Institut für Formale Methoden der Informatik

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 99

Neue Bedienkonzepte für mobile Routenplaner

Stefanie Bahle

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Stefan Funke
Betreuer/in:	Dipl.-Inf. Daniel Bahrdt
Beginn am:	16. September 2013
Beendet am:	18. März 2014
CR-Nummer:	H.1.2, D.2.0

Kurzfassung

Bei Routenplanern werden Start und Ziel typischerweise über eine Display-Tastatur eingegeben, was insbesondere bei Smartphones mit kleinem Display mühsam sein kann. Vor allem unter erschwerten Bedingungen, wie während der Fahrt auf dem Fahrrad oder mit Handschuhen, ist eine Eingabe über die Display-Tastatur kaum möglich.

Aus diesem Grund wurden in dieser Arbeit Bedienmethoden entwickelt, welche die Bedienung unter erschwerten Bedingungen erleichtern sollen.

Die Ansätze sind verschiedene Suchen mit zwei, drei oder vier Buttons zur Auswahl des Zielortes sowie eine Methode die die Lautstärketasten des Smartphones nutzt. Hierbei muss der gesuchte Ort alphabetisch eingeordnet werden und der entsprechende Button ausgewählt.

Diese Bedienmethoden wurden in einer Benutzerstudie mit bekannten Eingabemethoden verglichen um festzustellen, ob die neuen Methoden eine bessere Alternative darstellen. Dies konnte zwar nicht bestätigt werden, dennoch sollte ein Weiterverfolgen der zugrunde liegenden Idee nicht ausgeschlossen werden.

Abstract

In route planners start and finish are typically entered via a touchscreen keypad, which can be hard exceptionally using smartphones with a small display. Especially under difficult conditions, such as while riding a bike or wearing gloves, entering a command via the on-screen keyboard is hardly possible.

For this reason, control methods which should lighten the operation under severe conditions, have been developed.

The approaches are different search options with two, three or four buttons to select the destination as well as a method which uses the volume buttons on a smartphone.

In a user study, these control methods were compared with well-known input methods to determine whether the new methods are better alternatives. While this could not be confirmed, a follow-up of the underlying idea should not be excluded.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1. Einleitung

1.1. Motivation

Bei Routenplanern werden Start und Ziel typischerweise über eine (virtuelle) Tastatur eingegeben, was insbesondere bei Smartphones mit kleinem Display mühsam sein kann. Vor allem unter erschwerten Bedingungen wie während der Fahrt auf dem Fahrrad oder mit Handschuhen ist die Eingabe des Zielortes über die Display-Tastatur kaum möglich. Aus diesem Grund wurden Bedienmethoden entwickelt, die die Auswahl des Start- und Zielortes unter erschwerten Bedingungen erleichtern sollen.

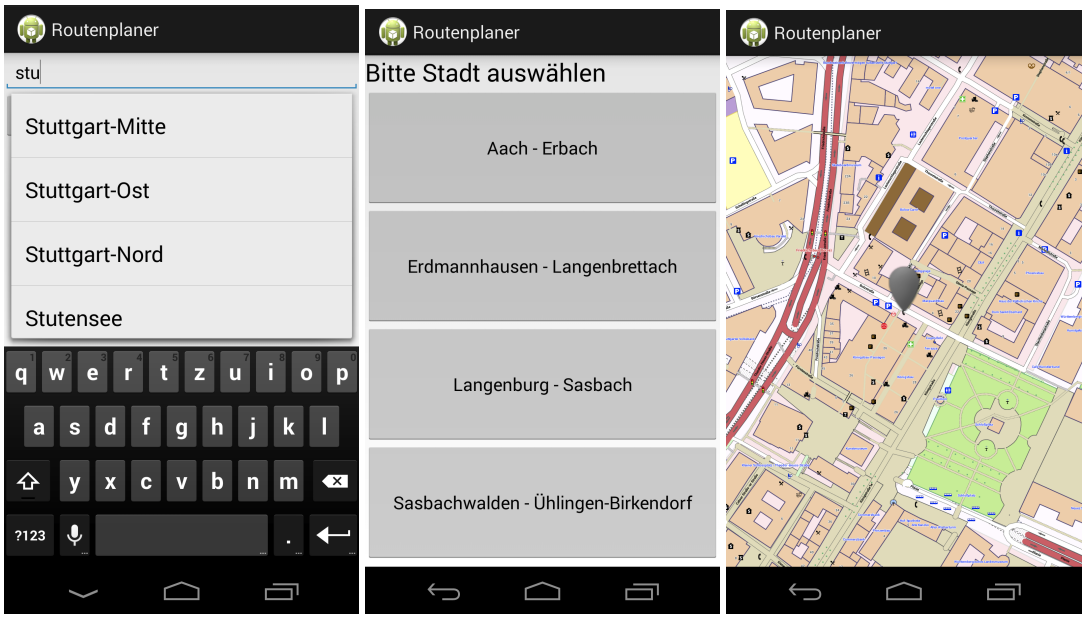


Abbildung 1.1.: Screenshots
links: Eingabe über Tastatur
Mitte: Quaternäre Suche
rechts: Kartenansicht

1.2. Aufgabenstellung

Im Rahmen dieser Bachelorarbeit wurden alternative Bedienkonzepte für die Eingabe von Orten untersucht, wobei insbesondere auf Konzepte eingegangen wurde, die auch auf kompakten Smartphones und unter erschwerten Bedingungen genutzt werden können. Es sollten eine binäre, eine ternäre und eine quaternäre Suche nach Orts- und Straßennamen sowie eine Methode die die Lautstärketasten des Smartphones nutzt implementiert werden. Im Vergleich hierzu sollten außerdem zwei Eingaben über die Android-Display-Tastatur erstellt werden. Um festzustellen ob die neuen Methoden eine Alternative zu bekannten sein könnten, sollten die verschiedenen Bedienmethoden der Android-Anwendung in einer Benutzerstudie verglichen werden.

1.3. Aufbau des Dokuments

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen: Hier werden werden die Grundlagen dieser Arbeit beschrieben.

Kapitel 3 – Implementierung: Dieses Kapitel erläutert die Implementierung der einzelnen Komponenten der Android-Anwendung.

Kapitel 4 – Benutzerstudie: Hier wird die Durchführung sowie die Ergebnisse der Benutzerstudie beschrieben.

Kapitel 5 – Zusammenfassung fasst die Ergebnisse der Arbeit zusammen.

2. Grundlagen

In diesem Kapitel wird zunächst auf verwandte Arbeiten eingegangen, welche sich bereits mit alternativen Eingabemethoden beschäftigt haben, sowie anschließend einige technische Grundlagen, die zum Verständnis der Arbeit notwendig sind, beschrieben.

2.1. Verwandte Arbeiten

In diesem Abschnitt werden verwandte Arbeiten beschrieben, die sich bereits mit alternativen Bedienmethoden an Smartphones beschäftigt haben. In der Arbeit "Usability Evaluation of Text Input Methods for Smartphone among the Elderly" [HN13] von Hamano et al. wird die Usability von verschiedenen Eingabemethoden mit Buchstaben an Smartphones von älteren Nutzern verglichen, mit dem Ziel eine neue Eingabemethode zu entwickeln. Während der Studie hatten ältere Teilnehmer viele Probleme und es war geplant die selbe Studie nochmals mit jungen Teilnehmern durchzuführen um dann eine neue Methode zu entwickeln.

Des Weiteren findet sich eine Arbeit von MacKenzie et al. [MSH11] in welcher die den Autoren zufolge effizienteste und schnellste 4-Tasten-Text-Eingabemethode entwickelt wurde, welche statt mit neun Tasten (bekannter T9-Modus) mit vier Tasten auskommt.

Weiterhin wurden keine Arbeiten gefunden, die sich mit alternativen Eingabemethoden zur qwertz-Tastatur beschäftigen, sondern lediglich mit Alternativen zu CJK-Schriftzeichen (chinesische, japanische und koreanische Schrift). Diese wurden nicht weiter verfolgt.

2.2. Technische Grundlagen

2.2.1. Android

Android ist ein Linux-basiertes Betriebssystem sowie eine Software-Plattform für mobile Endgeräte wie Smartphones oder Tablets. Android wird von der Open Handset Alliance, deren Hauptmitglied Google ist, entwickelt.

Android ist eine freie Software, die quelloffen entwickelt wird. Die bevorzugte Lizenz des Projekts ist Apache 2.0, unter welcher ein Großteil der Android-Software steht. [anda]

Die Architektur von Android baut auf einem Linux-Kernel auf, welcher für Speicher- und Prozessverwaltung zuständig ist. Die Laufzeitumgebung von Android basiert auf der Dalvik Virtual Machine.

Dalvik ähnelt der Java Virutal Machine, allerdings unterscheiden sie sich in der zugrundeliegenden virtuellen Prozessorarchitektur.

Anwendungen für die Android-Plattform werden in der Regel in Java geschrieben. Um eigene Programme für Android zu entwickeln, wird neben dem Java-SDK zusätzlich das Android-SDK benötigt. Die fertige Anwendung muss in ein *.apk-Paket (Android Package) gepackt werden und kann anschließend entweder direkt oder über einen Anwendungs-Shop auf einem Android-Gerät installiert werden.[andb]

2.2.2. Mapsforge

Das mapsforge-Projekt stellt freie Software (unter GNU Lesser GPL) für OpenStreetMap basierte Anwendungen bereit.

OpenStreetMap ist ein Projekt mit dem Ziel einer freien Weltkarte. Es werden weltweit Daten über alles gesammelt, das auf Karten zu sehen ist. OpenStreetMap-Daten dürfen lizenzkostenfrei eingesetzt und beliebig weiterverarbeitet werden. [osm]

Seitens mapsforge wird eine Bibliothek für sofortiges Kartenrendering auf Android-Geräten angeboten, welche für die im Laufe der Arbeit erstellte Anwendung in Version 0.3.0 genutzt wurde und unter [map] zu finden ist.

2.2.3. Karten-Daten

Die Kartendaten liegen als CSV-Datei (comma seperated value) vor. Dabei besteht eine Zeile aus "Breitengrad;Längengrad;Straße;Stadt". Die für die Studie sowie als Testdaten genutzten Kartendaten stammen von OpenStreetMap. Da hier Straßen nicht direkt Ortschaften zugeordnet sind, was aber für die Anwendung notwendig ist, wurden diese mit Hilfe von OsmFind [Bah13] zusammen mit den Koordinaten erstellt und als CSV-Datei gespeichert. Alternativ können die Kartendaten für Baden-Württemberg auch der Website des Landesamt für Geoinformation und Landentwicklung Baden-Württemberg [LGL] entnommen werden.

3. Implementierung

Diese Kapitel beschäftigt sich mit der Implementierung der Android-Anwendung. Zunächst wird eine kurze Übersicht über den Ablauf einer Zieleingabe gegeben um anschließend auf die einzelnen Komponenten einzugehen. Es werden die Datenverwaltung von Stadt, Straße und Koordinaten beschrieben und darauffolgend auf genutzte Algorithmen und die Benutzeroberfläche eingegangen.

Der komplette Quellcode sowie das Android Package (*.apk) der Anwendung sind dieser Arbeit als Anhang beigelegt.

3.1. Übersicht

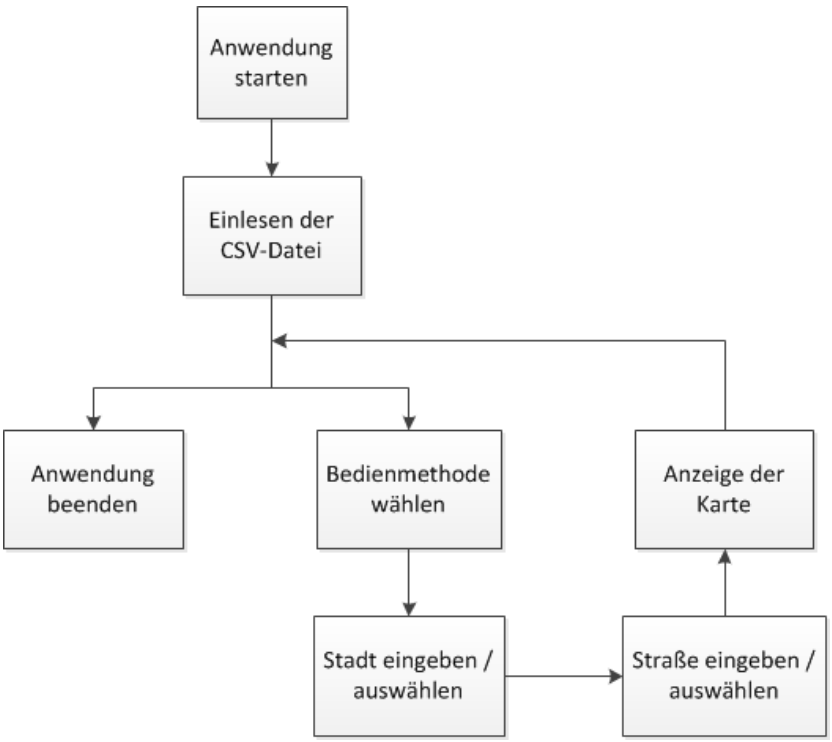


Abbildung 3.1.: Workflow der Anwendung

In Abbildung 3.1 wird eine Übersicht über den Workflow der Anwendung gegeben.

