Model-Based Testing für mobile Systeme
Ausarbeitung Grundseminar Wintersemester 2014

Alexander Piehl

Hamburg University of Applied Sciences,
Dept. Computer Science
Berliner Tor 7
20099 Hamburg, Germany
Email: alexander.piehl@haw-hamburg.de

I. Einleitung

Die Qualität von Software misst sich nicht alleine an dem Wie der Umsetzung der Anforderungen und einer guten Bedienung ggf. mithilfe einer schön designeten Oberfläche, sondern auch daran, dass die geforderten Anforderungen erfüllt sind und dies besonders auch fehlerfrei. Bis heute werden viele Softwareprodukte veröffentlicht, die zwar mit Features und schönem Design glänzen, aber sehr instabil laufen. Diese Instabilität könnte dafür sorgen, dass die Software beim Kunden in Ungnade verfällt und dahingehend nicht mehr verwendet wird. Dadurch kann die Software schnell zu einem Verlustgeschäft werden. Ein bekanntes Beispiel ist der Apple Kartendienst Apple Maps, welcher beim Release viele Fehler enthielt und daraufhin viele Nutzer abschreckte.

Mithilfe ausgiebigen Testens kann erreicht werden, dass eine Software stabiler und zuverlässiger funktioniert. Jedoch wird oft in der Praxis die Zeit dafür nicht investiert. Dies hat zumeist mehrere Gründe. Zu einem kostet das ausführliche Testen von Software viel Zeit und damit einhergehend auch Geld. Zum anderen kann es während der Entwicklung der Software zu Komplikationen kommen, die dafür sorgen, dass der Zeitplan nicht mehr eingehalten werden kann. In diesen Fällen wird häufig die Priorität darauf gesetzt, die vom Kunden gewünschten Anforderungen zu Ende zu implementieren, damit das Produkt bis zur Deadline fertig gestellt ist. Diese Entscheidung trägt auch häufig die Konsequenz mit, dass die Software nicht mehr ausführlich getestet wird bzw. werden kann. Dadurch kann eine instabile Software entstehen.

Die Problematik des fehlenden bzw. ausführlichen Testen zieht sich durch die gesamte Softwarebranche. Es spielt dabei keine Rolle, ob es sich um ein Computerspiel, Textverarbeitungsprogramm oder andere Software handelt.

Mit dem Testen von Software sollen nicht nur mögliche Fehler gefunden werden, die das Programm ggf. zum Absturz bringen. Es wird des Weiteren auch geprüft, ob die Anwendung zu den Spezifikationen konform ist. Der Prozess dieser Überprüfung wird Verifikation genannt. Die Anwendung wird also dahingehend verifiziert, ob sie ihren Spezifikation entspricht. Würde man dies als Frage definieren, würde die Frage ungefähr so lauten „Wird das Produkt richtig erstellt?“

Dazu im Gegensatz steht die Validierung. Bei der Validierung wird überprüft, ob das Programm seinen Anforderungen entspricht. Es wird im Prinzip validiert, ob die Anwendungen ihren Anwendungszweck erfüllt.

Dahingehend bedeutet gutes und ausführliches Testen nicht nur, dass Fehler gefunden werden, die während der Implementierung entstanden sind. Es bedeutet besonders auch, dass die Anwendung verifiziert wird, ob sie ihren Spezifikationen entspricht und validiert wird, ob sie ihre Anforderungen erfüllt. Eigentlich liegt der Hauptaugenmerk beim Testen nicht darauf Fehler zu finden, sondern um zu prüfen, dass das Programm seinen Anforderungen entspricht.

Beim Systemtest werden häufig manuell aus den Anforderungen an das Programm die Testfälle erstellt. Dabei bildet meist ein Testfall eine Anforderungen ab, die getestet werden soll, ob sie ordnungsgemäß implementiert wurden ist. Die Testfälle werden zudem meistens noch manuell ausgeführt und ausgewertet. Doch es gibt bereits viele Ansätze und Techniken, die einen automatisierten Ablauf dieses Vorganges versprechen. Eines dieser Techniken, die neben der Automatisierung noch ein effizientes und ausführliches Testen versprechen, ist das Model-based Testing. Model-based Testing wird generell als MBT abkürzt.

In dieser Ausarbeitung wird beschrieben, welche Motivation vorliegt, an diesem Thema zu arbeiten und warum diese Thema so eine wichtige Bedeutung hat. Des Weiteren wird es einen Überblick darüber geben, was Model-based Testing eigentlich im Detail bedeutet. Ergänzend dazu werden die einzelnen Schritte von Model-based Testing genauer beleuchtet und erklärt. Dazu kommend werden noch weitere Vorteile vorgestellt, die Model-based Testing mit sich bringt.

Des Weiteren wird in dieser Ausarbeitung auch die aktuelle Forschung wiedergegeben. Es wird dabei auch darauf eingegangen, in welche Richtung sich die momentane Forschung sich bewegt. Zudem werden in diesem Zusammenhang wichtige Forschungsgruppen vorgestellt, die sich mit dem Bereich von Model-based Testing beschäftigen.

Abschließend gibt es in dieser Ausarbeitung einen Ausblick darüber, welche Aufgaben und Ziele es für das Grundprojekt geben wird. Ergänzend dazu gibt es eine Abgrenzung der eigenen Arbeit im Bereich des modellbasierten Testens.

