Mobile App für Foto-Überweisungen

Inhalt

[1 Einleitung 4](#_Toc511725392)

[2 Entwicklung Mobiler Applikationen 4](#_Toc511725393)

[2.1 Anforderungsmodellierung 4](#_Toc511725394)

[2.2 Modellierung 5](#_Toc511725395)

[2.3 Testen 5](#_Toc511725396)

[3 Requirements Engineering 6](#_Toc511725397)

[4 Modellierungsphase 6](#_Toc511725398)

[5 Mobile Development 6](#_Toc511725399)

[5.1 Untersuchungskriterien 6](#_Toc511725400)

[5.2 Java 7](#_Toc511725401)

[5.3 Kotlin 8](#_Toc511725402)

[5.4 Android NDK 8](#_Toc511725403)

[5.5 Xamarin Android 8](#_Toc511725404)

[5.6 Corona 9](#_Toc511725405)

[5.7 PhoneGap 9](#_Toc511725406)

[6 Texterkennung 9](#_Toc511725407)

[6.1 Klassischer Ablauf 9](#_Toc511725408)

[6.2 Vorverarbeitung 11](#_Toc511725409)

[6.2.1 Glätten und verschwämmen 11](#_Toc511725410)

[6.2.2 Textkrümmung 12](#_Toc511725411)

[6.3 Auswahl einer OCR-Bibliothek 12](#_Toc511725412)

[6.3.1 Google Mobile-Vision OCR 13](#_Toc511725413)

[6.3.2 Tesseract 13](#_Toc511725414)

[6.3.3 OCRopy 14](#_Toc511725415)

[7 Implementierung 14](#_Toc511725416)

[8 Fazit und Ausblick 14](#_Toc511725417)

[9 Literaturverzeichnis 14](#_Toc511725418)

# Einleitung

Das Scannen von Dokumenten erfolgt meist als Bilddatei, was dazu führt, dass der digitalisierte Text weder kopiert, noch bearbeitet werden kann. Um dies zu ermöglichen kann der Text als Buchstaben erkannt und ausgegeben werden. Diese Methode der Texterkennung wird als Optical Character Recognition, kurz OCR, bezeichnet.

Moderne Banking Applikationen, beispielsweise die der Sparkasse, bieten die Möglichkeit an aus Fotos Überweisungsdaten wie IBAN, Betrag oder Verwendungszweck zu erkennen um den Überweisungsprozess zu vereinfachen.

Ziel dieses Projekts ist eine Applikation für Mobilgeräte zu entwickeln, die anhand eines Fotos Überweisungsdaten erkennt und diese auf dem Bildschirm ausgibt.

# Entwicklung Mobiler Applikationen

Der Entwicklungsprozess von Applikationen für mobile Geräte lehnt sich an die konventionelle Art des Softwareentwicklungsprozesses an, besitzt jedoch wichtige Unterschiede in beispielsweise Benutzeroberflächendesign oder der Testphase.

Der Aufbau des Projekts folgt dem V-Modell. In der Anforderungsspezifikation werden die Anforderungen des Kunden an die Software notiert. (SE Pressman, S.42)

(Bild) (SE Pressman S.43)

## Anforderungsmodellierung

Die Anforderungsmodellierung dient zur Kommunikation der Kundenanforderungen an den Softwareentwickler in Form von Text, sowie Diagrammen, sodass diese einfach zu verstehen sind. (SE Pressman, S.166)

Das V-Modell zeigt, dass es oft schwierig ist alle Anforderungen von Anfang an zu spezifizieren, weshalb durch verschiedene Testphasen die Anforderungen möglicherweise angepasst werden müssen. (SE Pressman, S.42)

Für dieses Projekt werden Szenariobasierte-, (Verhaltens-?) und Klassenmodelle verwendet, um die Anforderungen darzustellen.