Nach Starten der Anwendung werden die Kartendaten (CSV-Datei) eingelesen. Anschließend wird die Bedienmethode gewählt, dann Stadt und Straße entsprechend ausgewählt und auf der Karte angezeigt. Die Anwendung kann vom Auswahlscreen beendet werden.

Die Androidanwendung wurde mit den Android Developer Tools in Version 22.0.1. entwickelt. Hier sind das Android SDK (Version 22.3) und Eclipse in Version 4.2 (Juno) enthalten. Die Anwendung wurde während des Implementierens auf einem Google Nexus 5 ausgeführt und getestet.

3.2. Datenverwaltung

Die Daten zu Ort, Straße, Längengrad und Breitengrad müssen für die Anwendung als CSV-Datei vorliegen. Es muss immer folgendes Format eingehalten werden: *Breitengrad;Längengrad;Straße;Stadt*. Ist dies nicht der Fall, so muss entweder die CSV-Datei oder die in Listing 3.1 dargestellten Codezeilen der Klasse "FileReader.java" entsprechend angepasst werden. Die CSV-Datei muss sich im Stammverzeichnis des externen Speichers des Smartphones befinden und den Dateinamen "bw.csv" haben. Ist dies nicht der Fall so können Pfad und Dateiname in der Klasse *FileReader.java* angepasst werden.

Listing 3.1 Ausschnitt der Klasse FileReader.java

```
// lat : 0 | lon : 1 | street : 2 | town : 3
if (!townList.containsKey(arr[3])) {
      t = new Town(arr[3]);
      townList.put(arr[3], t);
} else {
      t = townList.get(arr[3]);
}
Street street = new Street(arr[2], Float.valueOf(arr[1]),
Float.valueOf(arr[0]));
```

Die Daten der CSV-Datei werden in der Klasse FileReader.java eingelesen.

Die CSV-Datei wird mit Hilfe eines BufferedReader [buf] zeilenweise eingelesen und in verschieden Datenobjekten gespeichert. Ein BufferedReader liest Text über einen *character-input stream* ein, er puffert dabei Buchstaben (characters) um zum Beispiel *characters* und *arrays* effizient einzulesen. Die Struktur wird im Folgenden beschrieben und bezieht sich auf Abbildung 3.2.

Die Daten werden in einer HashMap [has] gespeichert. Eine HashMap dient zum Speichern von *Key-Value*-Paaren (Schlüssel-Wert) sowie zum Auslesen eines *values* mit Hilfe des *key*.

Als Struktur wurde eine bzw. wurden zwei HashMaps gewählt, da diese im Vergleich zu einer ArrayList einige Vorteile haben sowie die Nachteile nicht relevant sind. Bei einer HashMap ist im Vergleich zu einer ArrayList keine Ordnung der Elemente gegeben, wodurch ein Iterieren über alle Elemente bedeutend komplizierter ist als bei einer ArrayList. Die Vorteile der HashMap sind, dass hier ein Eintrag schnell gefunden werden kann sofern der key bekannt ist [Ull11]. Da dies immer der Fall ist, und weder eine Ordnung der Elemente benötigt wird noch über die Liste iteriert wird, wurde eine HashMap gewählt.

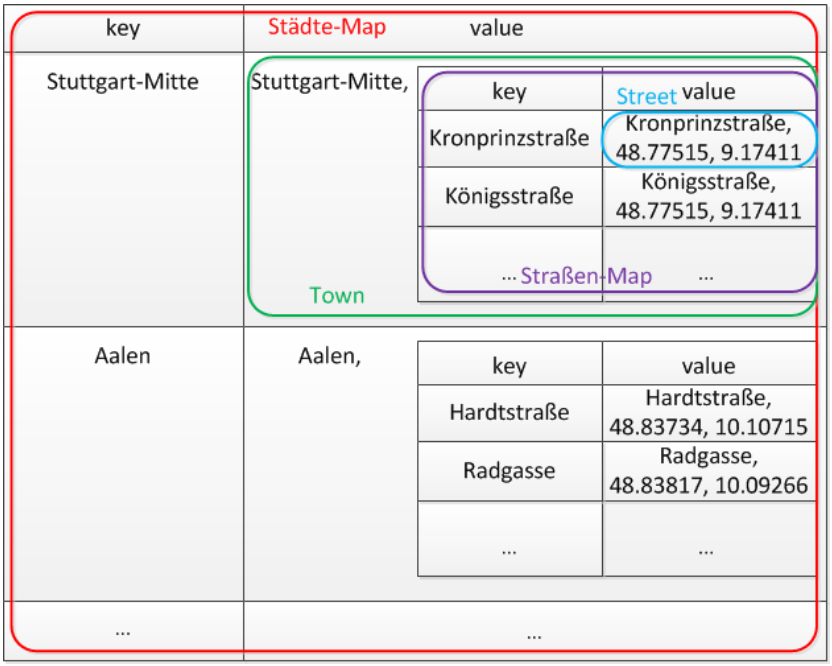


Abbildung 3.2.: Aufbau der Datenhaltung

Es wird ein *key* angegeben zu welchem ein *value* gespeichert wird. Der *key* ist üblicherweise eine ID oder ein String und muss eindeutig sein, es dürfen also nicht mehrere *values* mit dem gleichen *key* eingepflegt werden. Da dies allerdings notwendig ist (eine Stadt hat mehrere Straßen), wurde es folgendermaßen gelöst:

Es wird eine *HashMap* für Städte (Städte-Map) erstellt, welche als *key* den Stadtnamen und als *value* ein Objekt vom Typ *Town* hat. Das Objekt des Typs *Town* besteht aus einem String (erneut dem Stadtnamen) sowie einer weiteren *HashMap* (Straßen-Map) mit dem Straßennamen als *key* und einem Objekt vom Typ *Street* als *value*. Das *Street*-Objekt fasst einen String, den Straßennamen, sowie zwei Werte vom Typ float, die Längen- und Breitengrad darstellen (lon und lat).

Die *HashMap* verhindert, dass Städte oder Straßen mehrfach eingelesen und überschrieben werden.

Der Zugriff auf die Daten erfolgt von allen Klassen über die Klasse DataHolder, welche das Singleton-Entwurfsmuster realisiert. Hierdurch wird garantiert, dass es nur eine Instanz der Klasse geben kann, was dafür sorgt, dass die Städte-Map nur einmal erzeugt wird.

In der Klasse DataHolder sind Methoden angelegt, welche ein Array mit allen Städten (*getTowns()*) oder ein Array mit allen Straßen einer Stadt (*getStreets(String town)*) zurückgeben. Diese Methoden werden in der Implementierung jeder Bedienmethode aufgerufen, wenn die entsprechende Liste benötigt wird.

3.3. Algorithmen

3.3.1. Eingabe über Tastatur mit autocomplete

Für die Autovervollständigungsfunktion stellt Android selbst eine API zur Verfügung, sodass hier kein Algorithmus implementiert werden musste.

Eine *autocompletetextview* ist ein editierbares Textfeld, welches automatisch Vorschläge zur Vervollständigung liefert während der Nutzer schreibt. Die Liste der Vorschläge wird in einem Dropdown-Menü dargestellt, aus welchem der Nutzer ein Element wählen kann, welches dann als Inhalt des Textfeldes übernommen wird. [aut]

Die Liste der Vorschläge wird einem Datenadapter entnommen. Dieser Adapter dient als Brücke zwischen einer View und den dahinterliegenden Daten der View. Der Adapter realisiert Zugriff auf die Datenelemente.

Ein Beispiel für die Nutzung der AutoCompleteTextView findet sich in Listing 3.2. Es wird ein ArrayAdapter *adapter* mit einem Array von Strings erstellt und mit den Ländern (COUNTRIES) befüllt. Außerdem wird eine AutoCompleteTextView *textview* initialisiert, an welche anschließend der Adapter gefügt wird.

Listing 3.2 Beispielcode für die Nutzung der Android-API AutoCompleteTextView [aut]

```
public class CountriesActivity extends Activity {
    protected void onCreate(Bundle icicle) {
       super.onCreate(icicle);
       setContentView(R.layout.countries);

       ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
             android.R.layout.simple_dropdown_item_1line, COUNTRIES);
       AutoCompleteTextView textView = (AutoCompleteTextView)
             findViewById(R.id.countries_list);
       textView.setAdapter(adapter);
    }

    private static final String[] COUNTRIES = new String[] {
       "Belgium", "France", "Italy", "Germany", "Spain"
    };
 }
```

3.3.2. k-näre Suchen

Die k-nären Suchen verfolgen folgende Idee:

Alle Städte bzw. alle Straßen einer Stadt sind in einer eigenen Liste gespeichert. Zu Anfang wird in der Liste aller Städte die gewünschte Stadt gesucht. Nach Auswahl der Stadt wird die Straße gesucht.

Im folgenden Unterkapitel wird auf die einzelnen Suchen (binäre, ternäre und quaternäre Suche) nach der jeweiligen Stadt eingegangen, die Suche nach der Straße erfolgt äquivalent nach Auswahl der Stadt.

Für die Suche nach einer Stadt wird die alphabetisch sortierte Liste aller Städte in k Unterlisten geteilt. Der Nutzer wählt nun über einen Button eine Unterliste aus (in welcher sich die gesuchte Stadt befindet) und diese wird wiederum in k Unterlisten geteilt. Auf diese Weise wird die Liste so lange geteilt, bis dies nicht mehr trivial möglich ist. Bei k = 2 tritt dies ab einer Listenlänge kleiner gleich drei ein, bei k = 3 ab einer Listenlänge kleiner gleich fünf und bei k = 4 kleiner gleich sieben.

Binäre Suche

Bei der binären Suche wird die gesamte Liste immer in zwei Unterlisten geteilt. Die verwendeten Listen sind einfache Arrays. Arrays können nicht geteilt werden, stattdessen werden mit Hilfe von System.arraycopy [arr] neue Listen (Unterlisten) erstellt.

Sofern die Liste eine ohne Rest durch zwei teilbare Länge hat, ist dies problemlos möglich. Hier werden zwei neue Listen erstellt:

- Die erste Unterliste besteht aus den Städten an den Stellen 0 bis ausschließlich *Länge / 2*.
- Die zweite Unterliste besteht aus den Städten an den Stellen *Länge / 2* bis ausschließlich *Länge*.

In Abbildung 3.3 (links) ist ein Beispiel für eine Liste mit vier Städten dargestellt. Die Länge der Liste wird durch zwei geteilt, das Ergebnis ist *mezzo* = 2. Nun werden aus der ursprünglichen Liste zwei neue Listen erstellt, die erste enthält die Stellen 0 und 1 der ursprüngliche Liste, die zweite Unterliste die Stellen 2 und 3.

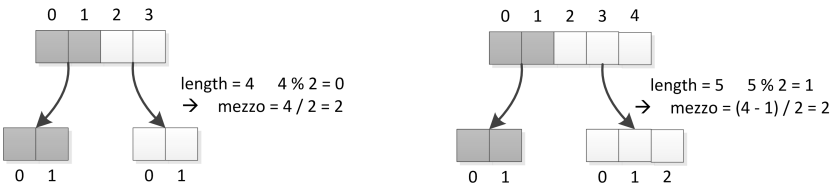


Abbildung 3.3.: Schematische Darstellung der Aufteilung der Listen bei der binären Suche

Ist die Länge der Liste nur mit Rest durch zwei teilbar, so werden folgende Unterlisten gebildet:

- Die erste Unterliste beinhaltet die Städte an den Stellen 0 bis ausschließlich *(Länge - 1) / 2*

- Die zweite Unterliste beinhaltet die Städte an den Stellen *(Länge - 1) / 2* bis ausschließlich *Länge*
  Ein Beispiel für eine Liste der Länge fünf ist in Abbildung 3.3 (rechts) dargestellt.