II. Motivation

Die Welt ist im Wandel. Auf jeden Fall, wenn man sich die gestiegenen Nutzungszahlen von mobilen Geräten anschaut. Wenn etwa noch vor 10 Jahren hauptsächlich Software für stationäre Endgeräte wie Computer und Laptop entworfen worden ist, wird heutzutage sehr viel Software für mobile

Endgeräte wie Tablets und Smart Phones entwickelt.Der Markt dafür wird sehr wahrscheinlich in den nächsten Jahren noch weiter steigen.

Die Entwicklung von Apps ist nicht nur durch den gestiegenen Markt interessant, sondern auch unter anderen Aspekten. Die Entwicklung von Apps unterscheidet sich von der Entwicklung von Programmen für normale Computer, wie einem Desktop Computer. Die Entwicklungszyklen sind bei der Entwicklung von Apps häufig viel geringer. Dazu kommend muss bei der Entwicklung von nativen Apps darauf geachtet werden, dass sie mit den vorhandenen Ressourcen umsichtig umgehen. Mobile Geräte laufen normalerweise immer über einen Akku. Deswegen muss bei der Entwicklung darauf geachtet werden, dass die entwickelte Software, den Akku nicht übermäßig belastet. Des Weiteren stehen häufiger weniger Arbeitsspeicher und kleinere und schwächere Prozessoren zur Verfügung.

Besonders bei Apps gibt es einen hohen Konkurrenzdruck. Es gibt für viele Anwendungsfälle mehrere Apps, die die gleichen Features bieten. Sucht man im Google Play Store nach einer App mit Wecker-Funktion, bekommt man mehr als 200 Ergebnisse. Dahingehend ist es für den Kunden schon ein Entscheidungsmerkmal, ob die Software stabil funktioniert. Sollte sie instabil sein, würde der Kunde sehr wahrscheinlich zu einem anderen Produkt greifen, welches die gleichen Features bietet, aber dazu noch stabil funktioniert.

Aus dem oben beschriebenen hohen Konkurrenzdruck entsteht auch ein hoher Zeitdruck. Da es häufig mehrere Apps für einen Anwendungsfall gibt, besteht auch immer die Gefahr das ein Konkurrenzprojekt zum eigenen Projekt entwickelt wird. Dementsprechend wird versucht das eigene Produkt so früh wie möglich zu veröffentlichen, weil man nicht weiß, wie weit der Entwicklungsstand bei möglichen Konkurrenzprojekten ist und man unbedingt als erster im Markt sein möchte. Daher möchte man schon sein Produkt am Markt etabliert haben, bevor das Konkurrenzprodukt veröffentlicht wurden ist. Dazu kommend werden Apps häufig für geringe Beträge verkauft. Die Kosten für die Implementierung müssen durch diese Beträge getragen werden. Daher ist es enorm wichtig, dass nicht zu viel Zeit für die Implementierung und somit Geld verbraucht wird.

Auch unter den verschiedenen Software Studios herrscht ein Konkurrenzdruck. Daher kann es vorkommen, dass man versucht einen Auftrag zu bekommen, indem man ein günstigeres Angebot vorlegt. Das günstigere Angebot kann unter anderem dadurch erreicht werden, indem man eine geringere Entwicklungszeit vorsieht. Dies kann den Zeitdruck zusätzlich erhöhen.

Für das ausführliche Testen kann der Zeitdruck negative Folgen haben. Es kann dazu führen, dass das Testen der Software nicht mehr in der geplanten Ausführlichkeit durchgeführt wird. Dies kann selbstverständlich Folgen für die Qualität der Anwendung haben. Es kann dahingehend sein, dass Anforderungen nicht richtig implementiert werden oder eklatante Fehler unentdeckt bleiben.

Wie schon beschrieben, haben native Apps einen geringeren Entwicklungszyklus. Um diesen Zyklus nicht zu verlängern, wäre es sinnvoll das Testen innerhalb dieses Zyklus durchzuführen. Dazu kommt die Besonderheit, beim Testen von Apps für Android, dass die Hardware unbekannt. Das Android-Betriebssystem läuft auf sehr viel verschiedenen Geräten, die alle unterschiedliche Eingenschaften haben und sich in ihrem Leistungsumfang unterscheiden. Eine weitere Besonderheit beim Testen von Apps ist, dass die GUI auch ausführlich getestet werden muss. Besonders bei Apps spielt die Benutzeroberfläche eine sehr wichtige Rolle. Dahingehend ist es wichtig, dass die Benutzeroberfläche und somit die Interaktion mit dem Nutzer ordnungsgemäß funktionieren.

Für das Testen von Android Apps hat Google ein eigenes Framework entwickelt, um das Testen von Android Apps zu regeln. Das Android Testing Framework wird von Google selbst als ein integraler Bestandsanteil der Entwicklung bezeichnet[9]. Dabei soll das Framework den Entwicklern beim Testen unterstützen, indem es die Architektur und und Tools zum Testen bereitstellt. Mithilfe dieses Framework soll es möglich sein, jeden Aspekt der Anwendung zu testen. Das Framework und die dazugehörigen Tools befinden sich direkt in der Eclipse Variante mit den Android Developer Tools. Die Tools können auch in anderen IDEs verwendet werden.