Ein Beispiel für ein Szenariobasiertes Modell ist das Use-Case-Diagramm:

(unser UseCase)

Ein Beispiel für ein Verhaltensdiagramm ist das Aktivitätsdiagramm. Es soll die schon im Use-Case-Diagramm vorhandenen Informationen auf kompaktere Weise darstellen, indem es einen Interaktionsfluss zeigt. (SE Pressman, S.99f)

(unser Aktivitätsdiagramm)

Folgendes Klassendiagramm soll das Klassenmodell darstellen:

(unser Klassendiagramm)

## Modellierung

(Unsere Anforderungen)

## Testen

Das Testen einer Mobilen Applikation kann wie bei konventioneller Software durch Unit-, Integration- und Systemtests durchgeführt werden (SE Pressman, S.483). Die Unittests sollen möglichst automatisiert durchgeführt werden und Fehler in einzelnen Funktionen oder Modulen aufdecken (SE Pressman, S.473). Durch Integrationstest wird geprüft, ob einzelne Komponenten korrekt miteinander zusammenarbeiten. Beispielsweise ob die Kameraansicht der Applikation das aufgenommene Foto korrekt an das Texterkennungsmodul weiterleitet (SE Pressman, S.475f).

Da die Entwickler der Software unbewusst dazu tendieren Tests durchzuführen, die zeigen, dass die Software fehlerfrei funktioniert ist ein weiterer wichtiger Bestandteil die Applikation von Nutzern testen zu lassen und deren Feedback bezüglich Benutzerfreundlichkeit und Navigation in den Softwareentwicklungsprozess einfließen zu lassen.

Da mobile Geräte Eigenschaften aufweisen, die auf Desktop Computer nicht zutreffen, beispielsweise viele verschiedene Formfaktoren, unterschiedliche Betriebssystemversionen und limitierte Akkukapazität und Speicherplatz, ist es notwendig spezifische Tests in Bezug auf Gerätekompatibilität, Performance und Netzwerkverfügbarkeit durchzuführen. (SE Pressman, S.483)

Das Android Developers Team schlägt vor die Testphase in 70% Unit-, 20% Integrations- und 10% UI-Tests zu unterteilen, wie aus der „Testing Pyramid“ in Abbildung X zu erkennen ist. Weiterhin ist in der Grafik zu sehen, dass die die Tests nach oben hin komplexer werden, aber im Gegenzug auch repräsentativer für die einwandfreie Funktion der Applikation sind. (https://developer.android.com/training/testing/fundamentals)

(Abbildung)

# Requirements Engineering

Pflichtenheft, Richtlinien, Testkonzept, Funktionsspezifikation

# Modellierungsphase

Anwendungsfälle, Klassendiagram, (Aktivitäts- und Sequenzdiagramme)

# Mobile Development

Die erste Entscheidung die zur Entwicklung mobiler Applikationen getroffen werden muss, ist welche Geräte und Plattformen unterstützt werden sollen. Die in diesem Projekt zu erstellende Applikation, soll auf Smartphones und Tablets lauffähig sein. Aufgrund der limitierten verfügbaren Hardware, wird als Plattform nur Android API-Level XX und höher unterstützt, also Android 6, Marshmallow, und höher.

Um eine Applikation für die Android Plattform zu entwickeln werden zahlreiche Möglichkeiten geboten. Folgend sollen diverse bekannte und weniger bekannte Methoden analysiert und verglichen werden um eine Auswahl zu treffen.

## Untersuchungskriterien

Pressman definiert sieben Kriterien um eine für mobile Applikationsentwicklung geeignete Entwicklungsumgebung und Programmiersprache zu wählen.

1. Die Entwicklungsumgebung sollte Editierung, Projektmanagement, Debugging, Architekturdesign, Dokumentation und Unit-Testing unterstützen.
2. Die Möglichkeit externe (Third-party) Bibliotheken (API/SDK) einzubinden sollte gegeben sein um in nur einer Entwicklungsumgebung arbeiten zu können.
3. Der Quellcode sollte von der Entwicklungsumgebung für bestimmte Endgeräte optimiert werden.
4. Das Testen der Applikation soll in der Entwicklungsumgebung möglich sein.
5. ???
6. Extensive Dokumentation und Tutorials sollten frei verfügbar und einfach verständlich sein.
7. Die Entwicklungsumgebung verfügt über die Funktion Benutzeroberflächen grafisch anzeigen und bearbeiten zu können.

## Java

Die wahrscheinlich bekannteste Methode zur Entwicklung einer Android Applikation ist die Nutzung der Programmiersprache Java in der Entwicklungsumgebung Android Studio.

Android Studio unterstützt Code-Editierung mit autocompletion, zeigt also Vorschläge an, wie man ein angefangenes Wort zu Ende schreiben würde. Dies reduziert Fehler, da man nicht versehentlich einen Variablennamen falsch schreiben kann. Zudem sind Debugging und Unit-Test Features, sowie die Möglichkeit externe Bibliotheken einzubinden implementiert. Die Applikation kann extensiv in der Entwicklungsumgebung getestet und auf verschiedenen Endgeräten emuliert werden.

Aufgrund der großen Beliebtheit der Programmiersprache Java sind viele Features schon implementiert und gut dokumentiert. Es existieren also viele Tutorials zur Android Programmierung in Java.

## Kotlin

Am 17. Mai 2017 hat das Android Team Kotlin als offizielle Android Programmiersprache anerkannt (<https://android-developers.googleblog.com/2017/05/android-announces-support-for-kotlin.html>). Kotlin Programmierung wird, wie auch Java, in Android Studio umgesetzt. Die Funktionalitäten der Entwicklungsumgebung bleiben daher gleich. Der Vorteil von Kotlin ist, dass man durch weniger Code, dasselbe Ergebnis wie mit Java erreichen kann und dabei der Code sogar sicherer gegen beispielsweise NullPointerExceptions ist. Kotlin erweitert die Funktionalitäten von Java ohne dessen Vorteile zu verlieren, da Java Code in Kotlin verwendbar ist. Da jedoch Kotlin in Relation zu Java recht neu ist, sind Dokumentationen und Tutorials nicht so extensiv verfügbar wie für Java. (https://developer.android.com/kotlin/index.html)

## Android NDK

Obwohl Android auf dem Linux Kernel basiert, welcher hauptsächlich aus C-Code besteht, hat sich Java aufgrund der großen Beliebtheit als Programmiersprache für Android Applikationen durchgesetzt. Android Studio unterstützt jedoch durch das Android Native Development Kit (NDK) die Programmierung in C und C++. Dies kann zum einen hilfreich sein, wenn man versucht die Applikation so performant wie möglich zu gestalten, oder wenn man Bibliotheken verwenden will, die auf C oder C++ basieren. (https://developer.android.com/ndk/guides/index.html)

## Xamarin Android

Eine Grundlegend andere Art der Android Programmierung bietet Xamarin Android. Die Programmiersprache ist in diesem Fall C# und die Entwicklungsumgebung Visual Studio. Ähnlich zu Android Studio, bietet Visual Studio Tools zur Editierung mit intelligenter autocompletion und Debugging. Die Applikation kann ebenso in der Entwicklungsumgebung getestet und emuliert werden. Ein grafischer Editor zur für visuelle Elemente ist ebenso verfügbar. (https://docs.microsoft.com/de-de/xamarin/android/get-started/)

## Corona

Corona ist ein Framework zur Entwicklung von Applikationen, das auf der Skriptsprache Lua basiert. Der Texteditor ist frei wählbar und es ist möglich in Echtzeit auf einem emulierten Gerät zu sehen, wie sich die Applikation verhält. Zudem enthält Corona Performanceoptimisierungen bei der Compilierung und native Bibliotheken, ob Java, C oder C++, können jederzeit aufgerufen werden. (https://coronalabs.com/product/)

## PhoneGap

Ein weiteres Framework zur Entwicklung mobiler Applikationen ist PhoneGap, welches den Ansatz der Web-Entwicklung nutzt. Durch HTML, CSS und Javascript können Android Applikationen erstellt werden. Sollten native Bibkiotheken notwendig sein, können diese ebenso verwendet werden. (<https://phonegap.com/products/>)

## Auswahl

Aufgrund der Beschränkung auf Android Geräte zur Entwicklung der Applikation erscheint es sinnvoll eine Entwicklungsumgebung und Sprache zu wählen, die ausschließlich zur Applikationsentwicklung auf Android Betriebssystemen optimiert ist. Obwohl Xamarin eine ansprechende Option ist, wenn die zu entstehende Applikation auf verschiedenen Plattformen verfügbar sein soll, wird im Rahmen dieses Projektes Android Studio als Entwicklungsumgebung verwendet.

Die zur Auswahl stehenden Programmiersprachen grenzen sich somit auf Java, Kotlin und das Android NDK, welches sich aus C und C++ zusammensetzt, ein. Obwohl Kotlin vergleichsweise neu ist und daher im Vergleich zu älteren Programmiersprachen weniger Beispiele zur Verfügung stehen, besteht in der Nutzung von Kotlin kein Nachteil gegenüber zu Java, da die hundertprozentige Interoperabilität sicherstellt, dass jederzeit zu Java Code zurückgefallen werden kann. Android Studio bietet zudem die Möglichkeit automatisch Java Code in Kotlin Code umzuwandeln, was die Einsteigerfreundlichkeit erheblich steigert.

Aufgrund der wachsenden Popularität und der einfachen Implementierbarkeit soll daher Kotlin in Verbindung mit Android Studio die Entwicklungsgrundlage des Projektes bieten.

# Texterkennung

Das Ziel dieser Studienarbeit ist es eine Rechnung mit einem mobilen Endgerät zu erkennen und vorhandene Kontoinformationen auszulesen. Dafür muss ein Dokument zunächst digitalisiert vorliegen. Dies geschieht beim Mobiltelefon mithilfe einer eingebauten Kamera. Das Bild liegt als Matrix von Pixelinformationen vor, in dem für jeden Pixel ein Farbwert gespeichert ist. Für die Texterkennung wird diese Matrix weiterverarbeitet.

## Klassischer Ablauf

Es wird zunächst generell die Funktion der klassischen Texterkennung ohne Deep Learning beschrieben. Der erste Schritt für die Weiterverarbeitung ist eine Binarisierung (OCR, Chapter 2 S.5). Dafür werden zunächst aus den Farbwerten Grauwerte berechnet und im nächsten Schritt wird ein Threshhold-Algorithmus angewandt. Ganz einfach kann zum Beispiel der Wert w = 127 als Schwelle festgelegt werden. Alle Werte kleiner/gleich diesem Wert würden dann zu einer 0 und alle Werte höher als w würden zu einer 1. Mit der geschickten Wahl und Parametrisierung des Threshhold-Algorithmus wird der spätere Mustervergleich verbessert. Nach diesem Schritt können eventuell noch weitere Filterungen durchgeführt werden um Störungen wie Rauschen zu eliminieren und die Qualität der Zeichen zu verbessern. Die Binarisierung des Bildes vereinfacht auch die darauffolgende Layout-Analyse. Hier wird der semantische Inhalt des Dokuments verstanden und in Tabellen oder Felder aufgeteilt. [BILD eines Beispiel-Feldes] Es kann eine Datenbank mit Modellen genutzt werden, die Parameter für unterschiedliche Dokumenttypen speichert und die Form- und spätere Syntaxanalyse unterstützt. Die erkannten Felder werden dann einzeln auf den Inhalt untersucht. Das Feld kann zunächst anhand der Formelemente, wie zum Beispiel begrenzende Linien ausgerichtet werden. Dafür wird untersucht, ob Linien ungerade sind und das Feld wird dementsprechend gedreht. Dann werden Formelemente herausgeschnitten um für jedes Feld nur noch die Zeichen zu untersuchen. Ein Text wird dann in die einzelnen Zeichen aufgeteilt.

Für die möglichen Zeichen muss es eine Datenbank mit Mustern geben. Diese Datenbank kann nach Sprache und Zeichensatz aufgeteilt sein. Jedes Zeichen wird dann Pixel für Pixel mit den Mustern verglichen und es wird eine Wahrscheinlichkeit für jedes Muster berechnet. QUELLE Außerdem kann noch eine Syntaxanalyse durchgeführt werden. Wird zum Beispiel für ein Feld im ausgewählten Modell eine Email-Adresse verlangt, kann dies überprüft werden. Mit einer weiteren Bibliothek kann auch eine Kontextanalyse durchgeführt werden. Wenn zum Beispiel das Wort „8aum“ erkannt werden, kann daraus das Wort „Baum“ abgeleitet werden.

[Mustererkennung Beispiel]

Es fehlt:

## Vorverarbeitung

In diesem Kapitel werden Methoden beschrieben um die Qualität eines Bildes vor der Texterkennung zu verbessern.

### Glätten und verschwämmen

In der Computer-Bildverarbeitung gibt es Algorithmen für das Glätten und Verschwämmen von Bildern. Dies dient dazu Kanten zu glätten, wie im ersten Bild zu sehen. Dadurch werden die Zeichen für die Maschine besser zu lesen. Außerdem können Störungen reduziert werden, wie im zweiten Bild zu sehen.

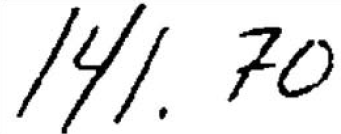
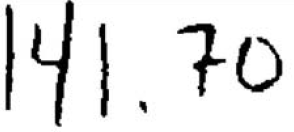




### Textkrümmung

Um den Text für die Maschine besser lesbar zu machen, können Neigungen korrigiert werden. So kann das gesamte Dokument anhand der Textlinien analysiert werden und ein Offset-Winkel errechnet werden mit dem korrigiert wird. Dies kann auch mit einem einzelnen Schriftzug geschehen, wie in der unteren Abbildung zu sehen.





## Auswahl einer OCR-Bibliothek

In diesem Kapitel soll beschrieben werden, welche Möglichkeiten es gibt optische Texterkennung auf einem mobilen Endgerät zu implementieren. Eine naheliegende Methode wäre es, die in den vorherigen Kapiteln beschrieben technischen Aspekte in einer eigenen Bibliothek zu implementieren. Dafür werden jedoch größere Entwicklungsressourcen benötigt, als sie für dieses Projekt gegeben sind. Um effektive Texterkennung zu ermöglichen, wird eine große Menge an Daten benötigt. Diese Daten setzten sich aus unterschiedlichen Zeichensätzen und unterschiedlichen Schriftarten zusammen. Derartige Daten zu sammeln ist zeitintensiv. Zum anderen wäre die Entwicklung der vorher beschrieben Algorithmen zeitaufwändig. Wenn dieses Risiko aufgenommen wird, kann in abschätzbarer Zeit eine OCR-Bibliothek entwickelt werden, diese wäre aber funktional begrenzt und ungenau. Fertige OCR-Bibliotheken haben dahingegen eine größere Funktionalität und eine höhere Genauigkeit, aufgrund der größeren Entwicklungszeit und -Ressourcen. Außerdem werden viele OCR-Bibliotheken aktiv weiterentwickelt, was einen weiteren Vorteil bietet. Zum anderen wäre die Entwicklung der vorher beschrieben Algorithmen zeitaufwändig. Wenn dieses Risiko aufgenommen wird, kann in abschätzbarer Zeit eine OCR-Bibliothek entwickelt werden, diese wäre aber funktional begrenzt und ungenau. Für kommerzielle Anwendungen werden meist Lizenzen von proprietäre SDKs eingekauft, da diese meist genauer sind. Eine Lizenz für die Aspire OCR SDK kostet aber zum Beispiel in der Lite-Version für Java schon 5000 Dollar pro Entwickler [2]. Aufgrund der hohen

Wie in den vorherigen Kapiteln beschrieben ist die Ausführung von Texterkennungs-Algorithmen rechenintensiv. Wenn die Texterkennung direkt auf den mobilen Endgeräten ausgeführt wird kann die begrenzte Rechenkraft ein Hindernis darstellen. Ein Weg um mögliche Performance-Probleme zu umgehen, wäre die Texterkennung in eine Cloud-Lösung auslagern. Die Anwendung würde also ein Foto aufnehmen und dann für die optische Charaktererkennung einen Webservice nutzen. Die Anwendung lädt also das aufgenommene Foto hoch, dieses wird auf Online-Servern von OCR-Software analysiert und der erkannte Text wird wieder heruntergeladen. Mit diesem Ansatz braucht die Anwendung jedoch eine aktive Internetverbindung. Da ein Bild hochgeladen werden muss, würde dies meist nur über eine aktive WLAN-Verbindung funktionieren, da sonst die Geschwindigkeit nicht ausreichend ist. Außerdem kann nicht garantiert werden, dass der OCR-Server immer erreichbar und nicht überlastet ist. Bei einer Offline-Lösung würde die Anwendung zwar mehr Rechenkraft benötigten, der Engpass über die Internetverbindung würde aber wegfallen.

### Google Mobile Vision

Google Mobile Vision ist ein Framework, welches bildverarbeitende Funktionen für mobile Endgeräte bereitstellt. Mit diesem Framework können Gesichter erkannt, Barcodes gescannt oder Text erkannt werden. Dieses Framework kann ohne aktive Internetverbindung genutzt werden und eignet sich für Echtzeit-Anwendungen(<https://android-developers.googleblog.com/2016/06/android-mobile-vision-text-api.html>)

Die Texterkennungs-API erkennt jede Sprache, die auf dem römischen Alphabet basiert. Der auf einem Bild erkannte Text wird in Blöcke, Linien und Worte aufgeteilt und bietet damit auch relative Positionsinformationen der erkannten Wörter, die später zur Dokumentanalyse genutzt werden können.

(<https://developers.google.com/vision/android/text-overview>)

TESTERGEBNISSE

Genauigkeit

Schnelligkeit

Repräsentation der Daten

### Tesseract

Tesseract ist eine OCR-Engine, die ursprünglich von HP entwickelt wurde und seit 2006 von Google weiterentwickelt wird [3]. Tesseract ist in C++ entwickelt, es gibt auf GitHub jedoch Bibliotheken für unterschiedliche Programmiersprachen, die ein Schnittstelle bieten. Tesseract kann über 100 unterschiedliche Sprachen erkennen und unterstützt eine Menge an Ausgabe- und Eingabeformaten. Mit einer Wertung von aktuell 17700 Stars ist Tesseract die beliebteste freie OCR-Bibliothek auf GitHub. Es gibt auf GitHub einen Fork des Tesseract-Repositories für Android (<https://github.com/rmtheis/tess-two>)

# Implementierung

Was haben wir genau gemacht? Anwendung vorstellen! Bezug auf Modellierungsphase nehmen

# Fazit und Ausblick

Wo gab es Probleme ? Was hat gut geklappt, was könnte man besser machen? Was könnte in Zukunft noch umgesetzt werden?

# Literaturverzeichnis

[1] Buch, Author, Jahr – Titel (Abk.)

[2] <https://asprise.com/royalty-free-library/java-ocr-source-code-open-order.html>

[3] <https://github.com/tesseract-ocr/tesseract>

[4] <https://github.com/tmbdev/ocropy>

[5] <https://developers.google.com/vision/android/text-overview>

[6] <https://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html>