Auf diese Art und Weise wird die Liste immer entsprechend weiter aufgeteilt, und die neu erhaltene Liste wieder. Dies wird bis zu einem Wert größer drei durchgeführt, anschließend werden die Fälle einzeln behandelt:

- Länge der Liste = 3
  Zwei Unterlisten:
  1. Stelle 0 und 1 der ursprünglichen Liste
  2. Stelle 2 der ursprünglichen Liste
- Länge der Liste = 2
  Zwei Unterlisten:
  1. Stelle 0 der ursprünglichen Liste
  2. Stelle 1 der ursprünglichen Liste

Hat eine Liste die Länge eins, so wird die Stadt als gesuchte Stadt weiterverarbeitet. Dies gilt für alle k-nären Suchen.

Ist die Stadt gefunden, so wird die Straßensuche mit allen Straßen dieser Stadt äquivalent zur Städtesuche druchgeführt. Die Straße wird dann auf einer Karte dargestellt.

Ternäre Suche

Im Prinzip wurde die ternäre Suche gleich implementiert wie die binäre Suche, wobei die Liste nun immer in drei Unterlisten geteilt werden muss. Dadurch ändert sich die Länge ab der die Liste besonders behandelt werden muss ebenso wie die Anzahl der Sonderfälle. Nun ergibt sich beim Teilen der Liste durch drei entweder der Rest null, eins oder zwei. Der Code zum Erstellen der Unterlisten wird in Listing 3.3 dargestellt. Je nachdem ob die Länge mit Rest 0, 1 oder 2 durch 3 geteilt wird, wird der Wert für *third* unterschiedlich berechnet. Anschließend werden solange die Listenlänge großer als fünf ist die Unterlisten dynamisch erstellt.

Listing 3.3 Ausschnitt aus TernaryActivity.java: Teilen der Listen bei der ternären Suche

```
if (towns.length % 3 == 0) {
                  third = towns.length / 3;
            } else if (towns.length % 3 == 1) {
                  third = (towns.length - 1) / 3;
            } else { // if (towns.length % 3 == 2
                  third = (towns.length - 2) / 3;
            }

if (towns.length > 5) {

                  towns1 = Arrays.copyOfRange(towns, 0, (third));
                  towns2 = Arrays.copyOfRange(towns, third, (third * 2));
                  towns3 = Arrays.copyOfRange(towns, (third * 2), (towns.length));
// ...
}
```

Sobald die Listenlänge fünf oder kleiner ist können die Unterlisten nicht mehr dynamisch erstellt werden sondern müssen als Sonderfälle einzeln behandelt werden. Wie die Liste in diesen Fällen geteilt wird, zeigt Abbildung 3.4. Bei einer Listenlänge von 5 (Abbildung 3.4 rechts unten) haben die ersten beiden Listen die Länge 2, die dritte Liste die Länge 1. Bei einer Listenlänge von 4 (Abbildung 3.4 links unten) fasst die erste Unterliste zwei Elemente, die zweite und dritte jeweils eines. Hat die Liste die Länge 3 (Abbildung 3.4 rechts oben), so entstehen drei Unterlisten mit je einem Element. Bei einer Liste mit zwei Elementen entstehen zwei Listen mit einem Element, eine dritte Liste existiert nicht. Dies wurde in der Abbildung durch $\perp$ gekennzeichnet.

Ansonsten verhält sich die ternäre Suche wie die bereits in Kapitel 3.3.2 (Binäre Suche) beschrieben.

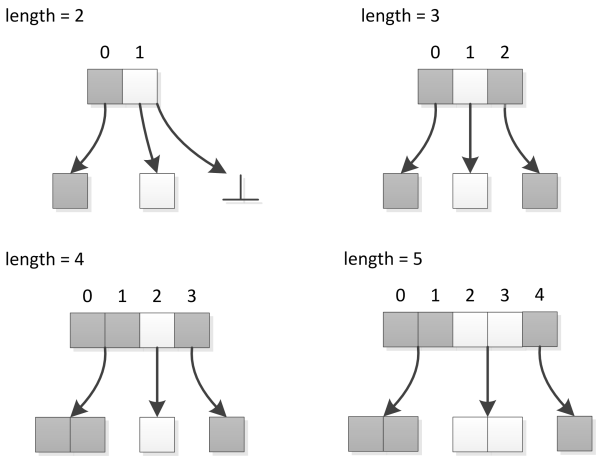


Abbildung 3.4.: Schematische Darstellung der Aufteilung der Listen in den Sonderfällen der ternären Suche

Quaternäre Suche

Auch die quaternäre Suche entspricht im Prinzip der binären und ternären Suche, mit dem Unterschied dass eine Liste nun in vier Unterlisten unterteilt wird und Sonderfälle ab einer Listenlänge von sieben behandelt werden müssen.

In Listing 3.4 ist die Aufteilung der Listen dargestellt. Wie bereits bei binärer und ternärer Suche wird auch hier ein Wert, in diesem Fall *quarter* abhängig vom Ergebnis (Listenlänge mod 4) berechnet. Anschließend werden die vier Unterlisten in Abhängigkeit zu *quarter* erstellt.

In Abbildung 3.5 befindet sich die Behandlung der Sonderfälle, welche die Listenlängen sieben bis zwei betreffen. Wie bereits bei der ternären Suche ist $\perp$ das Zeichen für eine leere bzw. nicht existente Liste.

Listing 3.4 Ausschnitt aus QuaternaryActivity.java: Teilen der Listen bei der quaternären Suche

```
if (towns.length \% 4 == 0) {
                  quarter = towns.length / 4;
            } else if (towns.length % 4 == 1) {
                  quarter = (towns.length - 1) / 4;
            } else if (towns.length % 4 == 2) {
                  quarter = (towns.length - 2) / 4;
            } else { // if (towns.length % 4 == 3)
                  quarter = (towns.length - 3) / 4;
            }

if (towns.length > 7) {

            towns1 = Arrays.copyOfRange(towns, 0, (quarter));
            towns2 = Arrays.copyOfRange(towns, quarter, (quarter * 2));
            towns3 = Arrays.copyOfRange(towns, (quarter * 2), (quarter * 3));
            towns4 = Arrays.copyOfRange(towns, (quarter * 3), (towns.length));
// ...
}
```

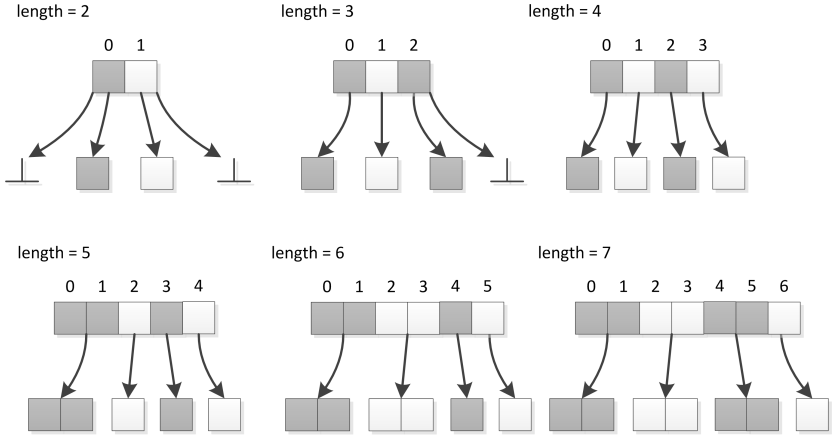


Abbildung 3.5.: Schematische Darstellung der Aufteilung der Listen in den Sonderfällen der quaternären Suche

3.4. Benutzeroberfläche

In der Android-Entwicklung wird für das Erstellen und Agieren mit Oberflächen eine spezielle Java-Klasse, eine Activity benötigt. Eine Activity ist eine *Kontrollinstanz* die die Darstellung von Texten, Schaltflächen und Menüoptionen übernimmt sowie auf Eingaben des Anwenders reagiert. Für jede Bildschirmseite (Screen) muss eine eigene Activity implementiert werden. [BP09]

Jedes Element das auf dem Screen dargestellt wird wird von der Klasse *android.view.View* abgeleitet. Es gibt spezielle Views, sogenannte *ViewGroups*, welche andere Views enthalten können.

Ein Screen ist ein Baum von Views, welcher als XML-Datei erstellt wird. Ein Bildschirmdialog besteht aus einem Screen, also aus der XML-Datei, welche mit einer Activity, einer Java-Klasse, verbunden ist.

Ein Screen kann mit Hilfe einer XML oder mit Java-Code implementiert werden, auch eine Kombination von XML und Java-Code ist möglich.

In der Anwendung wurde die Oberfläche soweit möglich in XML erstellt und lediglich die dynamischen Teile als Java-Klasse. Dies betrifft die Beschriftung der Buttons bei binärer, ternärer und quaternärer Suche.

Ein Beispiel für den Aufbau einer Bildschirmseite in XML findet sich in Listing 3.5. Hier wird zunächst ein LinearLayout erstellt, in welches eine TextView (Anzeige eines feststehenden Textes) und zwei Buttons eingefügt werden. Den Buttons wird in der XML-Datei kein String als Text zugewiesen, da dieser dynamisch in der BinaryActivity.java vergeben wird. Der hierzu notwendige Code findet sich in Listing 3.6.

Listing 3.5 ActivityBinary.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >

   <TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/WaehleStadt"
      android:textAppearance="?android:attr/textAppearanceLarge"
      android:textSize="25.0sp" />

   <Button
      android:id="@+id/binaryButton1"
      android:layout_width="fill_parent"
      android:layout_height="180dp"
      android:onClick="onClick" />

   <Button
      android:id="@+id/binaryButton2"
      android:layout_width="fill_parent"
      android:layout_height="180dp"
      android:onClick="onClick" />

</LinearLayout>
```

Listing 3.6 Ausschnitt aus BinaryActivity.java

```
Button binaryButton1 = (Button) findViewById(R.id.binaryButton1);
String button1 = towns1[0] + " - " + towns1[(towns1.length - 1)];
binaryButton1.setText(button1);

