementiert Das *Framework* soll alle mit einem typischen Smartphone möglichen Eingabemethoden implementieren. Allerdings konnte ich zur Entwicklungszeit nur die normale Touch-Bedienung und die Sprachsteuerung verwenden. Gestensteuerung (über den gyroskopischen Sensor) oder Multitouch-Bedienung waren noch nicht so implementiert, dass sie mit angemessenem Aufwand hätten genutzt werden können.

kein brauchbares Debugging Als Entwickler war es sehr schwierig zu verfolgen was das *Framework* tut bzw. woher Defekte kamen und ungewolltes Verhalten zurückzuverfolgen. Insbesondere, weil das *Framework* selbst noch nicht hinreichend defektarm war.

Memos werden auf Gerät gespeichert Der Einfachheit halber werden *Sprachmemo*s noch auf dem Gerät gespeichert und nicht von der Applikation verwaltet. Die Verwaltung hat das *Framework* bei meinen Versuchen nicht zugelassen (wie das Löschen von *Sprachmemo*s). Um das genannte Problem zu lösen, reicht jedoch ein externer Dienstanbieter, der die Dateien speichert

wiederkehrende Termine werden nicht unterstützt Laut der *CalDAV*-Spezifikation gibt es die Möglichkeit Termine zu erzeugen, die periodisch wieder kehren (in [3] "recurring events" genannt). Der Google Calendar hat für diese Termine die falschen Start- und Enddaten zurückgegeben, darum werden sie vom Prototypen ignoriert.

Die Nutzeroberfläche ist nicht auf Nutzer optimiert Ich habe während der Entwicklung keinen Wert auf die Nutzeroberfläche gelegt bis alles funktioniert hat. Die Evaluation soll genau dieses Problem lösen.

2 Evaluationsprozess

Nachdem nun der Prototyp für die erste Testphase beschrieben ist, folgt die Evaluation selbst. Dazu folgen in diesem Abschnitt die Ziele der Evaluation, die verwendeten Methoden, das Vorgehen bei den beiden Iterationen und dann die Ergebnisse aus diesen Iterationen.
Aus Zeitgründen waren nur 2 Iterationen und damit eine Anpassung der Applikation möglich. Außerdem sind auch einige lösbare Probleme übrig geblieben wie im vorhergehenden Abschnitt beschrieben.
Mit der Evaluation beginnt die gezielte Verbesserung der Applikation mit Hinsicht auf den potenziellen Nutzer. D.h., unter Verwendung der Methoden der "*Usability-Evaluation*" (nach [1]), habe ich versucht in dieser frühen Entwicklungsphase die Qualität der Applikation gezielt zu verbessern. Also, *Usability*-Probleme und technische Probleme zu identifizieren und auszuräumen.

> "«Evaluation» allgemein bezeichnet eine systematische und möglichst objektive Bewertung eines geplanten, laufenden oder abgeschlossenen Projekts [...] Die formative Evaluation wird vorgenommen, um die Schnittstelle im Rahmen eines iterativen Designprozesses zu verbessern." [1]

Die Studie fand unter folgender Prämisse statt:
Diese Applikation befindet sich in einem frühen Entwicklungsstadium, enthält aber die meisten Funktionalitäten. Sie soll, wenn möglich, einen Zustand der Marktreife erreichen, also generelle Anforderungen von Nutzern an die Funktionalitäten und die *Nutzbarkeit* dieser Applikation erfüllen.

2.1 Ziele

Das ureigene Ziel für mich als Softwareentwickler ist es, Applikationen zu entwickeln, die für den potenziellen Nutzer "brauchbar" sind, also ihn dabei unterstützen effektiv und effizient zu tun, was er möchte und ihn zufrieden stellen. Die so reagieren wie er erwartet und die das tun, das er von ihnen im Rahmen der Möglichkeiten verlangt.
Das Ziel für diese Applikation war es, zu zeigen, dass es möglich ist, mit diesem *Framework* brauchbare Applikationen zu entwickeln.
Und das Ziel dieser Arbeit ist, vom Stand des Prototypen ausgehend, die *Nutzbarkeit* dieser Applikation zu verbessern und damit auch zu zeigen, dass die Applikation brauchbar ist bzw. werden kann, also brauchbare Applikationen mit dem *Framework* machbar sind.

2.2 Methoden

Die Methode, die ich zur Erreichung der Ziele hauptsächlich verwendet habe, ist der induktive *Nutzertest* zum Finden von Schwachstellen mit dem Proto-

typen in der Umgebung von Nutzern (siehe [1]): “Die Testaufgaben müssen außerdem dem typischen Aufgabenkontext des zukünftigen Endbenutzers entsprechen, da die Beurteilung sonst nicht auf dieses Einsatzfeld übertragen werden kann.” “working partial system:funktionierende Version des Systems, die aber noch nicht die komplette Funktionalität umfasst (Rosson Carroll, 2002)” ([1]).
Die Studie bestand aus 2 mal 4 Tests, die sich aus folgenden 3 Elementen zusammensetzten.

Nutzertests Vorgegebene Szenarien, die eine typische Anwendung der Applikation vorgeben und den Nutzer auf alle zu testenden Funktionalitäten hinweist (nach [1], orientiert an Beispielen aus [8] und [9]). Unter Zuhilfenahme der “think-aloud”-Methode (“Methode des lauten Denkens” in [1]), bei der Nutzer ihre Gedanken während der Benutzung artikulieren, lassen sich insbesondere Schwächen in der Intuitivität finden. Dazu habe ich die Testsitzung jeweils per Mikrofon aufgenommen.

***SUS* als Zufriedenheitsmaß** Das “System Usability Scale” (*SUS*) besteht aus 10 Aussagen und sollen als Maß der Zufriedenheit eines Nutzers dienen. Es handelt sich bei *SUS* um ein subjektives Maß der *Usability*. Die 10 Aussagen stehen unter 5.7 in einer deutschen Übersetzung und wie aus den Antworten das Maß bestimmt wird, steht unter 3.1. Die Aussagen werden vom Testnutzer nach Ausführung aller Szenarien bewertet. Die Anforderung für *SUS* war, dass insbesondere im industriellen Kontext die allgemeine *Usability* eines Systems mit relativ geringem Aufwand mit seinen Vorgängerversionen oder Konkurrenzprodukten verglichen werden kann. Die Auswahl der 10 *SUS*-Aussagen geschah auf Basis zweier Systeme. Eines wurde vorab als “sehr einfach zu nutzen”, das andere als unmöglich benutzbar bewertet. Mit Hilfe dieser zwei Systeme wurden aus 50 Aussagen, die die *Usability* betreffen, diejenigen Aussagen gewählt, die durch 20 Testpersonen die extremsten Bewertungen erhielten (siehe [10]). *SUS* korreliert laut [10] zudem mit anderen subjektiven *Usability*-Maßen wie z.B. SUMI, auf das ich hier nicht näher eingehen werde.

Fragen Zuletzt folgen einige Fragen in Form eines Interviews zu den Dingen, die für den Nutzer während des Tests besonders schwierig bzw. ungewöhnlich waren. Und zusätzlich was ihm an dieser Applikation fehlt.

Die Nutzerauswahl basierte auf einem einfachen Profil. Ein Nutzer sollte ein Smartphone besitzen und verwenden, sowie eine Kalenderapplikation oder eine vergleichbare Terminverwaltung nutzen und vor allem, die Applikation nicht kennen, sodass ich besonders hochwertige Ergebnisse im Bereich der Intuitivität erhalten kann, die ich jedoch nicht direkt gemessen habe. Die Altersgruppen haben sich auf 18-24 und 25-34 verteilt. Diese Altersgruppen habe ich aus [11] übernommen. Die Geschlechtsverteilung war in beiden Iterationen 3 zu 1. Da

es bei dieser Größenordnung allerdings unmöglich ist präzise quantitative Ergebnisse zu erhalten, habe ich mich auf die qualitativen konzentriert wie bei der formativen Evaluation üblich. Die *SUS*-Aussagen sollten dann als objektiver Indikator für die Zufriedenheit dienen. Da ich jedoch nur sehr wenige Nutzer hatte, sind die sich ergebenden Zahlen wenig aussagekräftig. Zumindest sollten sie aber eine Tendenz in der Qualität der Applikation zeigen.
Es sind noch viele weitere Methoden zur gezielten Steigerung der *Nutzbarkeit* bekannt. Insbesondere einige Heuristiken (siehe dazu [1]), die mir bei der Entwicklung sehr geholfen hätten, mir zu jener Zeit aber noch unbekannt waren. Dafür kamen diese Heuristiken in der Phase der Beseitigung gefundener Probleme zum Einsatz. Die Idee dahinter ist, dass *Usability*-Probleme sich in eine geringe Anzahl Kategorien (Heuristiken) einteilen lässt. Diese Heuristiken geben grob vor, wie sich ein System den Nutzer gegenüber verhalten soll. Und, wenn ich ein Problem gefunden habe, das aus Verhalten entsteht, das einer solchen Heuristik widerspricht, so habe ich es als schwerwiegender eingestuft und zugleich eine Vorgabe zur Lösung erhalten.
Die Heuristiken sind wie folgt bezeichnet: (für die Beschreibung, siehe [1])

Nr.	Heuristik
1	Sichtbarkeit des Systemstatus
2	Übereinstimmung zwischen System und realer Welt
3	Benutzerkontrolle und Freiheit
4	Konsistenz und Standards
5	Fehler vermeiden
6	Erkennen vor Erinnern
7	Flexibilität und effiziente Nutzung
8	Ästethisches und minimalistisches Design
9	Unterstützung beim Erkennen, Verstehen und Bearbeiten von Fehlern
10	Hilfe und Dokumentation

Der Ablauf der Studie war zusammenfassend die 1. Iteration Tests durchzuführen, die Probleme zu identifizieren, von diesen möglichst schwerwiegende zu beseitigen, die 2. Iteration Tests durchzuführen und zu identifizieren, was sich verbessert hat, welche neuen Schwächen sich ergeben haben und was noch fehlt.

2.3 Iterationen

Wie bereits beschrieben, bestand jede Iteration aus 4 Tests mit insgesamt 8 möglichst unterschiedlichen Nutzern. Die Unterschiedlichkeit habe ich nach eigenem Ermessen und formlos bestimmt, da es sich bei jedem Nutzer um einen mir recht gut bis sehr gut bekannten Menschen handelte. Dies ist wichtig, um möglichst viele unterschiedliche Probleme zu finden ([1]). Nach [12] reichen 5 Testnutzer aus, um einen Großteil der vorhandenen Probleme zu finden. Allerdings ist diese Aussage mit großer statistischer Abweichung behaftet. Ich vermute, dass dies davon abhängt, wie unterschiedlich die Testpersonen sind.

2.3.1 1. Iteration

Vorbereitung Für die 1. Iteration hatte die Applikation den in Abschnitt 1.1.2 beschriebenen Zustand. Ich selbst war mit diesem Stand insbesondere im Bezug auf die Reaktionszeit und das Laufzeitverhalten der Applikation unzufrieden. Leider ist mir erst viel später die Ursache dafür aufgefallen.
In Vorbereitung auf diese Iteration habe ich innerhalb der Applikation einen *Logger* für entscheidende Stellen eingesetzt, um entscheiden zu können, ob Szenarien fehlgeschlagen sind. Ich habe mir die Szenarien anhand der Funktionalitäten der Applikation sowie in Anlehnung an Szenarien aus meinen Quellen ([8], [9]) überlegt.

Ergebnis Mit dieser Iteration habe ich etwa 42 *Usability*-Probleme gefunden. Die genaue Anzahl der Probleme hängt davon ab, ob ein fehlender Sprachbefehl auch als ein ganzes Problem gezählt wird. Die genauen Ergebnisse sind in folgender Tabelle festgehalten:

Schl. Schlüssel des Problems

Anzahl der Auftritte des Problems

Fat. *Fatalität* des Problems

Tabelle 1: gefundene Probleme aus der ersten Iteration

Schl.	Problem	#	Fat.	Kommentar
Bed1	Nutzer erwartet Übersicht über Termine (alle oder für Zeitraum)	4	4	durch Markteinfluss von 3 erhöht
Bed2	Was der Erinnerungs-*Button* machen soll, kann der Nutzer nicht erkennen	4	1	Nach der ersten Benutzung ist klar, wozu der *Button* gut ist
Bed3	Die Applikation reagiert sehr langsam	4	4	
Bed4	Beim Abspielen einer Memo sollte Stoppen oder Spulen möglich sein	1	2	durch Markteinfluss von 1 erhöht

Bed5	Der Mikrofon-*Button* könnte auch zum Aufnehmen einer *Sprachmemo* verwendet werden	1	0	kein *Usability*-Problem, sondern eine Funktionalität
Ein1	Nutzer erwartet einen neuen Dialog beim Verknüpfen eines Kontakts, bekommt aber nur *Select-Element*	1	1	Markteinfluss: von 0 erhöht
Ein2	das *Select-Element* sollte nach Benutzung nicht mehr farblich hinterlegt sein	1	1	
Ein3	beim Aufnehmen einer *Sprachmemo* wird ein Dialog erwartet	1	2	Markteinfluss: von 0 erhöht
Ein4	die Rückmeldung beim Speichern der *Sprachmemo* ist leicht kryptisch	1	2	der Nutzer sollte die Anwendung immer leicht verstehen können
Ein5	Es sollte erkennbar sein, wenn eine *Sprachmemo* angefügt wurde; sie änderbar sein.	2	3	Heuristik 1
Ein6	Bei Spracheingabe sollte der Titel des Termins nicht versehentlich geändert werden können	2	3	Heuristiken 5 und 3
Ein7	Die Zeitzone des Datums ist nicht ersichtlich	1	1	
Ein8	Nach Aufnehmen einer *Sprachmemo* kann die Fußzeile verschwinden	1	2	
Ein9	Dass noch Bedienelemente unterhalb von “Speichern” stehen, irritiert	1	1	
Ein10	“Network Access Failed”-Meldung bei fehlender Netzwerkverbindung und damit Datenverlust	1	4	Heuristik 9
Ein11	Nutzer möchte auch keine Erinnerung einstellen können	1	0	kein Problem, sondern Funktionalität
TAb1	Nutzer möchte alle Termine an einem Tag auf einmal sehen	1	3	Markteinfluss: von 2 erhöht
TAb2	Nutzer kann nicht erkennen, was *Buttons* bedeuten	4	3	
TAb3	Das Eingabefeld sollte das Datum des angezeigten Tags enthalten	2	2	Heuristik 1
TAb4	Die Meldung “keine Termine” sollte auch das Datum dazu enthalten	1	2	Heuristik 6
TAb5	Nutzer erkennt nicht, dass es noch weitere Termine an einem Tag gibt	1	2	Heuristik 6

TAb6	Nutzer möchte über Spracheingabe inhaltlich nach Terminen suchen können	1	0	nur eine Funktionalität
Not1	Das Erinnerungs*popup* sollte die Daten eines Termins enthalten	3	3	Heuristik 6
Not2	Der Schließen-*Button* des *Popup*s scheint nicht richtig zu funktionieren	4	2	
Not3	Die *Erinnerungsübersicht* sollte chronologisch sein	2	3	Heuristik 4
Not4	Die *Erinnerungsübersicht* sollte Überschneidungen von Terminen darstellen	1	0	kein *Usability* Problem, eher Funktionalität
Not5	Die *Erinnerungsübersicht* sollte über den Erinnerungs-*Button* verlassen werden können	1	1	Das bisherige Verhalten irritiert, da der Bildschirm erneut angezeigt wird.
Not6	Nutzer möchte durch Klicken auf die Erinnerungen in der Terminübersicht auf den entsprechenden Termin gelangen	2	2	Heuristik 6
SSt1	Anzeige des erkannten Texts sollte unten oder in einem kleinen *Popup* erfolgen	1	1	diese Anzeige ist eigentlich zu Debug-Zwecken eingeführt worden, kann aber praktisch werden
2SSt2	Spracherkennung hängt manchmal	3	4	Macht Bedienung unangenehm und Heuristik 1
SSt3	Bei Spracheingaben sollten die Rückmeldungen von allein schließen oder per Sprachsteuerung	1	2	macht Sprachsteuerung sonst wenig sinnvoll
SSt4	“Tracking-Mode” für Sprachsteuerung, also, dass sie aktiv bleibt ohne den Mikrofon-*Button* betätigen zu müssen	1	2	würde Sprachsteuerung wesentlich mehr Sinn geben
SSt5	Es ist dem Nutzer nicht bewusst, dass er nach Drücken des Mikrofon-*Button*s schon etwas sagen muss	1	1	
Wid1	Das *Datums-Widget* behindert sehr stark und führt z.B. zu Fehleingaben	4	4	führt zu vielen Fehlschlägen
Wid2	Das *Datums-Widget* hat einen Anzeigefehler bei Umlauten	1	1	
Wid3	Statt “Datum einstellen” besser “OK” als Bestätigung verwenden	1	0	hat keine Probleme verursacht

Spr1	“Erstelle einen Termin am 15.02 von 16 bis 19:30 Uhr, Titel: Abschlussfeier”	1	1	
Spr2	Nutzer möchte mehrere Sprachbefehle durch “und” verknüpfen	1	1	
Spr3	“Erinnere mich bitte 6 Stunden vorher daran”; “erinnern 6 Stunden”; “6 Stunden vorher erinnern”	1	1	
Spr4	“Termin erstellen am 15. Februar von 16 Uhr bis 19:30Uhr, Abschlussfeier”	1	1	
Spr5	“...,dankeschön”	1	1	
Spr6	Die Erkennung funktioniert teilweise miserabel	3	4	
Spr7	Es fehlt eine Rückmeldung, falls eine Anweisung unbekannt war	ich	2	Heuristiken 1 und 9

Die zwei Hauptprobleme während dieser Iteration waren das schlechte Laufzeitverhalten und daraus resultierende schlechte Spracherkennung. Ich hatte diese kurz vor dem ersten Test unwissentlich selbst in dieser Schwere hervorgerufen, indem ich eine zu aufwändige Animation in die Benutzerschnittstelle eingefügt hatte. Außerdem habe ich zu Gunsten der Einfachheit bei der Entwicklung in der Benutzerschnittstelle auf Darstellungen verzichtet, die in vergleichbaren Applikationen verwendet werden. Also auf eine Kalenderansicht bzw. eine *Agenda-Ansicht*. Beides wurde von den Testnutzern während der Befragung nach dem Test als fehlend angegeben.

2.3.2 Anpassung

Bewertung der Probleme - *Fatalität* Zur Bewertung der Probleme habe ich folgende Punkte nach [1] berücksichtigt:

Frequenz des Auftretens also einmalig bis laufend

Einfluss auf die Arbeitsabläufe also Nichtigkeit bis schwerwiegende Störung

Persistenz des Auftretens also zufälliges bis regelmäßiges Auftreten

Markteinfluss also, ob ein insgesamt ungünstiger Eindruck entsteht

Diese Kriterien sind wenig formal definiert, damit sind auch die letztlich gemachten Bewertungen der Probleme subjektiv, also durch mich selbst geschätzt.

Anpassung der Applikation Vor der Anpassung habe ich für jedes Problem eine oder mehrere Ideen gesammelt, um sie zu lösen. Für die Anpassung habe ich zuerst 14 der Probleme mit höchster *Fatalität*, die ich voraussichtlich in der verfügbaren Zeit hätte lösen können, ausgewählt. Am Ende habe ich von diesen zwar nicht jedes gelöst, zusätzlich aber noch kleinere Änderungen gemacht, die andere Probleme gelöst haben, da an manchen Tagen meine Zeit nicht mehr für die ausgewählten Probleme gereicht hätte, jedoch um kleinere Änderungen im Bezug auf andere Probleme zu machen. Dabei bin ich so vorgegangen, dass ich bei technischen Problemen die Applikation logisch so geändert habe, dass diese nicht mehr auftreten. Bei den häufigeren Problemen in der Benutzerschnittstelle, habe ich die angesprochenen Heuristiken verwendet. Die Entwurfsprinzipien für Android [5] oder von Apple [6] waren für meine Probleme zu abstrakt. Nur bei der Behandlung eines Netzwerk-Fehlers (siehe Problem Ein10) stellten die Android-Entwurfsprinzipien eine Vorlage dar.
Es folgt eine Übersicht über alle Lösungsideen und darauf die zur Lösung priorisierten Probleme mit Erläuterung zur Priorität:

Tabelle 2: Die Lösungsideen und ihre Erläuterungen

Schlüssel	Lösungsidee	Erläuterung der Lösung
Bed1	Führe eine Kalenderansicht oder *Agenda-Ansicht* ein	Solche Ansichten werden einerseits in ähnlichen Applikationen eingesetzt und wurden zweitens darum auch von den Testnutzern gefordert

[5] http://developer.android.com/design/get-started/principles.html
[6] http://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/AppDesign/AppDesign.html

Bed2	Führe Hilfe-Funktion ein (z.B. eine Einführung in Bedienelemente) oder konsultiere einen Designer	Dieser *Button* ist offensichtlich nicht intuitiv. Er ist nicht beschriftet sondern nur mit einem Bild versehen. Diesem Bild hinreichende Aussagekraft zu verleihen ist eine Design-Aufgabe. Und sollte es nicht möglich sein, ein intuitives Bild zu erzeugen, so kann nur noch Beschriftung z.B. in Form einer Hilfe-Funktion helfen.
Bed3	höchstwahrscheinlich Problem des *Framework*s (hat sich nachträglich als zum großen Teil mein Fehler herausgestellt)	(Der Fehler lag zum Teil in einer aufwändigen Animation, der andere Teil wird noch durch das *Framework* verursacht.)
Bed4	Füge einen Stopp-*Button* ein, ähnlich wie er bei der Aufnahme verwendet wird	technisches Problem: die Funktion muss nur kopiert werden; Spulen wäre unter Verwendung eines neuen Elements implementierbar
Bed5	eine Hilfe-Funktion	einfache Abbildungen können dieses Missverständnis nicht verhindern, möglicherweise liegt die Ursache aber auch beim *Sprachmemo*-Aufnahmeelement; das explizite Angeben der Funktionalität über Hilfe würde das Problem allerdings vollständig lösen
Ein1	Suche nach einem Standarddialog für solche Listenauswahlen	Ich habe keinen Entwurf für einen solchen Dialog; die Einbindung anderer Applikationen wie z.B. den Kontakten über Cordova ist ebenfalls nicht möglich
Ein2	Suche nach Ursache, ggf. den Fokus des Elements deaktivieren	technisches Problem
Ein3	Versuche den Standarddialog des Betriebssystems, den Cordova anbindet; allerdings ist dies schon einmal fehlgeschlagen	Aus anderen Applikationen ist bekannt, dass sie einen Dialog zur Sprachaufnahme (z.B. von Memos) verwenden, der vom Betriebssystem angeboten wird. Eine Anbindung dieses Dialogs besteht über Cordova. Die Verwendung schlug bei mir jedoch bisher fehl.
Ein4	Zeige den Dateinamen nicht an; setzt voraus, dass sich die Applikation um die Dateiverwaltung der aufgenommenen *Sprachmemo*s kümmert	Die Applikation zeigt im Moment die Meldung “Memo als <Dateiname> auf Gerät gespeichert”. Der Dateiname ist für den Nutzer jedoch irrelevant und dessen Angabe stammt noch aus Entwicklungszeiten für mich zu Defektidentifikation. Allerdings verwaltet die Applikation die Dateien bisher noch nicht.

Ein5	Füge eine Anzeige (z.B. Haken) für angefügte *Sprachmemo* hinzu, füge Abspiel-*Button*, Hinzufügen- und Ändern-*Button*s ein	Die Heuristik fordert, dass ein System seinen Status anzeigt, darum muss dem Nutzer auch zur Kenntnis gegeben werden, dass bereits eine *Sprachmemo* angehängt wurde.
Ein6	Füge speziellen Befehl für die Änderung des Titels über die Spracheingabe ein	Nach Heuristik 5 sollen Fehler vermieden werden, Heuristik 3 besagt, dass der Nutzer Kontrolle haben soll. Die Applikation erkennt jedoch eine ihr unbekannte Spracheingabe fast immer als Titeleingabe und ändert damit den Titel. Dies nimmt dem Nutzer die Möglichkeit zu entscheiden, wann er den Titel ändern möchte und provoziert Fehler (Die Titeländerung wird nicht explizit hervorgehoben).
Ein7	zeige die Zeitzone irgendwo mit an (z.B. in der Titelleiste)	Ähnlich Heuristik 1, sollte der Status angezeigt werden, also auch die verwendete Zeitzone
Ein8	Suche die Ursache des Problems	technisches Problem
Ein9	Vermutlich haben beide Anordnungen Vor- und Nachteile, die ich im Moment nicht abschätzen kann, also abwarten	Hier handelt es sich um ein Problem des Designs der Benutzerschnittstelle und ich habe kein Prinzip gefunden, das für diesen Fall Empfehlungen gibt. Ich erwäge jedoch diese Ansicht vollständig zu ändern, da sie auch noch andere Probleme verursacht. Es wäre also ein neuer Entwurf vonnöten.
Ein10	Fange den Netzwerkfehler ab und verhindere, dass die Daten verloren gehen	technisches Problem: Es werden Ausnahmen geworfen, jedoch bisher nur mit einer Rückmeldung (, also nicht sinnvoll) behandelt.
Ein11	füge eine Option für gar keine Erinnerung hinzu	technisch: neue Funktionalität z.B. in Form eines Kontrollkästchens (Checkbox)
TAb1	Zeige kleine Tagesübersicht unterhalb des angezeigten Termins an	Da einige Nutzer an einem Szenario gescheitert sind, weil sie die Termine eines Tags nicht gefunden haben, sollen diese Termine in Form einer Teil-*Agenda-Ansicht* (dem Ausschnitt aus einem Tag) angezeigt werden. Ich habe mich für unterhalb entschieden, da sich so die Bedienelemente zwischen den Ansichten nicht so sehr verschieben, weil die Elemente außer der Tagesübersicht ihre Größe beibehalten können.

TAb2	Ordne *Buttons* nach Aufgabe an (siehe Wikipedia “Principles of user interface design” [7])	Bedienelemente, die sich aufeinander beziehen sollten zusammengefasst werden
TAb3	Passe den Feldinhalt immer an	technisches Problem: Dies sollte eigentlich schon geschehen, aber ich habe noch einen Defekt im Code
TAb4	Passe die Meldung an	Nach Heuristik 6 sollen Systeme Nutzer darauf hinweisen, wenn sich auf Daten aus anderen Dialogen beziehen. Die bisherige Meldung erfordert, dass der Nutzer genau weiß, an welchem Tag die Applikation keine Termine gefunden hat, während das Feld, in dem der Tag steht nicht sichtbar ist.
TAb5	über die Tagesübersicht zu lösen	Der Nutzer muss bisher auf einen anderen Bildschirm wechseln, um einen weiteren Termin am selben Tag zu sehen. Er muss sich also selbst merken, welche Termine an einem bestimmten Tag sind. Dies sollte ihm die Applikation abnehmen.
TAb6	vermutlich nicht zu lösen, aber auch kein Problem	technisch eine neue Funktionalität, für die ich keine Lösung finde
Not1	Setze die Daten einfach ein	Der Nutzer sollte sich nicht daran erinnern müssen, um welchen Termin es sich genau handelt, wenn es ihm von der Applikation gezeigt werden kann. (Heuristik 6)
Not2	Suche die Ursache und entferne sie entsprechend (trivial)	technisches Problem: Ich bevorzuge jedoch ein Standard-*Popup* einzusetzen
Not3	Sortiere die Erinnerung vor Rendern	technisches Problem: Ich habe vergessen, die Erinnerungsdaten nach Start zu sortieren, bevor sie dargestellt werden.
Not4	Würde eine Umbau der Datenstruktur und des Renderns erfordern, allerdings ist dies auch kein Problem	Andere Applikationen zeigen z.B. Termine chronologisch an. Dies ist also standardkonform, im Gegensatz zu der Darstellung, die ich implementiert habe. Heuristik 4 fordert von einem System konsistent und standardkonform zu arbeiten, auch wenn sie sich auf Begriffe bezieht.

[7] http://en.wikipedia.org/w/index.php?title=Principles_of_user_interface_design&oldid=532111297

Not5	Füge diese Eingabemöglichkeit in den Zustandsautomaten ein; implementiere ein dynamische Zurück-Funktion	technisches Problem
Not6	Füge eine Verknüpfung zu entsprechendem Bildschirm hinzu; dazu muss das Terminfinden erweitert werden, sodass auch automatisch navigiert werden kann	technisches Problem, aber auch die Möglichkeit der Applikation den Nutzer auf Daten anderer Bildschirme zu verweisen und damit einerseits praktisch und andererseits laut Heuristik 6 wünschenswert.
SSt1	Verschiebe das Anzeige-Element oder suche nach passender *Popup*-Lösung	technisches Problem
SSt2	Vermutlich ein Problem des *Framework*s, muss noch untersucht werden	technisches Problem (hat sich als mit dem schlechten Laufzeitverhalten zusammenhängend herausgestellt)
SSt3	Verwende statt *Alert*s *Popup*s, die dann z.B. nach Timeout selbst schließen	technisches Problem: *Alert*s sollten zwar von vornherein nicht für diese Funktion verwendet werden. Sie lassen sich jedoch auch nicht automatisch schließen, müssen also unbedingt ersetzt werden.
SSt4	(zu jener Zeit keine Idee, nach der 2. Iteration: Lösung durch Änderung des Verhaltens der Aktivierung der Spracheingabe)	technisches Problem
SSt5	Hilfe-Funktion	Hier kann die Hilfe-Funktion auf jeden Fall helfen. Die genaue Ursache für die Verwirrung des Nutzers konnte ich aber nicht finden.
Wid1	Ersetze das *Datums-Widget* z.B. durch Tastatureingabe; Überlege anderes Konzept	Wie schon erwähnt, machte der gesamte Dialog “Termin erstellen” viele Probleme. Unter anderem, dass die Bedienung der einzelnen Elemente nicht konsistent ist, also z.B. nach “Weiter”-Drücken von der Titeleingabe aus, die Datumseinstellung übersprungen wird, da diese nicht mit Tastatur bedienbar ist. Die Datumseinstellung hat in ihrer bisherigen Form den Nutzern sehr viele Probleme, teilweise Stress verursacht (nach Aussagen wie etwa “[...]nicht schon wieder dieses Mistding[...]” und ähnliches)

Wid2	Anpassung in den Wörterbuchdaten des *Datums-Widget*s (Umstellung auf UniCode)	technisches Problem: Durch Änderung der Codierung einer Datei zu beheben
Wid3	wie in Wid2 eine Anpassung in den Wörterbuchdaten	technisches Problem: Änderung in selben Datei wie in Wid2
Spr1-5	einfach in die Grammatik einsetzen, teilweise Anpassung in der Verarbeitung der Daten	technisches Problem, wesentlich interessanter wäre die "Sprache des Nutzers" zu finden, also Formulierungen, die ein potenzieller Nutzer intuitiv verwendet. (siehe Heuristik 2)
Spr6	Gilt es zu untersuchen	technisches Problem: (Auf schlechtes Laufzeitverhalten zurückzuführen)
Spr7	Suche nach Möglichkeit zwischen ausführbaren und nicht-ausführbaren Spracheingaben zu unterscheiden	technisches Problem und gewünschtes Verhalten nach Heuristiken 1 und 9

Tabelle 3: Die vorläufige Priorisierung mit Umsetzung

Priorität	Problem	Umsetzung
1	Wid1	War nicht umsetzbar, weil ich keinen nachweisbar guten Ersatz für das *Datums-Widget* gefunden habe; allerdings wurde das Problem hauptsächlich durch das schlechte Laufzeitverhalten verursacht, dieses ist inzwischen besser, aber nicht hinreichend gut.
2	Ein10	Netzwerkfehler werden abgefangen und der Nutzer gefragt, ob er die Anfrage wiederholen oder abbrechen möchte.
3	Not2	Das Problem war nicht der Schließen-*Button*, sondern der Wechsel des *Popup*-Inhalts; das *Popup* wurde durch simplemodal[8] ersetzt
4	TAb1, TAb5	Es werden nun alle Termine an einem Tag angezeigt und der oben angezeigte wird markiert (wegen Rückmeldung des Zustands - siehe Heuristik 1)
5	TAb2	Um den *Button*s für nächster und vorheriger mehr Sinn zu geben, wurden sie mit "Suche Termine" versehen und sind vom funktional unterschiedlichen "Termine ansehen"-*Button* getrennt

[8] (http://www.ericmmartin.com/projects/simplemodal/

6	Not3	Erinnerungen werden auf ihrem Bildschirm immer noch in einer Rasteranordnung angezeigt, jedoch werden sie nun nach Start sortiert.
7	Ein5	bei angefügter *Sprachmemo* wird ein Symbol im *Button* angezeigt; um eine angefügte *Sprachmemo* anzuhören, wird bei erneutem Drücken ein Dialog angezeigt, der auch das Löschen und Ändern ermöglicht.
8	Bed1	aus Zeitgründen nicht mehr umgesetzt; Änderung 4 (für TAb1, TAb5) sollte dies aber schon ausreichend gemildert haben, sodass dieses Problem eher zu einer fehlenden Funktionalität wird
9	Ein6	Es muss nun explizit gesagt werden, dass der Titel geändert werden soll
10	Not1	es werden im *Popup* nun die gleichen Daten wie in der Terminansicht angezeigt und *Sprachmemo*abspielen ist auch möglich
11	Bed4	es wird die gleiche Technik für alle Abspielknöpfe verwendet, d.h., sie ermöglichen Stoppen durch erneutes Drücken
12	Not6	statt der Weiterleitung wird beim Klicken das Erinnerungs*popup* erneut angezeigt (bei der 2. Iteration hat sich diese Funktionalität als störend herausgestellt und es sollte weitergeleitet werden)
13	Spr7	aus Zeitgründen nicht mehr umgesetzt
14	SSt3	aus Zeitmangel nicht mehr umgesetzt

Diese Tabelle enthält nur einen Teil der Änderungen. Was im Detail gemacht wurde, steht unter 5.9 im Anhang.

2.3.3 2. Iteration

Für die 2. Iteration konnte ich endlich eine neuere Version des *Framework*s verwenden, das mit dieser schon wesentlich verbessert wurde. Die genaueren Verbesserungen sind nicht Thema dieser Arbeit. Zumindest hat sich die CPU-Last, die die Applikation erzeugt verringert. Außerdem konnte ich das zu aufwändige Element ausfindig machen und entfernen. Allein diese Maßnahmen haben das Gefühl bei der Bedienung schon stark verbessert. Im Vergleich zu vorher sind 12 Probleme erneut gefunden worden und 8 der ungelösten nicht noch einmal, was nur Zufall gewesen sein kann, da sich ja noch vorhanden sind.

Die Anzahl der gefundenen katastrophalen Probleme hat sich zwar nicht reduziert, allerdings sind 3 der alten Probleme nicht mehr aufgetreten, da ich sie technisch gelöst habe. Die Probleme, die noch bestehen, sind die Bedienprobleme des *Datums-Widget*s, das schlechte Laufzeitverhalten, die fehlende Hilfe-Funktion und einige kleinere technische Feinheiten, die den Nutzer irritieren könnten, weil die Applikation ihren Status falsch anzeigt bzw. unerwartete Fehlermeldungen wirft.

Ergebnisse Betrachtet man die Bewertung der Applikation über *SUS* (siehe 3.1), so stellt man fest, dass nach der Umarbeitung die Bewertung gleichmäßiger ausfällt und auch wesentlich besser ist. Das könnte zwar auch an der Auswahl der Nutzer liegen, jedoch sind auch nur noch die 2 Szenarien manchmal fehlgeschlagen, bei denen das *Datums-Widget* benutzt werden musste. In der ersten Iteration gab es außer diesen Problemen mit dem *Datums-Widget* noch massive Probleme bei der Spracheingabe und mittlere Probleme beim Navigieren durch die Termine.
Die groben Probleme mit der Applikation beschränkten sich damit nur noch auf das *Datums-Widget* und die Kritik am Laufzeitverhalten der Applikation. Für mich ist damit schon eine deutliche Verbesserung erkennbar, die sich im *SUS*-Wert niederzuschlagen scheint. Bis zur Marktreife müssen jedoch auf jeden Fall noch alle verbleibenden als katastrophal eingestuften Probleme gelöst werden und damit auch noch weitere Iterationen gemacht werden.
Folgende Tabelle führt alle gefundenen Probleme aus der zweiten Iteration auf. Die Probleme sind fortlaufend nummeriert. Bei Bezügen auf alte Probleme werden diese in Klammern hinter der Problembeschreibung angegeben.

Schl. Schlüssel des Problems

Anzahl der Auftritte des Problems

Fat. *Fatalität* des Problems

Tabelle 4: gefundene Probleme aus der zweiten Iteration

Schl.	Problem	#	Fat.	Kommentar
Bed6	Nutzer erkennt Funktion des Erinnerungs-*Button*s noch nicht gleich (Bed2)	1	1	Die Abbildung ist nicht intuitiv genug, ein Designer könnte dies vielleicht beheben
Bed7	Nutzer sieht *Alert*s immer als Fehler an	1	3	Da hierdurch massive Missverständnisse entstehen, da *Alert*s allgemein nur für Fehlermeldungen gedacht sind.
Bed8	Nutzer erwartet“zurück” eher oben	1	1	Es gilt hier noch zu untersuchen bei wie viele Menschen dies irritiert.
Bed9	Die Daten von heute und morgen durch die Darstellung “heute” bzw. “morgen” ersetzen	1	1	Dadurch würde die Lesbarkeit verbessert und ebensolche Daten leichter erfassbar
Bed10	Erinnerungs-*Button* sollte auch anzeigen (z.B. farblich), dass die *Erinnerungsübersicht* schon gesehen wurde	1	0	sehr praktische zusätzliche Funktion
Bed11	Rückmeldungen der Applikation (im Moment *Alert*s) sollten über die Sprachsteuerung zu schließen sein (SSt3)	1	2	
Ein12	Bei der Spracheingabe sollte der Titel auch korrekt geschrieben werden und nicht, wie jetzt, mit Kleinbuchstaben	1	1	Dieser Umstand war für den Nutzer unerwartet
Ein13	Nutzer möchte über eine Kalenderansicht Termine eintragen können (Bed1)	2	3	eigentlich eine zusätzliche Funktionalität, aber da andere Kalenderapplikation soetwas ermöglichen, von 0 erhöht
Ein14	Die Applikation gibt eine Fehlermeldung zurück, wenn Spracheingabe und Aufnahme einer *Sprachmemo* gleichzeitig aktiv sind.	1	4	Die Applikation kann diesen Umstand selbst behandeln.

Ein15	Die Bildschirmtastatur ist mit “Öffnen” statt Return beschriftet, wenn der Titel bearbeitet wird	1	2	Diese Beschriftung ist für den Nutzer irritierend.
Ein16	Die Aufnahme der *Sprachmemo* mit Hilfe des Mikrofon-*Button*s beenden	1	0	Funktionalität und eigentlich auch unpraktisch, weil sie dem *Button* eine Funktionalität geben würde, die nicht zu seinem eigentlichen Verhalten passt.
Ein17	Nutzer erwartet Speichern-*Button* ganz am Ende des Formulars	1	1	So ist es an sich auch üblich. Ich vermutete bei der Entwicklung, dass das *Sprachmemo*aufnehmen oder Personenverknüpfen nicht immer gemacht werden und habe die Anordnung deshalb so gelassen.
Ein18	Durch die Anordnung der *Button*s (Ein17) ergibt sich, dass ein Nutzer durch den Mikrofon-*Button* und “füge Sprachmemo an” irritiert wird.	1	1	
TAb7	Nutzer wünscht sich die Termine von heute auf dem Startbildschirm	1	0	eine Funktionalität
TAb8	Nutzer wünscht sich *Agenda-Ansicht* (Bed1)	3	0	eine Funktionalität
TAb9	Nutzer möchte Termine auch bearbeiten können	2	4	ist wichtig und wegen Markteinfluss auch notwendig, deshalb von 3 erhöht
TAb10	Nutzer möchte Termine durch Klicken auf Tagesübersicht anzeigen können	3	3	eigentlich kein Problem, aber Nutzer erwarten diese Funktionalität
TAb11	Statt auf “Anzeigen”-*Button* drücken zu müssen, sollte diese Aktion gleich mit dem Einstellen des Datums passieren	1	1	Der “Anzeigen”-*Button* ist wirklich gar nicht notwendig und stört (Heuristik 8)

Not7	Nutzer sieht Erinnerungspopup teilweise als Fehlermeldung an	2	2	Eine Beschriftung des *Popup*s als Erinnerung scheint hier angebracht zu sein, sonst ist der Kontext nicht ersichtlich (siehe "Android Design Principles"[9])
Not8	*Erinnerungsübersicht* ist für den Nutzer zu unübersichtlich (Not4)	1	2	Die Übersicht ist tatsächlich nicht standardkonform.
Not9	Nutzer möchte beim Klicken auf eine Erinnerung zur Terminübersicht des entsprechenden Termins gelangen (Not6)	1	2	Die Applikation sollte eine Verbindung zwischen verschiedenen Dialogen schaffen, wenn möglich, statt dies dem Nutzer zu überlassen (Heuristik 6)
SSt6	Nutzer möchte Spracheingabe durch nochmaliges Drücken beenden - löst damit aber eine neue aus	2	3	Dieses Verhalten ist für den Nutzer unerwartet.
SSt7	Die Applikation sollte anzeigen, wenn sie eine Spracheingabe verarbeitet	2	3	Heuristik 1
SSt8	Sprachbefehle können nur benutzt werden, wenn sie bekannt sind, also fehlt eine Hilfe-Funkton	1	4	Ohne Hilfe-Funktion ist die Sprachsteuerung nicht so leicht zu verwenden.
SSt9	Nutzer möchte Befehle durch "und" verknüpfen (Spr2)	1	1	
SSt10	Der Mikrofon-*Button* zeigt manchmal den falschen Zustand; die Spracheingabe hängt manchmal (SSt2)	4	4	Heuristik 1 und Markteinfluss, da Spracheingabe sauber funktionieren muss
Wid4	Verzögerungen bei der Bedienung behindern stark (Wid1)	4	4	
Wid5	Datum und Uhrzeit sollten getrennt eingegeben werden (Wid1)	2	4	die Bedienung ist durch diesen Umstand schwieriger als nötig
Wid6	"Datum einstellen" irritiert, darum bevorzugt der Nutzer "OK" oder "Datum aktualisieren" (Wid3)	1	2	

[9] http://developer.android.com/design/patterns/confirming-acknowledging.html

Spr8	Nutzer wünscht sich einen Befehl für die Aufnahme einer *Sprachmemo*	1	2	
Spr9	Orientierung an Siri für die Sprachbefehle	1	3	wegen Markteinfluss
Spr10	Erkennung von “von 1820 bis 1940” als Uhrzeiten, wenn sinnvoll	1	1	Es ist üblich, Uhrzeiten so abzukürzen

3 Auswertung

Nun, da die Applikation 2 Iterationen Evaluation und eine Anpassung durchlaufen hat, werden in diesem Abschnitt die Erkenntnisse aus dieser Studie zusammengefasst.

3.1 Ergebnisse der *SUS*-Befragung

3.1.1 Iteration 1

Aussage	Nr.	Nutzer 1	Nutzer 2	Nutzer 3	Nutzer 4
Ich denke, dass ich dieses System gerne häufig nutzen würde.	1	1	0	3	0
Ich fand das System unnötig komplex.	2	0	4	0	1
Ich denke, das System war einfach zu benutzen.	3	4	1	2	1
Ich denke, ich würde die Hilfe eines Technikers benötigen, um das System benutzen zu können.	4	0	1	0	0
Ich halte die verschiedenen Funktionen des Systems für gut integriert.	5	3	1	1	2
Ich halte das System für zu inkonsistent.	6	2	3	2	2
Ich kann mir vorstellen, dass die meisten Leute sehr schnell lernen würden, mit dem System umzugehen.	7	3	2	3	2
Ich fand das System sehr mühsam zu benutzen.	8	4	4	2	3
Ich fühlte mich bei der Nutzung des Systems sehr sicher.	9	3	1	2	2
Ich musste viele Dinge lernen, bevor ich das System nutzen konnte.	10	0	3	0	1
	Ergebnis	28	10	27	20
	SUS-Wert	70	25	$67,5$	50
	Mittelwert:	$53,125$	Varianz:	$28,125$	

3.1.2 Iteration 2

Aussage	Nr.	Nutzer 5	Nutzer 6	Nutzer 7	Nutzer 8
Ich denke, dass ich dieses System gerne häufig nutzen würde.	1	0	3	1	2
Ich fand das System unnötig komplex.	2	2	1	0	0
Ich denke, das System war einfach zu benutzen.	3	2	3	3	4
Ich denke, ich würde die Hilfe eines Technikers benötigen, um das System benutzen zu können.	4	0	0	0	0
Ich halte die verschiedenen Funktionen des Systems für gut integriert.	5	3	4	4	4
Ich halte das System für zu inkonsistent.	6	1	0	0	0
Ich kann mir vorstellen, dass die meisten Leute sehr schnell lernen würden, mit dem System umzugehen.	7	3	2	4	4
Ich fand das System sehr mühsam zu benutzen.	8	1	3	1	1
Ich fühlte mich bei der Nutzung des Systems sehr sicher.	9	3	2	2	3
Ich musste viele Dinge lernen, bevor ich das System nutzen konnte.	10	1	0	0	0
	Ergebnis	26	30	33	36
	SUS-Score	65	75	$82,5$	90
	Mittelwert:	$78,125$	Varianz:	$13,125$	

3.2 Verbesserungen, übrige Schwächen

Verbesserungen Für mich persönlich ist die Applikation spürbar besser geworden. Ob sich die *Usability* erhöht hat, schlägt sich in den Teilaspekten Zufriedenheit, Effektivität und Effizienz wider. Für die Zufriedenheit hatte ich *SUS* als Maß verwendet und es hat sich gezeigt, dass sich dieser Wert deutlich erhöht und stabilisiert hat. Da die geringe Anzahl von Nutzern keine quantitative Analyse der Ergebnisse erlaubt, habe ich auf weitergehende Analysen verzichtet. Für die Effektivität und Effizienz war ursprünglich vorgesehen die Dauer

der Szenarien zu messen. Jedoch ist dies einerseits zusammen mit der “think aloud”-Methode wenig praktikabel (siehe [1]), andererseits habe ich die Nutzer wirklich in ihrem natürlichen Umfeld getestet, in dem sie teilweise massiv abgelenkt waren. Aus diesem Grund ist auch folgende Aussage etwas vorsichtig zu behandeln: In der ersten Iteration dauerte ein Test durchschnittlich etwa $30,75$ Minuten und in der zweiten Iteration etwa 22 Minuten. Weiterhin sind, wie schon erwähnt, während der zweiten Iteration Szenarien ausschließlich aufgrund des *Datums-Widget*s fehlgeschlagen.

Schwächen Die neben der fehlenden internen Hilfe noch viele mögliche Formulierungen, die die Sprachsteuerung annehmen sollte. Die einzelnen, übrigen Probleme sind in Tabelle 4 nachzulesen.

3.3 Barrieren durch das *Framework*

Ein besonderes Problem, das mir dieses *Framework* macht, ist die Ladeanzeige. Nicht nur, dass es durch zu hohe CPU-Last zu schlechtem Laufzeitverhalten kommt, es ist nicht in allen Fällen möglich, anzuzeigen, dass die Applikation gerade einen neuen Bildschirm lädt. Damit kann der Nutzer häufig nicht unterscheiden, ob die Applikation gerade seine Eingabe verarbeitet oder er es noch einmal versuchen soll oder sie womöglich gar nicht mehr läuft. Unter Umständen kann dieses Defizit durch bestimmte Methoden bei der Entwicklung gelöst werden. Diese sollte das *Framework* aber vorgeben, falls sie vorhanden sind.
Ein weiteres Problem ist die relativ primitive Art der Sprachsteuerung. Für mich war es nur möglich Befehle einzufügen und je nachdem welche Daten eingegeben wurden, bestimmte Ereignisse zu erzeugen. Damit ist jede Spracheingabe vom Zustand der Applikation und von anderen gemachten Spracheingaben unabhängig. Da ich allerdings teilweise Sprachbefehle benötigte, die voneinander abhängig sind, habe ich dies auf eigene Art primitiv implementiert. Es sind noch weitere Funktionalitäten im *Framework* geplant, die diesen Umstand verbessern sollen und meine Implementierung ersetzen. Ohne eine solche Verbesserung sind die Sprachsteuerungsmöglichkeiten nicht konkurrenzfähig.
Zudem habe ich relativ am Anfang darauf verzichtet mehrere Sprachen in der Benutzerschnittstelle zu unterstützen. Die Oberfläche kann zwar ohne Probleme in mehrere Sprachen übersetzt werden und diese Übersetzung auch dynamisch verändert werden, aber die Spracheingabe lässt sich mit meiner Version des *Framework*s nicht dynamisch ändern. Es gibt noch nicht einmal eine Möglichkeit mehrere Grammatiken einzusetzen.
Als ein weiterer Punkt hat mir das *Framework* keinerlei Möglichkeiten gegeben andere Applikationen auf dem Gerät des Nutzers zu verwenden, sodass ich bisher keinen Nutzen für die Verknüpfung mit Personen innerhalb der Applikation erzeugen konnte.
Das sind die Hauptschwachstellen, auf die ich gestoßen bin und die noch im-

mer bestehen.

3.4 Bewertung des *Framework*s

Positives Nichtsdestotrotz ist es nicht so schwierig eine funktionsfähige Applikation mit diesem *Framework* zu bauen. Ich hatte keinerlei Dokumentation, sondern nur eine kurze Einführung des Hauptentwicklers, sodass ich mehrere 100 Stunden für den ersten Prototypen benötigt habe. Mit Dokumentation, einem vollständig funktionierenden *Framework*, sehr guter Javascript- und Webentwicklungserfahrung ist es aber sehr leicht zu bedienen.

Probleme Als sehr unangenehm haben sich die fehlenden Debug-Möglichkeiten erwiesen, die es mir während der Entwicklung oft schwierig gemacht haben Defekte zu lokalisieren. Insbesondere war dies bei Fehlverhalten in der Oberfläche der Fall. Daneben bestand noch die schon erwähnte Barriere den Zustand der Applikation anzuzeigen.
Ein weiteres Problem, das aber aus der Anforderung entstanden ist, dass das *Framework* plattformunabhängig ist, ist die schlechte Unterstützung nativer Bedienelemente wie z.B. dem Sprachaufnahmedialog, der eigentlich funktionieren sollte.
Und die schlimmste Schwäche ist natürlich die CPU-Last, die das *Framework* verursacht. Solange es das tut, ist es unmöglich damit eine wirklich nutzbare Applikation zu erzeugen. Und für mich war dieser Umstand während des gesamten Entwicklungsprozesses sehr unangenehm und hat mich wirklich gestresst.

4 Fazit

Was wurde erreicht? Im Rahmen dieser Arbeit ist eine Kalenderapplikation entstanden, die die meisten Funktionalitäten hat, die Nutzer an eine solche Art von Applikation stellen. Für diese Applikation wurde eine Nutzbarkeitsstudie durchgeführt, die zu sichtbaren Verbesserungen in der *Nutzbarkeit* der Applikation geführt hat, aber auch gezeigt hat, dass bis zu Marktreife noch Funktionalitäten fehlen und Probleme bestehen, die dringend gelöst werden müssen. Die Studie hat qualitative Schwächen der Applikation teilweise aufgedeckt. Das Beispiel dieser Applikation hat gezeigt, dass das *Framework* selbst noch große Schwächen in der *Nutzbarkeit* hat. Und zwar in Bezug auf den Entwickler als Nutzer und in der Wirkung des *Framework*s auf die *Nutzbarkeit* der entstandenen Applikation. Wie hiermit gezeigt, hat das *Framework* leider zu sehr schlechter *Nutzbarkeit* geführt, die einen praktischen Nutzen ausschließt. Das heißt allerdings nicht, dass eine neue Version des *Framework*s diese Probleme nicht ausräumen kann. Ich habe ja auch nur eine *Entwicklungsversion* untersucht.

Was fehlt? Die Studie muss noch über weitere Iterationen fortgeführt werden, denn solange noch Probleme auftreten, gibt es Erkenntnisse zu gewinnen. Insbesondere muss die Applikation noch um Funktionalitäten wie eine Kalenderübersicht oder *Agenda-Ansicht* und das Bearbeiten von Terminen erweitert werden und die Szenarien um neue Elemente erweitert werden, um auch diese Funktionalitäten standardisiert testen zu können. Das *Framework* benötigt ebenfalls noch Verbesserungen in der Effizienz, sodass sie keine dauerhafte CPU-Last mehr verursacht. Sobald die *Nutzbarkeit* ausreichend ist, also z.B. keine Probleme der *Fatalität* größer 2 mehr auftreten, kann untersucht werden, warum Nutzer die Applikation nicht verwenden wollen, falls sie keine Akzeptanz findet.

Was nicht getan wurde Ich habe in dieser Arbeit nicht explizit aufgeführt, welche Szenarien aus welchem Grund fehlgeschlagen sind, da ich dies durch Fehler in der Vorbereitung und technische Probleme beim *Logging* nun nicht mehr nachvollziehen kann. Damit sind die Fehlschläge in den Szenarien nur implizit in den *Fatalität*en widergespiegelt.
Aus der Auswertung gehen nicht alle Aussagen der Testnutzer bezüglich ihrer Vorstellungen an die Funktionalitäten der Applikation hervor. Besonders wichtige Funktionalitäten, die schon während der Ausführung der Szenarien nachgefragt wurden (wie z.B. die Kalenderansicht) sind zusammengefasst aufgeführt. Jedoch habe ich auf die Angabe gewünschter Funktionalitäten, die meiner Meinung nach offensichtlich keine Usability-Steigerung bewirken, verzichtet. Es sollte hier ausschließlich um Usability gehen.

Erkenntnisse Als wichtige Erkenntnis möchte ich hier hervorheben, dass bei der Entwicklung einer Applikation für unterschiedliche Nutzer immer die Heuristiken beachtet werden sollten. Also, dass bei der Entwicklung systematisch auf mögliche *Usability*-Probleme geachtet wird. Dadurch lassen sich sehr viele Probleme schon während der Entwicklungszeit vermeiden. Weiterhin reicht es in einem primitiven Stadium einer Applikation wirklich nur wenige *Nutzertest*s pro Iteration zu machen. Ich behaupte sogar, dass es in einem sehr frühen Stadium wie es hier der Fall war, nur einen Testnutzer zu verwenden, um die offensichtlichen Probleme für eine Iteration schnell entfernen zu können. Denn bei mir sind viele kleine Probleme aufgetreten wie z.B. der nicht korrekt arbeitende Schließen-*Button* oder das Laufzeitproblem oder die fehlende Tagesübersicht aufgetreten, die mir vor dem ersten Test nicht so deutlich aufgefallen sind. Diese Unsauberkeiten haben insbesondere Probleme in den Bedienungsabläufen verdeckt, die für mich wesentlich wichtiger waren. Das heißt, vor einer längeren Studie, wäre es sicherlich sinnvoll die groben Probleme ausgeräumt zu haben. Dass mir diese Fehler unterlaufen sind, lag einmal an fehlender Erfahrung in diesem Bereich und der neuen Technologie, die mich leider in keiner Weise unterstützt hat.

5 Anhang

5.1 Definitionen verwendeter Begriffe

Agenda-Ansicht Ich meine mit einer Agenda-Ansicht eine Ansicht, in der die Elemente in einer Liste angeordnet sind und nach einem Schlüssel gruppiert sind. Für Termine bedeutet das, dass unter einem Tag alle Termine dieses Tags stehen. Und es werden alle Tage, an denen mindestens ein Termin ist, gelistet.

Alert bezeichnet eine Dialogbox, die mit "Alert" beschriftet ist, normalerweise eine Fehlermeldung oder einen Hinweis enthält und sich über "OK" schließen lässt. Dies ist ein Standarddialog, den Javascript unterstützt.

Benutzbarkeit auch Usability

Button Mit Buttons bezeichne ich Schaltflächen, die durch Berühren des Touch-Screens betätigt werden können.

CalDAV ist eine Erweiterung des WebDAV-Protokolls, auf das ich nicht weiter eingehen werde. CalDAV spezifiziert Standardoperationen für die Arbeit mit Kalendern. (siehe [3] für die Spezifikation)

Datums-Widget ist ein spezielles visuelles Bedienelement, um ein Datum auszuwählen. Gegebenenfalls kann auch zusätzlich die Uhrzeit damit ausgewählt werden.

Entwicklungsversion Eine Version eines Produkts, das sich noch in Entwicklung befindet. D.h., eine Version, die noch vor der ersten veröffentlichten liegt.

Erinnerungsansicht siehe Erinnerungsübersicht

Erinnerungsübersicht auch Erinnerungsansicht, ist ein Bildschirm in meiner Applikation, auf dem die Termine, an die erinnert wurde und die noch nicht beendet sind, angezeigt werden.

Fatalität die Schwere eines Problems

Framework auch Rahmenwerk oder Rahmenstruktur genannt. In diesem Fall stellt das Framework eine Menge von Strukturen, Methoden und Werkzeugen bereit, um die Entwicklung zu unterstützen.

Logger ist das Modul, das Methoden zum Logging definiert.

Logging meint den Vorgang Daten, die dazu dienen den Programmablauf teilweise nachzuvollziehen, in menschenlesbarer Form zu sammeln.

MVC Abkürzung für Model View Controller. Ist ein Software-Entwurfsmuster, bei dem Software in die 3 Einheiten Datenmodell, Präsentation und Programmsteuerung eingeteilt werden (siehe dazu Wikipedia[10] als Einstieg).

Nutzbarkeit auch Benutzbarkeit

Nutzertest auch Nutzbarkeitstest oder Usability-Test

"Polling bezeichnet in der Informatik die Methode, den Status eines Geräts aus Hard- oder Software oder das Ereignis einer Wertänderungen mittels zyklischem Abfragen zu ermitteln."[11]

Popup ist ein visuelles Dialogelement, das sich nicht in die Struktur vorhandener Bedienelemente einordnet, sondern in deren Vordergrund "schwebt". Es kann selbst noch Bedien- bzw. Dialogelemente enthalten.

Select-Element ist ein Standard-HTML-Element, spezifiziert durch das W3C im HTML-Standard[12].

Sprachmemo ist eine Form von Audiodaten. Eine Sprachmemo speichert eine kurze Aufnahme gesprochenen Texts. Sie soll gespeichert werden können und zu einem späteren Zeitpunkt wieder angehört werden können.

SUS Abkürzung für "System Usability Scale", ein Skalar zur Bewertung der Usability auf Basis von 10 Aussagenn, die über eine Likert-Skala bewertet werden. ([10])

Usability ist eine Eigenschaft eines Produkts. Sie umfasst Effizient, Effektivität und Zufriedenheit in der Nutzung. (siehe [13])

Usability-Evaluation bezeichnet eine systematische möglichst objektive Bewertung eines Produkts hinsichtlich seiner Usability (nach [1])

5.2 Anforderungen an die Applikation bei der Entwicklung

Für die Arbeit ist eine Anwendung, die an das mobile Dialogsystem angeschlossen ist und in diese einen (Web-)Kalender integriert, zu erstellen.

Die Operationen, die die Anwendung durchführen soll, sind Erzeugen, Auffinden und Anzeigen von Events und sie soll den User abhängig von den Daten im Event erinnern.

[10] http://de.wikipedia.org/w/index.php?title=Model_View_Controller&oldid=114824177

[11] http://de.wikipedia.org/w/index.php?title=Polling_%28Informatik%29&oldid=111908429

[12] http://www.w3.org/TR/REC-html40/interact/forms.html#h-17.6

Ein Event muss dabei Start- und Enddatum und Links zu Daten (z.B. Bilder und Sprachdateien) enthalten.

Die Anwendung soll weitestgehend plattformunabhängig umgesetzt werden und wird mit Phonegap (inzwischen Cordova) in Javascript und HTML5 entwickelt.
Folgende Sprachbefehle wären denkbar (Beispiele):

- Zeige mir meine Termine am 15.06.12
- Was sind meine Termine am 15.06.12
- Habe ich Termine am 15.06.12
- Trage einen Termin am 23.06.12 um 15 Uhr ein
- Trage einen Termin für den 23.06.12 ein
- Termin für 20.06.12 zum Thema Besichtigung eintragen
- Trage heute um 15 Uhr Besichtigung ein
- Trage jetzt Blumengießen ein
- Spiele Sprachnotiz ab
- Sprachnotiz aufnehmen
- Füge Sprachnotiz zum Kalendereintrag hinzu
- Trage einen Termin ein und füge eine Sprachnotiz hinzu
- Vorwärts
- Zurück

Der Kalender, der integriert werden soll, muss gefunden werden und soll folgende Anforderungen erfüllen:

- Erinnerungen an Events des Users ermöglichen
- die Daten fassen können
- möglichst plattformunabhängig sein
- die Daten der Anwendung vor Korrumpierung schützen

5.3 Die Kalenderschnittstelle

Hier steht die Definition der Schnittstelle des Kalenders in Java, statt Javascript.

```
/*
  interface: CalendarClient<EventType>
*/
/**
 * Adds an Event to the connected calendar and then returns an Event Object to
 *  identify the added one.
 * @param event CalendarEvent to add
 * @return an Event, that may be used to refer to the added in the Calendar.
 */
public EventType addEvent (CalendarEvent event) throws IOException;

/**
 * Retrieve all Events from the connected calendar within start and end bounds.
 * @param start lower bound date to search for events
 * @param end upper bound date to search for events
 * @return a List of Events, that refer to the calendar's and
 *      occur between start and end
 */
public List<EventType> findEvents (Date start, Date end)
                       throws IOException;

/**
 * Retrieve all Events from the connected calendar with
 *   Reminders within start and end bounds.
 * @param start lower bound date to search for reminders
 * @param end upper bound date to search for reminders
 * @return a List of Events, that refer to the calendar's and that have
 *      Reminders occurring between start and end
 */
public List<EventType> findEventsByReminders (Date start, Date end)
                       throws IOException;

/**
 * Remove the given Event from the connected calendar. It has no effect,
 *   if the given Event does not exist.
 * @param event Event to delete
 */
public void deleteEvent (EventType event) throws IOException;
```

5.4 Die CalDAV-Anfragen des Prototypen

5.4.1 addEvent

```
PUT <url> HTTP/1.1
[...]
BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//DFKI//javascript CalDav Client//EN
CALSCALE:GREGORIAN
BEGIN:VEVENT
UID:<uuid>
DTSTAMP:<formatDate(now)>
DTSTART:<formatDate(start)>
DTEND:<formatDate(end)>
SUMMARY:<summary>
DESCRIPTION:<description>
END:VCALENDAR
```

Wobei uuid eine generierte Identifikationszeichenkette ist. now, start und end sind Daten, formatDate erstellt aus ihnen das Zeitformat nach der Spezifikation [3]. summary und description sind einfache Zeichenketten. url ist der Bezeichner des Kalenders wie in Server Verbindung aufgeführt.

5.4.2 findEvents

```
REPORT <url> HTTP/1.1
[...]
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
 <C:calendar-query xmlns:C="urn:ietf:params:xml:ns:caldav" xmlns:D="DAV">
  <D:prop>
   <C:calendar-data>
    <C:expand start="<fStart>" end="<fEnd>" />
   </C:calendar-data>#
  </D:prop>
  <C:filter>
   <C:comp-filter name="VCALENDAR">
    <C:comp-filter name="VEVENT">
      <C:time-range start="<fStart>" end="<fEnd>" />
    </C:comp-filter>
   </C:comp-filter>
  </C:filter>
</C:calendar-query>
```

Wobei fStart und fEnd die nach Spezifikation formatierten Daten sind.

5.4.3 findEventsByReminders

Implementiert das Finden ähnlich findEvents. Normalerweise wäre es möglich einen Filter für Start und Ende der Erinnerungen zu verwenden, jedoch hat der Google Calendar diese Funktion im Januar 2013 plötzlich nicht mehr unterstützt.

5.4.4 deleteEvent

```
DELETE <url+uuid>.ics HTTP/1.1
```

url+uuid bezeichnet die Konkatenation des Kalenderbezeichners und der Identifikationszeichenkette des Ereignisses (auf Ebene der Applikation Termin genannt).

5.5 Bilder des Prototypen

5.5.1 Vor der Anpassung

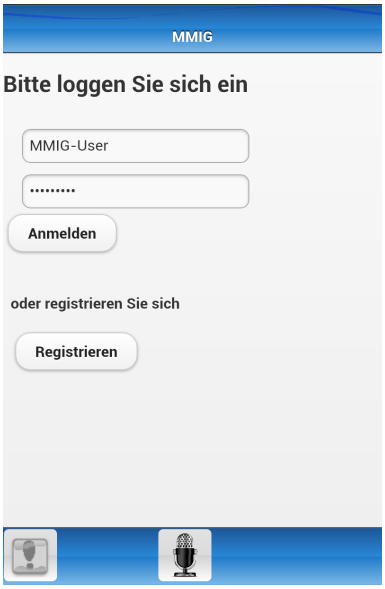
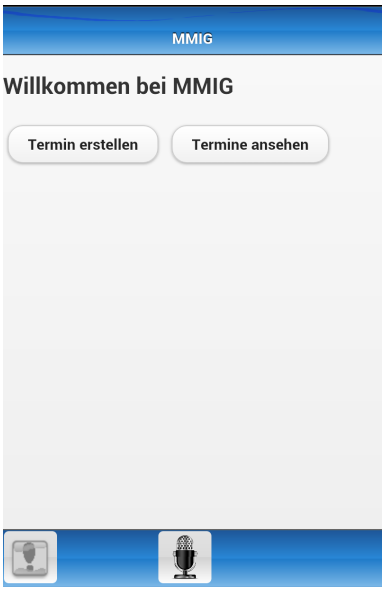


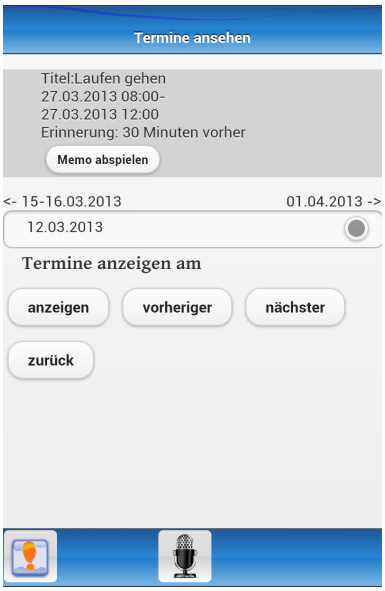
Abbildung 1: der Login-Bildschirm, dieser wurde mir gegeben

5.5 Bilder des Prototypen



(a) Der Hauptbildschirm

(b) Der “Termin erstellen”-Bildschirm



(c) Der “Termine ansehen”-Bildschirm

(d) Die Erinnerungsanzeige

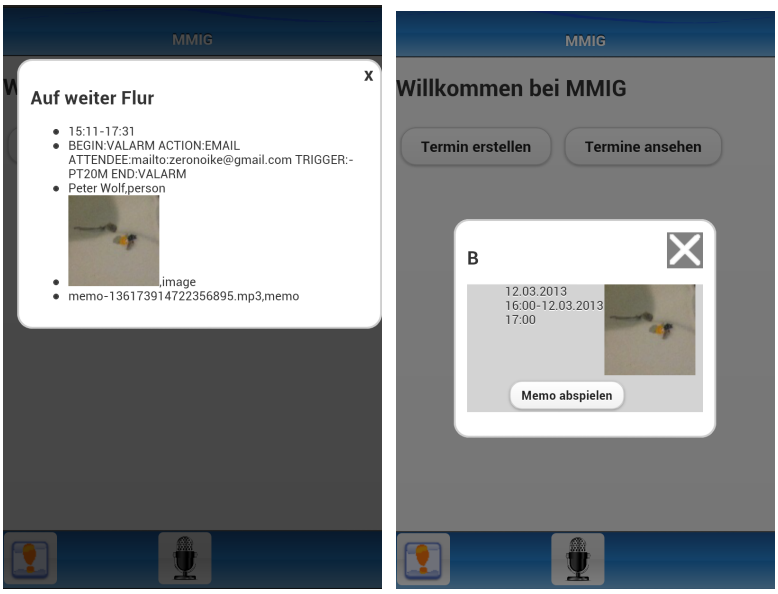
Abbildung 2: Die Bildschirme; jeweils unten links im Bild ist der Erinnerungs-Button zu sehen

5.5.2 Das Datums-Widget

(a) mit Uhrzeit (b) Ohne Uhrzeit

Abbildung 3: Das Datums-Widget, in der Form, die ich verwendet habe

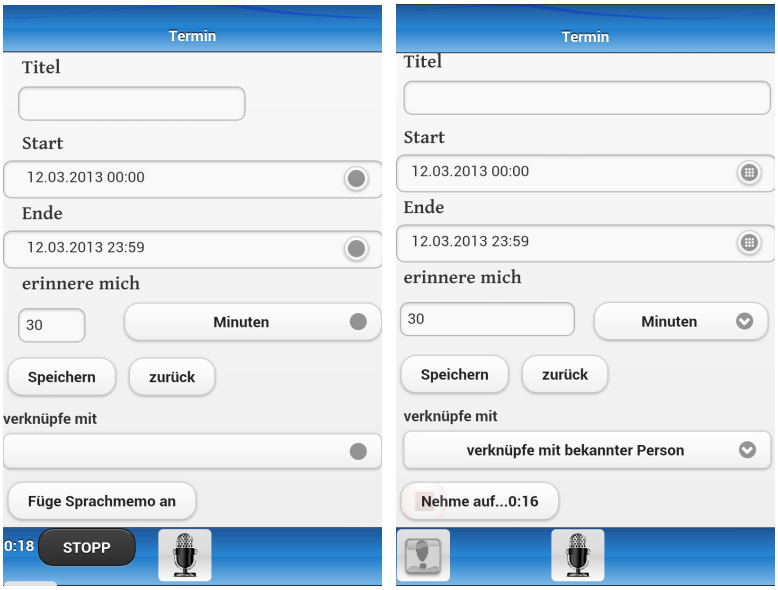
5.5.3 Das Popup - vorher - nachher



(a) altes Popup (b) neues Popup

Abbildung 4: Das Popup, das bei Erinnerungen verwendet wird

5.5.4 Die Sprachmemoaufnahme - vorher - nachher

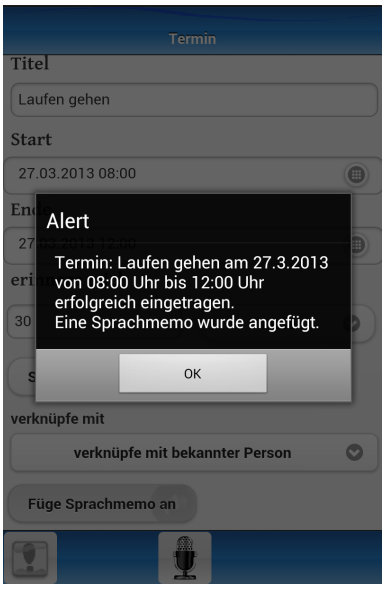


(a) alte Variante der Aufnahme mit zusätzlichen Button

(b) neue Variante - alles in einem Bedienelement

Abbildung 5: Die Sprachmemoaufnahme und meine Änderung im Vergleich

5.5.5 Einige Rückmeldungen der Applikation nach der Anpassung



(a) erfolgreiches Eintragen

(b) Speichern einer Sprachmemo

(c) Ändern einer Sprachmemo

(d) Netzwerkfehler

Abbildung 6: Beispiele von Rückmeldungen, die die Applikation erzeugt

5.5.6 Die Tagesübersicht

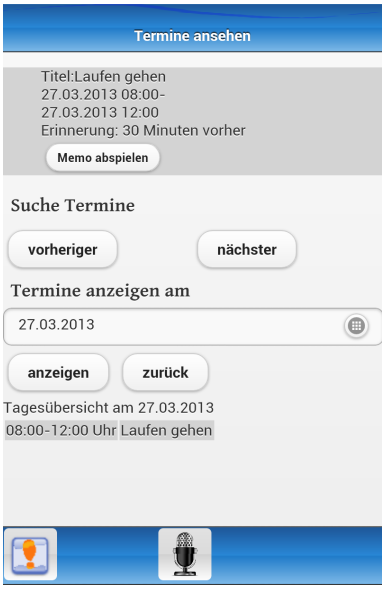


Abbildung 7: Der Bildschirm “Termine ansehen” nach der Anpassung, nun mit Tagesübersicht und besser angeordneten Buttons

5.6 Die Szenarien der Studie

Szenario 1 - Einloggen Nach dem Start der Applikation sehen Sie den Login-Bildschirm. Username und Passwort sind schon eingetragen, Sie brauchen nur noch zu bestätigen. Der nächste Bildschirm ist der Willkommensbildschirm. Was denken Sie, müssen Sie tun, um einen neuen Termin einzutragen? Was glauben Sie, außerdem mit den Schaltflächen machen zu können?

Szenario 2 - Termin eintragen Ein nahe wohnender Freund (Peter Wolf) hat Sie gebeten heute seine Blumen zu gießen. Um dies nach einem weiteren harten Arbeitstag nicht zu vergessen, tragen Sie sich einen Termin heute von 18:20 Uhr bis 19:45 Uhr ein.

Titel Blumen gießen

Start <heute> 18:20 Uhr

Ende <heute> 19:45 Uhr

erinnere mich 30 Minuten vorher

Szenario 3 - Einen verknüpften Termin eintragen Melanie Hoppe hat am 13. März Geburtstag und Sie sagten, sie wollten sie dann nach der Arbeit anrufen. Zudem wollten Sie ihr sagen, dass Sie das Geschenk zusammen abholen können, weil Sie dies vorhin vom Händler erfahren haben.
Tragen Sie bitte diesen Termin ein und verknüpfen ihn mit Melanie Hoppe, um gleich an ihren Eintrag im Adressbuch weitergeleitet werden zu können. Fügen Sie zudem eine Sprachmemo hinzu, dass Sie ihr außer dem Glückwunsch noch sagen, dass Sie zusammen das Geschenk abholen gehen werden.

Titel Melanies Geburtstag

Start 13.03.2013 16:30

Ende 13.03.2013 16:30

erinnere mich 30 Minuten vorher

verknüpfe mit Melanie Hoppe

Szenario 4 - Termine finden und durch sie navigieren

- Sie hatten in letzter Zeit sehr viel Stress und würden sich morgen gern etwas zurücknehmen. In Ihrer Anspannung haben Sie vergessen, ob oder welche Termine Sie morgen haben. Suchen Sie bitte nach morgigen Terminen.
- An einem hängt eine Sprachmemo an. Hören Sie diese an. Woran wollten Sie sich erinnern?

Szenario 5 - Erinnerung erleben Sie wurden an einen Termin erinnert, der demnächst beginnt. Sehen Sie sich diesen noch einmal genauer an. Wie heißt dieser Termin und wann beginnt er? Woran wurden Sie in letzter Zeit noch erinnert?

Szenario 6a - Die andere Steuerung Haben Sie den Mikrofonknopf bemerkt? Versuchen Sie sich an die "Abschlussfeier" am 15. April 2013 von 16Uhr bis 19:30Uhr erinnern zu lassen, indem Sie die Sprachsteuerung verwenden. Sie möchten 6 Stunden vorher Bescheid gesagt bekommen, um genügend Vorbereitungszeit zu haben. Sehen Sie sich den eingetragenen Termin dann auch gleich per Sprachsteuerung an. Wie würden Sie sich die "Unterhaltung" mit der Applikation wünschen/ vorstellen?

- Sagen Sie etwas wie "Erstelle mir bitte einen Termin am 15. April von 16 Uhr bis 19:30 Uhr zum Thema Abschlussfeier"
- Dann "Und erinnere mich bitte 6 Stunden vorher".
- Und dann "bitte speichern"
- Zum Abrufen "bitte zeige meine Termine am 15. April"

Szenario 6b - Die andere Steuerung Wie Sie sicherlich bemerkt haben, gibt es auf jedem Bildschirm auch mehrere Schaltflächen. Versuchen Sie sich an die "Abschlussfeier" am 15. April 2013 von 16Uhr bis 19:30Uhr erinnern zu lassen, indem Sie nur die Schaltflächen verwenden. Sie möchten 6 Stunden vorher Bescheid gesagt bekommen, um genügend Vorbereitungszeit zu haben. Sehen Sie sich den eingetragenen Termin dann auch gleich an.

5.7 Fragebogen (SUS) und dessen Bewertung

Die Aussagen

1. Ich denke, dass ich dieses System gerne häufig nutzen würde.
2. Ich fand das System unnötig komplex.
3. Ich denke, das System war einfach zu benutzen.
4. Ich denke, ich würde die Hilfe eines Technikers benötigen, um das System benutzen zu können.
5. Ich halte die verschiedenen Funktionen des Systems für gut integriert.
6. Ich halte das System für zu inkonsistent.

7. Ich kann mir vorstellen, dass die meisten Leute sehr schnell lernen würden, mit dem System umzugehen.
8. Ich fand das System sehr mühsam zu benutzen.
9. Ich fühlte mich bei der Nutzung des Systems sehr sicher.
10. Ich musste viele Dinge lernen, bevor ich das System nutzen konnte.

(Übersetzung nach [13])

Die Bewertung Die Bewertung der einzelnen Aussagen geschieht nach [10]. Jede Aussage x ($x = 1..10$) wurde durch den Nutzer von 0 bis 4 (auf einer Likert-Skala) bewertet, wobei 0 "nicht zutreffend" und 4 "zutreffend" entspricht. Dieser Wert heißt F_x. Zur Auswertung wird für jede Aussage x ein Ergebniswert E_x bestimmt. Das Ergebnis ergibt sich dann wie folgt:

- für ungerade x ist $E_x = F_x - 1$
- für gerade x ist $E_x = 5 - F_x$
- Das Ergebnis ist die Summe der Ergebniswerte: $\sum_{x=1}^{10} E_x$

5.8 Interviewfragen

Nutzerhintergrund

- Wie lange nutze Sie Smartphones inzwischen? Welche Typen haben Sie bisher verwendet?
- Wofür verwenden Sie Ihren Kalender hauptsächlich?
- Wie lange verwenden Sie ihn inzwischen, haben Sie andere versucht? Wenn ja, welche?

Nutzermeinung

- Was ist das erste, an das Sie bei dieser Applikation denken? Was ist Ihnen als erstes aufgefallen?
- Wenn Sie die Schwierigkeiten nehmen, die Sie hatten, welche wiegen für Sie persönlich besonders schwer?
- Gab es Momente bei der Benutzung, in die Applikation besonders unerwartet reagiert (Sie irritiert) hat? Und welche?
- Empfanden Sie die Oberfläche als ansprechend?

[13] http://www.wi.uni-muenster.de/pi/lehre/ws0809/seminarse2/ausarbeitungen/usabilitytesting.pdf

Nutzerideen

- Gibt es Dinge, die Ihnen spontan einfallen, die Ihnen gefehlt haben oder finden Sie es war alles Nötige vorhanden?
- Im Vergleich zu den Applikationen, die Sie nutzen, was fehlt Ihnen hier?

5.9 Veränderungen nach der ersten Iteration

Es folgen die tatsächlichen Änderungen an der Applikation aufgeschlüsselt nach Problem, das jede Änderung gelöst oder verbessert hat.

Schlüssel	Fat.	Lösungsidee
Bed3	4	Dieses Problem habe ich durch Aktualisierung des Frameworks und Entfernen einer Animation abgeschwächt.
Bed4	2	Beim Memoabspielen wird der Button zum Abspielen auch als Stopp-Button verwendet, Spulen ist der Einfachheit halber nicht implementiert und das wird vermutlich auch kein Problem darstellen. Die alte Technik, die ich bei der Aufnahme der Sprachmemo eingesetzt hatte, verwendete einen zusätzlichen Stopp-Button. Allerdings kann immer nur entweder abgespielt oder gestoppt werden. Also ist diese doppelte Belegung des Buttons in Ordnung. Viele Abspielgeräte verwenden ja auch eine vergleichbare Technik mit dem Unterschied, dass sie pausieren und nicht stoppen.
Ein3	2	Die vorherige Technik wurde durch ein einfaches Bedienelement ersetzt, das Aufnehmen, Abspielen, Löschen, Ändern und Stoppen in sich vereint. Der Standarddialog über Cordova hat wie erwartet nicht funktioniert. Dafür konnte ich die selbe Technik für alle Abspiel-Buttons und diesen Aufnahme-/Abspiel-Button verwenden.
Ein5	3	Es wird nun durch ein Symbol angezeigt, dass eine Sprachmemo angefügt ist. Damit erfüllt die Applikation hier auch Heuristik 1. Wie in Ein3 erwähnt, ist diese nun auch lösch-, abspiel- und änderbar.
Ein6	3	Im “Termin erstellen”-Dialog muss nun explizit angegeben werden, dass der Titel über die Spracheingabe geändert werden soll. Z.B. durch “Titel: ...”
Ein8	2	Wegen der Umstellung auf das Bedienelement aus Ein3 verschwindet nun auch die Fußzeile nicht mehr.
Ein10	4	Falls ein Netzwerkfehler auftritt, fragt die Applikation nun nach, ob die Anfrage wiederholt werden soll und verhält sich damit nach Heuristik 9.
TAb1	3	Es werden alle Termine eines Tags in einer Tagesübersicht angezeigt (behandelt teilweise auch Bed1). Dieses Problem führte auch dazu, dass ein Szenario fehlschlägt.
TAb2	3	Buttons sind nun besser beschriftet und nach Ähnlichkeit gruppiert.
TAb3	2	Eingabefeld enthält nun immer das Datum des aktuell angezeigten Termins

TAb4	2	“keine Termine”-Meldung enthält nun auch das Datum, an dem keine Termine gefunden wurden.
TAb5	2	siehe TAb1
Not1	3	Das Erinnerungspopup enthält nun alle Daten, die auch die Terminansicht anzeigt und überlässt es nicht mehr dem Nutzer sich daran erinnern zu müssen.
Not2	2	Der Schließen-Button funktioniert nun immer. Zudem habe ich meine einfache Popup-Implementierung durch das “simplemodal”[14] ersetzt.
Not3	3	Die Erinnerungen in der Übersicht sind nun chronologisch. Andere Applikation sortieren Termine auch chronologisch. Ich hatte dieses Problem scheinbar nicht bemerkt.
Not5	1	Über den Erinnerungs-Button geht es aus der Erinnerungsübersicht zurück zur Startseite.
SSt1	1	Das Eingesprochene wird immer noch oben angezeigt, allerdings so, dass sich die Bildschirmelemente nicht mehr verschieben. Dadurch, dass sich die Bedienelemente nicht mehr bewegen, wenn es der Nutzer nicht will, ist er auch weniger irritiert.
Wid1	4	Konnte ich zwar nicht lösen, habe das Problem jedoch auf die gleiche Weise wie Bed1 abgeschwächt.
Wid2	1	Die Beschriftung im Datums-Widget ist nun korrekt.
Spr3	1	Die Befehle “Erinnere mich bitte x Stunden vorher daran”, “erinnern x Stunden”, “x Stunden vorher erinnern” werden nun unterstützt.
Spr6	4	Das Fehlverhalten bei der Spracherkennung ist vermutlich aus der sehr hohen CPU-Last entstanden. Nach der Aktualisierung des Frameworks und dem Entfernen einer Animation war sie wieder hervorragend.

[14] (http://www.ericmmartin.com/projects/simplemoda

Literatur

[1] H. Brau und F. Sarodnick, *Methoden der Usability Evaluation: Wissenschaftliche Grundlagen und praktische Anwendungen*. Verlag Hans Huber, Hogrefe AG, 2006.

[2] W3C, "State chart xml (scxml): State machine notation for control abstraction." http://www.w3.org/TR/2012/WD-scxml-20121206/, Dez. 2012.

[3] C. Daboo, B. Desruisseaux, und L. Dusseault, "Calendaring Extensions to WebDAV (CalDAV)." RFC 4791 (Proposed Standard), März 2007. Updated by RFCs 5689, 6638.

[4] Google, "Google calendar caldav developer's guide." https://developers.google.com/google-apps/calendar/caldav, März 2013. besucht 2013-03-06.

[5] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, und L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication." RFC 2617 (Draft Standard), Juni 1999.

[6] Google, "About using caldav to sync ical or sunbird." http://support.google.com/calendar/bin/answer.py?hl=de&answer=99358#sunbird, März 2013. besucht 2013-03-06.

[7] J. H. Martin und D. Jurafsky, "Speech and language processing," 2000.

[8] B. Kahlert, "Verbesserung der Out-Of-Box-Experience in Saros mittels Heuristischer Evaluation und Usability-Tests," Master's thesis, Freie Universität Berlin, Inst. für Informatik, 2011.

[9] C. M. Barnum, *Usability testing essentials: ready, set... test*. Morgan Kaufmann, 2010.

[10] J. Brooke, "Sus-a quick and dirty usability scale," *Usability evaluation in industry*, vol. 189, p. 194, 1996.

[11] K. Johnson, "Smartphone demographics, smartphone penetration highest among users aged 25-34 in the us and europe." http://velositor.com/2012/03/30/smartphone-penetration-highest-among-users-aged-25-34-in-the-us-and-eu5-but-there-is-surprisingly-increased-reach-among-age-55-users-than-age-18-24-users-in-the-eu5/, März 2012. besucht 2013-03-06.

[12] J. Nielsen und R. L. Mack, "Usability inspection methods. 1994."

[13] "What is usability?." http://www.usabilitybok.org/what-is-usability. besucht 2013-03-08.