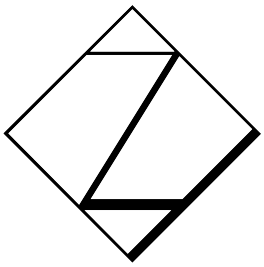


Westsächsische Hochschule Zwickau
University of Applied Sciences
HOCHSCHULE FÜR MOBILITÄT | UNIVERSITY FOR MOBILITY

BACHELORARBEIT

EINE KATEGORISIERTE SAMMLUNG UND PROTOTYPISCHE IMPLEMENTIERUNG VON STANDARD-TESTFÄLLEN IM BEREICH APP-ENTWICKLUNG

Jakob Burger

20. November 2020

Betreuer:

Prof. Dr.-Ing. Rainer Wasinger
Westsächsische Hochschule Zwickau

André Gorzel (B.Sc. Wirtschaftsinformatik)
T-Systems Multimedia Solutions

Danksagung

Ich möchte Prof. Dr. Wasinger und André Gorzel danken, für die Betreuung und Begleitung dieser Arbeit. Besonderer Dank gilt weiterhin Marcus Trepte und Oliver Löwe, welche bei Fragen und Problemen immer behilflich waren und sich außerdem am Entstehungsprozess der hier beigefügten Standard-Tests beteiligten.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ISTQB International Software Testing Qualifications Board.

MDC Mobile Device Cloud.

MMS T-Systems Multimedia Solutions.

TIC Test and Integration Center.

1 Einleitung

1.1 Motivation und Ziel der Arbeit

Das Testen von Software ist integraler Bestandteil ihrer Entwicklung. Ein frühes Anfertigen von Software-Testfällen kann Ressourcen sparen und gleichzeitig zu besserer Software führen.[Moh14]
Das Entwickeln und Testen von mobilen Anwendungen stellt die Entwickler vor zusätzliche Herausforderungen: Um eine möglichst große Zielgruppe zu erreichen, muss eine App auf möglichst vielen unterschiedlichen Endgeräten lauffähig sein. Dabei unterscheiden sich diese mobilen Endgeräte in vielerlei Hinsicht: Betriebssystem, Hardware, Bildschirmauflösung und weitere Faktoren erzeugen eine große Menge an verschiedenen Betriebsszenarien.
Mit dem wachsenden Erfolg von mobilen Endgeräten hat sich das Testen von Apps zu einer eigenständigen Branche entwickelt. Viele Entwickler lassen ihre Apps heutzutage von dafür spezialisierten Dienstleistern testen. Bereits 1999 gründete die *T-Systems Multimedia Solutions (MMS)* eine solche Einrichtung zum Testen von Software: Das *Test and Integration Center (TIC)*. Unter anderem wird mit der *Mobile Device Cloud (MDC)* auch eine Cloud Plattform angeboten, welche Zugriff auf über 200 reale mobile Endgeräte anbietet.

Beim Testen von Software wird unter anderem zwischen manuellem Testen und automatisiertem Testen unterschieden. Beim automatisierten Testen wird die Software mithilfe von Automatisierungstools und selbstgeschriebenem Code getestet. Beim manuellen Testen werden Testfälle von menschlichen Testern ausgeführt und die entsprechenden Ergebnisse ausgewertet. Diese beiden Arten des Testens sollen nicht als gegeneinander konkurrierend verstanden werden, sondern einander unterstützen und verbessern.

Wie im Folgenden dargelegt wird, bieten viele Apps dieselben Funktionalitäten an, welche allerdings oft auf unterschiedliche Weise implementiert und unterschiedlich eingesetzt werden. Ob eine Funktionalität getestet werden muss, hängt unter anderem von ihrer Bedeutsamkeit für die App und ihrer

Komplexität ab. Erfahrungen im Bereich App-Testing zeigen jedoch, dass gewisse Funktionalitäten zwingend getestet werden müssen, um eine korrekte Nutzbarkeit der App zu gewährleisten.

Zwei Probleme sind an dieser Stelle erkennbar:

1. Aufgrund der Tatsache, dass viele Apps dieselben Funktionalitäten bereitstellen, ähneln sich auch die zu den Apps erstellten Testfälle oft sehr. Ein Dienstleister, welcher für mehrere Apps automatisierte Testfälle entwickelt, muss oft mehrmals einen inhaltlich identischen Testfall für verschiedene Apps schreiben.

2. Dadurch, dass zu jeder Funktionalität einer App erneut überdacht werden muss, welche Testfälle dazu geschrieben werden sollen, entsteht ebenfalls erhöhter Arbeitsaufwand.

Ziel dieser Arbeit ist es eine potenzielle Lösung für diese beiden Probleme zu geben. Dazu wird eine Liste mit Standard-Testfällen erstellt. Diese Liste wird nach den jeweiligen potenziellen Funktionalitäten einer App sortiert. Außerdem werden ausgewählte Standard-Testfälle prototypisch implementiert. Dabei wird besonders darauf geachtet, die prototypisch implementierten Testfälle möglichst generisch zu gestalten.

1.2 Methodisches Vorgehen

Um geeignete Standard-Testfälle zu erstellen wird eine Kombination aus mehreren Verfahren angewandt. Zuerst wird der Gegenstand eines Testfalls festgelegt. Dazu muss der typische Aufbau der zu testenden Funktionalität bekannt sein. Der Markt für Apps ist groß und wächst immer noch stark an. [Ann20] Deswegen stellt es sich als Herausforderung dar, belastbare und aktuelle Fakten über den Aufbau von Apps und die bereitgestellten Funktionalitäten zu finden. Aus diesem Grund wird eine eigene Auswertung der aktuell am meisten benutzten Apps erstellt.

Die Auswertung ist für die spätere Arbeit aus zwei Gründen wichtig: Erstens bietet sie einen Überblick über die häufig von Apps bereitgestellten Funktionalitäten. Daraus werden im späteren Verlauf Testfälle erstellt. Zweitens kann aus der Auswertung abgeleitet werden, welche UI-Elemente häufig verwendet werden. Aus diesem zweiten Aspekt werden wertvolle Kenntnisse zur Erstellung der prototypischen Testfälle gewonnen.
Bei der Auswertung wird eine Auswahl der aktuell am meist benutzen Apps miteinander verglichen. Die Auswertung besteht im Kern aus einer Navigation durch die jeweilige App. Dabei wird nach jeder Interaktion mit der App ein Screenshot aufgezeichnet. Diese Screenshots werden später auf UI-Elemente und Funktionalitäten der App untersucht. Die Ergebnisse sind in Unterabschnitt 2.2 *Analyse populärer Apps* grafisch dargestellt.

Die aus der Auswertung gewonnenen Testfälle werden im nächsten Schritt nach Rücksprache mit Experten des TIC verbessert und erweitert. Die Testfälle sind nach abgewandelten Richtlinien des *International Software Testing Qualifications Board (ISTQB)* ausformuliert. Anschließend werden sie nach den entsprechenden Funktionalitäten kategorisiert und in Form einer Checkliste zusammengestellt.
Eine Auswahl der Testfälle wird in prototypischer Weise implementiert. Dabei wird besonders darauf geachtet, einen möglichst generischen Ansatz zu wählen. Die prototypischen Testfälle sind so entwickelt, dass möglichst viele Apps ohne große Abwandlung gegen sie getestet werden können.

Bei den prototypischen Testfällen handelt es sich um Unit-Testfälle, die in *Java* geschrieben sind. Zusätzlich werden die Test-Automatisierungs-Frameworks *TestNG* und *Appium* verwendet.
Um einen möglichst generischen Ansatz der Testfälle zu gewährleisten, werden die vom Nutzer benötigten Angaben über die zu testende App minimiert. Beispielsweise muss der Nutzer beim Testen des Logins seiner App lediglich die IDs der beiden Loginfelder und Anmeldedaten zu seiner App vor Programmstart eingeben. Daraufhin werden alle Standard-Testfälle für die Funktionalität „Login“ ausgeführt.

Hintergrund der meisten Testfälle ist ein selbst programmierter App-Crawler. Dieser analysiert vor Ausführung des Testfälle den Aufbau der App. Während des Testfalls ist es dadurch möglich, an eine beliebige Stelle innerhalb der App zu navigieren.

1.3 Definitionen

Innerhalb dieser Arbeit werden Begriffe verwendet, welche im allgemeinen Sprachgebrauch oft mehrere Bedeutungen haben. Um Fehlinterpretationen vorzubeugen sind im Folgenden alle eventuell missverständlichen Begriffe definiert. Teilweise werden die Definitionen anderer Autoren herangezogen:

Testfall:
An dieser Stelle wird auf die Definition eines Testfalls laut ISTQB verwiesen:
„[Ein Testfall] umfasst folgende Angaben: die für die Ausführung notwendigen Vorbedingungen, die Menge der Eingabewerte [...], die Menge der vorausgesagten Ergebnisse, sowie die erwarteten Nachbedingungen. Testfälle werden entwickelt im Hinblick auf ein bestimmtes Ziel bzw. auf eine Testbedingung, wie z.B. einen bestimmten Programmpfad auszuführen oder die Übereinstimmung mit spezifischen Anforderungen zu prüfen [...]“.[IST]
Insbesondere bezieht sich die Arbeit auf Testfälle im Bereich App-Entwicklung. Also auf Testfälle, welche für mobile Anwendungen (Apps) geschrieben wurden.

Standard-Testfall:
Als Standard-Testfall ist im Sinne der Arbeit ein Testfall zu verstehen, welcher für viele Programme, beziehungsweise Apps, zu implementieren ist. Das Wort „viele“ kann an dieser Stelle nicht näher spezifiziert werden.

Funktionalität:
Hitchins definiert eine Funktion folgendermaßen:
„An action, a task, or an activity performed to achieve a desired outcome.“ [Hit07]

Diese interdisziplinäre Definition lässt sich gut verwenden, um eine Funktion eines Programms zu definieren. Um Verwechslungen mit dem Programmkonstrukt einer Funktion zu vermeiden, wird stattdessen das Wort „Funktionalität“ benutzt.

Prototypisch implementierte Testfälle:
„Ein Software-Prototyp ist ein ausführbares Modell mit wesentlichen Eigenschaften des Zielsystems, das Grundlage für die Systemspezifikation ist und die Kommunikation zwischen Kunden und Entwickler unterstützt. [...]“[Gus92] In dieser Arbeit wird auf „prototypisch implementierte Testfälle“ verwiesen. Damit sind Testfälle gemeint, welche prinzipiell ausführbar sind, allerdings nicht alle gewünschten Eigenschaften besitzen, sodass sie im momentanen Zustand nicht innerhalb eines Software-Projekts eingesetzt werden sollten.

Gültigkeit & Allgemeingültigkeit:
In der Arbeit werden diese Begriffe vor allem in Bezug auf eine „möglichst gültige Implementierung“ beziehungsweise eine „allgemeingültige Implementierung“ von Testfällen verwendet. Die Gültigkeit eines Testfalls meint in diesem Sinne die Anzahl an Apps, die sich mittels des Testfalls testen lassen, in Relation zu allen existierenden Apps. Selbstverständlich ist die Gültigkeit eines Testfalls nur ein hypothetischer Wert. Ein allgemeingültiger Testfall wäre demnach ein Testfall, mit dem sich jede existierende App testen lässt. Die Erstellung eines allgemeingültigen Testfalls ist nicht Teil dieser Arbeit.

Screen:
Ein „Screen“ einer App sei hier als die Summe aller zu einem gewissen Zeitpunkt dargestellten Informationen definiert. Insbesondere enthält ein Screen mehrere UI-Elemente. Die Interaktion mit einem dieser UI-Elemente kann zur Navigation zu einem neuen Screen führen. Zwei Screens sind dann identisch, wenn sie dieselben Informationen darstellen.

2 Auswahl der Standard-Testfälle

2.1 Auswahlverfahren

Als Standard-Testfall ist ein Testfall anzusehen, der für so gut wie jede App zu implementieren ist. Testfälle sind in der Regel an die Funktionalitäten der App gebunden. Es existieren zu den meisten Funktionalitäten auch Standard-Testfälle. Sobald feststeht, welche Funktionalitäten eine App enthalten soll, können und sollten daraus Testfälle generiert werden. Aus diesem Grund war es wichtig, die hier beschriebene Auflistung von Standard-Testfällen so zu konzipieren, dass sie zu einer möglichst frühen Phase der Softwareentwicklung verwendet werden kann.
Aus diesem Gedankengang heraus wurden die Testfälle nach den potenziellen Funktionalitäten einer App ausgewählt. Funktionalitäten, welche häufig von Apps angeboten werden, wurden durch eine Analyse der aktuell am meisten heruntergeladenen Apps erforscht. Siehe hierzu Unterabschnitt 2.2 *Analyse populärer Apps*.

Diese Funktionalitäten dienten als Kategorien für die Auflistung der Standard-Testfälle. Im Anschluss wurden zu jeder dieser Kategorien Testfälle erstellt und dokumentiert. Dabei wurden eigene Erfahrungen im Bereich App-Testing verwendet. Um eine möglichst hohe Relevanz der Testfälle zu gewährleisten, wurden die wesentlichen Aspekte der jeweiligen Funktionalität getestet.

Ein Standard-Testfall soll möglichst essenzielle Anforderungen der entsprechenden Funktionalität überprüfen. Die Auflistung der Standard-Testfälle wurde so entworfen, dass die Standard-Testfälle der jeweiligen Funktion zwingend umgesetzt werden sollten, um eine anforderungsgerechte Funktion sicherzustellen. Dementsprechend wurden die Standard-Testfälle ausgewählt.
Außerdem war eine möglichst hohe Gültigkeit der Auflistung ebenfalls wichtig bei der Auswahl der Standard-Testfälle. Damit die Aufzählung der Standard-Testfälle bei möglichst vielen Projekten verwendet werden kann, sollte sie einen möglichst hohen Grad an Gültigkeit aufweisen. Die Standard-Testfälle

selbst können sich also jeweils nur auf die typische Implementierung ihrer jeweiligen Funktionalität beziehen. Sollte eine Funktionalität anders als für die entsprechenden Standard-Testfälle angenommen implementiert worden sein, müssen eventuell auch die dazugehörigen Standard-Testfälle umgeschrieben, erweitert oder gar ignoriert werden.
Für manche Funktionalitäten gibt es keine allgemein übliche Implementierung. Als Beispiel sei hier etwa die Funktionalität „Navigation“ genannt. Die Navigation einer App hängt stark von ihrem Aufbau ab, welcher nur schwer verallgemeinert werden kann. Dementsprechend allgemein muss die Dokumentation eines Standard-Testfalls zu dieser Funktionalität ausfallen. Dennoch sollten auch solch allgemein gehaltenen Standard-Testfälle implementiert werden, um eine anforderungsgerechte Funktion zu garantieren.