Button binaryButton2 = (Button) findViewById(R.id.binaryButton2);
String button2 = towns2[0] + " - " + towns2[(towns2.length - 1)];
binaryButton2.setText(button2);
```

Die Anwendung ist so aufgebaut, dass nach dem Laden der Anwendung, bei welchem ein Lade-Screen angezeigt wird, zunächst ein Auswahlscreen dargestellt wird, in welchem die Bedienmethode ausgewählt werden muss. Dieser findet sich in Abbildung 3.6 (links).

Je nachdem welche Bedienmethode gewählt wurde, findet sich der Nutzer dann bei einer der Bedienmethoden:

- Eingabe über Tastatur mit autocomplete (siehe Abbildung 3.6, Mitte)
- Eingabe über Tastatur one autocomplete (siehe Abbildung 3.6, rechts)
- Binäre Suche (siehe Abbildung 3.7, links)
- Ternäre Suche (siehe Abbildung 3.7, Mitte)
- Quaternäre Suche (siehe Abbildung 3.7, rechts)

Nachdem der gesuchte Ort gewählt wurde, wird dieser mit Hilfe von mapsforge auf einer Karte angezeigt. Die mapsforge-Karte für den entsprechenden Bereich muss sich im Stammverzeichnis des externen Speichers des Smartphones befinden und den Dateinamen "baden-wuerttemberg.map" haben. Ist dies nicht der Fall, so können Verzeichnis und Dateiname in der Klasse *DisplayMapActivity.java* angepasst werden.

Abbildung 3.6.: Screenshots
links: Auswahlscreen
Mitte: Eingabe über Tastatur mit autocomplete
rechts: Eingabe über Tastatur ohne autocomplete

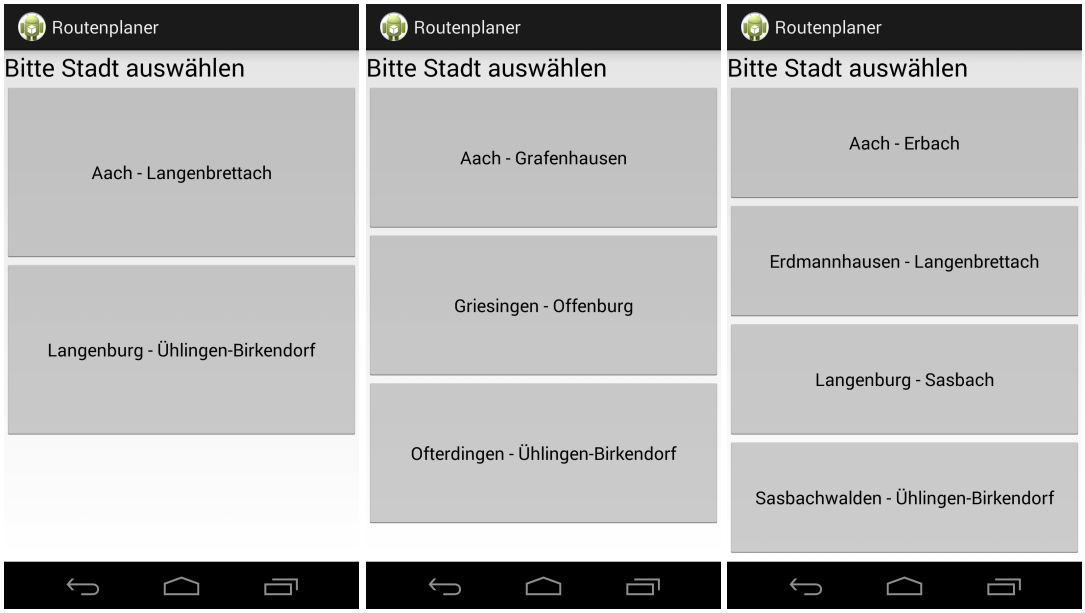


Abbildung 3.7.: Screenshots
links: Binäre Suche
Mitte: Ternäre Suche
rechts: Quaternäre Suche

4. Benutzerstudie

In der Benutzerstudie wurden die sechs implementierten Bedienmethoden von freiwilligen Teilnehmern getestet und bewertet.

In diesem Kapitel wird zunächst auf das Studiendesign sowie die Bewertungsmethoden der Studie eingegangen. Diese bestehen neben der Zeitmessung aus zwei Fragebögen, welche vom Teilnehmer auszufüllen sind. Der erste Fragebogen dient zur Untersuchung der subjektiven Usability (System Usability Scale), der zweite zur Bewertung der Anforderungen (NASA Task Load Index). Es wird die Durchführung der Studie beschrieben sowie die ausgewerteten Ergebnisse dargestellt und erläutert.

In diesem Kapitel wird zunächst vorgestellt, wie die Eingabemethoden getestet wurden sowie auf die Bewertungsmethode eingegangen. Darauf folgend wird die Durchführung der Studie beschrieben. Anschließend werden die erwarteten sowie ausgewerteten Ergebnisse dargestellt sowie abschließend diskutiert.

4.1. Methode

4.1.1. Design

In der Studie musste jeder Teilnehmer die selben Aufgaben für jede Bedienmethode lösen, da sonst nicht sichergestellt werden kann, dass die Aufgaben sonst äquivalent sind. Es wurden alle Aufgaben für eine Bedienmethode gelöst, anschließend alle Aufgaben für die nächste Methode. Analog wurden alle Methoden getestet. Da die Ergebnisse einer Eingabemethode möglicherweise von einer bereits durchgeführten beeinflusst werden sowie unter Umständen ein Lerneffekt auftreten kann, darf die Reihenfolge des Testens der Eingabemethoden nicht bei jedem Teilnehmer die selbe sein. Diese Reihenfolge wurde durch ein sogenanntes "Latin Square"festgelegt.

Balanced Latin Squares

Ein "balanced Latin Square" ist die ausbalancierte und damit bessere Version des "Latin Square". In 4.1 ist ein *balanced Latin Square* für vier mögliche Testzustände dargestellt. Bei vier Testzuständen werden vier gleich große Testgruppen benötigt, um jede Zustandsfolge gleich häufig durchzuführen. Die Zahl der Teilnehmer muss also durch vier teilbar sein.

Auf diese Weise kommt jeder Zustand genau einmal in jeder Reihe und Zeile vor, was auch bei einem Latin Square der Fall ist, zusätzlich kommt im *balanced Latin Square* jeder Zustand gleich häufig und nach jedem anderen Zustand vor.

A	B	D	C
B	C	A	D
C	D	B	A
D	A	C	B

Tabelle 4.1.: 4 x 4 balanced Latin Square

Da in der Studie sechs verschiedene Bedienmethoden untersucht wurden, war hier ein 6 x 6 balanced Latin Square (siehe 4.2) sowie eine durch sechs teilbare Anzahl an Teilnehmern notwendig.

A	B	F	C	E	D
B	C	A	D	F	E
C	D	B	E	A	F
D	E	C	F	B	A
E	F	D	A	C	B
F	A	E	B	D	C

Tabelle 4.2.: 6 x 6 balanced Latin Square

System Usability Scale

Der System Usability Scale (kurz: SUS) ist ein im Jahre 1986 von John Brook entwickelter Fragebogen [Bro96]. Er umfasst zehn Fragen mit einer Antwortskala von eins bis fünf ("Stimme überhaupt nicht zu" bis "Stimme voll zu"). Das Ergebnis der SUS soll einen allgemeinen groben Überblick über die subjektive Usability geben. Die Fragestellungen sind stark an die Definition von Usability der EN ISO 9241 Norm angelehnt. Im Original ist der Fragebogen nur als englische Version zu finden, er wurde aber auch ins Deutsche übersetzt [Rau11]. Der in der Studie genutzte SUS-Fragebogen findet sich in Anhang A.1 auf Seite 45.
Nach einer Testreihe mit mehreren Probanden wird aus allen Fragebögen ein Durchschnittswert ermittelt, welcher als Prozentwert interpretiert werden kann:

- 100 % - perfektes System ohne Usability-Probleme
- über 80 % - gute bis exzellente Usability
- 60 % - 80 % - grenzwertige bis gute Usability
- unter 60 % - erhebliche Usability-Probleme

John Brook bezeichnet seine Methode im Titel seines Papers "SUS: a "quick and dirty" usability scale" [Bro96] selbst als *quick and dirty*. Die Bearbeitungszeit ebenso wie die Dauer der Auswertung des Fragebogens sind kurz, wodurch er besonders gut für eine Benutzerstudie mit eingeschränkter Durchführungszeit nutzbar ist. Auch *dirty* im Sinne von unscharf trifft auf die Methode zu, da beispielsweise der Unterschied zwischen 79 % und 81 % keine Aussagekraft hat. Der SUS-Score ist also mehr als Tendenz zu sehen.

NASA Task Load Index

Der NASA Task Load Index (TLX) ist ein multi-dimensionales Bewertungsverfahren, welches eine allgemeine Anforderungsbewertung basierend auf einem gewichteten Durchschnitt von Bewertungen auf sechs Unterskalen liefert:

- Geistige Anforderung
- Körperliche Anforderung
- Zeitliche Anforderung
- Leistung
- Anstrengung
- Frustration

Die genauen Definitionen der einzelnen Skalen können [Gro] entnommen werden.

Für die Auswertung in der zu dieser Arbeit zugehörigen Studie wurden die Bewertungen der Skalen gleich stark gewichtet.

Wie bei SUS (Kapitel 4.1.1, S.25) wird der originale Fragebogen offiziell nur in englischer Sprache angeboten. Hier wurde eine deutsche Übersetzung aus einer Untersuchung des Deutschen Zentrum für Luft- und Raumfahrt e.V. aus dem Jahre 2000 gewählt [VSM00]. Der TLX-Fragebogen findet sich in Anhang A.1 auf Seite 46.

4.1.2. Bedienmethoden

Eingabe über Tastatur mit Autocomplete

Bei dieser Methode wird die Stadt bzw. Straße über die klassische Android-Display-Tastatur eingegeben. Es werden Vorschläge entsprechend den bisher eingegebenen Buchstaben gemacht, welche ausgewählt und über den "Los"-Button bestätigt werden können. Dem Aktivitätsdiagramm in Abbildung 4.1 kann der Ablauf der Eingabe zum Anzeigen des gewünschten Ortes entnommen werden.

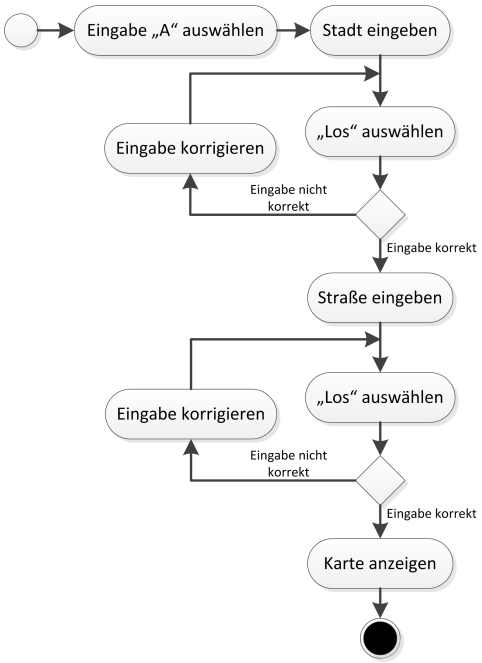


Abbildung 4.1.: *Aktivitätsdiagramm* - Eingabe über Tastatur mit autocomplete