Das Android Testing Framework bietet verschiedene einzelne Bereiche an, um die versprochene Testabdeckung zu erreichen. Die Erstellung der Testfälle beruht dabei auf JUnit. JUnit ist ein Framework, welches es ermöglicht Units-Tests durchzuführen. Bei Unit-Tests wird direkt der Source Code gestestet, ob er den Anforderungen entspricht.

Mit dem normalen JUnit ist es nur möglich Klassen zu testen, welche nicht die Android API verwenden. Um auch Code zu testen, welcher die Android API verwenden, gibt es von Google die Android JUnit Extension. Mit dieser Erweiterung können dann auch Code-Abschnitte getestet werden, die die Android API verwenden.

Des Weiteren bietet diese Erweiterung an Mock-Objekte zu erstellen. Mock-Objekte ersetzen im Prinzip ein richtiges Objekt, welches zum Zeitpunkt des Testen entweder noch nicht existiert oder nur schwer anzubinden ist. Häufig werden Mock-Objekte dafür verwendet, um Datenbankzugriffe zu simulieren. Zudem werden Mechanismen zur Verfügung gestellt, um den Lebenszyklus eines Objekts zu kontrollieren. Dazu bietet es auch Mechanismen an, um die Benutzerfläche der Anwendung zu testen.

Zusammenfassend kann man sagen, dass Google für das Testen von Android Anwendungen schon ein sehr umfassendes Framework anbietet. Dahingehend stellt sich die Frage, ob das Framework wirklich alles das leistet. was es verspricht. Zudem kommt noch die Frage auf, ob das Testen von Android Apps nicht vielleicht mit Model-Based Testing besser und vor allem effizienter abläuft. Denn die Testfälle innerhalb des Android Testing Frameworks müssen manuell erstellt werden.

Da die Programmierung von Software immer unter Zeitdruck steht und somit auch das Testen, ist es wichtig das Testen Zeiteffizient zu gestalten. Unter anderem aus dieser Überlegung heraus, ist Model-based Testing entstanden.

Kurz zusammengefasst bedeutet Model-based Testing (MBT) die automatische Generierung von Testfällen anhand von Modellen, welche auf die zu testende Software beruhen, und deren automatische Durchführung. Mit der Automatisierung des Testen möchte man ein Testen erreichen, was effizient mit der Zeit umgeht. Des Weiteren soll auch damit erreicht werden, dass die Software beim Testen komplett abgedeckt wird. Das bedeutet, dass jede einzelne Komponente getestet werden soll, damit möglichst viele Fehler entdeckt werden und

geprüft werden kann, ob die Komponente ihren Anforderungen entspricht.

Model-Based Testing ist aus mehreren Punkte sehr interessant. Es verspricht ein zeiteffizientes Testen auf Grundlage der Automatisierung. Daher stellt sich hier schon die Frage, wie viel Zeit wird wirklich durch die Automatisierung eingespart und hält es somit was es verspricht. Dieser Aspekt wäre schon interessant für eine genauere Untersuchung.

Zudem ist Model-Based Testing durchaus interessant unten dem generellen Aspekt der Automatisierung. Die Testfälle werden direkt aus den Modellen erstellt. Deswegen wäre es schon aufschlussreich, wie dieser Prozess genau funktioniert. Auch die Erstellung der Modelle ist ein wissenswerter Punkt. Denn es stellen sich hier mehrere Fragen, wie z.B. welche Modellierungssprachen sich für das Erstellen der Modelle am Besten eignet und wie viel Zeit das Erstellen der Modelle benötigt.

Diese und andere Fragen kann man am leichtesten an einem konkreten Beispiel beantworten. Es bieteten sich mehrere verschiedene Beispiele von Software an, welche man für das Überprüfen von Model-Based Testing nutzen könnte. Man könnte zu einem eine übliche Software wie ein Warenwirtschaftssystem testen oder schauen, wie sich Model Based Testing im Bereich von Embedded Systems verhält. Der persönlich jedoch interessantes Bereich, ist der Bereich der Software für mobile Geräte, also Apps. Da auch der Bereich der Apps sehr breit gefächert ist, wäre eine Konzentration ausschließlich auf Android Apps sehr sinnvoll.

Genau aus diesen oben aufgeführten Punkten bietet es sich an genauer zu analysieren, wie sich Model-based Testing beim Testen von Software von mobilen Geräten verhält. Sollte Model-based Testing das halten was es verspricht, also Zeit effizientes Testen und komplette Abdeckung der Software, wäre es für das Testen von Apps sehr passend. Dazukommend müsste aber auch geprüft werden, ob das ModelbBased Testing den gleichen oder ggf. den größeren Testumfang als das Standard Testverfahren von Android Anwendungen bietet.

III. Model-based Testing

Model-based Testing ist eine Technik für die automatische Erstellung von Testfällen aus einem Modell oder mehreren Modellen, welche das Verhalten eines Systems beschreiben [1]. Das System, welches getestet werden soll, trägt in diesem Zusammenhang die Bezeichnung System-under-Test, kurz SUT. Die daraus entstehenden Testfälle werden zudem auch automatisch ausgeführt [2]. Model-based Testing wird auch als 4.Generation des automatischen Testens betitelt[3].

Den Prozess des Model-Based Testings kann man generell in 3 Hauptschritte unterteilen. Im ersten Schritt werden die formalen Anforderungen an das SUT modelliert. Im darauf folgenden Schritt werden die Testfälle automatisch aus den Testfällen erstellt. Im letzten Schritt werden dann die erstellten Testfälle ausgeführt und testen somit das SUT. Zu diesem Schritt gehört auch die Evaluierung der Testergebnisse. [4] Der grobe Ablauf ist in Abbildung 1 nochmal als Übersicht abgebildet.

Bei der Modellierung der Spezifikation vom SUT muss alles beachtet werden, was das System im Endeffekt leisten soll[4]. Dies ist die Voraussetzung dafür, um aus diesem Modellen Testfällen erstellen zu lassen, welche auch alle

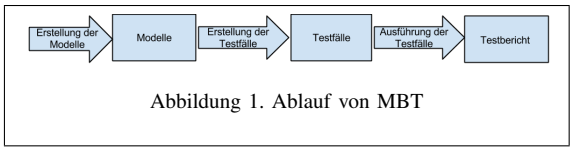

Abbildung 1. Ablauf von MBT

Anforderungen testen. Dahingehend ist es wichtig eindeutige Modelle zu haben, denn nur so bekommt man auch eindeutige Testfälle[4]. Ergänzend dazu, müssen die Modelle abstrakter sein als das SUT [5].

Das System-under-Test kann auf drei unterschiedliche Arten modelliert werden. Es kann zu einem unter dem Aspekt des Daten Inputs modelliert werden. Diese Modellierung wird als Data Model verstanden[1]. Zum anderen kann es unter dem Aspekt der Interaktionen mit dem Nutzer modelliert werden. Zusammengefasst wird diese Modellierung unter dem Begriff Tester Model [1]. Die letzte Möglichkeit wäre die Modellierung des SUT unter dem Aspekt des dynamischen Verhalten des Systems, dem sogenannten Design Model [1].

Für die Modellierung der Modelle stehen sehr viele verschiedene Modellierungssprachen zur Verfügung. Man kann diese Modellierungssprachen generell in drei 3 Gruppen unterteilen [1] und zwar in formale, semi-formale und nicht-formale Modellierungssprachen. In der Industrie werden am häufigsten semi-formale Modellierungssprachen, wie UML verwendet[1]. Besonders UML findet eine große Verwendung beim Model-Based Testing [6].

Sobald die Modelle erstellt wurden sind, können aus diesen Modellen die Testfällen erstellt werden. Für die Erstellung der Testfälle gibt es mehrere Programme, die diese Aufgaben übernehmen. Dabei gibt es kommerzielle Programme genauso freie Programme.

Die Programme produzieren häufig als Ergebnis mehrere sogenannten Abstract-Test-Cases[7]. Diese erstellten Testfälle sind noch keine ausführbare Testfälle. Jedes Abstract-Test-Cases ist dabei eine Operation, die vom System gefordert wird, mit den entsprechenden Input Daten und den zu erwarteten Ergebnissen [7].

Im nächsten Schritt müssen dieses Abstract-Test-Cases in ausführbare Testskripte umgewandelt werden[7]. Dieser Prozess kann von einem Programm übernommen werden, kann aber auch manuell durchgeführt werden [4]. Diese Transformation bzw. Konkretisierung der Testfälle findet auf Grundlage von Templates und Übersetzungstabellen statt, die meist vom Tester bereitgestellt werden. Diese Templates und Übersetzungstabellen beschreiben das SUT genauer und sorgen dementsprechend dafür, dass aus den abstrakten Modellen ausführbare Testfälle werden [7].

Nun kann mit dem erstellen und ausführbaren Testfälle das SUT getestet werden. Dies geschieht mithilfe eines Programmes, wobei dieser Prozess meistens automatisiert stattfindet[7][4]. Grundsätzlich werden die von den Testfällen festgelegten Eingaben mit der dazugehörigen Operation festgehalten und anschließend ausgewertet [4]. Der Tester enthält am Ende eine Übersicht, welche Fehler im SUT entdeckt wurden sind. Diese Übersicht unterscheidet sich von Programm zu Programm.

Model-based Testing soll einige Vorteile mit sich bringen im Gegensatz zum kompletten manuellen Testen. Wie schon

mehrmals beschrieben, soll mit Model-based Testing das Testen effizienter gestaltet werden, was die Zeit angeht. Dies bedeutet, dass das Model-based Testing weniger Zeit beansprucht als das normale Testverfahren. Die Zeitersparnis soll durch die Automatisierung erfolgen. Dahingehend kann man die Automatisierung des Testens auch als einen vermeintlichen Vorteil betrachten.

Damit das Testen automatisiert werden kann, müssen zunächst die Modelle erstellt werden. In den Modellen sollen sich die kompletten Anforderungen an das System widerspiegeln. Damit soll erreicht werden, dass die komplette Software abgedeckt wird, was wiederum recht vorteilhaft wäre. Dazu kommend sollen die Modelle auch den Vorteil bieten, dass Änderungen schnell ins Testverfahren eingepflegt werden können [3]. Bei Änderungen an den Anforderungen können diese einfach in die bestehende Modelle eingepflegt werden. Auch Änderungen an der Testumgebung sollen schnell eingepflegt werden können, da im Prinzip nur die Transformation zwischen den abstrakten und ausführbaren Testfälle angepasst werden muss [3].

IV. Aktueller Stand der Forschung

Zurzeit wird bei Model-based Testing in viele verschiedene Richtungen geforscht. So finden sich mehrere Forschungsarbeiten dazu, wie man Modelle besser und einfacher erstellen kann. Des Weiteren gibt es Studien, die sich damit befassen, ob Model-based Testing seine vermeintlichen Vorteile einhält und wie es sich in einem realen Kontext verhält. Ein weiterer Forschungsschwerpunkt ist, wie GUI's mit Model-based Testing getestet werden können.

In der Ausarbeitung von Wendland, Hoffmann und Schieferdecker mit Title Fokus!MBT: a multi-paradigmatic test modeling environment"geht es darum, wie die Erstellung der Modelle mit UML leichter gestaltet werden können [10]. Sie gehen in ihrer Arbeit davon aus, dass die Tester selten sehr erfahren mit der Anwendung von UML seien. Da aber die Erstellung der Modelle nicht sehr komfortabler sei und man ein gutes Wissen über UML bräuchte, um die Modelle ordnungsgemäß zu erstellen, wäre dies ein Problem in der Praxis. Dahingehend haben sie ein Tool in Verbindung mit dem Fraunhofer-Institut FOKUS entwickelt, welches dieses Problem lösen soll. Das entwickelte Tool soll Mechanismen anbieten, welche das Erstellen und Authentifizieren von Modellen unterstützen soll. Dabei basiere das Tool auf UML Testing Profile. UML Testing Profile sei eine OMG(Object Management Group) definierte Notation für Model-Based Testing [10].

In einem aktuellen Papier von 2014 von Schulze, Ganesan, Lindvall, Cleaveland und Goldmann geht es um einen Vergleich zwischen Model-based Testing und manuellen Testen ohne irgendwelche Automatisierung. Das Papier trägt den Titel Ässessing model-based testing: an empirical study conducted in industry"[2]. Als Testsubjekt diente ihnen dabei eine Anwendung, welche professionell entwickelt sei. Dabei sei die Anwendung eine Web-basierte Daten Sammlung, die auch in der Praxis eingesetzt würde. Beide Ansätze seien unter zwei Punkten verglichen wurden und zwar unter dem Punkt, wie viele Fehler gefunden wurden und wie viel Aufwand der Ansatz benötigte. Als Ergebnis kam das Papier zum Schluss, dass manuelles Testen zwar weniger Aufwand benötigte, aber auch weniger Fehler finden würden als Model based-Testing. In einem selbst definierten Punktesystem bekam Model-based Testing 60 Prozent mehr Punkte als das manuelle Testen[2].

Ein großer Forschungsschwerpunkt bei Model-Based Testing ist das Testen von Benutzeroberflächen, also GUIs. Es finden sich mehrere Arbeiten, die sich mit diesen Bereich befassen. Zu dieser Thematik gibt es unter anderen die Ausarbeitung Ïntroducing Model-Based Testing in an Industrial Scrum Project"von Entin, Winder, Zhang, Christmann. In diesem Papier geht es zu einem darum, wie sich Model-based Testing sich im Bereich der agilen Software Entwicklung verhält, in diesem Beispiel Scrum. Dabei ging es auch unter anderem darum wie Modelle definiert werden und wie diese gewartet bzw. aktualisiert werden können. Es geht aber in diesem Papier auch darum, wie man mit Model-Based Testing die Benutzeroberfläche testen könne[11].

Auch in der Ausarbeitung Äutomated functionality testing through GUIs"von Nguyen, Strooper und Süß geht es um das Testen von der GUI im Kontext von Model-based Testing. Dabei beruht das Paper auf den Ansatz Model-based GUI Testing, kurz MGT. Dieser Ansatz modelliert auch die Events und die dazugehörigen Interaktionen [12]. Jedoch hätte dieser Ansatz nur limitierte Möglichkeit die abstrakten Aktionen mit der GUI abzubilden. Daher haben die Verfasser dieses Papers ein Framework entwickelt, welches dem Tester helfe die abstrakten Details der GUI in verhaltensorientierte Modelle darzustellen. Damit sei es dem Tester möglich zeitgleich die Spezifikationen, also die Anforderungen an das System und die Benutzeroberfläche zu testen [12].

Auch im Bereich im Testen von Android Apps gibt es viele aktuelle Arbeiten die sich mit diesem Themenbereich befassen. Besonders das Automatisieren von Testen ist ein Schwerpunkt, der in vielen Ausarbeitungen behandelt wird.

In dem Papier Ïmproving code coverage in android apps testing by exploiting patterns and automatic test case generation"von Amalfitano, Amatucci, Fasolino, Gentile, Mele, Nardone, Vittorini und Marrone geht es unter anderem darum ein Verfahren zu finden, welches die Testabdeckung besonders bei automatischen GUI-Prüfungen zu verbessern. Dabei wird in diesem Ansatz auch auf die Verfahren von Model-based Testing zurückgegriffen. Um die bessere Testabdeckung zu bewerkstelligen, sollen weitere Testfälle definiert werden. Diese Testfälle sollen dabei darauf beruhen, ein automatisiertes Verfahren zu verwenden, welches einen bestehenden GUI-Testansatz in Verbindung mit dem Ansatz pattern basedïn einem anderen Kontext verwendet[14].

Im Papier Äutomated Testing with Targeted Event Sequence Generation"von Jensen, Prasad und Møller geht es um die Automatisierung des Testen von Anwendungen auf Android. Es wird in diesem Papier ein Ansatz verfolgt, welcher sich von Model-based Testing unterscheiden soll. Die Verfasser von diesem Papier gehen davon aus, dass Anwendungen auf mobilen Geräten Event-basiert seien. Dahingehend stellen sie einen Ansatz vor, welcher Ereignissequenzen erstellt, um bestimmte Abschnitte im Code zu erreichen. Besonders wollen sie damit Bereiche im Source Code erreichen, die nur durch komplexe Ereignissequenzen erreicht würden können. Dies können angebliche bisherige Ansätze und Testverfahren nicht erreichen [15].

Von Griebe und Gruhn gibt es das Papier Ä Model-Based Approach to Test Automation for Context-Aware Mobile App-

lications". Im diesem Papier geht es darum, wie man zusätzlich die Kontextsensitivität, also das Verhalten in einem bestimmten Kontext, automatisiert Testen kann. Kontext bedeutet in diesem Zusammenhang z.B. welche GPS Position das Gerät hat. Es wird in diesem Papier ein Ansatz vorgestellt, welcher modellbasiert ist und die Testfälle aus sogenannten "design-time system modelsërstellt werden[16].

A. Konferenzen und Forschungsgruppen

Es gibt viele Konferenzen und Forschungsgruppen, die sich mit dem Testen besonders mit dem Model-based Testing auseinander setzen. Da Testen ein Unterthema von Software Engineering ist, ist im Prinzip jede Konferenz und Forschungsgruppe zu Software Engineering interessant, weil auch dort immer wieder Model-Based Testing ein Thema ist. Die wichtigsten Konferenzen und Forschungsgruppen sollen hier nun vorgestellt werden.

Eine der wichtigsten Konferenzen ist die jährlich stattfindende Ïnternational Conference on Software Engineering"kurz ICSE. Die ICSE findet jedes Jahr ungefähr im Frühling statt. Der Ort der Konferenz wechselt jährlich, wobei die Konferenz am Häufigsten in den USA stattfindet. Die nächste Konferenz findet vom 16. Mai - 24. Mai in Florenz in Italien statt. Zwar gibt gibt es keine Keynote zu Model-based Testing, aber trotzdem werden wichtige Aspekte zu Model-Based Testing bei der Konferenz besprochen[17].

Die Konferenz Test and Proofs, kurz TAP hat sich in den letzten Jahren immer wieder mit Model-based Testing unter den verschiedensten Aspekten beschäftigt. Die TAP findet jährlich an wechselnden Ort statt. Die nächste Konferenz findet vom 20. - 24. Juli in L'Aquila in Italien statt[18].

Auch Jährlich findet die Konferenz Ïnternational Symposium on Software Testing and Analysis", kurz ISSTA statt. Dieses Jahr findet die Konferenz in Baltimore in den USA vom 12. - 17. July statt. Zurzeit steht noch nicht fest, welche Themen bei der diesjährigen Konferenz behandelt werden[19].

Eine weitere wichtige Konferenz ist die DATE. DATE steht für Design, Automation and Test in Europe. Auch diese Konferenz findet jährlich statt. Das nächste mal findet sie vom 09. - 13. März in Grenoble in Frankreich statt[20].

Bei ACM gibt es die Forschungsgruppe SIGSOFT, welche sich unter anderem mit Model-based Testing beschäftigt. SIGSOFT beschäftigt sich generell mit dem kompletten Bereich des Software Engineerings. Da, wie schon geschrieben, das Testen ein Unterbereich von Software Engineering ist, setzt sich diese Forschungsgruppe auch mit Model-Based Testing auseinander[21].

In Deutschland gibt es noch zudem die Fachgruppe TAV. TAV steht dabei für Test, Analyse und Verifikation von Software. Die Fachgruppe TAV ist eine Einrichtung im Fachbereich der Softwaretechnik von der Gesellschaft für Informatik. Bei TAV werden mehrere Themen behandelt, die sich mit Testen auseinander setzen. Zudem gibt es eine eigene Arbeitsgruppe, die sich mit Model-based Testing beschäftigt[22].

Des Weiteren gibt es in Deutschland den Arbeitskreis Software-Qualität und -Fortbildung (ASQF). Es gibt innerhalb dieses Arbeitskreises eine Fachgruppe zum Thema Testen und zwar die Fachgruppe Software-Test. Diese Fachgruppe beschäftigt sich mit vielen verschiedenen Aspekten des Software Testens, aber auch mit Model-based Testing[23].

V. Ziele und Arbeitsplan fürs Grundprojekt

Für das im nächsten Semester stattfindende Grundprojekt stellen sich mehrere Aufgaben. Zunächst muss erst mal eine Software programmiert oder gefunden werden, die man mit Model-based Testing testen kann. Wie schon in der Motivation beschrieben, soll sich diese Software im Bereich der Software für mobile Geräte bewegen. Dabei würde es sich anbieten eine Kooperation mit HAWAI durchzuführen. HAWAI steht für HAW Labor für Anwenungsintegration. Im Rahmen der Software Engineering Vorlesung des Studiengangs Wirtschafts-Informatik sollen Anwendungen entwickelt werden, welche sich im Rahmen der HAWAI bewegen. Es werden dabei auch Anwendungen für mobile Geräte entwickelt.

Dies wäre eine gute Möglichkeit Model-Based Testing an einem praktischen Beispiel zu testen. Es würde sich nicht um irgendeine Beispiel Anwendung handeln, sondern um eine realistische Software. Der große Vorteil wäre dabei, dass man ausschließlich als Software-Tester agieren würde. Man wäre nicht direkt in die Implementierung mit eingebunden, was wiederum den Vorteil mit sich bringt, dass man einen Abstand zur Implementierung hat.

Dahingehend wäre der Arbeitsplan für das nächste Semester die von den Wirtschafts-Informatikern entwickelte Software für Android Anwendungen mit Model-based Testing zu testen. Dabei sollen mehrere Fragestellungen beantwortet werden.

Es stellt sich in diesem Zusammenhang vor allem erstmal die Frage wie verhält sich Model-based Testing in der Umgebung von der Entwicklung von Software für mobile Geräte. Ist diese Technik überhaupt ausreichend dafür oder ist es sogar zu viel des Guten und man schafft sich eigentlich mehr Arbeit.

Der Aspekt wird besonders noch interessanter, wenn Model-based Testing im Vergleich zum Android Testing Framework betrachtet wird. Bietet Model-based Testing die besseren Mechanismen, um Android Apps zu testen oder bietet Goolge mit seinem Test Framework die besseren Lösungen an.

Interessant ist auch die Fragestellung, wie gut ist die Testabdeckung der Android App durch Model-based Testing. Werden mit Model-Based Testing alle Bereiche abgedeckt? Wie sieht es überhaupt in diesem Zusammenhang mit der GUI aus? Sind die zur Verfügung gestellten Mechanismen überhaupt ausreichend, um die GUI zu testen? Dazu wäre es auch interessant, ob man mit Model-based Testing die Kontextsensitivität testen kann.

Des Weiteren kann man sich die Erstellung der Modelle genauer anschauen. In diesem Bereich gibt es viele interessante Fragestellungen, wie z.B. wie aufwendig ist die Erstellung der Modelle. Dazu kommt noch die Frage, ob mit den Modellen wirklich die gesamte Software abgedeckt werden kann? Ergänzend dazu ist auch die Fragestellung interessant, ob und wie nicht-funktionale Anforderungen in den Modellen abgebildet werden können.

Am Ende des nächsten Semesters sollen diese Fragen beantwortet sein. Des Weiteren soll auch als Ergebnis feststehen, ob Model-based Testing überhaupt geeignet für das Testen von Android Apps ist oder ob es ggf. erweitert werden muss. Damit man Android Apps wirklich sinnvoll mit Model-based Testing testen kann.

A. Abgrenzung der eigenen Arbeit

Wie im vorigen Kapitel schon beschrieben, würde schon einige Bereiche bei Model-based Testing erforscht oder werden gerade erforscht. Dabei liegt der momentane Forschungsschwerpunkt auf den Modellen, besonders auf eine effizientere Erstellung der Modelle. Ein weiterer Schwerpunkt bei der Forschung von Model-Based testing liegt auf dem automatischen Erstellen von Testfällen zum Testen der GUI.

Die geplante Forschung meinerseits legt den Schwerpunkt aber darauf, wie geeignet Model-Based Testing für Android Apps ist. Es gibt im Bereich der Forschung zum Testen von Android Apps Ansätze zum automatisierten Testen. Jedoch beschäftigt sich keiner dieser Ansätze ausschließlich mit Model-Based testing. Teilweise wird ein bestimmter Teilbereich von Model-based Testing verwendet, aber es wird nie ausschließlich Model-based Testing verwendet. Daher grenzt sich diese Forschungsarbeit schon von den bisherigen Forschungsarbeiten ab. Es gibt bislang keine Forschungsarbeiten dazu, wie gut Model-Based testing für mobile Anwendung besonders für Android Anwendungen geeignet ist. Es gibt zwar genug Forschungsarbeiten wie generell Model-based Testing geeignet ist, jedoch gibt es keine in Verbindung mit mobilen Anwendungen.

Dies betrifft auch andere Forschungsaspekte. Zwar gibt es schon Arbeiten dazu, wie aufwendig und kompliziert die Modell Erstellung ist und wie man dies vereinfachen könnte, aber im Bereich der Entwicklung von Apps gibt es dazu keine Forschungsarbeit zu. Da die Entwicklung von Apps sich unter anderem dadurch auszeichnet, dass sie kurze Entwicklungszyklen haben, wird dieser Punkt durchaus wieder interessant.

Beim Testen von mobilen Anwendungen spielt die GUI eine wichtige Rolle. Dahingehend wird im nächsten Semester auch behandelt, wie gut man die GUI mit Model-Based Testing testen kann. Es wird in diesem Bereich auf jeden Fall Berührungspunkte zur bisherigen Forschungsarbeiten geben. Jedoch liegt mein Augenmerk mehr auf dem Gesamtbereich und nicht auf einen einzelnen Aspekt. Dazu ergänzend soll das Model-based Testing eher im Vergleich zum Android Testing Framework stehen. Dieser Aspekt wurde bisher nicht erforscht.

LITERATUR

[1] Monalisa Sarm, P. V. R. Murthy, Sylvia Jell, Andreas Ulrich. *Model-based testing in industry: a case study with two MBT tools*. Proceedings of the 5th Workshop on Automation of Software Test, Pages 87-90, May 3-4, 2010, Cape Town, South Africa

[2] Christop Schulze, Dharmalingam Ganesan, Mikael Lindvall, Rance Cleaveland, Daniel Goldman. *Assessing model-based testing: an empirical study conducted in industry*. Companion Proceedings of the 36th International Conference on Software Engineering, Pages 135-144, May 31 – June 7, 2014, Hyderabad, India

[3] Mark Blackburn, Robert Busser, Aaron Nauman *Why Model-Based Test Automation is Different and What You Should Know to Get Started*. PSQT 2004 East, Washington D.C., USA.

[4] Mark Timmer, Ed Brinksma, and Mariëlle Stoelinga *Model-based testing*. In: Software and Systems Safety: Specification and Verification. NATO Science for Peace and Security Series - D: Information and Communication Security. IOS Press, Amsterdam (2011)

[5] A. Pretschner , W. Prenninger , S. Wagner , C. Kühnel , M. Baumgartner , B. Sostawa , R. Zölch , T. Stauner *One evaluation of model-based testing and its automation*. Proceedings of the 27th international conference on Software engineering, May 15-21, 2005, St. Louis, MO, USA

[6] F. Bouquet , C. Grandpierre , B. Legeard , F. Peureux , N. Vacelet , M. Utting *A subset of precise UML for model-based testing*. Proceedings of the 3rd international workshop on Advances in model-based testing, p.95-104, July 09-12, 2007, London, United Kingdom

[7] Mark Utting , Bruno Legeard *Practical Model-Based Testing: A Tools Approach*. Elsevier Science/Morgan und Kaufmann, 2007, ISBN 0-12-372501-1.

[8] Chris Stoecker *Neues iPhone: Apple blamiert sich mit Karten-Desaster*. www.spiegel.de/netzwelt/gadgets/apple-maps-die-schoensten-schnitzer-des-ios-kartendienstes-a-857229.html, 21.09.2012, Zugriff am 24.02.2015

[9] *Testing Fundamentals*. developer.android.com/tools/testing/testing_android.html, Zugriff am 24.02.2015

[10] Marc-Florian Wendland, Andreas Hoffmann, Ina Schieferdecker *Fokus!MBT - a multi-paradigmatic test modeling environment*. Proceedings of the workshop on ACadeMics Tooling with Eclipse (ACME 2013), ACME'13, Montpellier, France, ACM

[11] Vladimir Entin, Mathias Winder, Bo Zhang, Stephan Christmann *Introducing Model-Based Testing in an Industrial Scrum Project*. Proceedings of the 7th International Workshop on Automation of Software Test, Pages 43-49

[12] Duc Hoai Nguyen, Paul Strooper, Jörn Guy Süß *Automated Functionality Testing through GUIs* . Proceedings of the Thirty-Third Australasian Conferenc on Computer Science, p.153-162, January 01-01, 2010, Brisbane, Australia

[13] Duc Hoai Nguyen, Paul Strooper, Jörn Guy Süß *Model-based testing of multiple GUI variants using the GUI test generator.* . Proceedings of the 5th workshop on automation of software test, AST '10 (pp.24–30). New York, NY, USA: ACM

[14] D. Amalfitano, N. Amatucci, A. Fasolino, U. Gentile, G. Mele, R. Nardone, V. Vittorini, S. Marrone *Improving code coverage in android apps testing by exploiting patterns and automatic test case generation.* . Proceedings of the 5th workshop on automation of software test, AST '10 (pp.24–30). New York, NY, USA: ACM

[15] Casper S. Jensen , Mukul R. Prasad , Anders Møller *Automated testing with targeted event sequence generation.* . Proceedings of the 2013 International Symposium on Software Testing and Analysis, July 15-20, 2013, Lugano, Switzerland

[16] Tobias Griebe, Volker Gruhn *Automated testing with targeted event sequence generation.* . Proceedings of the 29th Annual ACM Symposium on Applied Computing, March 24-28, 2014, Gyeongju, Korea.

[17] *The 37th International Conference on Software Engineering* . http://2015.icse-conferences.org/, Zugriff am 23.02.2015

[18] *9th International Conference on Tests and Proofs* . http://tap2015.in.tum.de/call.shtml, Zugriff am 23.02.2015

[19] *International Symposium on Software Testing and Analysis* . http://issta2015.cs.uoregon.edu/, Zugriff am 23.02.2015

[20] *Design, Automation and Test in Europe* . http://www.date-conference.com/, Zugriff am 23.02.2015

[21] *SIGSOFT* . http://www.sigsoft.org/, Zugriff am 23.02.2015

[22] *Fachgruppe Test, Analyse und Verifikation* . https://fg-tav.gi.de/, Zugriff am 23.02.2015

[23] *Arbeitskreis Software-Qualität und -Fortbildung* . https://www.asqf.de/, Zugriff am 23.02.2015