2.2 Analyse populärer Apps

2.2.1 Vorgehen

Im Jahr 2019 wurden 204 Milliarden Apps heruntergeladen.[Ann20] Im Vergleich zum Vorjahr wurden 2019 700.000 zusätzliche Apps im Apple App Store zum Download angeboten. [Cow20] Im Google Play Store waren es 200.000 Apps.[Mat20] Belastbare, aktuelle Daten zum Aufbau von Apps sind schwer zu finden. Konkret gesucht wurden Informationen zu Funktionalitäten der Apps und zu verwendeten UI-Elementen sowie deren Attribute.
In der durchgeführten Analyse wurden möglichst alle Screens einer App analysiert. Der Ablauf der Analyse bestand aus zwei Phasen:

In der ersten Phase wurde jede App einzeln heruntergeladen und installiert. Anschließend wurde manuell durch die App navigiert. Dabei wurde nach jeder Interaktion mit der App ein Screenshot aufgezeichnet. Als Interaktionen werden hier das Drücken auf einen Button, das Scrollen auf einem Screen, das Drücken des Zurück-Buttons unter Android, bzw. das Wischen nach rechts unter iOS, sowie die Eingabe von Text in Textfelder gezählt. Diese Systematik wurde bewusst gewählt, um das Vorgehen des später entwickelten App-

Crawlers zu imitieren. Allerdings wurden keine identischen Screenshots aufgezeichnet. Daraus folgt, dass ein Screen nicht zweimal analysiert wurde, selbst wenn ein mehrmaliges Aufrufen des Screens zur Navigation nötig war. Dadurch wurde die Verzerrung der Daten eingeschränkt.

In der zweiten Phase wurden die so erstellten Screenshots ausgewertet. Dabei wurden ausgewählte sichtbare UI-Elemente und deren Attribute (zum Beispiel Text oder Anklickbarkeit) erfasst. Zu jedem UI-Element wurde eine Beschreibung erstellt, welche den Aufbau und die Funktion des UI-Elements enthält. Dadurch können eventuelle Unklarheiten über die analysierten UI-Elemente aufgeklärt werden.
Als weiterer Schritt wurde eine Liste von Aktions-Stichwörtern erstellt. Dazu wurden häufig verwendete Worte wie „OK“, „Weiter“ oder „Abbrechen“ erfasst. Außerdem wurde jeder analysiere Screenshot zu einer Funktionalität der App zugeordnet.
Die Auswahl der zu erfassenden Elemente wurde mit Rücksicht auf den später entwickelten App-Crawler getroffen. Elemente wie Links, Buttons, Pop-ups oder Eingabefelder sind für die Entwicklung eines App-Crawlers wichtig. Diese Elemente ermöglichen die Interaktion mit der App.
Es wurde jeweils ausgewertet, ob ein Screenshot die ausgewählten Elemente enthält. Es wurde explizit nicht die Anzahl der Elemente in einem Screenshot gezählt. Als Beispiel wurden für einen Screenshot mit fünf Buttons nicht fünf, sondern nur ein Button erfasst. Dieses Vorgehen wurde gewählt, um einen möglichst unverzerrten Überblick über häufig verwendete UI-Elemente zu erhalten.

2.2.2 Analysierte Apps

Insgesamt wurden 13 verschiedene Apps ausgewertet. Dabei wurden die Apps mit den höchsten Downloadzahlen zum Zeitpunkt der Analyse, dem 23.09.2020 ausgewählt. Ausgewertet wurde nach Möglichkeit und Verfügbarkeit auf einem Android- sowie auf einem iOS-Gerät. Konkret wurde auf einem *iPod Touch 7. Generation* unter iOS 14 und einem *Google Pixel 2 XL* unter Android 10 ausgewertet. Die Auswahl der Geräte wurde nach ihrer Verfügbarkeit festgelegt.

Die folgenden 13 Apps wurden ausgewertet:

- **Lidl Plus** (Lidl Digital International GmbH & Co. KG)
- **Widgetsmith** (Cross Forward Consulting, LLC)
- **Color Widgets** (MM Apps, Inc.)
- **Pinterest** (Pinterest)
- **Microsoft Teams** (Microsoft Corporation)
- **Instagram** (Instagram, Inc.)
- **Google Maps - Transit & Essen** (Google LLC)
- **QR & Barcode Scanner (Deutsch)** (TeaCapps)
- **X Icon Changer** (ASTER PLAY)
- **Corona-Warn-App** (Robert Koch-Institut)
- **TikTok** (TikTok Pte. Ltd.)
- **eBay Kleinanzeigen** (eBay Classifieds Group)
- **Simple Photo Widget** (Life Simple)

Die App *Lidl Plus* war sowohl unter iOS als auch unter Android eine der am meisten heruntergeladenen Apps und wurden unter beiden Betriebssystemen ausgewertet. Die folgenden Apps zählten nur unter iOS zu den am meisten heruntergeladenen Apps und wurden deshalb nur unter iOS ausgewertet: *Widgetsmith*, *Color Widgets*, *Pinterest*, *Microsoft Teams*, *Instagram* und *Google Maps - Transit & Essen*. Dasselbe gilt in Bezug auf das Android Betriebssystem für die folgenden Apps: *QR & Barcode Scanner (Deutsch)*, *X Icon Changer*, *Corona-Warn-App*, *TikTok*, *eBay Kleinanzeigen* und *Simple Photo Widget*. Diese Apps wurden aus demselben Grund nur unter Android ausgewertet.
Die *Corona-Warn-App* konnte aus technischen Gründen nur unter Android ausgewertet werden. Die Apps *Photo Widget*, *WhatsApp Messenger* sowie die App *QR & Barcode Scanner* des Publishers *Gamma Play* konnten aus technischen Gründen nicht ausgewertet werden.

2.2.3 Ergebnisse

Abbildung 1 liefert eine Übersicht über alle insgesamt aufgezeichneten UI-Elemente über alle Apps hinweg. Dabei zeigt die vertikale Dimension die Häufigkeit des Vorkommens eines UI-Elements an. Diese sind im Diagramm nach ihrer Häufigkeit sortiert. Insgesamt wurden 294 Screenshots und somit ebenso viele unterschiedliche Screens ausgewertet. Die Summe aller Elemente ist jedoch deutlich höher, da viele Screens mehrere Elemente aufweisen. Bestimmte Elemente wurden weiter analysiert, da sie in gewissen Variationen auftreten. Die Farben der einzelnen Balken entsprechen dabei diesen unterschiedlichen Variationen der UI-Elemente. Mehr dazu in den folgenden Diagrammen.

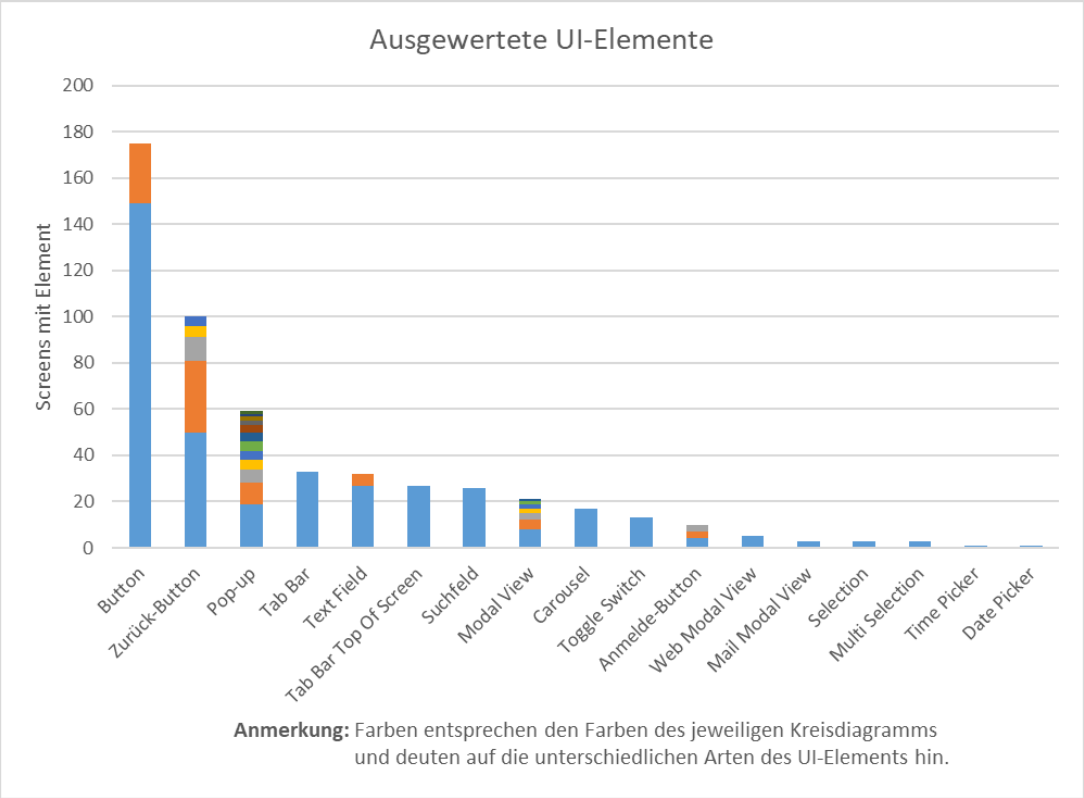


Abbildung 1: Ein Überblick über alle gezählten UI-Elemente

Zu beachten ist die eventuell irreführende Verteilung der Tab Bar-Elemente. Durch die in Unterunterabschnitt 2.2.1 *Vorgehen* beschriebene Vorgehensweise wurden alle relevanten UI-Elemente eines Screens erfasst. Tab Bar-Elemente sind jedoch derart designt, dass sie auf mehreren Screens vorkommen. Dieses Mehrfachvorkommen wurde erfasst, die Daten wurde dahingehend nicht bereinigt.

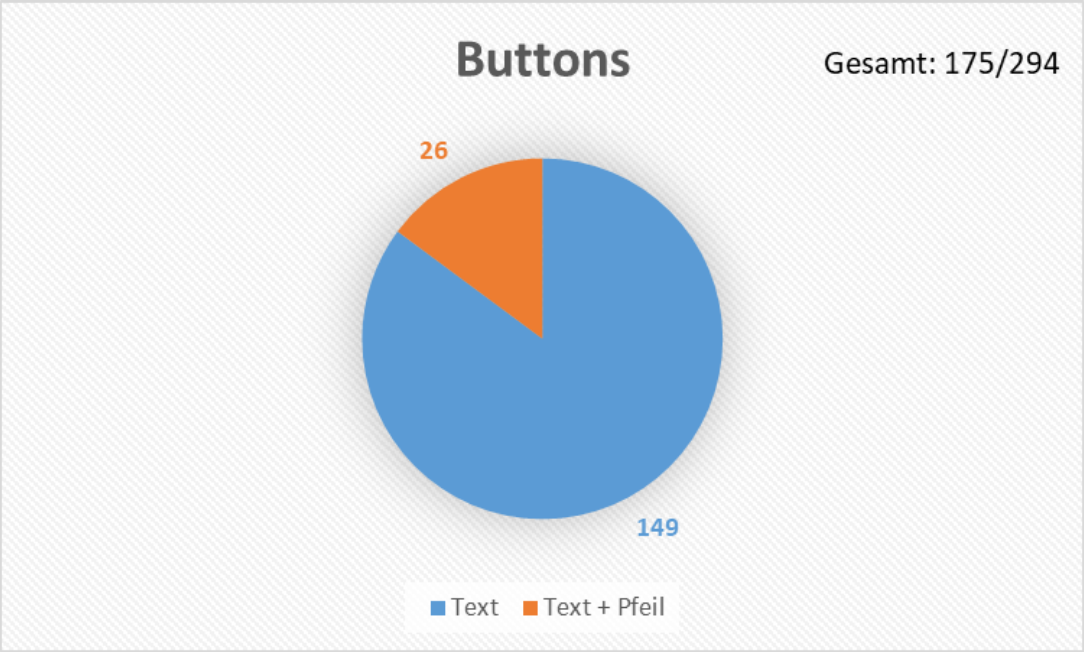


Abbildung 2: Screens, welche mindestens ein UI-Element „Button“ enthalten, sowie die Verteilung der beiden Variationen des UI-Elements

Abbildung 2 zeigt alle gezählten Button-Elemente sowie deren unterschiedliche Variationen. Bei der Analyse wurde zwischen Buttons mit Text und Buttons mit Text und zusätzlichem Pfeil nach rechts unterschieden.
Das Kreisdiagramm ist genauso aufgebaut wie alle folgenden Kreisdiagramme. Die verschiedenen Farben weisen auf die unterschiedlichen Variationen der UI-Elemente hin. In der rechten oberen Ecke wird aufgezeigt wie viele der 294 ausgewerteten Screens das im Diagramm dargestellte UI-Element enthalten. Außen an den Abschnitten des Kreisdiagramms ist jeweils angegeben, wie viele der insgesamt gezählten UI-Elemente zu der jeweiligen Variation gehören. Die Summe all dieser Zahlen ergibt somit die oben rechts angegebene Zahl an insgesamt gezählten UI-Elementen.
175 von 294 ausgewerteten Screens enthalten Buttons. Somit ist der Button das am häufigsten vorkommende UI-Element innerhalb der Analyse. Die dennoch verhältnismäßig geringe Anzahl folgt unter anderem auch aus dem Vorgehen während der Aufzählung:

Als Button galt ein UI-Element im Sinne der Analyse nur, falls es nicht Teil eines weiteren UI-Elements war. Elemente, wie zum Beispiel Tab Bar-Elemente, welche als eine Ansammlung verschiedener Buttons angesehen werden könnten, wurden separat gewertet. Auch die Buttons innerhalb einer Modal View oder einem Pop-up wurden nicht in die Wertung mitaufgenommen. Falls beim Klick auf einen Button zum vorherigen Screen navigiert wurde, galt der Button im Sinne der Analyse als Zurück-Button und wurde ebenfalls nicht als Button gewertet. Eine Übersicht über die unterschiedlichen Zurück-Buttons findet sich in Abbildung 3.

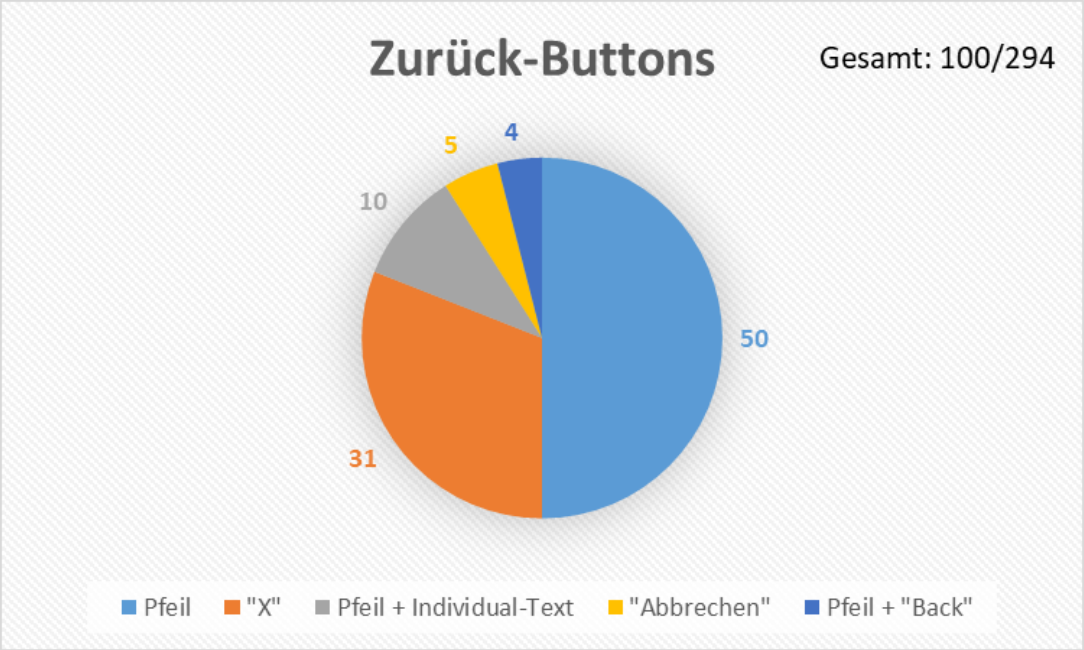


Abbildung 3: Screens, welche mindestens ein UI-Element „Zurück-Button“ enthalten, sowie die Verteilung der verschiedenen Variationen des UI-Elements

In Abbildung 3 ist erkennbar, dass über die Hälfte aller Zurück-Buttons einen nach links deutenden Pfeil besitzen. 50 von 100 gezählten Zurück-Buttons bestehen sogar nur aus einem Pfeil nach links.

Obwohl die Systemsprache auf beiden Testgeräten auf Deutsch gestellt war, wurden dennoch Zurück-Buttons mit englischsprachigem Text angezeigt. Das Wort „Back“ wurde viermal verwendet, aber auch unter den im Diagramm grau dargestellten Zurück-Buttons mit Individual-Text wurden Wörter wie „Cancel“ und „Exit“ aufgezeichnet.
Bemerkenswert ist weiterhin, dass jeder dritte ausgewertete Screen einen Zurück-Button enthielt. Google empfiehlt Android-Entwicklern keinen Back-Button innerhalb der UI einer App bereitzustellen.[And20]

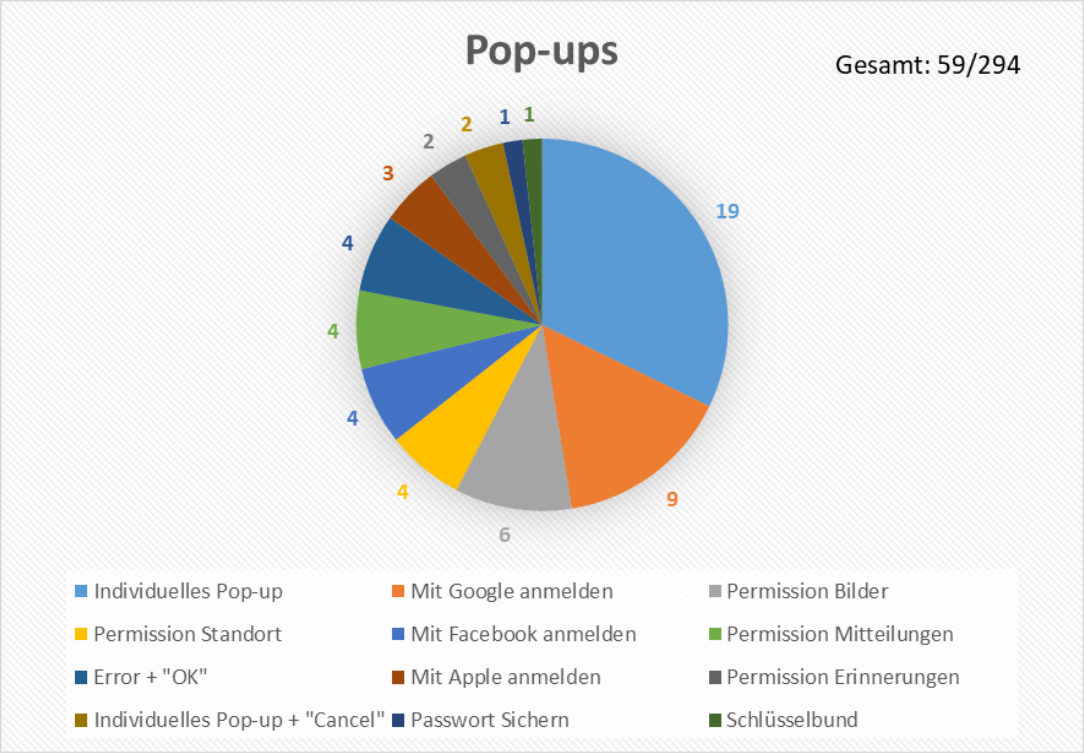


Abbildung 4: Screens, welche mindestens ein Element „Pop-up“ enthalten, sowie die Verteilung der verschiedenen Variationen des Elements

Abbildung 4 zeigt unterschiedliche Arten von Pop-ups, welche im Sinne der Analyse als eigenständige UI-Elemente angesehen wurden. Von allen näher untersuchten UI-Elementen weisen Pop-ups die größte Diversität auf. Ein Drittel aller gezählten Pop-ups konnten nicht standardisiert werden, sie sind un-

ter dem Bereich „Individuelles Pop-up“ zusammengefasst. Auch die restlichen Pop-ups ähneln sich nicht stark, was die Anzahl der Variationen als auch deren recht gleichmäßige Verteilung erklärt.

Dass die beiden jeweils einmal vorgekommenen Pop-ups „Passwort Sichern“ und „Schlüsselbund“ nicht auch zu den individuellen Pop-ups gezählt wurden, ergibt sich aus der Vorgehensweise der Analyse: Da die Apps nacheinander analysiert wurden, wurden die beiden Pop-ups als eigenständige Variationen gewertet, in der Annahme sie würden auch bei der Analyse anderer Apps vorkommen. Tatsächlich stehen die beiden Pop-ups mit Funktionalitäten des Betriebssystems in Verbindung, weshalb sie auch im weiteren Verlauf der Analyse als eigenständige Variationen gezählt wurden.
Außerdem bemerkenswert ist die ungleiche Verteilung an Pop-ups zur Anmeldung mit Drittanbietern, als auch von Pop-ups, welche Berechtigungen vom Nutzer erfragen. Das Pop-up zur Anmeldung mit Google kam in der Analyse doppelt so oft vor wie die entsprechenden Pop-ups der anderen Drittanbieter Facebook und Apple. Bei den Berechtigungs-Pop-ups wurde das Pop-up, welches Zugriff auf die Bilder des Nutzers erfragt, am häufigsten gezählt.

Abbildung 5: Screens, welche mindestens ein UI-Element „Textfeld“ enthalten, sowie die Verteilung der beiden Variationen des UI-Elements

In Abbildung 5 sind die unterschiedlichen Arten von Textfeldern aufgezeigt. Textfelder bieten im Vergleich zu den sonst gezählten UI-Elementen eine andere Art von Nutzer-Interaktion, nämlich der Eingabe von Text beziehungsweise Zahlen. Diese Unterscheidung wurde aus dem Grund gewählt, da beim Klick auf ein Textfeld zur Eingabe von Zahlen statt einer vollständigen Tastatur nur eine Telefontastatur angezeigt wird. Ähnlich wie bei Buttons wurden Suchfelder als eigenständige UI-Elemente gesehen und separat gezählt.
Textfelder zur Eingabe von Zahlen wurden während der Analyse ausschließlich mit der Aufforderung zur Eingabe einer Telefonnummer gezählt. Textfelder zur Eingabe von Text wurden allerdings ebenfalls zur Eingabe von Telefonnummern verwendet.

Ein Modal View-Element ähnelt einem Pop-up, legt sich allerdings mit einer Animation von unten über den vorherigen Screen. Mit einem Wisch nach unten kann das Modal View-Element geschlossen werden. Zusätzlich besitzen Modal Views oft Buttons zum Schließen der Modal View, meist am oberen linken Rand.

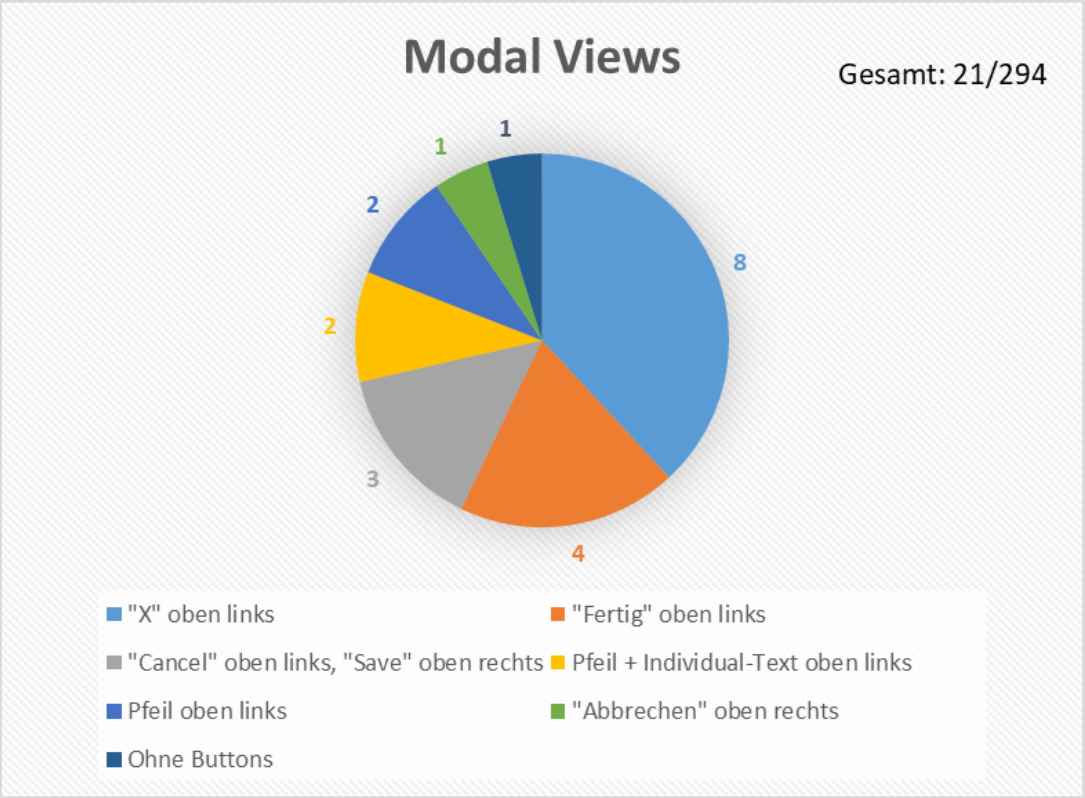


Abbildung 6: Screens, welche mindestens ein UI-Element „Modal View“ enthalten, sowie die Verteilung der verschiedenen Variationen des UI-Elements

In Abbildung 6 wurden die gezählten Modal Views, je nachdem welchen Text der Schließen-Button enthält, in verschiedene Variationen unterteilt. Wie bei der Auswertung der Zurück-Buttons in Abbildung 3, wurden von einigen Apps auch innerhalb von Modal Views englische Begriffe verwendet.
Nicht explizit gezeigt ist die Verteilung zwischen Modal Views mit Button am oberen rechten Rand und solchen mit Buttons am oberen linken Rand: Von

21 gezählten Modal Views besitzen 16 und somit drei Viertel lediglich einen Button am oberen linken Rand. Drei weitere oben links und oben rechts einen Button. Und jeweils eine Modal View wurde gezählt, die nur oben rechts beziehungsweise gar keinen Button besitzt.
Modal Views, die einen Link öffnen oder zum Versenden von Mails verwendet werden, wurden als separate UI-Elemente gewertet und sind somit nicht in Abbildung 6 enthalten. Diese Entscheidung wurde getroffen, da die genannten Web bzw. Mail Modal Views deutlich mehr Funktionalitäten bieten und sich somit zu sehr von den restlichen Modal Views unterscheiden.

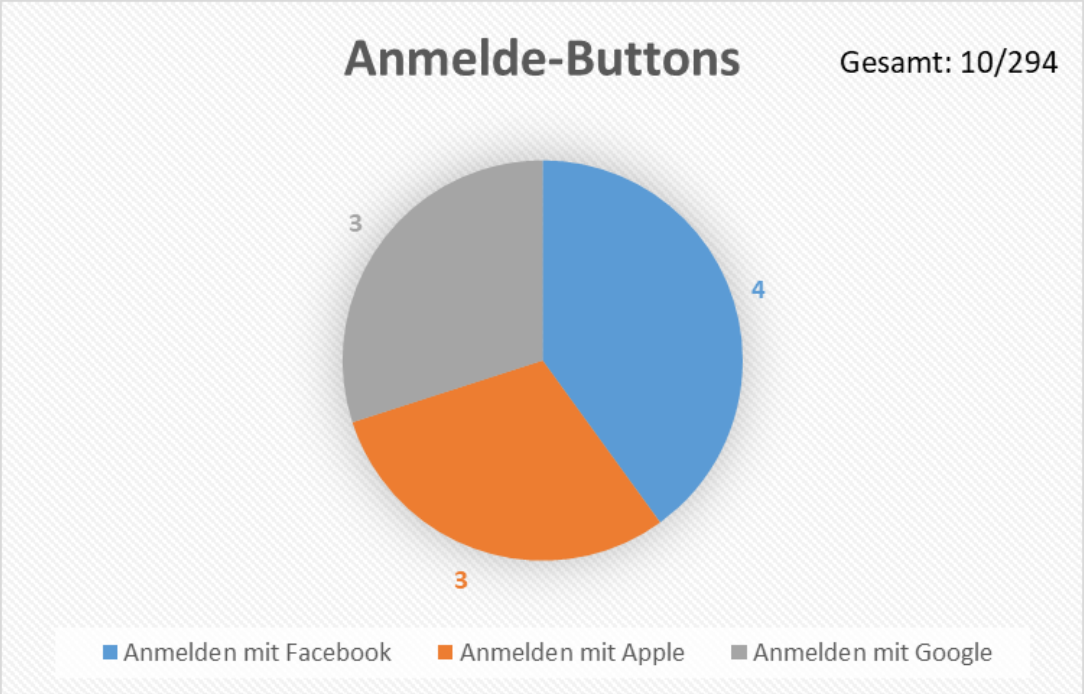


Abbildung 7: Screens, welche mindestens ein UI-Element „Anmelde-Button“ enthalten, sowie die Verteilung der verschiedenen Variationen des UI-Elements

Abbildung 7 zeigt die unterschiedlichen Variationen von Anmelde-Buttons. So wurden im Zuge der Analyse Buttons benannt, welche den Nutzer auffordern, sich mittels eines Drittanbieters anzumelden.
Wie in Abbildung 4 ist auch in Abbildung 7 eine ungleiche Verteilung der

Drittanbieter-Dienste zu erkennen. Allerdings fällt die Verteilung in diesem Fall anders aus. Das liegt daran, dass nicht jeder Anmelde-Button zu einem Pop-up führt und umgekehrt ein Anmelde-Pop-up auch ohne entsprechenden Anmelde-Button dargestellt werden kann.
Während die meisten Anmelde-Pop-ups zu der Variation „Mit Google anmelden“ gehören, ist die Verteilung der Anmelde-Buttons beinahe ausgeglichen. Obwohl ebenso viele Android wie iOS Apps analysiert wurden, wurden etwa gleich viele Anmelde-Buttons des Drittanbieters Apple wie die anderer Drittanbieter, gezählt. Wie in Abbildung 4 zu sehen, wurden jedoch im Vergleich zu anderen Drittanbietern deutlich weniger Anmelde-Pop-ups des Drittanbieters Apple gezählt.

Neben dem Zählen von UI-Elementen wurde eine Liste von Aktionswörtern erstellt, welche auf einen Button hindeuten. Um im Sinne der Analyse als Aktionswort zu gelten, musste ein Wort auf mindestens fünf unterschiedlichen Buttons stehen. Die aus der Analyse gewonnenen Aktionswörter sind im Folgenden alphabetisch aufgezählt:

Deutsch:
Abbrechen; Ablehnen; Aktivieren; Akzeptieren; Anmelden; Danke; Erlauben; Erstellen; Fortfahren; Ja; Nein; Nochmal; Sichern; Verstanden; Wählen; Weiter; Zugriff; Zurück

Englisch:
Adjust; Back; Cancel; Forward; No; OK; Save; Yes

Wie bereits erwähnt wurde sichergestellt, dass die Systemsprache der beiden für die Analyse gewählten Geräte auf „Deutsch“ eingestellt war. Die hier aufgeführten Begriffe wurden dennoch von einigen Apps verwendet, weshalb sie ebenfalls in die Liste von Aktionswörtern aufgenommen wurden. Eine fehlende Lokalisierung der entsprechenden Apps könnte ein naheliegender Grund für die Verwendung von englischen Begriffen sein.

2.3 Expertengespräche

Aus den in der Analyse gewonnenen Informationen wurden im nächsten Schritt Kategorien erstellt, welche nach den bereitgestellten Funktionalitäten der Apps gewählt wurden. Zu diesen Kategorien wurden einige rudimentäre Testfälle erstellt. Im Gespräch mit zwei Experten für Testentwicklung innerhalb des TIC wurden diese Testfälle dann verbessert und erweitert.
Für das Gespräch standen *Marcus Trepte* und *Oliver Löwe* bereit. Beide haben jahrelange Erfahrung im Testen von mobilen Anwendungen. Im Gespräch wurden die bis dato erstellten Testfälle diskutiert und weitere Testfälle in Betracht gezogen.

Insbesondere die Bedeutsamkeit der Testfälle wurde kritisch hinterfragt. Ein Standard-Testfall sollte eine möglichst hohe Bedeutsamkeit aufweisen und als solcher häufig implementiert werden und bereits häufig implementiert worden sein. Mit diesem Hintergrund wurden viele Testfälle verallgemeinert oder ganz verworfen.
Die Testfälle der beiden Kategorien „Sharing“ und „Deeplinks“ kamen erst im Expertengespräch auf und wurden daraufhin in die Auflistung aufgenommen. Die Funktionalität „Sharing“ wurde zwar von einigen Apps der Analyse bereitgestellt, während der Analyse allerdings nicht als häufig getestete Funktionalität eingestuft. Die Funktionalität von „Deeplinks“ konnte im Rahmen der Analyse nicht vorkommen, da Deeplinks für gewöhnlich kein Bestandteil einer App sind, sondern von außen auf die App verlinken.
Insgesamt wurden im Expertengespräch fünf Testfälle verworfen und sieben Testfälle überarbeitet. Weitere sechs Testfälle wurden im Laufe des Gesprächs in die Auflistung aufgenommen. Nach dem Gespräch fasste die Auflistung 41 Standard-Testfälle. Einer dieser Testfälle wurde nachträglich noch erweitert und zu zwei Testfällen umgeschrieben.

3 Kategorisierte Auflistung

Die Auflistung von Standard-Testfällen soll bei der Planung und Umsetzung von automatisierten Testfällen behilflich sein. Die Auflistung der Standard-Testfälle ist nach den potenziellen Funktionalitäten einer App kategorisiert, damit sie zu einem möglichst frühen Zeitpunkt der App-Entwicklung benutzt werden kann.
Die Auflistung ist nach dem Schema einer Checkliste erstellt worden. Dadurch wird eine einfache Benutzung gewährleistet. Je nach Präferenz können Testfälle abgehakt oder durchgestrichen werden. Auch eine Kombination ist möglich, etwa können zu implementierende Testfälle abgehakt und bereits implementierte Testfälle durchgestrichen werden.

3.1 Kategorien

Die Kategorien der Auflistung sind nach häufig auftretenden Funktionalitäten ausgewählt. Dabei wurden nur solche Funktionalitäten gewählt zu denen tatsächlich Standard-Testfälle gefunden werden konnten. Genau wie die Auswahl der Standard-Testfälle wurden auch die ausgewählten Kategorien noch einmal von den Experten des TIC geprüft und gemeinsam verbessert. Die einzelnen Kategorien sind im Folgenden beschrieben:

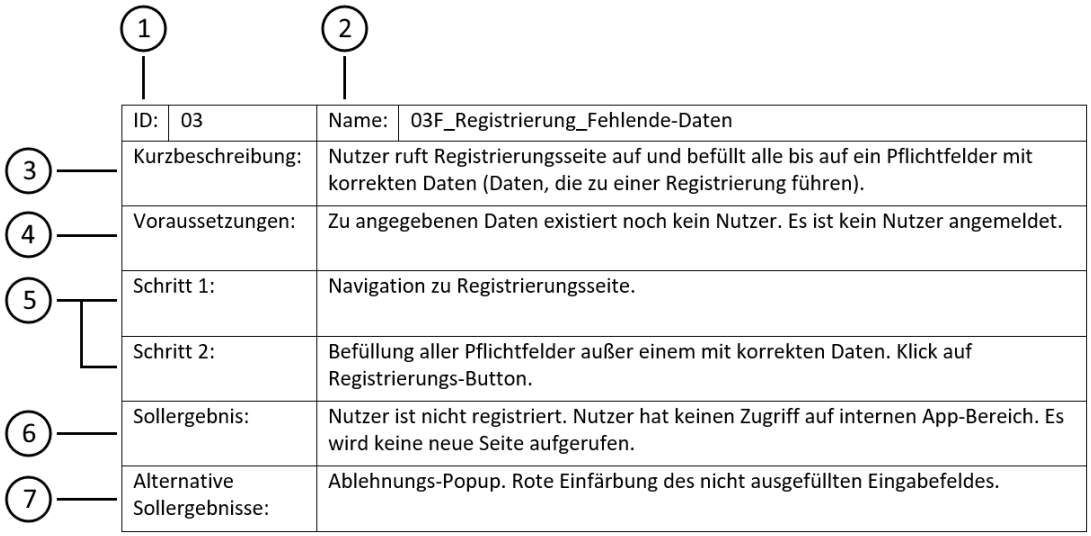
- **Registrierung:** Umfasst das Erstellen eines Benutzeraccounts unter Angaben persönlicher Daten. Zu testen ist u. a. die Registrierung mit korrekten und inkorrekten Daten.
- **Login:** Umfasst das Anmelden mit zuvor erstelltem Benutzeraccount. Zu testen ist u. a. das Anmelden mit korrekten und inkorrekten Anmeldedaten sowie das Nutzen der App vor und nach dem Login.
- **Navigation:** Umfasst das korrekte Verhalten in Reaktion auf Benutzereingaben. Zu testen ist u. a. das Vorwärts- und Rückwärts-Navigieren innerhalb der App sowie das Scrolling.

- **Suche:** Umfasst die Nutzung eines Suchfeld-Elements. Zu testen ist das Suchen eines korrekten sowie eines inkorrekten Suchbegriffs.
- **Werbung:** Umfasst Darstellung und Funktion von Werbe-Elementen. Zu testen ist u. a. die korrekte Darstellung sowie die korrekte Weiterleitung beim Klick auf ein Werbe-Element.
- **Darkmode:** Umfasst die Darstellung der App im Darkmode. Zu testen ist u. a. die korrekte Darstellung nach Aktivieren und erneutem Deaktivieren des Darkmode.
- **Fremdinhalte:** Umfasst die Darstellung von Fremdinhalten sowie die Funktionalität der bereitgestellten Operationen zu diesen Fremdinhalten. Zu testen ist u. a. das Abspielen von Videos, das Betrachten von Dokumenten und die jeweilig bereitgestellten Operationen.
- **Netzwerkverbindung:** Umfasst die Verwendbarkeit der App im Falle einer beeinträchtigten Netzwerkverbindung. Zu testen ist u. a. die Nutzung der App bei Störung und Ausfall der Netzwerkverbindung und des Ortungsdienstes.
- **Updates:** Umfasst das korrekte Verhalten der App im Falle eines Updates. Zu testen ist u. a. das Erhalten wichtiger Daten und das Beibehalten von Einstellungen nach einem Update der App.
- **Sharing:** Umfasst das Teilen von Inhalten mit anderen Nutzern über Drittanbieter-Dienste. Zu testen ist u. a. die Verfügbarkeit und die Funktionalität der implementierten Sharing-Operationen.
- **Deeplinks:** Umfasst das korrekte Verhalten beim Klick auf Deeplinks. Zu testen ist u. a. das Ziel eines Deeplinks bei installierter sowie bei nicht installierter App.
- **Sonstiges:** Umfasst alle sonstigen Testfälle. Zu testen ist Verschiedenes.

Die Kategorien „Sharing“ und „Deeplinks“ waren ursprünglich nicht in der Auflistung enthalten und ergaben sich auch nicht aus der Analyse populärer Apps. Vielmehr wurden diese Kategorien nach Absprache mit den Experten des TIC nachträglich hinzugefügt. In der Kategorie „Sonstiges“ sind alle Testfälle zusammengefasst, welche nicht zu Funktionen anderer Kategorien zugeordnet werden konnten. Es handelt sich um die folgenden drei Testfälle:
40_Sonstiges_Landscape-Modus
41F_Sonstiges_Inputfeld-Grenzwert
42_Sonstiges_Pull-to-refresh-Funktion
Es wurde sich der Übersicht halber dagegen entschieden, für jeden dieser Testfälle eine eigene Kategorie zu erstellen. Dennoch handelt es sich um wichtige Testfälle, welche implementiert werden sollten, falls die entsprechende Funktionalität angeboten wird.

3.2 Aufbau eines Testfalls

Die einzelnen Testfälle wurden nach den firmeninternen Richtlinien der MMS erstellt. Der Aufbau eines einzelnen Testfalls orientiert sich dabei an den Richtlinien des ISTQB. Dabei wird ein Testfall über verschiedene Attribute definiert.



ID:	03	Name:	03F_Registrierung_Fehlende-Daten
Kurzbeschreibung:		Nutzer ruft Registrierungsseite auf und befüllt alle bis auf ein Pflichtfelder mit korrekten Daten (Daten, die zu einer Registrierung führen).	
Voraussetzungen:		Zu angegebenen Daten existiert noch kein Nutzer. Es ist kein Nutzer angemeldet.	
Schritt 1:		Navigation zu Registrierungsseite.	
Schritt 2:		Befüllung aller Pflichtfelder außer einem mit korrekten Daten. Klick auf Registrierungs-Button.	
Sollergebnis:		Nutzer ist nicht registriert. Nutzer hat keinen Zugriff auf internen App-Bereich. Es wird keine neue Seite aufgerufen.	
Alternative Sollergebnisse:		Ablehnungs-Popup. Rote Einfärbung des nicht ausgefüllten Eingabefeldes.	

Abbildung 8: Der Aufbau eines Testfalls am Beispiel des Testfalls „03F_Registrierung_Fehlende-Daten“

Allerdings mussten diese Richtlinien aufgrund der standardisierten Natur der Testfälle angepasst werden. Da es sich um Standard-Testfälle handelt, welche teilweise nur sehr allgemein beschrieben werden können, mussten einige Attribute abgeändert oder gar verworfen werden. Einige neue Attribute wurden mit derselben Begründung hinzugefügt. Abbildung 8 zeigt den Aufbau eines Testfalls. Die endgültigen Attribute eines Testfalls werden im Folgenden in Kombination mit Abbildung 8 aufgeschlüsselt:

① **ID:** Fortlaufende zweistellige Nummerierung der Testfälle startend bei „1“. Verwendbar für Datenverarbeitungszwecke und zur einfachen Identifikation eines Testfalls.

② **Name:** Name des Testfalls nach abgewandelter Namenskonvention der MMS: [ID][F]_[Funktionalität]_[Ausprägung]

③ **Kurzbeschreibung:** Beschreibung der Ausprägung der zu testenden Funktionalität in wenigen Worten.

④ **Voraussetzungen:** Zum Testen der Ausprägung benötigten Voraussetzungen. Sind vor der Ausführung des Testfalls durchzuführen.

⑤ **Schritte:** Aktionen, die zur Prüfung der Ausprägung durchzuführen sind. Erfordern Interaktion mit der App. Können in beliebiger Anzahl auftreten.

⑥ **Sollergebnis:** Erwartetes Ergebnis nach Durchführung aller Schritte. Das Eintreten aller Sollergebnisse entscheidet über das Ergebnis des Testfalls.

⑦ **Alternative Sollergebnisse:** Menge von alternativen Sollergebnissen, welche je nach Spezifikation der App ebenfalls eintreten müssen. In der Regel visuelle Effekte oder anderweitige sekundäre Aktionen.

Die Firmenrichtlinien zum Aufbau eines Testfalls sehen zu jedem angegebenen Schritt das Dokumentieren des zu erwartenden Ergebnisses vor. Da viele Standard-Testfälle jedoch nur sehr allgemein formuliert werden konnten, wurde sich dazu entschieden, lediglich das Sollergebnis des letzten Prüfschritts anzugeben.
Aufgrund einer möglichst hohen Gültigkeit der Testfälle wurde das Attribut „Alternative Sollergebnisse“ zur Beschreibung eines Testfalls hinzugefügt. Alternative Sollergebnisse sollen weitere eventuell eintretende Aktionen beschreiben, welche ebenfalls als Sollergebnisse angesehen werden müssen. Falls die Spezifikation der App ein oder mehrere alternative Sollergebnisse nicht vorsieht, müssen diese auch nicht geprüft werden. Die Zusammenstellung der alternativen Sollergebnisse basiert auf häufig beobachteten Verhalten beim Ausführen der jeweiligen Funktionalität in diversen Apps. Die Liste von alternative Sollergebnissen ist weder als abgeschlossen noch als vollumfänglich zu betrachten. Vielmehr soll sie Hilfestellung zur eventuellen Erweiterung der Sollergebnisse bilden. Dabei enthalten die alternative Sollergebnisse keine

system- oder sicherheitstechnisch relevanten Ergebnisse. Es wurde darauf geachtet, nicht den Eindruck der Irrelevanz solcher Ergebnisse zu erzeugen.

Die Namenskonvention der Testfälle wurde ebenfalls leicht abgewandelt. Es wurde auf die Nennung des Moduls verzichtet, da diese bei regulären Testfällen auf eine Testelementliste verweist, welche für Standard-Testfälle nicht erstellt wurde. Der Name eines Testfalls wurde so gewählt, dass bereits durch ihn ablesbar ist, was getestet wird. Im Folgenden wird die Benennung der Testfälle weiter erläutert:

- **[ID]** bezieht sich auf die ID des Testfalls.
- Mit einem optionalen **[F]** werden Negativtestfälle gekennzeichnet. Damit sind Testfälle gemeint, welche keine Anforderung an die App testen, sondern vielmehr vorgesehene Fehler innerhalb der App erzeugen und die Behandlung dieser Fehler überprüfen.
- **[Funktionalität]** bezieht sich auf die getestete Funktionalität, zu welcher es mehrere Testfälle geben kann.
- Die **[Ausprägung]** beschreibt den Inhalt des Testfalls in möglichst wenigen Worten. Besteht die Ausprägung aus mehreren Worten, werden diese mit einem Bindestrich getrennt.

Manche Testfälle sind sehr allgemein formuliert, da die zu prüfenden Funktionalitäten von App zu App sehr unterschiedlich implementiert werden können. Es handelt sich um eine Auflistung von Standard-Testfällen. Beim Nutzen der Auflistung im Test-Vorgang kann es vorkommen, dass manche Testfälle angepasst werden müssen. Die Auflistung bietet eine Reihe von Standard-Testfällen an, welche für *viele* Apps implementiert werden sollten. Daraus folgt nicht, dass all diese Testfälle für *alle* Apps zu implementieren sind.
Falls die Spezifikation einer App einen Testfall ungültig oder überflüssig macht, muss dieser Testfall nicht implementiert werden. Ebenso können Testfälle abgeändert oder erweitert werden, falls sie sich dadurch besser für die Spezifikation der zu testenden App eignen.

An dieser Stelle sei noch einmal erwähnt, dass es sich bei der Auflistung von Standard-Testfällen insbesondere *nicht* um eine allumfassende Sammlung von Testfällen handelt. Vielmehr stellt sie ein Mindestmaß an zu implementierenden Testfällen zur Verfügung. Je nach Spezifikation der App sind auf jeden Fall weitere Testfälle zu implementieren.

4 Prototypische Implementierung

Um eine potenzielle allgemein nützliche Implementierung der erstellten Testfälle aufzuzeigen, wurde einige Testfälle prototypisch implementiert. Dabei handelt es sich um automatisierte Testfälle die mithilfe von *Appium* auf realen Geräten innerhalb der MDC durchgeführt werden können. Umgesetzt wurden alle zu der Funktionalität „Login“ gehörigen Testfälle. Insgesamt wurden die folgenden sechs Testfälle implementiert:

- **05_Login_Credentials-korrekt:** Nutzer ruft Loginseite auf und befüllt die Eingabefelder mit korrekten Anmeldedaten.
- **06F_Login_Credentials-inkorrekt:** Nutzer ruft Loginseite auf und befüllt die Eingabefelder mit inkorrekten Anmeldedaten.
- **07F_Login_Fehlende-Credentials:** Nutzer ruft Loginseite auf und befüllt nur eines der Eingabefelder mit korrekten Anmeldedaten.
- **08_Login_Nutzer-bleibt-eingeloggt-nachdem-App-geschlossen:** Angemeldeter Nutzer schließt die App und öffnet die App wieder.
- **09_Login_Weiterleitung-zu-Loginseite-bei-Klick-auf-Login-Feature:** Nutzer navigiert zu einer Seite, die ohne Login nicht nutzbar ist.
- **10_Login_Features-ohne-Login-zugänglich:** Nutzer navigiert zu einer Seite, die auch ohne Login nutzbar ist.

Diese Testfälle wurden ausgewählt, da sie leicht verständlich, einfach zu implementieren und auch einfach zu überprüfen sind. Dabei handelt es sich ausdrücklich nur um eine prototypische Implementierung. Diese soll lediglich eine Möglichkeit aufzeigen, wie einige der Standard-Testfälle allgemeingültig implementiert werden könnten. Die prototypisch implementierte Testfälle soll ausdrücklich nicht das Erstellen von eigenen Testfällen ersetzen. Sie können allerdings dafür genutzt werden, den Prozess des App-Testens zu verbessern und zu vereinfachen, indem sie in Kombination mit selbst geschriebenen Testfällen verwendet werden.

Die Testfälle sind in *Java* geschrieben und verwenden die Testautomatisierungstools *Appium* und *TestNG*. Durch die Verbindung zur MDC ist es möglich, die Testfälle auf über 200 verschiedenen realen mobilen Geräten durchzuführen. Jedoch wurden die Testfälle vorrangig für mobile Geräte des Android-Betriebssystems entwickelt und auch nur auf Android Geräten ausgeführt.

4.1 App-Crawler

Grundlage für alle implementierten Standard-Testfälle ist ein selbst programmierter App-Crawler. Dieser ist bei Weiterentwicklung der prototypisch implementierten Testfälle durch einen länger und besser entwickelten App-Crawler zu ersetzen. Die selbstständige Entwicklung eines App-Crawlers bot jedoch die Möglichkeit einige essenzielle Interaktionsmöglichkeiten einfach zu implementieren.

Während der App-Crawler durch die App navigiert, erstellt er parallel ein Modell des Aufbaus der App. Dieses kann als gerichteter Graph verstanden werden, dessen Knoten jeweils einen Screen der analysierten App enthalten. Eine Kante stellt in diesem Modell eine Interaktion mit der App dar, welche potenziell zu einem anderen Screen führt. Für jede Interaktion wird dabei gespeichert, ob sie bereits analysiert, das heißt bereits durchgeführt wurde und ob so bereits bekannt ist, zu welchem Screen sie führt. Als Interaktion wird analog zu Unterabschnitt 2.2 *Analyse populärer Apps* etwa das Drücken auf einen Button, das Interagieren mit einem Textfeld oder das Drücken der Zurücktaste verstanden. In einem Knoten werden dabei die jeweiligen Interaktionsmöglichkeiten des Screens gespeichert. Hierfür liefert Appium ein Objekt mit den nötigen Informationen wie ID, Xpath und Text, welches dann gespeichert wird. Da die meisten Interaktionen mit der App etwa dieselbe Zeit in Anspruch nehmen, ist der Graph ungewichtet.
Anhand der unterschiedlichen Elemente wird auch die Eindeutigkeit eines Screens definiert. In Bezug auf generisch generierte Screens wird so verhindert, dass sich der Crawler in endlosen Zyklen scheinbar unterschiedlicher Screens verliert. Der Crawler hat die App vollständig analysiert, sobald alle

Interaktionsmöglichkeiten aller Screens mindestens einmal durchgeführt wurden. Als Pfadfindungsalgorithmus innerhalb des Graphen wurde eine simple Variante des Dijkstra-Algorithmus gewählt.

Nachdem die App analysiert wurde stellt der Crawler verschiedene hilfreiche Methoden zur Verfügung. So ist es möglich anhand einer ID eines Elements zu dem entsprechenden Screen, auf dem sich das Element befindet, zu navigieren. Auch können alle oder ausgewählte Elemente eines Screens zurückgegeben werden. Dabei verwendet der Crawler die durch *Appium* definierten Klassen, was das Erstellen von Testfällen sehr erleichtert. Sollte der App-Crawler im Zuge von Erweiterungen der Testfälle ausgetauscht werden, ist darauf zu achten, dass der neue App-Crawler diese Methoden ebenfalls zur Verfügung stellt. Falls dem nicht so sein sollte, müssten entsprechende Hilfsmethoden geschrieben werden.

4.2 Vorgehensweise & Hindernisse

Der Ablauf der prototypisch implementierten Testfälle ist beinahe immer derselbe: Zuerst navigiert der App-Crawler durch die App und erstellt ein Modell des Aufbaus der App. Danach wird der eigentliche Testfall ausgeführt. Dabei werden die zusätzlich von dem App-Crawler angebotenen Methoden verwendet. Am Ende folgt der Assert-Schritt, wodurch festgestellt wird, ob der Testfall erfolgreich war oder fehlgeschlagen ist.

Die meisten Testfälle benötigen diverse Informationen. Dabei handelt es sich meist um IDs diverser für den Testfall benötigter Elemente. Diese Informationen werden mittels der TestNG-Annotation „@Parameters“ übergeben. Für einige Testfälle ist Vorarbeit notwendig, etwa das Anlegen eines Testnutzers.

Aufgrund der potenziell allgemein nützlichen Implementierung kann der Assert-Schritt nur sehr oberflächlich ausfallen. Beispielsweise ist es sehr schwierig festzustellen, ob ein Nutzer angemeldet ist oder nicht. Je nach App verfügt ein angemeldeter Nutzer über diverse Rechte und/oder kann auf diverse vorher

nicht zugängliche Screens zugreifen. Dazu eine allgemeingültige Assertion zu schreiben stellt eine Herausforderung dar, welche über den Umfang dieser Arbeit hinaus geht.
Im konkreten Fall wurde eine Negativ-Lösung gewählt. Um festzustellen, ob die Anmeldung eines Nutzers fehlgeschlagen ist, vergleicht der Testfall die Screens vor und nach der Anmeldung. Handelt es sich um den gleichen Screen, ist die Anmeldung aus Sicht des Testfalls fehlgeschlagen.
Dieser Ansatz hängt stark von der Implementierung des App-Crawlers ab. Genauer, woran der App-Crawler zwei Screens differenziert. Oftmals erhält ein Nutzer bei fehlgeschlagener Anmeldung einen Hinweis. Dieser Hinweis kann dazu führen, dass der App-Crawler den Screen nicht mehr als identisch zum Screen vor dem Login betrachtet. Das Resultat ist ein potenziell inkorrekter Testfall. Bei der aktuellen Implementierung des App-Crawlers könnte dieser Fall zum Beispiel dann auftreten, wenn die getestete App nach inkorrekter Anmeldung ein Pop-up anzeigt.

Allgemein steigt die Qualität der implementierten Testfälle mit der Qualität des App-Crawlers. Aus der Entwicklung der ausgewählten Standard-Testfälle entstand die Erfahrung, dass ein fehlschlagender Testfall nicht unbedingt aufgrund eines Fehlverhaltens der App fehlschlug. Vielmehr führten Probleme beim Crawlen innerhalb der App oftmals dazu, dass der Crawling-Prozess abgebrochen werden musste und dadurch zu einem negativen Testergebnis. Die eigenständige Entwicklung eines App-Crawlers stellte während der Entwicklung der Testfälle eine große Herausforderung dar. Im Folgenden werden einige bekannte Problematiken in der momentanen Implementierung des App-Crawlers aufgezählt.

Jeder Button einer App kann potenziell einen neuen Screen öffnen. Deshalb analysiert der App-Crawler so lange die App, bis er alle Buttons zumindest einmal angeklickt hat. Allerdings besitzen einige Apps Interaktionsmöglichkeiten, die nur einmal aufrufbar sind, allerdings mehr als einen Button besitzen. Ein häufiges Beispiel dafür sind Pop-ups zur Abfrage von Zugriffsrechten. Das Pop-up, welches die Erlaubnis auf Standortdaten zuzugreifen einholt, besitzt

etwa unter Android zwei Buttons und unter iOS meist drei. Sobald allerdings einer der Buttons angeklickt wurde, ist das erneute Aufrufen des Pop-ups nicht trivial. In Folge dessen scheitert der App-Crawler oft an Apps mit solchen Interaktionsmöglichkeiten.

Das Benutzen der Zurück-Taste kann nur hinreichend originalgetreu simuliert werden. Dies gilt sowohl für Android als auch analog für Apple. Appium erlaubt das Simulieren der Zurück-Taste prinzipiell. Allerdings können gewisse Screens ab einem gewissen Zeitpunkt nicht mehr durch das simple Drücken der Zurück-Taste erreicht werden. Das Drücken der Zurück-Taste führt in diesem Fall nicht zu dem vom App-Crawler erwarteten Screen. Das kann zu Problemen während des Crawling führen.

Eine weitere Herausforderung stellt das Auffinden von Interaktionsmöglichkeiten im Allgemeinen dar. Um zu erkennen, welche Interaktionsmöglichkeiten auf einem Screen vorhanden sind, betrachtet der App-Crawler momentan die Hierarchie der UI-Elemente und sucht dort nach den gewünschten Klassen. Allerdings verwenden einige Apps nicht mehr die typischen UI-Elemente wie „Button“ oder „Text Field“. Auch das Suchen nach geeigneten Elementeigenschaften wie „clickable“ oder „enabled“ führte bei der Entwicklung des App-Crawlers zu Problemen.
Der momentan implementierte App-Crawler verwendet eine Kombination aus beiden besprochenen Lösungsansätzen. Falls dadurch noch immer keine Interaktionsmöglichkeiten gefunden werden können, sucht der App-Crawler nach häufig verwendeten Stichwörtern. Dazu wird die erstellte Liste an Aktions-Stichwörtern aus Unterabschnitt 2.2 *Analyse populärer Apps* verwendet.
Diese letzte Vorgehensweise ist potenziell sehr zeitaufwändig, da auch Elemente erfasst werden, die keine Interaktionsmöglichkeit bieten. Diese werden später durch den App-Crawler erforscht und somit angeklickt. Da sich der Screen dabei nicht verändert, werden sie als Referenz auf denselben Screen gewertet und sind für den weiteren Prozess unerheblich.

5 Fazit

Im Rahmen dieser Arbeit wurde eine nach Funktionalitäten kategorisierte Liste von Standard-Testfällen erstellt. Dazu wurden einige der zum Zeitpunkt der Arbeit am meisten heruntergeladenen Apps analysiert. Die Analyse ergab eine Reihe von Funktionalitäten, die von mehreren der analysierten Apps implementiert wurden. Zu diesen Funktionalitäten wurden Standard-Testfälle dokumentiert. Die so entstandene Liste an Testfälle wurden im Gespräch mit Test-Experten des TIC der MMS erweitert und überarbeitet.

Aus dieser Liste wurden einige Testfälle ausgewählt, welche prototypisch implementiert wurden. Das Augenmerk der Entwicklung lag auf einer möglichst gültigen Implementierung. Ziel war es, Testfälle zu entwickeln, mit denen möglichst viele Apps ohne große Vorarbeit getestet werden können. Dazu nutzen die meisten der so implementierten Testfälle zu ihrer Durchführung einen selbst geschriebenen App-Crawler.

Die so entstandene Liste von Standard-Testfällen und die prototypische Implementierung einiger dieser Testfälle können beide zur Verbesserung beziehungsweise Vereinfachung des Test-Vorgangs genutzt werden. Dabei soll keines der beiden Ergebnisse den bisherigen Test-Vorgang ersetzen. Vielmehr können sie in Kombination mit selbst geschriebenen Testfällen benutzt werden.

5.1 Schlussfolgerungen

Die Analyse ausgewählter Apps ergab eine Liste von Funktionen, welche häufig von den analysierten Apps implementiert wurden. Namentlich: *Registrierung*, *Login*, *Navigation*, *Suche*, *Werbung*, *Darkmode*, *Fremdinhalte*, *Netzwerkverbindung* und *Sharing*. Diese Funktionalitäten wurden als Kategorien für die erstellte Liste von Standard-Testfällen gewählt. Die Kategorie *Deeplinks* wurde ebenfalls als wichtig erachtet und auch in die Liste mitaufgenommen. Alle Standard-Testfälle, die nicht zu einer der obigen Funktionalitäten

zugeordnet werden konnten, wurden unter der Kategorie *Sonstige* zusammengefasst.
Zu diesen Kategorien konnte in Absprache mit Experten des TIC eine Liste von Standard-Testfällen erstellt werden. Die Erkenntnis des Existierens von Standard-Testfällen konnte dadurch bestätigt werden. Die kategorisierte Liste hängt der Arbeit an. Sie stellt eine Konzentration von Test-Erfahrungen dar, welche in vielerlei Hinsicht benutzt werden kann.
Die Analyse gab auch Aufschluss über einige häufig benutzte Aktionswörter, diese wurden zur Programmierung des App-Crawlers benutzt. Die gesamte Liste der benutzten Aktionswörter ist in Unterabschnitt 2.2 *Analyse populärer Apps* zu finden. Sie kann neben der Entwicklung von App-Crawlern auch anderweitig genutzt werden.

Die prototypische Entwicklung einiger ausgewählter Standard-Testfälle liefert einen Ansatz einer möglichst gültigen Implementierung eines Testfalls. Die so implementierten Testfälle können theoretisch bereits in ihrem momentanen Zustand zur Verbesserung des Test-Vorgangs genutzt werden. Für eine Implementierung mit einem höheren Grad an Gültigkeit, um also mehr Apps durch sie testen zu können, sollten die Testfälle allerdings erweitert und verbessert werden. Bei der Erstellung der Testfälle wurde die Schlussfolgerung getroffen, dass eine allgemeingültige Implementierung zum momentanen Zeitpunkt nicht realistisch ist. Auch eine völlige Ablösung selbstgeschriebener Testfälle ist momentan nicht absehbar.

5.2 Ausblick

5.2.1 Eine Einschätzung der Machbarkeit

Die prototypischen Implementierungen der Standard-Testfälle wurden als zusätzliches Tool zum Testen von Apps entwickelt. Sie können und sollen nicht die Entwicklung selbst geschriebener automatisierter Testfälle ersetzen. Selbst bei Weiterentwicklung der prototypisch implementierten Testfälle scheint eine allgemeingültige Implementierung nicht realistisch.

Das erste Hindernis zu einem allgemeingültigen Testfall stellt der App-Crawler dar. Die momentane Implementierung des App-Crawlers ist ebenso wie die Implementierungen der Standard-Testfälle nur prototypisch entwickelt worden. In Hinblick auf länger entwickelte App-Crawler wird an dieser Stelle auf die Bachelor-Arbeit von Florian Hübscher verwiesen. Eine von ihm erstellte Auswertung mehrerer, dem TIC zur Verfügung stehender App-Crawler kommt zu dem Schluss, dass keiner der getesteten App-Crawler perfekt implementiert wurde: „Alle Crawler haben wiederum auch Grenzen. So können noch keine spezifischen Benutzerinteraktionen, die ein Ereignis erst interpretieren müssen, wie CAPTCHAs gelöst werden“.[Hü20]

Eine weitere Herausforderung stellen die implementierten Standard-Testfälle selbst dar. Beim Entwickeln der Testfälle mussten einige Vermutungen über den Aufbau der App getätigt werden. Beispielsweise beschränkt sich der Testfall „05_Login_Credentials-korrekt“ momentan auf ein Anmeldevorgang bei dem sich die beide Eingabefelder für E-Mail und Passwort auf demselben Screen befinden. Falls sich die Eingabefelder auf zwei nacheinander folgenden Screens befinden, schlägt der Testfall fehl. Eine Erweiterung diesbezüglich erscheint schlüssig. Allerdings würde damit nur eine weitere von mehreren Arten von Anmeldevorgängen unterstützt. Alle Arten von Anmeldevorgängen zu unterstützen stellt sich als schwierig heraus. Auch weitere bereits besprochene Probleme, wie das Erstellen einer allgemeingültigen Assertion, verhindert momentan eine ausschließliche Nutzung der prototypischen Implementierungen der Standard-Testfälle.
Jedoch ist eine den Testprozess unterstützende Nutzung der prototypisch implementierten Testfälle auch ohne allgemeingültige Implementierung sinnvoll. Auch eine Weiterentwicklung der Testfälle kann für deren Nutzung sinnvoll sein. Desto besser die Testfälle implementiert sind, desto mehr Apps lassen sich verlässlich mit ihnen testen.

Die Standard-Testfälle der Funktionalität „Login“ wurden auch deshalb für die prototypische Implementierungen gewählt, da sie einfach umzusetzen sind. Dies gilt nicht für alle ermittelten Standard-Testfälle. Einige der Standard-

Testfälle richten sich zu sehr nach der Spezifikation der zu testenden App, wodurch eine allgemeine Implementierung schwierig umzusetzen ist. Andere Testfälle lassen sich nicht mit demselben Ansatz, also mithilfe eines App-Crawlers implementieren.

Die Standard-Testfälle wurden so speziell wie möglich und so allgemein wie nötig formuliert. Einige Testfälle sind zwar zwingend zu implementieren, jedoch ist die zu testende Funktion stark von ihrer Implementierung abhängig. Daraus resultiert ein sehr allgemein verfasster Standard-Testfall. Das Sollergebnis des Testfalls „11_Navigation_Vorwärts“ gibt beispielsweise folgendes an: „Navigation funktioniert nach Spezifikation. Jeder Navigations-Button führt zu der vorgesehenen Seite.“. Eine allgemeingültige Implementierung des Testfalls ist nur schwer vorstellbar. Selbst eine lediglich automatisierte Implementierung dieses Testfalls ist nicht trivial. Dennoch handelt es sich um einen wichtigen Testfall, der für die meisten Apps umgesetzt werden sollte.

Ein herkömmlicher App-Crawler erreicht potenziell nicht alle Screens einer App. Viele Apps teilen sich in einen externen und einen internen Bereich, wobei letzterer nur nach einem Login des Nutzers erreichbar ist. Um den internen Bereich einer App zu erreichen, muss der App-Crawler Direktiven ausführen können. Diese Direktiven bestimmen das Verhalten des Crawlers sobald einen bestimmten Screen erreicht ist. Im Beispiel könnte eine Direktive dem App-Crawler erlauben einen Login durchzuführen und so den internen Bereich der App zu erreichen. Ohne eine solche Direktive ist es nicht möglich den internen Bereich zu erreichen. Folglich können dann auch keine Funktionen getestet werden, die den internen Bereich der App benutzen.
Allerdings ist eine allgemeine Umsetzung einer nicht-trivialen Direktive, also eine Direktive, die dasselbe komplexe Verhalten für alle Apps durchführt, schwierig zu programmieren. Folglich ist auch ein Testfall, der für seine Durchführung eine Direktive benötigt, da er eine Funktion testet, die im internen Bereich der App liegt, nur schwer allgemeingültig implementierbar.

5.2.2 Weitere Ansätze & Erweiterungen

Bei einer potenziellen Weiterentwicklung der Standard-Testfälle sollte in erster Linie eine Implementierung der Testfälle mit höherer Gültigkeit angestrebt werden. Dabei ist die Reduzierung der False Positive Ergebnisse von besonderer Wichtigkeit. False Negative Ergebnisse führen zu eventuellem Mehraufwand, da die benutzte Funktion noch einmal getestet beziehungsweise begutachtet wird. False Positive Ergebnisse jedoch könnten dazu führen, dass die entsprechende Funktionalität als funktionierend angesehen werden könnte, obwohl sie potenziell nicht oder nur teilweise korrekt funktioniert.
Auch das Implementieren weiterer Standard-Testfälle bietet sich an. Die bereits implementierten Testfälle können dazu als Vorlage dienen. Wie schon in Unterunterabschnitt 5.2.1 *Eine Einschätzung der Machbarkeit* besprochen, lassen sich jedoch nicht alle Standard-Testfälle nach demselben Schema allgemeingültig implementieren.

Der App-Crawler, der zur Durchführung der Standard-Testfälle benutzt wird, ist nur prototypisch implementiert worden. Im Zuge einer potenziellen Erweiterung sollte dieser durch einen weiterentwickelten App-Crawler ersetzt werden. Auch die Weiterentwicklung des bereits implementierten App-Crawlers ist denkbar.
Falls es zum Ersatz des bereits implementierten App-Crawlers kommen sollte, muss darauf geachtet werden, dass der neue App-Crawler alle für die Testfälle benötigten Schnittstellen zur Verfügung stellt. Falls dem nicht so ist, müsste der neue App-Crawler dementsprechend angepasst werden. In diesem Fall muss der Arbeitsaufwand der Anpassung des neuen App-Crawlers gegen den Arbeitsaufwand der Weiterentwicklung des alten App-Crawlers aufgewogen werden. Dabei wird die Anpassung des neuen App-Crawlers nach Autorenmeinung wahrscheinlich weniger Arbeitsaufwand erzeugen als die Weiterentwicklung des alten App-Crawlers. Aus diesem Grund wird der Austausch des bereits implementierten App-Crawlers, wie schon mehrfach erwähnt, nahegelegt.
Eine weitere potenzielle Erweiterung stellt die Anpassung der prototypisch

implementierten Standard-Testfälle in Bezug auf iOS dar. Diese unterstützen im momentanen Entwicklungsstand Android-Apps. Es wurde aktuell auch nur Android-Apps mithilfe der prototypisch implementierten Standard-Testfälle getestet. Eine Unterstützung von iOS-Apps wurde bisher noch nicht implementiert. Allerdings sollte eine Anpassung der Testfälle, sodass beide Betriebssysteme unterstützt werden, ohne zu großen Arbeitsaufwand möglich sein.

Selbstständigkeitserklärung

Hiermit erklären ich, dass ich diese vorliegende Arbeit ohne fremde Hilfe und mit keinen anderen als den angegebenen Hilfsmitteln angefertigt habe. Insbesondere versichere ich, dass alle fremden Zitate als solche gekennzeichnet wurden und dass auch sinngemäße Inhalte aus anderen Werken kenntlich gemacht wurden.

Zwickau, den 20. November 2020

_____ Jakob Burger

Literatur

[And20] Android. Provide custom back navigation. `https://developer.android.com/guide/navigation/navigation-custom-back`, 2020. Aufrufdatum: 20. November 2020.

[Ann20] App Annie. State of Mobile - 2020. `https://www.appannie.com/de/go/state-of-mobile-2020/`, 2020. Aufrufdatum: 20. November 2020.

[Cow20] Ric Cowley. Count of Active Applications in the App Store. `https://www.pocketgamer.biz/metrics/app-store/app-count/`, 2020. Aufrufdatum: 20. November 2020.

[Gus92] Gustav Pomberger, Wolfgang Pree und Alois Stritzinger. Methoden und Werkzeuge für das Prototyping und ihre Integration. *Informatik Forschung und Entwicklung*, 1992.

[Hü20] Florian Hübscher. Design und Umsetzung eines Crawling-Systems in automatisierten Systemtests mobiler Applikationen. Bachelorarbeit, 2020. Universität Potsdam.

[Hit07] Derek Hitchins. *Systems Engineering: A 21st Century Systems Methodology*. John Wiley & Sons, Hoboken, New Jersey, USA, 2007.

[IST] ISTQB. ISTQB Glossary - Testfall. `https://glossary.istqb.org/de/term/testfall/`. Aufrufdatum: 20. November 2020.

[Mat20] Mathijs Vogelzang und Uwe Maurer. Number of Android apps on Google Play. `https://www.appbrain.com/stats/number-of-android-apps`, 2020. Aufrufdatum: 20. November 2020.

[Moh14] Mohd. Ehmer Khan und Farmeena Khan. Importance of Software Testing in Software Development Life Cycle. *International Journal of Computer Science Issues*, 2014.

Checkliste für Standarttestfälle

Registrierung

- ☐ 01_Registrierung_Daten-korrekt
- ☐ 02F_Registrierung_Daten-inkorrekt
- ☐ 03F_Registrierung_Fehlende-Daten
- ☐ 04F_Registrierung_Bereits-vergebene-Daten

Login

- ☐ 05_Login_Credentials-korrekt
- ☐ 06F_Login_Credentials-inkorrekt
- ☐ 07F_Login_Fehlende-Credentials
- ☐ 08_Login_Nutzer-bleibt-eingeloggt-nachdem-App-geschlossen
- ☐ 09_Login_Weiterleitung-zu-Loginseite-bei-Klick-auf-Login-Feature
- ☐ 10_Login_Features-ohne-Login-zugänglich

Navigation

- ☐ 11_Navigation_Vorwärts
- ☐ 12_Navigation_Rückwärts
- ☐ 13_Navigation_Scrolling
- ☐ 14_Navigation_Scrolling-Seitwärts

Suche

- ☐ 15_Suche_Eingabe
- ☐ 16_Suche_Suchbegriff-korrekt
- ☐ 17F_Suche_Suchbegriff-inkorrekt

Werbung

- ☐ 18_Werbung_Darstellung-korrekt
- ☐ 19_Werbung_Weiterleitung-korrekt
- ☐ 20_Werbung_Darstellung-für-Premiumnutzer

Darkmode

- ☐ 21_Dark-Mode_Darstellung-korrekt
- ☐ 22_Dark-Mode_Light-Mode-Darstellung-korrekt
- ☐ 23_Dark-Mode_Wie-System

Fremdinhalte

- ☐ 24_Fremdinhalte_Videos-abspielbar
- ☐ 25_Fremdinhalte_Video-Operationen
- ☐ 26_Fremdinhalte_Video-Sound
- ☐ 27_Fremdinhalte_Video-Fullscreen
- ☐ 28_Fremdinhalte_Dokument-Darstellung
- ☐ 29_Fremdinhalte_Dokument-Operationen

Netzwerkverbindung

- ☐ 30F_Netzwerkverbindung_Ausfall
- ☐ 31F_Netzwerkverbindung_Störung
- ☐ 32F_Netzwerkverbindung_Ortungsdienst-Ausfall
- ☐ 33F_Netzwerkverbindung_Start-im-Offlinemodus

Updates

- ☐ 34_Update_Daten-bleiben-erhalten
- ☐ 35_Update_Einstellungen-bleiben-erhalten

Sharing

- ☐ 36_Sharing_Optionen-verfügbar
- ☐ 37_Sharing_Funktionalität

Deeplinks

- ☐ 38_Deeplinks_Ziel-korrekt
- ☐ 39_Deeplinks_Ziel-korrekt-App-nicht-installiert

Sonstiges

- ☐ 40_Sonstiges_Landscape-Modus
- ☐ 41F_Sonstiges_Inputfeld-Grenzwert
- ☐ 42_Sonstiges-Pull-to-refresh-Funktion

ID:	01	Name:	01_Registrierung_Daten-korrekt
Kurzbeschreibung:		Nutzer ruft Registrierungsseite auf und befüllt die Pflichtfelder mit korrekten Daten (Daten, die zu einer Registrierung führen).	
Voraussetzungen:		Zu angegebenen Daten existiert noch kein Nutzer. Es ist kein Nutzer angemeldet.	
Schritt 1:		Navigation zu Registrierungsseite.	
Schritt 2:		Befüllung der Pflichtfelder mit korrekten Daten. Klick auf Registrierungs-Button.	
Sollergebnis:		Nutzer ist registriert. Nutzer ist angemeldet. Nutzer kann auf internen Bereich der App zugreifen.	
Alternative Sollergebnisse:		Weiterleitung zu Startseite. Bestätigungs-Popup. Bestätigungsmail an E-Mail-Adresse.	

ID:	02	Name:	02F_Registrierung_Daten-inkorrekt
Kurzbeschreibung:		Nutzer ruft Registrierungsseite auf und befüllt die Pflichtfelder mit inkorrekten Daten (bspw. Fehlendes "@" in E-Mail-Adresse).	
Voraussetzungen:		Zu angegebenen Daten existiert noch kein Nutzer. Es ist kein Nutzer angemeldet.	
Schritt 1:		Navigation zu Registrierungsseite.	
Schritt 2:		Befüllung der Pflichtfelder mit inkorrekten Daten. Klick auf Registrierungs-Button.	
Sollergebnis:		Nutzer ist nicht registriert. Nutzer hat keinen Zugriff auf internen App-Bereich. Es wird keine neue Seite aufgerufen.	
Alternative Sollergebnisse:		Ablehnungs-Popup. Rote Einfärbung von inkorrekt ausgefüllten Eingabefeldern.	

ID:	03	Name:	03F_Registrierung_Fehlende-Daten
Kurzbeschreibung:		Nutzer ruft Registrierungsseite auf und befüllt alle bis auf ein Pflichtfelder mit korrekten Daten (Daten, die zu einer Registrierung führen).	
Voraussetzungen:		Zu angegebenen Daten existiert noch kein Nutzer. Es ist kein Nutzer angemeldet.	
Schritt 1:		Navigation zu Registrierungsseite.	
Schritt 2:		Befüllung aller Pflichtfelder außer einem mit korrekten Daten. Klick auf Registrierungs-Button.	
Sollergebnis:		Nutzer ist nicht registriert. Nutzer hat keinen Zugriff auf internen App-Bereich. Es wird keine neue Seite aufgerufen.	
Alternative Sollergebnisse:		Ablehnungs-Popup. Rote Einfärbung des nicht ausgefüllten Eingabefeldes.	

ID:	04	Name:	04F_Registrierung_Bereits-vergebene-Daten
Kurzbeschreibung:		Nutzer ruft Registrierungsseite auf und befüllt die Pflichtfelder mit korrekten Daten, zu denen bereits ein anderer Nutzer registriert ist.	
Voraussetzungen:		Zu angegebenen Daten existiert bereits ein Nutzer. Es ist kein Nutzer angemeldet.	
Schritt 1:		Navigation zu Registrierungsseite.	
Schritt 2:		Befüllung der Pflichtfelder mit Daten zu denen bereits ein Nutzer existiert. Klick auf Registrierungs-Button.	
Sollergebnis:		Nutzer ist nicht registriert. Nutzer hat keinen Zugriff auf internen App-Bereich. Es wird keine neue Seite aufgerufen.	
Alternative Sollergebnisse:		Ablehnungs-Popup. Rote Einfärbung der falsch ausgefüllten Eingabefelder. Hinweis-Mail an eingegebene E-Mail-Adresse.	

ID:	05	Name:	05_Login_Credentials-korrekt
Kurzbeschreibung:		Nutzer ruft Loginseite auf und befüllt die Eingabefelder mit korrekten Credentials.	
Voraussetzungen:		Zu angegebenen Credentials existiert bereits ein Nutzer. Es ist kein Nutzer angemeldet.	
Schritt 1:		Navigation zu Loginseite.	
Schritt 2:		Befüllung der Eingabefelder mit korrekten Credentials. Klick auf Login-Button.	
Sollergebnis:		Nutzer ist angemeldet. Nutzer kann auf internen Bereich der App zugreifen. Weiterleitung zu Startseite.	
Alternative Sollergebnisse:		—	

ID:	06	Name:	06F_Login_Credentials-inkorrekt
Kurzbeschreibung:		Nutzer ruft Loginseite auf und befüllt die Eingabefelder mit inkorrekten Credentials.	
Voraussetzungen:		Zu angegebenen Credentials existiert bereits ein Nutzer. Es ist kein Nutzer angemeldet.	
Schritt 1:		Navigation zu Loginseite.	
Schritt 2:		Befüllung der Eingabefelder mit inkorrekten Credentials. Klick auf Login-Button.	
Sollergebnis:		Nutzer ist nicht angemeldet. Nutzer kann nicht auf internen Bereich der App zugreifen.	
Alternative Sollergebnisse:		Ablehnungs-Popup. Rote Einfärbung der falsch ausgefüllten Eingabefelder.	

ID:	07	Name:	07F_Login_Fehlende-Credentials
Kurzbeschreibung:		Nutzer ruft Loginseite auf und befüllt nur eines der Eingabefelder mit korrekten Credentials.	
Voraussetzungen:		Zu angegebenen Credentials existiert bereits ein Nutzer. Es ist kein Nutzer angemeldet.	
Schritt 1:		Navigation zu Loginseite.	
Schritt 2:		Befüllung eines der Eingabefelder mit korrekten Credentials. Klick auf Login-Button.	
Sollergebnis:		Nutzer ist nicht angemeldet. Nutzer kann nicht auf internen Bereich der App zugreifen.	
Alternative Sollergebnisse:		Ablehnungs-Popup. Rote Einfärbung der nicht ausgefüllten Eingabefelder.	

ID:	08	Name:	08_Login_Nutzer-bleibt-eingeloggt-nachdem-App-geschlossen
Kurzbeschreibung:		Angemeldeter Nutzer schließt die App und öffnet die App wieder.	
Voraussetzungen:		Es ist ein Nutzer angemeldet.	
Schritt 1:		Navigation zu einer Seite. Aufzeichnen dieser Seite.	
Schritt 2:		Schließen der App.	
Schritt 3:		Öffnen der geschlossenen App.	
Sollergebnis:		Nutzer ist angemeldet. Nutzer kann auf internen Bereich der App zugreifen.	
Alternative Sollergebnisse:		Nutzer ist nicht mehr angemeldet. Weiterleitung auf Startseite.	

ID:	09	Name:	09_Login_Weiterleitung-zu-Loginseite-bei-Klick-auf-Login-Feature
Kurzbeschreibung:		Nutzer navigiert zu einer Seite, die ohne Login nicht nutzbar ist.	
Voraussetzungen:		Es ist kein Nutzer angemeldet. App besitzt Bereich, der ohne Login erreichbar ist.	
Schritt 1:		Navigation zu Seite mit Link zu interner Seite.	
Schritt 2:		Klick auf Link zu interner Seite.	
Sollergebnis:		Nutzer wird zur Loginseite weitergeleitet.	
Alternative Sollergebnisse:		Nutzer wird zu Registrierungsseite weitergeleitet.	

ID:	10	Name:	10_Login_Features-ohne-Login-zugänglich
Kurzbeschreibung:		Nutzer navigiert zu einer Seite, die auch ohne Login nutzbar ist.	
Voraussetzungen:		Es ist kein Nutzer angemeldet. App besitzt Bereich, der ohne Login erreichbar ist.	
Schritt 1:		Navigation zu externer Seite.	
Sollergebnis:		Nutzer wird zu der gewünschten externen Seite weitergeleitet. Nutzer kann alle externen Features dieser Seite nutzen.	
Alternative Sollergebnisse:		—	

ID:	11	Name:	11_Navigation_Vorwärts
Kurzbeschreibung:		Nutzer navigiert durch die App indem er auf Navigations-Buttons klickt.	
Voraussetzungen:		—	
Schritt 1:		Navigation durch App.	
Sollergebnis:		Navigation funktioniert nach Spezifikation. Jeder Navigations-Button führt zu der vorgesehenen Seite.	
Alternative Sollergebnisse:		—	

ID:	12	Name:	12_Navigation_Rückwärts
Kurzbeschreibung:		Nutzer navigiert rückwärts durch die App indem er auf die entsprechenden Rückwärts-Buttons klickt.	
Voraussetzungen:		App besitzt Buttons zur Rückwärts-Navigation.	
Schritt 1:		Navigation durch App.	
Schritt 2:		Rückwärts-Navigation durch App.	
Sollergebnis:		Rückwärts-Navigation funktioniert nach Spezifikation. Jeder Rückwärts-Button führt zu der vorgesehenen Seite.	
Alternative Sollergebnisse:		—	

ID:	13	Name:	13_Navigation_Scrolling
Kurzbeschreibung:		Nutzer scrollt auf einer beliebigen Seite.	
Voraussetzungen:		Bildschirm des Testgeräts ist klein genug, um Scrolling zu erlauben.	
Schritt 1:		Navigation zu beliebiger Seite.	
Schritt 2:		Scrollen.	
Sollergebnis:		Scrolling funktioniert.	
Alternative Sollergebnisse:		—	

ID:	14	Name:	14_Navigation_Scrolling-Seitwärts
Kurzbeschreibung:		Nutzer scrollt seitwärts auf Seite mit entsprechendem Element.	
Voraussetzungen:		App besitzt Element, das Seitwärts-Scrolling erlaubt (bspw. Carousel).	
Schritt 1:		Navigation zu Seite mit entsprechendem Element.	
Schritt 2:		Seitwärts scrollen.	
Sollergebnis:		Seitwärts-Scrolling funktioniert in beide Richtungen.	
Alternative Sollergebnisse:		—	

ID:	15	Name:	15_Suche_Eingabe
Kurzbeschreibung:		Nutzer gibt Suchbegriff in Suchfeld ein.	
Voraussetzungen:		—	
Schritt 1:		Navigation zu Seite mit Suchfeld.	
Schritt 2:		Eingabe von Daten.	
Sollergebnis:		Eingabe von Daten ist möglich.	
Alternative Sollergebnisse:		Vorläufige Suchergebnisse werden angezeigt.	

ID:	16	Name:	16_Suche_Suchbegriff-korrekt
Kurzbeschreibung:		Nutzer sucht mittels Suchfeld nach einem korrekten Suchbegriff (Suchbegriff, der Ergebnis zurück liefert).	
Voraussetzungen:		Es existieren Daten nach denen gesucht werden kann.	
Schritt 1:		Navigation zu Seite mit Suchfeld.	
Schritt 2:		Eingabe von korrekten Daten. Klick auf Suchen-Button.	
Sollergebnis:		Suchergebnisse werden angezeigt.	
Alternative Sollergebnisse:		—	

ID:	17	Name:	17F_Suche_Suchbegriff-inkorrekt
Kurzbeschreibung:		Nutzer sucht mittels Suchfeld nach einem inkorrekten Suchbegriff (Suchbegriff, der kein Ergebnis zurück liefert).	
Voraussetzungen:		Es existieren Daten nach denen gesucht werden kann.	
Schritt 1:		Navigation zu Seite mit Suchfeld.	
Schritt 2:		Eingabe von inkorrekten Daten. Klick auf Suchen-Button.	
Sollergebnis:		Es werden keine Suchergebnisse angezeigt.	
Alternative Sollergebnisse:		Hinweis-Popup.	

ID:	18	Name:	18_Werbung_Darstellung-korrekt
Kurzbeschreibung:		Nutzer navigiert durch die App.	
Voraussetzungen:		—	
Schritt 1:		Navigation zu Seite mit Werbung.	
Sollergebnis:		Werbung wird korrekt dargestellt.	
Alternative Sollergebnisse:		—	

ID:	19	Name:	19_Werbung_Weiterleitung-korrekt
Kurzbeschreibung:		Nutzer klickt auf Werbung.	
Voraussetzungen:		—	
Schritt 1:		Navigation zu Seite mit Werbung.	
Schritt 2:		Klick auf Werbung.	
Sollergebnis:		Nutzer wird zu der Seite des Werbetreibenden weitergeleitet.	
Alternative Sollergebnisse:		—	

ID:	20	Name:	20_Werbung_Darstellung-für-Premiumnutzer
Kurzbeschreibung:		Premium-Nutzer navigiert durch App.	
Voraussetzungen:		App besitzt Bezahlmodell wonach Premium-Nutzern keine Werbung mehr angezeigt wird.	
Schritt 1:		Navigation durch App.	
Sollergebnis:		Werbung wird nicht dargestellt.	
Alternative Sollergebnisse:		—	

ID:	21	Name:	21_Dark-Mode_Darstellung-korrekt
Kurzbeschreibung:		Nutzer aktiviert den Dark Mode der App.	
Voraussetzungen:		Dark Mode ist noch nicht aktiviert.	
Schritt 1:		Navigation zu den entsprechenden Einstellungen.	
Schritt 2:		Aktivierung der Einstellung „Dark Mode“.	
Sollergebnis:		Alle Seiten werden korrekt im Dark Mode angezeigt.	
Alternative Sollergebnisse:		—	

ID:	22	Name:	22_Dark-Mode_Light-Mode-Darstellung-korrekt
Kurzbeschreibung:		Nutzer deaktiviert den Dark Mode der App.	
Voraussetzungen:		Dark Mode ist aktiviert.	
Schritt 1:		Navigation zu den entsprechenden Einstellungen.	
Schritt 2:		Deaktivierung der Einstellung „Dark Mode“.	
Sollergebnis:		Alle Seiten werden korrekt angezeigt. Keine Seite wird mehr im Dark Mode angezeigt.	
Alternative Sollergebnisse:		—	

ID:	23	Name:	23_Dark-Mode_Wie-System
Kurzbeschreibung:		Nutzer aktiviert die Einstellung „Dark Mode wie System“.	
Voraussetzungen:		App unterstütz die entsprechende Einstellung.	
Schritt 1:		Navigation zu den entsprechenden Einstellungen.	
Schritt 2:		Aktivierung der Einstellung „Dark Mode wie System“.	
Sollergebnis:		Alle Seiten werden nach Systemspezifikationen im Dark Mode bzw. Light Mode angezeigt.	
Alternative Sollergebnisse:		—	

ID:	24	Name:	24_Fremdinhalte_Videos-abspielbar
Kurzbeschreibung:		Nutzer spielt ein eingebettetes Video ab.	
Voraussetzungen:		Die App besitzt ein eingebettetes Video.	
Schritt 1:		Navigation zu Video.	
Schritt 2:		Klick auf Video.	
Sollergebnis:		Video wird abgespielt.	
Alternative Sollergebnisse:		Weiterleitung zur Website des eingebetteten Videos. Video wird stummgeschalten.	

ID:	25	Name:	25_Fremdinhalte_Video-Operationen
Kurzbeschreibung:		Nutzer pausiert ein Video, scrollt durch das Video und spielt das Video von der Stelle weiter ab.	
Voraussetzungen:		Die App besitzt ein eingebettetes Video.	
Schritt 1:		Navigation zu Video.	
Schritt 2:		Abspielen des Videos.	
Schritt 3:		Pausieren des Videos.	
Schritt 4:		Sprung zu einer anderen Zeitmarke.	
Schritt 5:		Fortsetzen des Videos.	
Sollergebnis:		Die Operationen werden korrekt durchgeführt. Das Video wird von der angegebenen Zeitmarke weiter abgespielt.	
Alternative Sollergebnisse:		—	

ID:	26	Name:	26_Fremdinhalte_Video-Sound
Kurzbeschreibung:		Nutzer spielt ein Video ab und schaltet es stumm.	
Voraussetzungen:		Die App besitzt ein eingebettetes Video.	
Schritt 1:		Navigation zu Video.	
Schritt 2:		Abspielen des Videos.	
Schritt 3:		Stummschalten des Videos.	
Sollergebnis:		Je nach Spezifikation wird das Video beim Abspielen stummgeschaltet oder der Ton ist zu hören. Nach dem Stummschalten ist der Ton nicht mehr zu hören.	
Alternative Sollergebnisse:		—	

ID:	27	Name:	27_Fremdinhalte_Video-Fullscreen
Kurzbeschreibung:		Nutzer spielt ein Video ab und aktiviert den Fullscreen-Modus.	
Voraussetzungen:		Die App besitzt ein eingebettetes Video.	
Schritt 1:		Navigation zu Video.	
Schritt 2:		Abspielen des Videos.	
Schritt 3:		Aktivieren des Fullscreen-Modus.	
Sollergebnis:		Das Video wechselt in den Fullscreen-Modus.	
Alternative Sollergebnisse:		Das Video wechselt in den Landscape-Modus.	

ID:	28	Name:	28_Fremdinhalte_Dokument-Darstellung
Kurzbeschreibung:		Nutzer öffnet ein Dokument innerhalb der App.	
Voraussetzungen:		Die App besitzt ein eingebettetes Dokument.	
Schritt 1:		Navigation zu Dokument.	
Schritt 2:		Klick auf Dokument.	
Sollergebnis:		Das Dokument wird korrekt angezeigt.	
Alternative Sollergebnisse:		—	

ID:	29	Name:	29_Fremdinhalte_Dokument-Operationen
Kurzbeschreibung:		Nutzer öffnet ein Dokument innerhalb der App und führt eine Operation aus (bspw. Kopieren oder Drucken).	
Voraussetzungen:		Die App besitzt ein eingebettetes Dokument. Die App bietet Operationen auf dem Dokument an.	
Schritt 1:		Navigation zu Dokument und Öffnen des Dokuments.	
Schritt 2:		Ausführen der jeweiligen Operation.	
Sollergebnis:		Die Operation lässt sich auf dem Dokument ausführen.	
Alternative Sollergebnisse:		—	

ID:	30	Name:	30F_Netzwerkverbindung_Ausfall
Kurzbeschreibung:		Nutzer führt Funktion aus, die eine Netzwerkverbindung benötigt. Dabei fällt die Netzwerkverbindung aus.	
Voraussetzungen:		Ausfall der Netzwerkverbindung kann simuliert werden.	
Schritt 1:		Navigation zu Funktion, die eine Netzwerkverbindung benötigt.	
Schritt 2:		Ausführen der Funktion.	
Schritt 3:		Ausfall der Netzwerkverbindung.	
Sollergebnis:		Die App reagiert nach Spezifikation.	
Alternative Sollergebnisse:		Der Vorgang wird abgebrochen. Hinweis-Popup. Der Vorgang kann später fortgesetzt werden.	

ID:	31	Name:	31F_Netzwerkverbindung_Störung
Kurzbeschreibung:		Nutzer führt Funktion aus, die eine Netzwerkverbindung benötigt. Dabei kommt es zu Störungen der Netzwerkverbindung.	
Voraussetzungen:		Störung der Netzwerkverbindung kann simuliert werden.	
Schritt 1:		Navigation zu Funktion, die eine Netzwerkverbindung benötigt.	
Schritt 2:		Ausführen der Funktion.	
Schritt 3:		Störung der Netzwerkverbindung.	
Sollergebnis:		Die Funktion kann trotzdem ausgeführt werden.	
Alternative Sollergebnisse:		Der Vorgang wird abgebrochen. Hinweis-Popup. Der Vorgang kann später fortgesetzt werden.	

ID:	32	Name:	32F_Netzwerkverbindung_Ortungsdienst-Ausfall
Kurzbeschreibung:		Nutzer führt Funktion aus, die Ortungsdienste benötigt. Dabei kommt es zu Störungen der Ortungsdienste.	
Voraussetzungen:		Die App besitzt Funktionen, die Ortungsdienste benötigt. Störung der Ortungsdienste kann simuliert werden. Die App hat Zugriff auf Ortungsdienste.	
Schritt 1:		Navigation zu Funktion, die Ortungsdienste benötigt.	
Schritt 2:		Ausführen der Funktion.	
Schritt 3:		Ausfall der Ortungsdienste.	
Sollergebnis:		Die App reagiert nach Spezifikation.	
Alternative Sollergebnisse:		Der Vorgang wird abgebrochen. Hinweis-Popup. Der Vorgang kann später fortgesetzt werden.	

ID:	33	Name:	33F_Netzwerkverbindung_Start-im-Offlinemodus
Kurzbeschreibung:		Nutzer startet die App, ohne dass eine Internetverbindung hergestellt ist.	
Voraussetzungen:		Es besteht keine Internetverbindung.	
Schritt 1:		Starten der App.	
Sollergebnis:		Die App funktioniert nach Spezifikation. Funktionen, die keine Internetverbindung benötigen, können ausgeführt werden.	
Alternative Sollergebnisse:		—	

ID:	34	Name:	34_Update_Daten-bleiben-erhalten
Kurzbeschreibung:		Nutzer installiert ein Update.	
Voraussetzungen:		Eine ältere Version der App ist installiert. Es wurden Daten hinterlegt, die nach dem Update erhalten bleiben sollen.	
Schritt 1:		Installation des Updates.	
Sollergebnis:		Die hinterlegten Daten sind nach dem Update erhalten geblieben.	
Alternative Sollergebnisse:		—	

ID:	35	Name:	35_Update_Einstellungen-bleiben-erhalten
Kurzbeschreibung:		Nutzer installiert ein Update.	
Voraussetzungen:		Eine ältere Version der App ist installiert. Es wurden Einstellungen getätigt, die nach dem Update erhalten bleiben sollen.	
Schritt 1:		Installation des Updates.	
Sollergebnis:		Die getätigten Einstellungen sind nach dem Update erhalten geblieben.	
Alternative Sollergebnisse:		—	

ID:	36	Name:	36_Sharing_Optionen-verfügbar
Kurzbeschreibung:		Nutzer teilt Inhalte über einen Drittanbieter-Dienst mit anderen Nutzern.	
Voraussetzungen:		Der entsprechende Drittanbieter-Dienst ist installiert und dort ist ein Nutzer angemeldet.	
Schritt 1:		Navigation zu Inhalt der geteilt werden soll.	
Schritt 2:		Klick auf Sharing-Button.	
Sollergebnis:		Es besteht die Möglichkeit den Inhalt über den entsprechenden Dienst zu teilen.	
Alternative Sollergebnisse:		—	

ID:	37	Name:	37_Sharing_Funktionalität
Kurzbeschreibung:		Nutzer teilt Inhalte über einen Drittanbieter-Dienst mit anderen Nutzern.	
Voraussetzungen:		Der entsprechende Drittanbieter-Dienst ist installiert und dort ist ein Nutzer angemeldet.	
Schritt 1:		Navigation zu Inhalt der geteilt werden soll.	
Schritt 2:		Klick auf Sharing-Button.	
Schritt 3:		Klick auf entsprechenden Dienst. Klick auf empfangenden Nutzer.	
Sollergebnis:		Der Inhalt wird geteilt. Der empfangende Nutzer erhält eine Nachricht über den entsprechenden Dienst.	
Alternative Sollergebnisse:		Bestätigungs-Popup.	

ID:	38	Name:	38_Deeplinks_Ziel-korrekt
Kurzbeschreibung:		Nutzer klickt auf einen Deeplink. Die App ist bereits installiert.	
Voraussetzungen:		—	
Schritt 1:		Navigation zu Deeplink.	
Schritt 2:		Klick auf Deeplink.	
Sollergebnis:		Der Deeplink führt an die richtige Stelle innerhalb der App.	
Alternative Sollergebnisse:		—	

ID:	39	Name:	39_Deeplinks_Ziel-korrekt-App-nicht-installiert
Kurzbeschreibung:		Nutzer klickt auf einen Deeplink. Die App ist nicht installiert.	
Voraussetzungen:		Die App ist nicht installiert.	
Schritt 1:		Navigation zu Deeplink.	
Schritt 2:		Klick auf Deeplink.	
Sollergebnis:		Der Deeplink führt zur Installationsseite der App.	
Alternative Sollergebnisse:		Hinweis-Popup.	

ID:	40	Name:	40_Sonstiges_Landscape-Modus
Kurzbeschreibung:		Nutzer dreht das Gerät, um den Landscape-Modus zu nutzen.	
Voraussetzungen:		Die App unterstützt den Landscape-Modus. Das Drehen des Geräts kann simuliert werden.	
Schritt 1:		Navigation zu Seite die im Landscape-Modus darstellbar ist.	
Schritt 2:		Drehen des Geräts.	
Sollergebnis:		Die Seite wird korrekt im Landscape-Modus angezeigt.	
Alternative Sollergebnisse:		—	

ID:	41	Name:	41F_Sonstiges_Inputfeld-Grenzwert
Kurzbeschreibung:		Nutzer gibt sehr langen Begriff in ein Eingabefeld ein.	
Voraussetzungen:		Die App besitzt Eingabefelder.	
Schritt 1:		Navigation zu Eingabefeld.	
Schritt 2:		Eingabe des Grenzwertes der Charakterlänge.	
Sollergebnis:		Die App funktioniert weiterhin. Der Vorgang wird abgebrochen. Hinweis-Popup.	
Alternative Sollergebnisse:		Rote Einfärbung des inkorrekt ausgefüllten Eingabefeldes.	

ID:	42	Name:	42_Sonstiges-Pull-to-refresh-Funktion
Kurzbeschreibung:		Nutzer zieht den Bildschirm nach unten (Pull-to-refresh-Geste), um den Inhalt der Seite zu aktualisieren.	
Voraussetzungen:		Die App besitzt eine Seite, die mit einer Pull-to-refresh-Geste aktualisiert werden kann.	
Schritt 1:		Navigation zu Seite die mit einer Pull-to-refresh-Geste aktualisiert werden kann.	
Schritt 2:		Ausführen der Pull-to-refresh-Geste.	
Sollergebnis:		Der Inhalt der Seite wird aktualisiert.	
Alternative Sollergebnisse:		—	