Nach Auswahl der Eingabemethode "Eingabe A" (siehe Abbildung 4.2 links) wird der Nutzer aufgefordert die Stadt einzugegeben (siehe Abbildung 4.2 Mitte). Sobald der erste Buchstabe eingegeben wurde, werden entsprechende Vorschläge gemacht, welche durch die Eingabe weiterer Buchstaben weiter eingegrenzt werden können (siehe 4.2 rechts).

Wird hier ein String eingegeben, der sich nicht im Speicher befindet, so wird beim Auswählen des "Los"-Buttons eine Fehlermeldung angezeigt und die Eingabe muss korrigiert werden (siehe 4.3 links). Ist die Eingabe korrekt, so wird der Nutzer äquivalent zur Stadt-Eingabe aufgefordert die Straße einzugeben. Nach korrekter Eingabe der Straße und Auswahl des "Los"-Buttons wird "Straße" in "Stadt" auf der Karte angezeigt (siehe 4.3 Mitte).

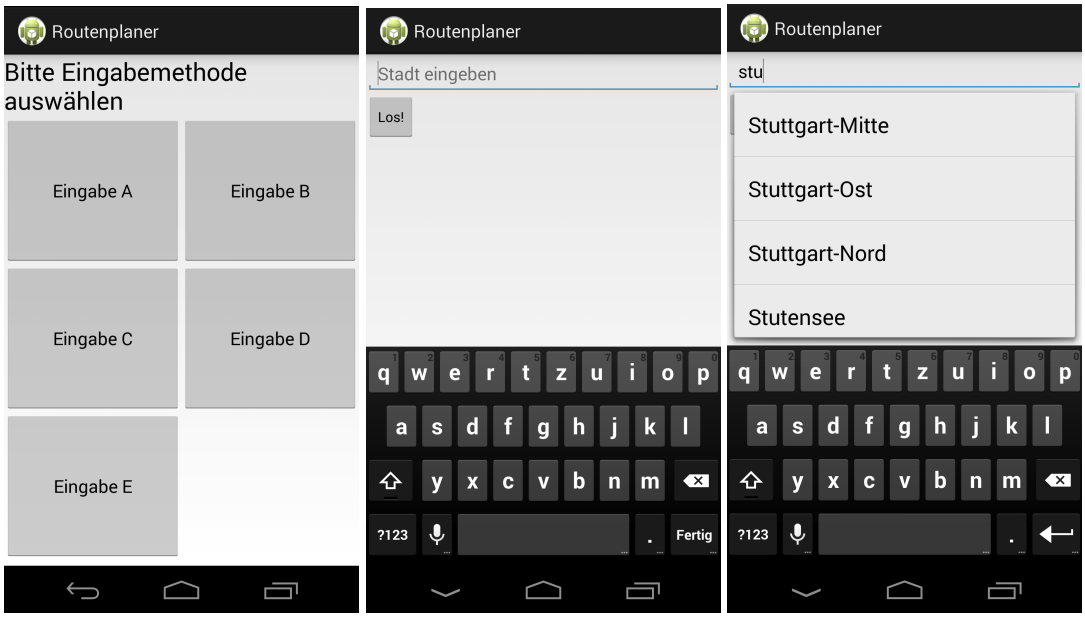


Abbildung 4.2.: Screenshots
links: Auswahlscreen
Mitte: Eingabeaufforderung
rechts: autocomplete

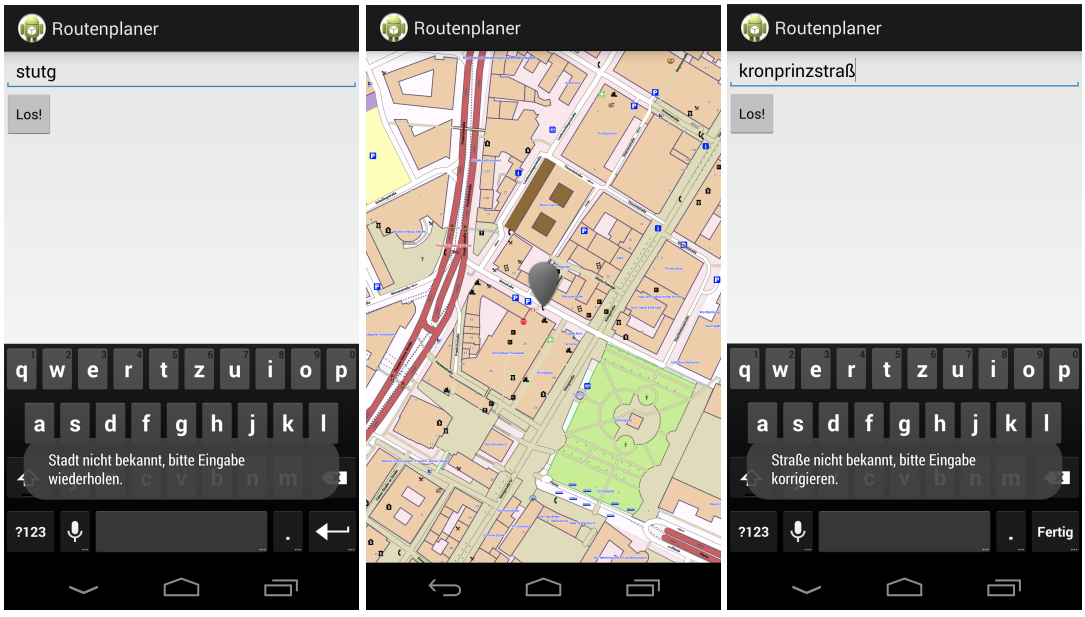


Abbildung 4.3.: Screenshots
links: Anzeige von "Eingabe wiederholen" bei Eingabe mit autocomplete
Mitte: Kartenansicht
rechts: Anzeige von "Eingabe wiederholen" bei Eingabe ohne autocomplete

Eingabe über Tastatur ohne Autocomplete

Auch bei dieser Methode wird die Stadt bzw. Straße über die klassische Android-Display-Tastatur eingegeben. Hier werden allerdings keine Vorschläge gegeben, stattdessen muss String muss exakt eingegeben werden (ausgenommen Groß-/Kleinschreibung). Dem Aktivitätsdiagramm in 4.4 kann der Ablauf der Eingabe zum Anzeigen des gewünschten Ortes entnommen werden.

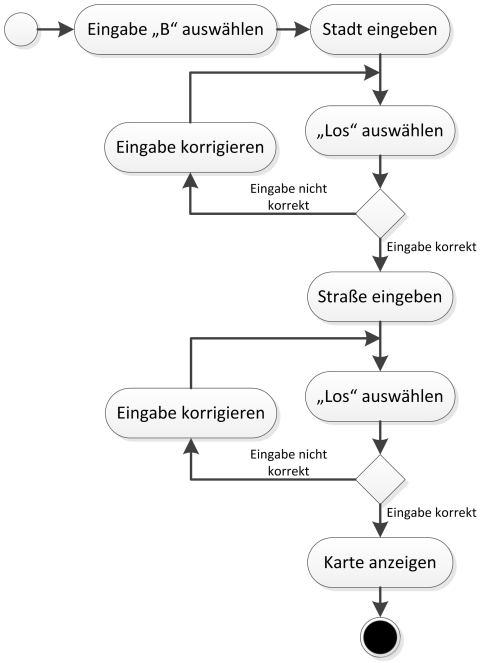


Abbildung 4.4.: *Aktivitätsdiagramm* - Eingabe über Tastatur ohne autocomplete

Nach Auswahl der Eingabemethode "Eingabe B" (siehe Abbildung 4.2 links) wird der Nutzer aufgefordert die Stadt einzugegeben (siehe Abbildung 4.2 Mitte). Wird hier ein String eingegeben, der sich nicht im Speicher befindet, so wird beim Auswählen des "Los"-Buttons eine Fehlermeldung angezeigt und die Eingabe muss korrigiert werden (siehe Abbildung 4.2 rechts). Ist die Eingabe korrekt, so wird der Nutzer äquivalent zur Stadt-Eingabe aufgefordert die Straße einzugeben. Nach korrekter Eingabe der Straße und Auswahl des "Los"-Buttons wird "Straße" in "Stadt" auf der Karte angezeigt (siehe Abbildung 4.3 Mitte).

Binäre Suche (Auwahl über zwei Buttons oder Lautstärketasten)

Bei der binären Suche werden Stadt und Straße mit Hilfe zweier Buttons ausgewählt. Hinter den Buttons liegen alphabetisch sortierte Listen. Dem Aktivitätsdiagramm in Abbildung 4.5 kann der Ablauf der Auswahl des gewünschten Ortes entnommen werden.

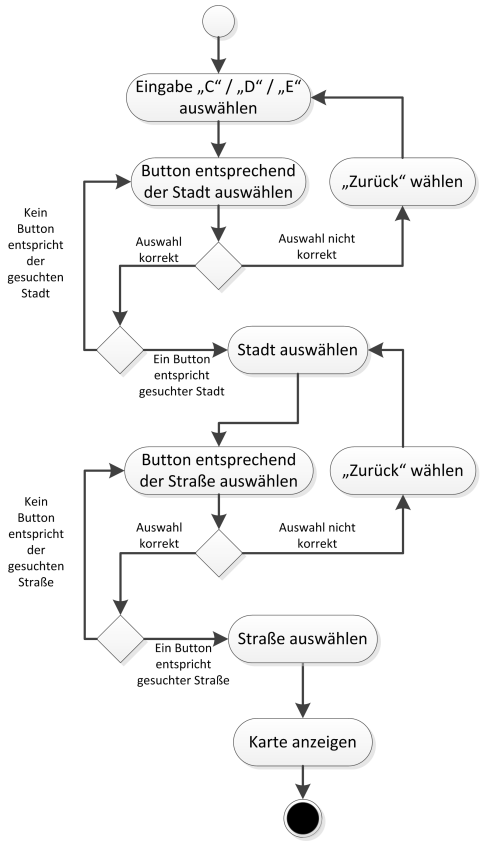


Abbildung 4.5.: *Aktivitätsdiagramm* - k-näre Suchen

Nach Auswahl der Eingabemethode "Eingabe C" (siehe Abbildung 4.2 links) wird der Nutzer aufgefordert die Stadt auszuwählen (siehe Abbildung 4.6 links). Die Liste aller Städte wurde alphabetisch sortiert und in zwei etwa gleich große Teile geteilt (siehe Kapitel 3.3.2). Der Name der ersten und letzten Stadt der ersten Liste werden auf dem ersten Button dargestellt, der Name der ersten und letzten Stadt der zweiten Liste auf dem zweiten Button. Nun muss bestimmt werden, ob die gesuchte Stadt alphabetisch in der ersten oder zweiten Liste liegt und der entsprechende Button gewählt werden. Die somit bestimmte Liste wird wieder geteilt und die Buttons entsprechend der neuen Listen beschriftet. Dies wird so lange wiederholt, bis entweder der falsche Button gewählt wurde oder nur die gesuchte Stadt auf einem der Buttons steht. War die Auswahl nicht korrekt, so muss "zurück" gewählt werden und der Nutzer muss erneut "Eingabe C" auswählen und die Stadt neu suchen. War die Auswahl durchgängig korrekt, so wählt der Nutzer den Button mit der gesuchten Stadt aus (siehe 4.6 rechts) und wird anschließend aufgefordert eine Straße auszuwählen. Dies geschieht äquivalent

zur Auswahl der Stadt, wobei das Auswählen von "zurück" nicht wie bei der Auswahl der Stadt zum Start-Screen führt sondern nur erneut die Zielstadt ausgewählt werden muss (siehe Abbildung 4.6 rechts). Wird der Button der Zielstraße ausgewählt, so wird "Straße" in "Stadt" auf der Karte angezeigt (siehe Abbildung 4.3 Mitte).

Abbildung 4.6.: Screenshots
links: Auswahl der Stadt bei binärer Suche
rechts: Auswahl der Stadt im letzten Schritt

Die Auswahl über Lautstärketasten funktioniert ebenso wie über die Buttons, nur dass der Button nicht durch tippen auf das Display sondern mit den Lautstärketasten ausgewählt wird. Um den oberen Button auszuwählen wird die Taste für "lauter" gewählt, um den unteren Button auszuwählen "leiser". In der Studie wurde hier die selbe Funktion genutzt und lediglich darauf geachtet, dass der Nutzer entsprechend nur über das Display oder die Lautstärketasten auswählt.

Ternäre Suche (Auswahl über drei Buttons)

Die ternäre Suche entspricht grundlegend der binäre Suche (Kapitel 4.1.2). Der Unterschied ist, dass die alphabetisch sortierte Liste hier in drei Teile geteilt wird und entsprechend drei Buttons zur Wahl stehen. Nach Auswahl der Bedienmethode "Eingabe D" werden also drei Buttons angezeigt (siehe Abbildung 4.7 links), die ggf. auf zwei reduziert werden (siehe Abbildung 4.7 Mitte). Die Begründung hierfür findet sich in Kapitel 3.3.2 (Implementierung ternäre Suche).

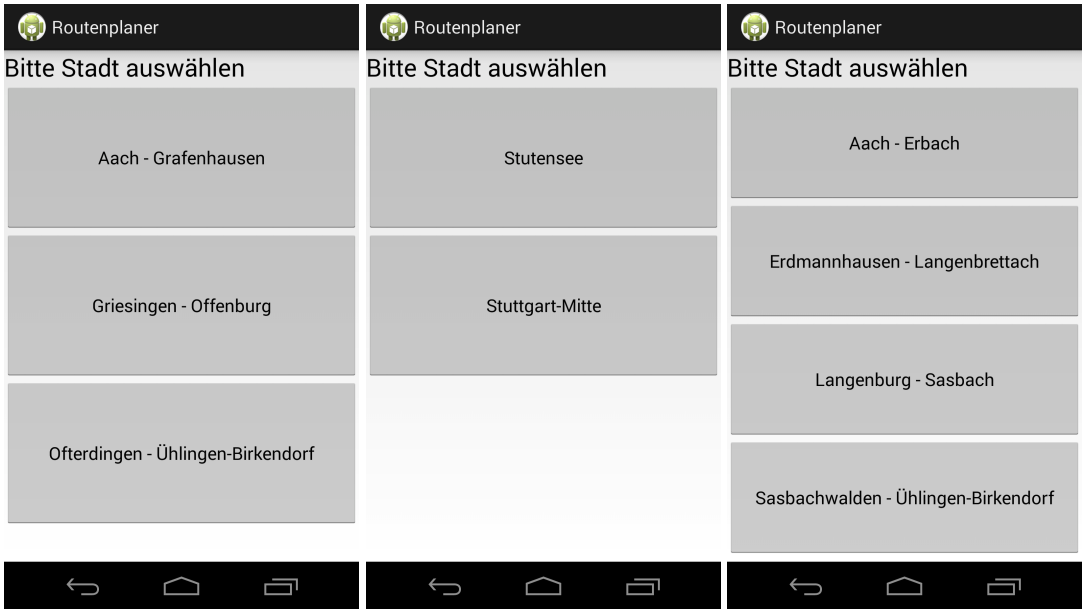


Abbildung 4.7.: Screenshots
links: Auswahl der Stadt bei ternärer Suche
Mitte: Reduzierte Ansicht (ternäre Suche)
rechts: Auswahl der Stadt bei quaternärer Suche

Quaternäre Suche (Auswahl über vier Buttons)

Auch die quaternäre Suche entspricht im Prinzip der binären (Kapitel 4.1.2) und ternären Suche (Kapitel 4.1.2) , allerdings wird die alphabetisch sortierte Liste in vier Listen geteilt und es stehen nach Auswahl der Bedienmethode "Eingabe E" vier Buttons zur Auswahl bereit (siehe Abbildung 4.7 rechts). Hier können die Buttons auf drei oder zwei reduziert werden, was in Kapitel 3.3.2 (Implementierung quaternäre Suche) begründet ist.

Bei allen Bedienmethoden wurde während der Eingabe bzw. dem Auswählen jeder einzelnen Stadt sowie Straße die Zeit mit Hilfe der Anwendung gestoppt und ausgegeben.

4.1.3. Teilnehmer

An der Studie haben insgesamt 18 Personen teilgenommen, davon zwei weibliche. Das durchschnittliche Alter lag bei 22,61 Jahren, die Standardabweichung bei 2,83 Jahren. Es haben ausschließlich Studierende teilgenommen, davon hatten 17 informationstechnischen Hintergrund, eine Person kam aus einem fachfremdem Studienbereich. Von den 18 Teilnehmern besitzen 16 ein Smartphone, wobei 15 dieses mehrmals täglich zur Eingabe von Text nutzen. 17 Teilnehmer nutzen ein Navigationssystem, zwölf davon selten, die anderen fünf Teilnehmer nutzen wöchentlich bis mehrmals in der Woche ein Navigationssystem. Jeder Teilnehmer erhielt für die Teilnahme an der Studie sieben Euro.

4.1.4. Ausrüstung

Die Studie fand immer in einer ruhigen Umgebung ohne offensichtliche Ablenkung durch Personen oder Geräusche statt. Die Studie wurde von allen Teilnehmern auf dem selben Smartphone, einem Google Nexus 5 (4,95"-Display, 1920 x 1080, 445 ppi, Full HD-IPS) [nex] , durchgeführt. Es wurde darauf geachtet, dass der Flugmodus eingestellt war, sodass der Teilnehmer nicht gestört wurde. Außerdem wurde die Helligkeit bei allen Teilnehmern auf höchste Stufe gestellt.

4.1.5. Einzugebende Strings

Als Strings wurden für die Studie drei verschiedene Orte gewählt. Es wurde darauf geachtet, dass sowohl längere als auch kürzere Strings gewählt wurden, Strings die sich im vorderen, hinteren und mittleren Bereich des Alphabets befinden ebenso wie Strings mit Sonderzeichen. Der für die Suchen zugrunde liegende Datensatz besteht aus Städten in Württemberg. Es handelt sich hierbei um 976 Städte. In Tabelle 4.3 findet sich die Anzahl der Klicks sortiert nach String und Bedienmethode für die k-nären Suchen.

	C	D	E
Stuttgart-Mitte	10	6	5
Kronprinzstraße	10	6	5
Aalen	9	7	5
Hardstraße	9	5	4
Esslingen am Neckar	10	7	5
Zeisigweg	10	7	5

Tabelle 4.3.: Anzahl Klicks für die entsprechenden Strings und Bedienmethoden
C: binäre Suche, *D:* ternäre Suche, *E:* quaternäre Suche

Wird ein Datensatz genutzt der mehr oder weniger Städte fasst, so werden für das Auswählen der Stadt entsprechend mehr oder weniger Klicks benötigt. Die Studie wurde mit den oben genannten Daten durchgeführt.

4.1.6. Ablauf

Im Folgenden wird die genaue Durchführung der Studie beschrieben, wie sie für jeden Teilnehmer durchgeführt wurde. Zunächst wurde dem Teilnehmer sinngemäß folgendes erklärt: Es handelt sich um eine Studie zu einer Abschlussarbeit zum Thema "Neue Bedienkonzepte für Mobile Routenplaner". Es wurden sechs Eingabemethoden implementiert, die alle durch den Teilnehmer getestet werden sollen. Dazu werden für jede Eingabemethode die selben drei Orte gesucht. Dabei wird jeweils die Zeit gestoppt. Die Reihenfolge der Eingabemethoden werden vorgegeben. Nachdem mit einer Methode alle drei Orte gefunden wurden, werden dem Teilnehmer zwei Fragebögen vorgelegt, die auszufüllen sind. Die Studie wird circa 30 min dauern und der Teilnehmer erhält sieben Euro.

Anschließend wurde von jedem Teilnehmer eine Einverständniserklärung (siehe Anhang Anhang A.1 auf Seite 43) unterzeichnet. Die Studie startete nun mit einem Fragebogen, in dem der Teilnehmer Angaben zu Alter, Geschlecht, Studiengang / Beruflichem Hintergrund machte sowie die Regelmäßigkeit der Nutzung eines Smartphones und Navigationsgerätes angab (siehe Anhang Anhang A.1 auf Seite 44).

Nun wurden in der durch das Balanced Latin Square vorgegebenen Reihenfolge die Orte "Stuttgart-Mitte, Kronprinzstraße", "Aalen, Hardtstraße" sowie "Esslingen am Neckar, Zeisigweg" gesucht, welche dem Teilnehmer gut lesbar als Ausdruck vorgelegt wurden. Nach Auswahl der Strings wurden vom Teilnehmer nach jeder Methode SUS- und TLX-Fragebogen ausgefüllt.

Abschließend erhielt der Teilnehmer sieben Euro und bestätigte dies durch seine Unterschrift.

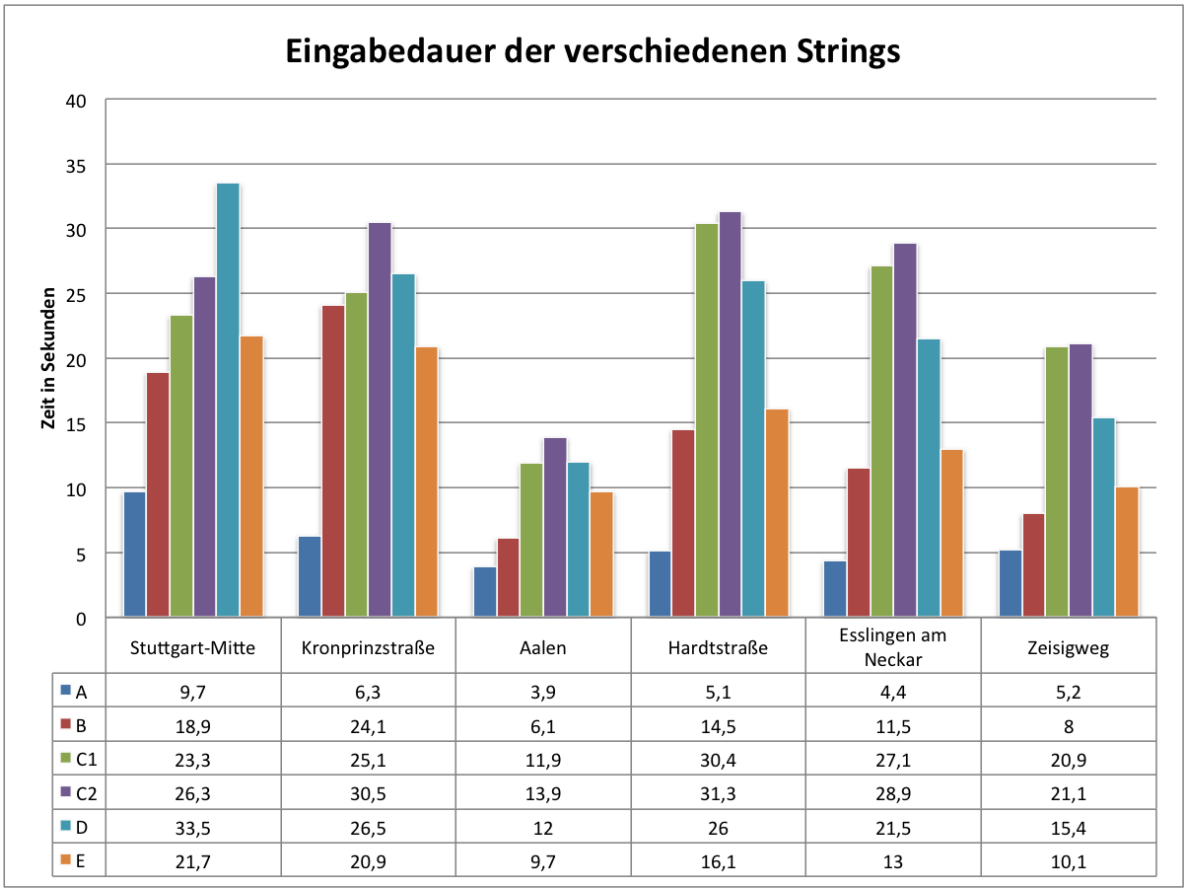
Damit war die Studie für den Teilnehmer beendet, und es wurden noch die Zeiten der String-Eingaben gesichert.

4.2. Ergebnis

4.2.1. Hypothesen

- Die Bedienmethode mit der Android-Display-Tastatur mit autocomplete ist im Mittel die schnellste Eingabemethode.
- Die Bedienmethoden mit Buttons sind einfach und intuitiv zu bedienen.
- Die Bedienmethoden mit Buttons sind unter erschwerten Bedingungen besser zu bedienen als die Displaytastatur.

4.2.2. Zeitmessung



	Stuttgart-Mitte	Kronprinzstraße	Aalen	Hardtstraße	Esslingen am Neckar	Zeisigweg
A	9,7	6,3	3,9	5,1	4,4	5,2
B	18,9	24,1	6,1	14,5	11,5	8
C1	23,3	25,1	11,9	30,4	27,1	20,9
C2	26,3	30,5	13,9	31,3	28,9	21,1
D	33,5	26,5	12	26	21,5	15,4
E	21,7	20,9	9,7	16,1	13	10,1

Abbildung 4.8.: Eingabedauer der verschiedenen Strings mit verschiedenen Eingabemethoden *A:* Tastatur mit autocomplete, *B:* Tastatur ohne autocomplete, *C1:* Binäre Suche, *C2:* Binäre Suche mit Eingabe über Lautstärketasten, *D:* Ternäre Suche, *E:* Quaternäre Suche

4.8 zeigt die mittleren Eingabedauern der verschiedenen Strings mit den verschiedenen Bedienmethoden. Die mittlere Eingabedauer der Strings mit der Bildschirmtastatur mit autocomplete beträgt ca. 5,77 s, ohne autocomplete ca. 13,85 s, mit zwei Buttons ca. 23,12 s, über die Lautstärke-Tasten ca. 25,33 s, mit drei Buttons ca. 22,48 s und mit vier Buttons ca. 15,25 s.

4.2.3. System Usability Scale

Die Mittelwerte des System Usability Scale finden sich in Abbildung 4.9.

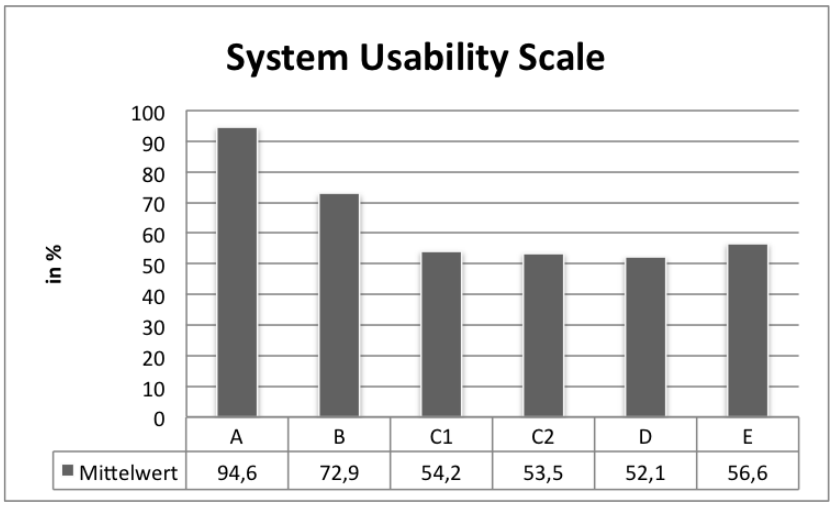


Abbildung 4.9.: Auswertung des System Usability Scale
A: Tastatur mit autocomplete, *B:* Tastatur ohne autocomplete, *C1:* Binäre Suche, *C2:* Binäre Suche mit Eingabe über Lautstärketasten, *D:* Ternäre Suche, *E:* Quaternäre Suche

Man kann erkennen, dass sich die Eingabe mit Tastatur mit autocomplete eindeutig als einzige im Bereich zwischen 80 % und 100 % befindet und damit (siehe Kapitel 4.1.1) als einzige Methode gute bis exzellente Usability vorweist. Die Eingabe mit Tastatur ohne autocomplete befindet sich mit 72,9 % im Bereich der grenzwertigen bis guten Usability. Hingegen haben alle für die Teilnehmer neuen und unbekannten Bedienkonzepte einen eindeutigen Wert unter 60 % erreicht, was für erhebliche Usability-Probleme spricht.

4.2.4. NASA Task Load Index

Die gesamten Mittelwerte für den NASA TLX, mit allen Unterskalen in gleicher Wertung, finden sich in Abbildung 4.10. Diagramme für jede Eingabemethode mit Aufteilung in die verschiedenen Unterskalen finden sich in Anhang A.2 auf Seite 47

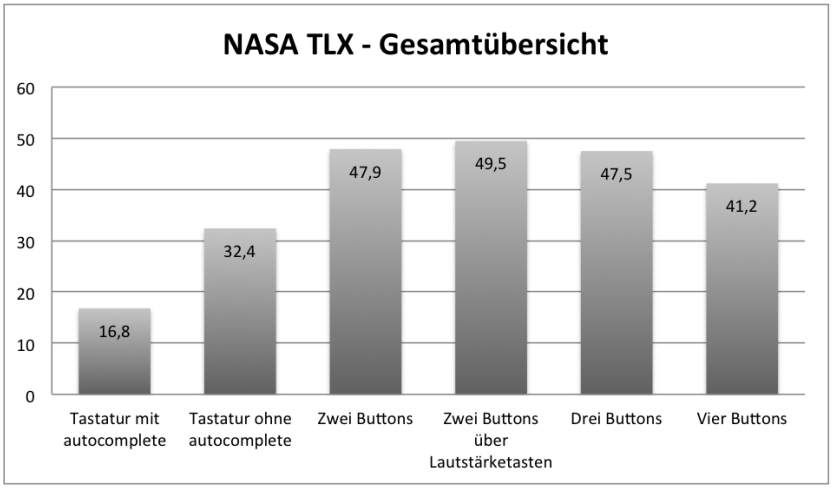


Abbildung 4.10.: Auswertung des NASA Task Load Index

In der Gesamtübersicht in Abbildung 4.10 kann man erkennen, dass die Teilnehmer die Anforderungen für die Eingabe mit Tastatur mit sowie ohne autocomplete als am geringsten empfanden, während die Anforderungen für die k-nären Suchen eindeutig als höher eingestuft wurden. Hier fällt auf, dass die binäre und ternäre Suche als anstrengender empfunden wurden als die quaternäre Suche.

4.2.5. Beobachtungen

Außer den messbaren Zeiten sowie den Fragebögen wurden die Teilnehmer während der Studie beobachtet. So wurde, nachdem ein Teilnehmer eine Bedienmethode nach etwa 60 Sekunden noch nicht anwenden konnte, die Funktion erklärt. Dies musste ausschließlich bei den k-nären Bedienmethoden gemacht werden, und betraf immer die erste k-näre Suche, die der Nutzer testete. Die darauf folgenden konnten anschließend ohne Erklärung bedient werden. Hierbei musste neun von 18 Teilnehmern die Funktionsweise der Bedienmethode erklärt werden. Die Funktionsweise musste im Schnitt 1,5-mal bei der binären Suche, drei mal bei der binären Suche mit Lautstärketasten, keinmal bei der ternären Suche sowie 1,5-mal bei der quaternären Suche erklärt werden. Hierbei kann kein Zusammenhang zur Regelmäßigkeit der Nutzung eines Smartphones oder eines Navigationssystems sowie zu Geschlecht, Alter oder Studienfach festgestellt werden. Außerdem wurde beobachtet, dass viele Teilnehmer bei den k-nären Eingabemethoden das Alphabet vor sich hin sagten, ein Teilnehmer schrieb sich Teile des Alphabets sogar auf. Einige Teilnehmer schienen die k-nären Suchen sehr unangenehm zu finden und teilten dies mit, ebenso schienen sie erleichtert bei den Eingaben über die Tastatur.

4.3. Diskussion

Zunächst ist anzumerken, dass nur 50% der Teilnehmer der Studie die unbekannten Bedienmethoden intuitiv bedienen konnten. In 50% der Fälle musste eine Anleitung gegeben werden, zuvor wurde eine Minute abgewartet, ob der Teilnehmer die Methoden eigenständig nutzen kann. Die Eingabe über die Display-Tastatur jedoch musste nicht erklärt werden und war vollkommen intuitiv. Der Teilnehmer konnte sofort damit beginnen die gestellte Aufgabe zu lösen.

Die Studie konnte nicht unter den Bedingungen durchgeführt werden, für die die Bedienmethoden gedacht sind. Aus diesem Grund lassen sich nur Aussagen für die Umgebung machen, in welcher die Studie durchgeführt wurde, nicht aber für den gedachten Anwendungszweck. Die Studie wurde in einem Raum ohne Ablenkung durchgeführt, der Nutzer saß an einem Tisch und konnte das Smartphone ruhig sitzend bedienen. Unter diesen Umständen war zu erwarten, dass der Nutzer die Aufgaben mit den bekannten Bedienmethoden am schnellsten durchführen kann. Dennoch ist auffällig, dass für die quaternäre Suche im Schnitt nur 1,4 Sekunde länger benötigt wurde als bei der Eingabe über die Display-Tastatur ohne autocomplete. Hier kann die Behauptung aufgestellt werden, dass die quaternäre Suche unter erschwerten Bedingungen schneller sein könnte als die Eingabe mit der Display-Tastatur (ohne autocomplete). Allerdings muss auch bedacht werden, dass der Nutzer die Städte und Straßen auf den Buttons lesen muss, bevor er den entsprechenden Button auswählen kann, was unter erschwerten Bedingungen möglicherweise zusätzliche Zeit in Anspruch nimmt und dadurch die Auswahl der Stadt/Straße insgesamt nicht verkürzt wird, obwohl die großen Buttons leichter auszuwählen sind als die einzelnen Buchstaben der Display-Tastatur. Dies sind jedoch nur weitere Hypothesen, die unter erschwerten Bedingungen untersucht werden müssten. Dies war im Rahmen dieser Arbeit jedoch nicht möglich. Daher lässt sich bezüglich der zeitlichen Anforderungen für die unter 4.1.3 beschriebene Teilnehmergruppe nur aussagen, dass Bedienmethoden die die Display-Tastatur nutzen schneller sind als die k-nären Suchen, wobei hier die benötigte Zeit im Schnitt mit zunehmender Anzahl an Buttons fällt. Die Bedienmethode über die Lautstärketasten war im Schnitt 2,21 Sekunden langsamer als die binäre Suche mit Auswahl über das Display. Da man das Display im Gegensatz zu den Lautstärketasten beispielsweise mit Handschuhen nicht bedienen kann, ist unter besonderen Umständen das Bedienen über die Lautstärketasten eine schnellere Alternative zum Bedienen über das Display, da vorher die Handschuhe ausgezogen werden müssten.

Unabhängig von der benötigten Zeit für die einzelnen Bedienmethoden muss jedoch auch die Usability und die Anforderungseinschätzung der Teilnehmer betrachtet werden. Im Fragebogen des System Usability Scale (siehe 4.1.1) zeigen die Bewertungen der k-nären Suchen erhebliche Usability-Probleme, wohingegen die Eingaben über die Display-Tastatur mit guter oder exzellenter Usability bewertet wurden. Die Bewertung der k-nären Suchen weißt dabei keine nennenswerten Unterschiede auf, wohingegen die Eingabe mit autocomplete als bedeutend Usability-freundlicher als die Eingabe ohne autocomplete bewertet wurde. Auch die Anforderungen wurden von den Teilnehmern für die Eingaben über die Display-Tastatur bedeutend geringer empfunden als für die k-nären Suchen. Bei der quaternären Suche fühlten sich die Teilnehmer weniger stark gefordert als bei den anderen Suchen.

Insgesamt sind der Studie zufolge die neu entwickelten Bedienmethoden nicht schneller als die bereits bekannten und wurden im Vergleich zu dein Eingaben über die Display-Tastatur schlechter bewertet.

Von den k-nären Suchen liefert jedoch die quaternäre Suche die besten Ergebnisse, was daran liegen könnte, dass hier weniger Klicks benötigt werden als bei binärer und ternärer Suche.

5. Zusammenfassung

Im Rahmen dieser Arbeit wurden sechs Methoden zur Spezifizierung des Zielortes eines Routenplaners als Android-Anwendung entwickelt. Von den entwickelten Methoden wurde bei zwei Methoden über die Display-Tastatur ausgewählt, bei einer über die Lautstärke-Tasten sowie bei drei über zwei, drei bzw. vier große Buttons über das Display. Die entwickelten Methoden wurden in einer Benutzerstudie auf zeitliche Anforderung, Usability und allgemeine Anforderungen getestet um herauszufinden ob der Ansatz der k-nären Suchen für spezielle Routenplaner, welche unter erschwerten Bedingungen genutzt werden sollen, weiter verfolgt werden sollte. Ein Studie unter Laborbedinungen ergab jedoch, dass die k-nären Suchen im Vergleich zu den Suchen über die Display-Tastatur unter Laborbedingungen zeitlich keinen Vorteil bringen und von den Teilnehmern als unangenehmer, umständlicher und unintuitiv wargenommen werden. Der quaternäre Ansatz der Studie führte zu den besten Ergebnissen unter den k-nären Suchen und sollte daher weiterverfolgt werden.

Ausblick

Es gibt weitere Ansatzpunkte für die Eingabe bei Routenplanern unter erschwerten Bedinungen. Es gibt viele andere Varianten des Anzeigens der Städte / Straßen, beispielsweise wäre es möglich nur die sich unterscheidenden Buchstaben auf den Buttons anzuzeigen, sodass der Nutzer weniger vergleichen muss. Möglich wäre es auch, den Nutzer durch Anzeigen des Alphabets an sinnvoller Stelle auf die Funktionsweise hinzuweisen, sodass die Suche keiner Erklärung mehr bedarf. Auch die Eingabe über Wischen statt Tippen könnte sich für den Nutzer als angenehmer und intuitiver herausstellen. Die Auswahl über die Lautstärketasten ergab keine guten Ergebnisse, dennoch ist es die einzige Möglichkeit einen Routenplaner z.B. mit Handschuhen zu bedienen. Aus diesem Grund könnte auch eine Eingabe über die Lautstärketasten weiter verfolgt werden. Auch die quaternäre Eingabe ist eine Bedienmethode, welche in verschiedenen Ausprägungen weiter verfolgt werden könnte um dem Nutzer für die Zukunft einen Routenplaner anzubieten, der auch unter erschwerten Bedingungen problemlos zu bedienen ist.

A. Anhang

A.1. Dokumente für die Durchführung der Studie

Einverständniserklärung

BESCHREIBUNG:

Sie wurden eingeladen an einer Benutzerstudie über Eingabemethoden bei mobilen Routenplanern teilzunehmen.

DAUER:

Ihre Teilnahme an der Studie wird ca. 30 Minuten dauern.

GESAMMELTE DATEN:

In dieser Studie werden sie mit 6 verschiedenen Methoden Orte suchen. Es werden Daten über die Dauer der Eingaben gesammelt, außerdem werden Sie einige Fragebögen ausfüllen.

RISIKEN UND NUTZEN:

Diese Studie hat keinerlei Risiken. Die gesammelten Daten werden sicher gespeichert. Wir garantieren, dass die gesammelten Daten nicht zweckendfremdet werden und Ihre Daten in jedem Fall geschützt werden. Eine Teilnahme oder nicht-Teilnahme an der Studie wird Ihre Studienleistungen in keiner Weise beeinflussen.

VERGÜTUNG:

Sie erhalten 7 € für die Teilnahme an der Studie.

RECHTE ALS TEILNEHMER:

Wenn Sie diese Erklärung gelesen haben und entscheiden, an der Studie teilzunehmen gilt für Sie folgendes: Die Teilnahme an dieser Studie ist freiwillig und kann jederzeit von Ihrer Seite ohne Folgen abgebrochen werden.

Sie können das Antworten auf einzelne Fragen verweigern. Die während dieser Studie gesammelten Daten werden für die wissenschaftliche Nutzung gesammelt und hierbei vertraulich und anonymisiert behandelt.

KONTAKTINFORMATIONEN:

Haben Sie Fragen, Anliegen oder Beschwerden zu dieser Studie, den Abläufen, Risiken und Nutzen, so kontaktieren sie eine der folgenden Personen:

Daniel Bahrdt (daniel.bahrdt@fmi.uni-stuttgart.de)

Stefanie Bahle (bahlese@studi.informatik.uni-stuttgart.de)

Mit der Unterschrift dieses Dokuments bestätige ich, dass ich den Bedingungen zustimme.

Name: _____ Datum, Unterschrift: _____

Fragebogen (Demographics)

Teilnehmer Nummer: _____

Alter: _____

Geschlecht: _____

Studiengang/Beruflicher Hintergrund: _____

Besitzen Sie ein Smartphone? ☐ Ja ☐ Nein

Wenn Ja: Wie oft nutzen Sie Ihr Smartphone zur Eingabe von Text?

☐ mehrmals am Tag ☐ mehrmals in der Woche

☐ täglich ☐ wöchentlich

☐ selten

Nutzen Sie ein Navigationssystem? ☐ Ja ☐ Nein

Wenn Ja: Wie oft nutzen Sie das Navigationssystem?

☐ mehrmals am Tag ☐ mehrmals in der Woche

☐ täglich ☐ wöchentlich

☐ selten

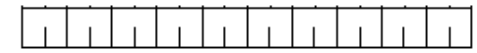
System Usability Scale Fragebogen

	Stimme überhaupt nicht zu				Stimme voll zu
1. Ich denke, dass ich das System gerne häufig benutzen würde.	1	2	3	4	5
2. Ich fand das System unnötig komplex.	1	2	3	4	5
3. Ich fand das System einfach zu benutzen.	1	2	3	4	5
4. Ich glaube, ich würde die Hilfe einer technisch versierten Person benötigen, um das System zu benutzen.	1	2	3	4	5
5. Ich fand, die verschiedenen Funktionen in diesem System waren gut integriert.	1	2	3	4	5
6. Ich denke, das System enthielt zu viele Inkonsistenzen.	1	2	3	4	5
7. Ich kann mir vorstellen, dass die meisten Menschen den Umgang mit diesem System sehr schnell lernen.	1	2	3	4	5
8. Ich fand das System sehr umständlich zu nutzen.	1	2	3	4	5
9. Ich fühlte mich bei der Benutzung des Systems sehr sicher.	1	2	3	4	5
10. Ich musste eine Menge lernen, bevor ich anfangen konnte das System zu verwenden.	1	2	3	4	5

NASA TLX

Geistige Anforderung

Wie hoch waren die geistigen Anforderungen der Aufgabe?



Sehr niedrig — Sehr hoch

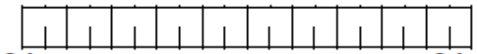
Körperliche Anforderung

Wie hoch waren die körperlichen Anforderungen der Aufgabe?

Sehr niedrig — Sehr hoch

Zeitliche Anforderung

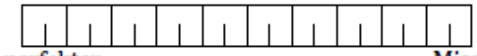
Wie hoch war das Tempo, mit dem die einzelnen Arbeitsschritte aufeinander folgen?



Sehr niedrig — Sehr hoch

Leistung

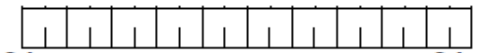
Wie erfolgreich haben Sie die geforderte Aufgabe ihrer Ansicht nach durchgeführt?



perfekter Erfolg — Misserfolg

Anstrengung

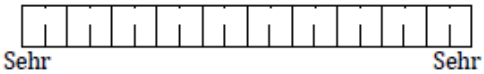
Wie sehr mussten Sie sich anstrengen um Ihre Leistung zu erreichen?



Sehr wenig — Sehr stark

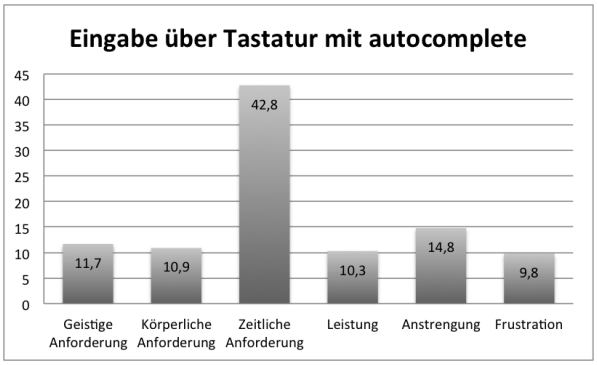
Frustration

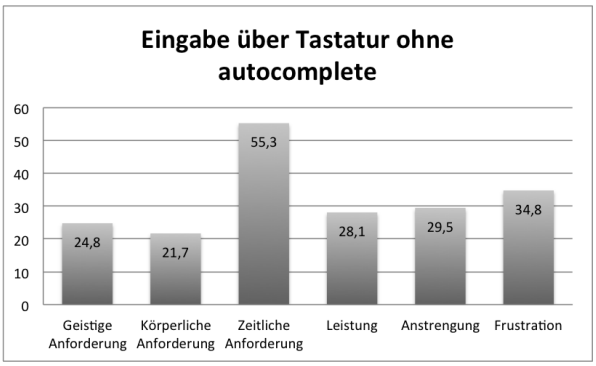
Wie verunsichert, entmutigt, gereizt und verärgert waren Sie?

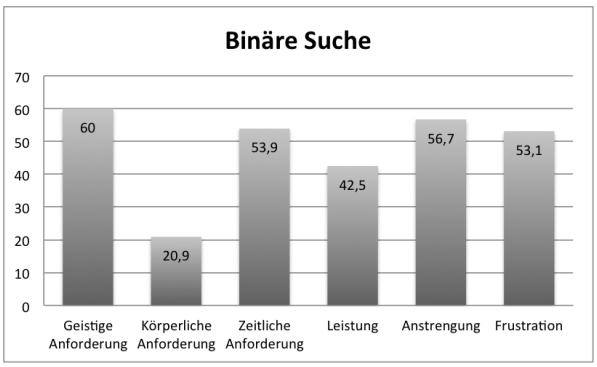


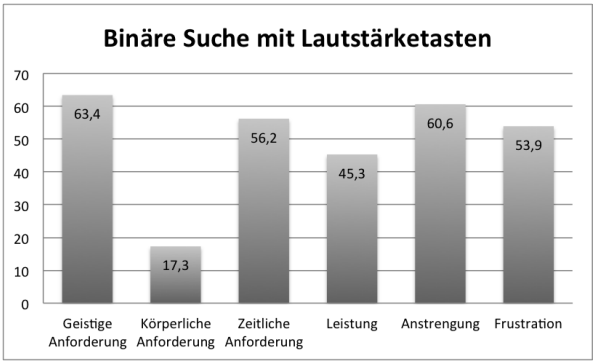
Sehr niedrig — Sehr hoch

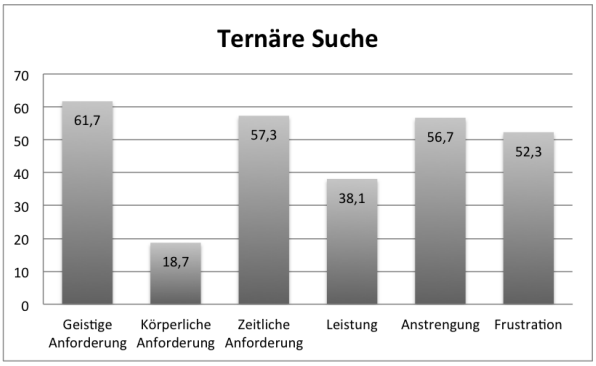
A.2. NASA Task Load Index - Auswertung der einzelnen Bedienmethoden

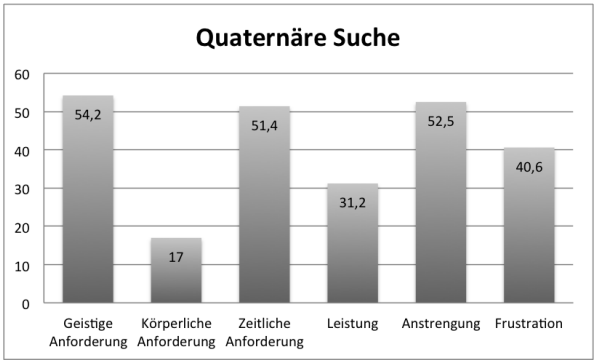







Binäre Suche mit Lautstärketasten
70
60
50
40
30
20
10
0
63,4
17,3
56,2
45,3
60,6
53,9
Geistige Anforderung
Körperliche Anforderung
Zeitliche Anforderung
Leistung
Anstrengung
Frustration


Ternäre Suche
70
60
50
40
30
20
10
0
61,7
18,7
57,3
38,1
56,7
52,3
Geistige Anforderung
Körperliche Anforderung
Zeitliche Anforderung
Leistung
Anstrengung
Frustration


Quaternäre Suche
60
50
40
30
20
10
0
54,2
17
51,4
31,2
52,5
40,6
Geistige Anforderung
Körperliche Anforderung
Zeitliche Anforderung
Leistung
Anstrengung
Frustration

Literaturverzeichnis

[anda] URL http://source.android.com/source/licenses.html. (Zitiert auf Seite 9)

[andb] URL www.developer.android.com. (Zitiert auf Seite 10)

[arr] URL http://docs.oracle.com/javase/7/docs/api/java/lang/System.html. (Zitiert auf Seite 15)

[aut] AutoCompleteTextView. URL http://developer.android.com/reference/android/widget/AutoCompleteTextView.html. (Zitiert auf Seite 14)

[Bah13] D. Bahrdt, 2013. URL https://theogit.fmi.uni-stuttgart.de/bahrdtdl/osmfind. (Zitiert auf Seite 10)

[BP09] A. Becker, M. Pant. *Android Grundlagen und Proframmierung*. dpunkt.verlag, 2009. (Zitiert auf Seite 19)

[Bro96] J. Brooke. SUS - A quick and dirty usability scale. Redhatch Consulting Ltd., 12 Beaconsfield Way, Earley, READING RG6 2UX, United Kingdom, 1996. URL http://hell.meiert.org/core/pdf/sus.pdf?/. (Zitiert auf Seite 25)

[buf] java.io Class BufferedReader. URL http://docs.oracle.com/javase/6/docs/api/java/io/BufferedReader.html. (Zitiert auf Seite 12)

[Gro] H. P. R. Group. NASA Task Load Index (TLX), Paper and Pencil Package. URL http://humansystems.arc.nasa.gov/groups/TLX/downloads/TLX_pappen_manual.pdf. (Zitiert auf Seite 26)

[has] java.util Class HashMap<K,V>. URL http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html. (Zitiert auf Seite 12)

[HN13] Y. Hamano, N. Nishiuchi. Usability Evaluation of Text Input Methods for Smartphone among the Elderly, 2013. (Zitiert auf Seite 9)

[LGL] URL https://www.lgl-bw.de/lgl-internet/opencms/de/07_Produkte_und_Dienstleistungen/Open_Data_Initiative/. (Zitiert auf Seite 10)

[map] URL https://code.google.com/p/mapsforge/. (Zitiert auf Seite 10)

[MSH11] I. S. MacKenzie, R. W. Soukoreff, J. Helga. 1 thumb, 4 buttons, 20 words per minute: design and evaluation of H4-writer, 2011. URL http://dl.acm.org/citation.cfm?id=2047196.2047258&coll=DL&dl=ACM. (Zitiert auf Seite 9)

[nex] URL http://www.google.de/nexus/5/. (Zitiert auf Seite 33)

[osm] OpenStreetMap. URL http://www.openstreetmap.de/. (Zitiert auf Seite 10)

[Rau11] M. Rauer. Quantitative Usablility-Analysen mit der System Usability Scale (SUS), 2011. URL http://blog.seibert-media.net/2011/04/11/usablility-analysen-system-usability-scale-sus/. (Zitiert auf Seite 25)

[Ull11] C. Ullenboom. *Java ist auch eine Insel*. Galileo Computing, 2011. URL http://openbook.galileocomputing.de/javainsel9/javainsel_13_008.htm. (Zitiert auf Seite 12)

[VSM00] M. Vejvoda, A. Samel, H. Maaß. Untersuchungen zur Beanspruchung des Kabinenpersonals auf transmeridianen Strecken. 2000. URL http://www.dlr.de/me/Portaldata/25/Resources/dokumente/flugphysiologie/fbkabine2000-32b_ge.pdf. (Zitiert auf Seite 26)

Alle URLs wurden zuletzt am 17. 02. 2014 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift