Test Cases Evolution of Mobile Applications

Lynda Ait Oubelli, Jean-Marie Mottu, Christian Attiogbé

Test Cases Evolution of Mobile Applications

Model Driven Approach

Author :

LYNDA AIT OUBELLI

Supervisors :

DOCTOR. JEAN-MARIE MOTTU
PROFESSOR. CHRISTIAN ATTIOGBE

A dissertation submitted to the University of Nantes in accordance with the requirements of the degree of MASTERS COMPUTER SCIENCE -ALMA- in the Department of Sciences and Technology.

AELOS Team LINA Laboratory
HTTP://GOO.GL/BKS7BJ

2 JULY 2015

Abstract

Mobile Applications Developers, with large freedom given to them, focus on satisfying market requirements and on pleasing consumer's desires. They are forced to be creative and productive in a short period of time. As a result, billions of powerful mobile applications are displayed every day. Therefore, every mobile application needs to continually change and make an incremental evolution in order to survive and preserve its ranking among the top applications in the market. Mobile apps Testers hold a heavy responsibility on their shoulders, the intrinsic nature of agile swift change of mobile apps pushes them to be meticulous, to be aware that things can be different at any time, and to be prepared for unpredicted crashes. Therefore, starting the generation or the creation of test cases from scratch and selecting each time the overridden or the overloaded test cases is a tedious operation. In software testing the time allocated for testing and correcting defects is important for every software development (regularly half the time). This time can be reduced by the introduction of tools and the adoption of new testing methods. In the field of mobile development, new concerns should be taken into account; among the most important ones are the heterogeneity of execution environments and the fragmentation of terminals which have different impacts on the functionality, performance, and connectivity. This project studies the evolution of mobile applications and its impact on the evolution of test cases from their creation until their expiration stage. A detailed case study of a native open source Android application is provided; describing many aspects of design, development, testing in addition to the analysis of the process of mobile apps evolution. This project based on model driven engineering approach where the models are serialized using the standard XMI. It presents a protocol for the adaptation of test cases under certain restrictions.

Keywords. Android mobile applications(apps), Software evolution, Test cases adaptation, Model-Driven Engineering, EMF Compare Framework.

Dedication and acknowledgements

First of all, I would like to say thank you to the Great Allah for being with me all this period, for giving me the courage to be here, and the endurance for finishing this year. I would like to thank my supervisors Doctor Jean Marie-Mottu and Professor Christian Attiogbe who continuously steered me in the right direction. I deeply appreciate their involvement and dedication. This work could not have been finished without their guidance and comments. Doctor Mottu who taught me how to be patient in research, and push me to give better results. Professor Christian with his kindness, wisdom, I really enjoyed the work with both of you. I want to say thank you to Mr.Cedric Guinoiseau and Mr.Damien Villeneuve from BeApp company for their time even when they were busy, the discussions and the meetings have great benefits on the project. I want to say thank you to the jury members for reading and evaluating my work. I want to thank all Aelos Team members for their comments and for giving me the chance to be a part of them along my trainee ship period. I would like to thank the responsible of Master ALMA Professor Mourad CHABANE OUSSALAH for his warm welcome from the first day at Nantes University, for his advises, and for taking a full attention in resolving ALMA student's issues. Thank you to Madame Hala Skaf for her great organization of our train ships project from the start until the end. Special thanks goes to Professor Abdelkrim AMIRAT from Souk Ahras University for his great support and for sacrificing his time to help me and encouraging me during my adventure by his advises even when he was too busy. Another thank you goes to Doctor Imed BOUCHRIKA, who makes me love research from the first work with him, it were a great honor to work with you Doctor. Thanks to my Parents ABDELAZIZ and FARIDA MOUHOUBI for giving me the chance to be here their endurance and encouragement during the good and bad days of this year. I would also wish to thank my brothers(NacerEddine and Salim) and sisters (Souhaila and Rahima) for their support during my career. Finally, I wish to thank all my uncles,friends, cousins for their kind support and I would like to dedicate this dissertation to Samir, Ali, R.Boujemaa, Touati Waheb, Tata Farida Boukhors, Zenir Hichem, Racha, Rahil, Waheb, Malika, Chahrazed, Lamine, Achref, Souad, Fouzi, Walid, Hala, Anas, Ayoub, Mohamed Amine Aouadhi,Narjes,Imene, Yosra, Moncif, Alexis Linard, Alexis Guilbaud, Yoann Vernageau, Kevin. Pierre Cyrille, yannis grego, Amel, My sister Samira, and my best freind Nora.

Author's declaration

I declare that this project and the work presented in it are my own, I confirm that: this work was done wholly or mainly while in candidature for a Master degree at University of Nantes, France; where no part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;where I have consulted the published work of others, this is always clearly attributed; where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work; I have acknowledged all main sources of help; where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;

SIGNED: Lynda AIT OUBELLI. DATE: 25/06/2015

TABLE OF CONTENTS

List of Tables

List of Figures

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Context

Mobile applications are becoming an integral part of our lives. Not long ago, many people used to feel uncomfortable without a cell phone, but nowadays most people feel uncomfortable without a smart phone, a survey says that 47% of adults couldn't last a day without smart phone, and this is mainly due to mobile applications. A great revolution in the world of mobile industry [26] . That has evolved very rapidly since its beginning, from simple tiny pieces of calendars and calculators to a huge growing number of modern applications that fulfill our everyday tasks such as shopping, business, banking, diary planning and social networking. This leads consumers to ask many questions what kind of mobile application do we need in our lives? How can we choose them? How can we decide whether to keep using or to throw an application? These questions and many others lead developers to investigate and ask what are basic challenges of developing a mobile application? How can we keep users away from uninstalling our apps? How to improve the quality of our apps? How to adapt our apps with their ecosystems during their evolution? What is a good or bad mobile application? Generally software evolution is the dynamic behavior of programming systems as they are maintained and enhanced over their life time.Lehman et al. [39], defined the phenomenon, but what is the specificity of mobile application evoution during its life cycle? When can we say that this application has evolved? Lehman et al. [40] also proposed laws for software evolution but are they applicable on mobile apps? For the best of our knowledge we say that mobile application evolution is a software evolution but with more challenges in improving quality. Because mobile app evolution takes into consideration during its life cycle many constraints such as ecosystem conditions, general context, diversity of devices...etc

1.2 Motivation

In order to gain insights into the evolution of mobile application, Minelli et al. [50], investigated either Lehman's laws that were proposed for traditional systems are available for mobile apps and they proved that mobile apps accept the law of continuing change to be applied. Second law of the increase in complexity is also accepted despite the quality declining law; the authors recommend further study to be done. And a question mark is still put; testing is usually performed to improve quality, doing this task during mobile apps evolution manually or automatically is a tedious and error prone operation even in the case of application developed individually or by a team of developers in the same area or in different ones. Change management systems are used for different reasons such as storing changes of evolving software systems, sharing source code with colleagues or backing up source code. Using change management systems for extracting that kind of information may leads to different problems such as degraded information, information loss or unordered information. During mobile apps development this has an impact on test cases, and force the tester to generate test cases from scratch for each new change. It takes a lot of time and will be an obstacle of putting the application out the door because we are not sure of the quality criteria which leads to break the confidence of the relation consumers/products and throw the application after five minutes of its installation.

1.3 Contribution

While most people think of new techniques for automating the generation of test cases for mobile applications, we tackle the adaptation of ancient test cases through the evolution of their systems. For example A.Memon [47] studied the problem of repairing regression test cases at the GUI level, M.Mirzaaghaei et al. [51] presented TCA test care assistant a framework based on eight algorithms of repairing test cases according the evolution of their systems in the code level.

By proposing, for the first time, a model driven approach for test cases evolution that are generated by a black box testing technique where everything is considered as a model and evolutionary informations are extracted from the analysis of the comparison between two or more developed versions of a mobile application, changes can influence on the interface or the code of such application or both of them in the same time any change can influence on the generated test cases. The advantage of our proposition is that we study simultaneously the evolution of test cases in both GUI and Code of android mobile application and in both compile and run time. Inspiring by Bart's thesis [22] where types of changes are derived from the specification of such a meta-model. We use a comparison model in analyzing mobile apps syntactic evolution.

1.4 Manuscript Organization

This dissertation starts with a presentation of the context of the project, motivations and main contributions after that an overview of several approaches for studying mobile applications and their testing manually or automatically, from their GUIs or codes, benefits and shortcomings regarding their usability, all these aspects are discussed in detail in Chapter 2. Chapter 3 seeks software evolution generally and mobile app and their test cases evolution specifically. Furthermore, Chapter 3 takes a closer look at the related works to our proposal. Next, in Chapter 4 we discuss the theoretical sound of the model driven approach of test cases evolution, architecture of the proposal, technique environment,... etc. Chapter 5 presents case study of our proposal (android mobile apps, kind of changes on the original application, the comparison pattern used in extracting the equivalences and differences between different versions, the generation of test suite from the original version and the definition of a protocol for adaptation of those test cases). We conclude by summarizing our work and outlining future works in Chapter 6.

1.5 Presentation of the laboratory/Team

My Train ship took place in LINA laboratory within Aelos (Architectures et Logiciels Surs) team. LINA result of the meeting in 2004 of the Institute for Research in Computer Science of the University of Nantes (IRIN) and the IT (Information Technology) computer science department of the School of Mines of Nantes, France. Initially FRE (Research Training Evolution), in January 2008 the LINA become l 'UMR 6241with three guardianships: the University of Nantes, School of Mines of Nantes and CNRS. It is attached to the Institute of Computer Sciences and their Interactions (INS2I) of CNRS which covers the 5 major themes of responsibility, either in 6th section for themes "data science", "constraints and optimization" , "distributed systems and software", in 7th Section for the theme "multilingual resources and applications." Since January 2013, INRIA is a partner of LINA. The laboratory is directed by Pierre Cointe. By the end of 2013, LINA housed nearly 180 people on two remote sites in a dozen kilometers, that of Lombarderie and that of the Chantrerie, the Lombarderie site houses a large half of the laboratory and its media services hosted by the Faculty of Science, the site of the Chantrerie meets the other half within two engineering schools: The School of Mines of Nantes and Polytech of Nantes. Today, LINA plays a central role in the development of IT in the area of Loire contries the laboratory is associated with IRCCyN (Research Institute in Communications and Cybernetics of Nantes) within the research federation AtlanSTIC (FR 2819). IRCCyN is the other regional UMR attached, inter alia, to the INS2I. LINA is also associated with the graduate school "Sciences and Technologies of Information and Mathematics" (ED-STEM) of PRES of Nantes university, Angers, Mans (L'UNAM) and regularly collaborates with two other laboratories of the federation LERIA((Laboratory of study and research in computer science of Angers University) in the in the field of optimization and constraints and LIUM (Laboratory of Informatics of the University of Maine) in the field of

speech processing. Next to the federation, LINA is an engine in the RFI process (Research-Training-Innovation) of loire countries Area, and participate in building a "digital RFI" which, beyond Nantes to involve cities such Angers, Le Mans and Laval and will dovetail with ongoing developments in Britain Finally, the LINA carries the griot regional bioinformatics project in conjunction with Biogenouest. The laboratory is also heavily involved in multiple inter-regional collaborations with Brittany, It is the Nantes laboratory involved in the selection committee and validation of cluster international projects "Images and Networks" transverse to both fields Brittany and Loire countries. Since its origin it also shares with the Centre INRIA Rennes Atlantic Britain project teams, Historically ATLAS and OBASCO in 2002 today ASCOLA and AtlanMod for the theme " software and distributed systems " and TASC to the theme "constraints and optimization." LINA is also involved in the Excellence Laboratory CominLabs with 9 other laboratories or research centers in Britain. It contributes significantly to the challenges "and distributed software systems," "constraints and optimization" and "multilingual resources and applications" with three projects currently underway (SecCloud for safe programming of the cloud, EPOC, for an efficient cloud energy, DESCENT to the decentralization of social networks) and a brand new project in early 2014, Limah, which deals with the discovery of similar language contained in multimodal documentations and opinion mining in multimedia collections. He finally participates in the training component with Labex including CominOpenCourseware project (COCo) on the provision of educational resources in the form of increased videos [42].

CHAPTER 2

STATE OF THE ART: MOBILES APPLICATIONS AND THEIR TESTING

2.1 Introduction

Mobile apps deployment is time consuming and it involves huge amount of expenses. yet critical to ensure that consumers have a positive experience when they use mobile applications. There are trade offs that need to be considered and choices that need to be made regarding the mix of different techniques and methods that will be used in mobile apps testing. In this chapter we present the state of the art of mobile apps and their testing paradigms. In the first section we talk about types of mobile apps, their life cycle, we give some statistics about their usability. In the second section we present an overview of the existing approaches for testing mobile apps from manual testing, Cloud testing, and the paradigms of automation testing of mobile apps.

2.2 Definition of Mobile Applications

mobile application is a software application designed to run on Smart phone, tablet, computers and other mobile devices and/or taking in input contextual information [17]. In mobile computing an application is considered to be mobile if it runs on an electronic device that may move (e.g., mp3 readers, digital camera, mobile phones). Satyanarayanan et al. [68] synthetically captures the uniqueness and conceptual difference of mobile computing through four constraints: Limited resources, security and vulnerability, performance and reliability variability, and finite energy source. Henry Mucciniin et al. [55] defines from a testing perspective an app four Mobile as an application that, driven by user inputs, runs on a mobile device with limited resources.

2.2.1 Types of Mobile applications

In the mobile technology , we hear often terms like native apps or Web apps, or even hybrid apps. In the following subsection we try to present similarities and differences between them.

Native apps

Native apps are installed through an application store (such as Google Play or Apple's app Store) on the device and are accessed through icons on the device home screen. They are developed specifically for one platform, and can take full advantage of all the device features, they can use the camera, the GPS, the accelerometer, the compass, the list of contacts, and so on. They can also incorporate gestures (either standard operating system gestures or new, app-defined gestures). Native apps can use the device's notification system and can work offline [13].

Mobile Web apps

Web apps are not real applications; they are really websites that, in many ways, look and feel like native applications, but are not implemented as such. We can run them by a browser and they are typically written in HTML5. Users first access them as they would access any web page: they navigate to a special URL and then have the option of installing them on their home screen by creating a bookmark to that page. Web apps became really popular when HTML5 came around and people realized that they can obtain native-like functionality in the browser. Today, as more and more sites use HTML5, the distinction between web apps and regular web pages has become blurry [13].

Hybrid apps

Hybrid apps are a part native apps and a part web apps (because of that, many people incorrectly call them web apps). Like native apps, they live in an app store and can take advantage of the many device features available. Like web apps, they rely on HTML being rendered in a browser, with the caveat that the browser is embedded within the app. Often, companies build hybrid apps as wrappers for an existing web page; in that way, they hope to get a presence in the app Store, without spending significant effort for developing a different app. Hybrid apps are also popular because they allow cross-platform development and thus significantly reduce development costs: that is, the same HTML code components can be reused on different mobile operating systems. Tools such as PhoneGap and Sencha Touch allow people to design and code across platforms, using the power of HTML.In a nutshell native and hybrid apps are installed from an app store, whereas web apps are mobile-optimized web pages that look like an app. Both hybrid and web apps render HTML web pages, but hybrid apps use app-embedded browsers to do that [13].

2.2.2 Mobile Application Life cycle Management

Mobile Application Life cycle Management has an aim to deliver the end product with higher quality. Hence, it has to repeat the steps in this process.

Planning and Designing

In order to take future decisions, it is important to consider the present and the future uses

of the applications. Due to large market of devices and hardware and software solutions, the decision making process becomes more difficult. It is also necessary to look into frequent changes and updates in the applications so as to make sure its quick deployment.To enhance the user experience, offline functionality must be given an important consideration.

Development and Testing

The important aspects of development stage are device detection, limited bandwidth and memory management. Offline data should be available and it should be updated. The next challenging task is to test the mobile applications. It involves huge amount of expenses. It is necessary to test the applications in various situations at different bandwidths. Pre-flight test, beta testing, scalability test, device test, automated unit test, etc. are different tests undertaken for various reasons.

Deployment

Deployment of applications is an easier task as they are delivered to the application stores. In-house applications are available in separate application stores. If the company applications are to be given to customers or distributors, it is even more difficult.

Updating

Dynamic updating of the applications is important. The users have to install the new application first. But some studies show that the users continue with the older versions of applications [36].

2.2.3 Statistics

The changes indicate that the mobile browser has become just a single application swimming in a sea of apps. Figure 2.1 examines which app categories remained most popular year-over-year.

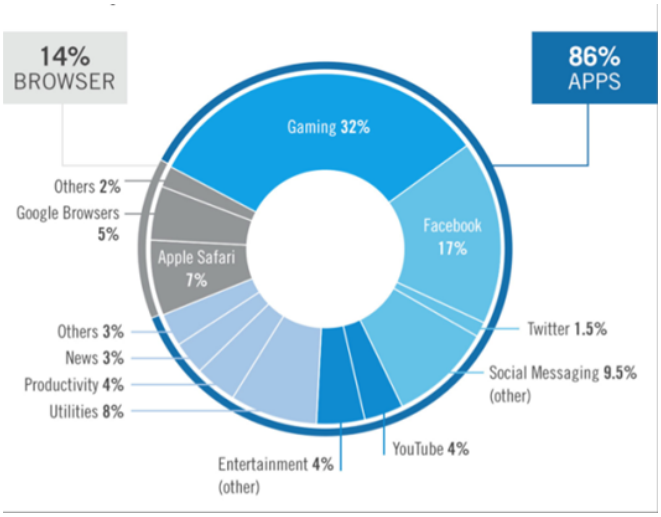

FIGURE 2.1. Statistics of most mobile applications. Figure reproduced from [60].

It shows that gaming still dominates mobile usage with 32% of time spent on iOS and Android

devices (same as last year). While Facebook remained a strong second with 17% of time spent on mobile. However, it's interesting to note that Facebook exhibited a slight decline year-over-year, going from 18% of time spent in 2013 to the 17% [60].

2.3 Testing mobile applications

2.3.1 Definition of testing

Testing is usually performed for the following purposes: (1) to improve quality, (2) for verification and validation, and (3) for reliability estimation. All of these points refer to the conformance of the specified design requirements. Being correct, the minimum requirement of quality, means performing as required under specified circumstances. Debugging, a narrow view of software testing, is intensively performed to discover design defects by the programmer. The imperfection of human nature makes it almost impossible to make a moderately complex program correct the first time around. Another important purpose of testing is verification and validation. Within the process of verification and validation, testing can serve as a metric. Software reliability has an important connection with many aspects of the software, including structure and the amount of testing it has been subjected to. Testing can also serve as a statistical sampling method to gain failure data for reliability estimation [58]. Testing is another important area for mobile software engineering research. For mobile devices it's insufficient to merely test an Android application on an emulator; it must be tested across many different Android devices running on different versions of the operating system over various telecom networks [74].

2.3.2 Layers of software testing

The verification and validation of requirements are a critical part of systems and software engineering. The importance of verification and validation is a major reason that the traditional development cycle underwent a minor modification to create the V model; that links early development activities to their corresponding later testing activities, quality engineers, and other stakeholders interested in the use of testing as a verification and validation method. Figure 2.2 shows the tester's single V model, which is oriented around these work products rather than the activities that produce them. In this case, the left side of the V model illustrates the analysis of ever more detailed executable models, whereas the right side illustrates the corresponding incremental and iterative synthesis of the actual system. Thus, this V model shows the executable work products that are tested rather than the general system engineering activities that generate them [29].

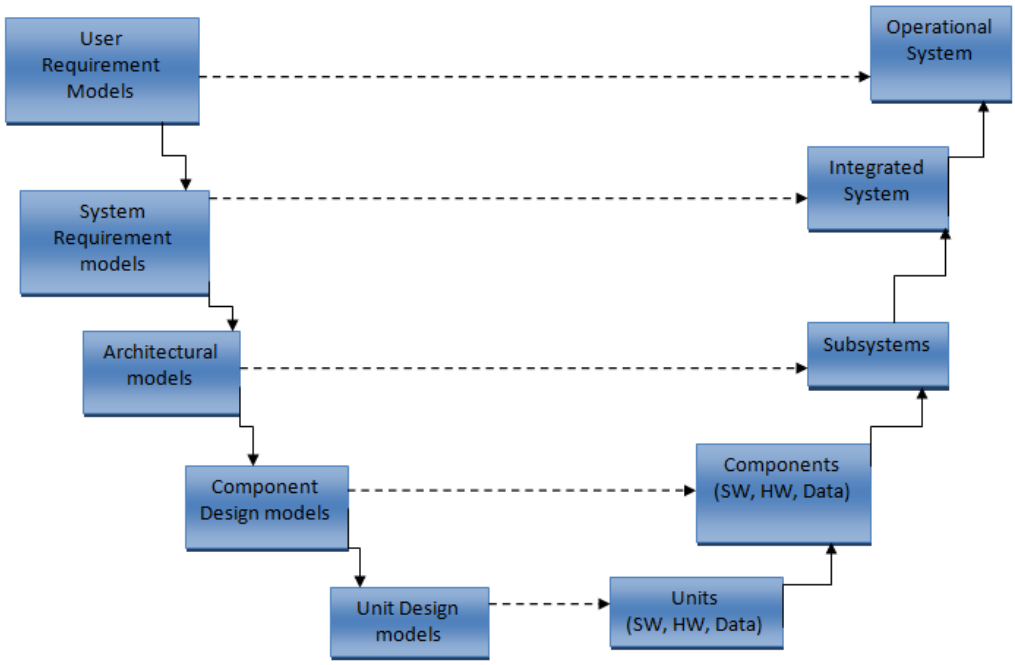

FIGURE 2.2. Single V-Model.Figure reproduced from [29].

2.3.3 Key challenges of mobile applications testing

The movement towards mobile devices has brought a whole different set of challenges to the testing world. Mobile apps and websites need to be rock solid before they are released to the market [31]. Six key challenges that apps developers and testers are facing are: (1) screen sizes : Mobile devices differ in screen sizes, input methods (QWERTY, touch, normal) with different hardware capabilities, (2) Connection types: Here are several standards for mobile data connections (3G, 4G,) as well as for Wifi. Sometimes there might be no connection available at all or the device is in flight mode, (3) Different OS versions: Diversity in Mobile platforms /OS- There are different Mobile Operating Systems in the market. The major ones are Android, iOS, BREW, BREWMP, Symbian, Windows Phone, and BlackBerry (RIM). Each operating system has its own limitations. Testing a single application across multiple devices running on the same platform and every platform poses a unique challenge for testers,(4) Power consumption and battery life: When testing mobile apps we need to make sure that the power consumption is kept minimal and the app is developed by keeping the best practices in mind, (5) Usability: It's challenging to keep the interaction clean and simple for the user, and at the same time display all the necessary information, (6) Internationalization : Testers should also take into account regional traits (locale settings, time zones) and target audience. Changing time while app is running might cause some interesting artifacts [77]. Table 2.1 highlights the distinctive characteristics and their implications on Testing [38].

Mobile apps Distinctive Characteristics	**Implications on Testing**
Connectivity	Functional, Performance, Security, Reliability testing through different networks
Convenience	GUI testing
Supported Device(diversity of physical device and OSs)	Test matrix based on Diversity Coverage testing
Touch Screens	Usability and Performance testing
New Programming Languages	White box and Black box testing, Byte-code analysis
Resource constraints	Functional and Performance monitoring testing
Context awareness	Context dependent Functional testing

Table 2.1: Mobile apps characteristics v Implications on Testing. Table reproduced from [38].

2.3.4 Test driven development (TDD)

In model driven development testing as soon as possible reduces the cost of the verification and validation (V&V) process [7]. Besides testing mobile apps software is a disciplined process that consists of evaluating the application (including its components) behavior, performance, and robustness. These V&V activities can be sub-classified as preventative, detective, or corrective measures of quality. Errors in conception and implementation of mobile applications controllers have caused dramatic problems especially in the area of space exploration and applications. On the other hand, the opportunities offered by technology for Mobile applications controllers are immense. Model-Driven Testing Techniques (MDT) is a part of software engineering based on model transformation principle. This implies increasing research on automation of the testing processes. Haeng Kon Kim et al. [37] designed and implemented the automation of testing parallel to system development. As shown in Figure 2.3, a platform independent system design models (PIM) can be transformed into platform specific test models (PIT). While PIMs focus on describing the pure functioning of a system independently from potential platforms that may be used to realize and execute the system, the relating PITs contain the corresponding information about the test. In another transformation step, test code may be derived from the PIT. Certainly, the completeness of the code depends on the completeness of the system design model and test model.

2.3.5 Testing mobile applications in the Cloud

Cloud based testing strategy enables a distributed, cost effective way of testing mobile applications on a multitude of devices B. Kirubakaran et al. [38], present two orthogonal aspects which exacerbate the need of automation in testing mobile applications:

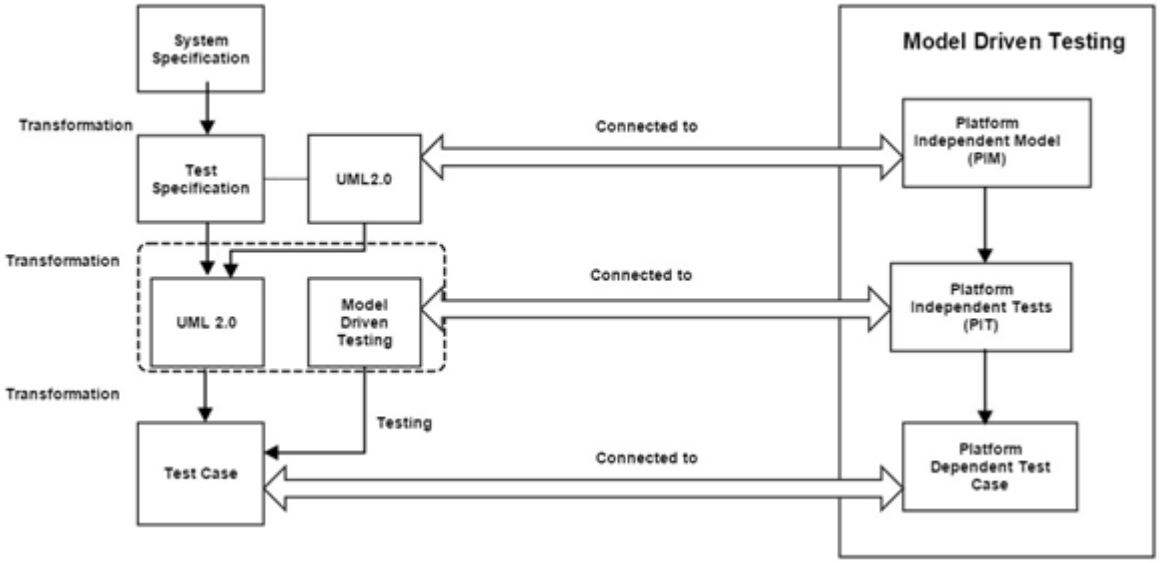

Figure 2.3: Mapping between test specification and MDT [37].

I. Cost of testing: Automation is certainly among the most important means for keeping the cost of testing low, while guaranteeing and adequate degree of dependability.

II. Testing through the layers: This remark highlights the need of testing automation towards all the different layers, and able to clearly separate application-level failures from application framework or OS failures. Perspective solution that may enable cost-effective testing of mobile applications include outsourcing, Cloud based testing solution where companies will start offering as-a- service software testing services (TaaS) [74].

2.3.5.1 Testing as a Service (TaaS)

Testing as a Service (TaaS) is an outsourcing model in which testing activities associated with some of an organization's business activities are performed by a service provider rather than employees. TaaS may involve engaging consultants to help and advice employees or simply outsourcing an area of testing to a service provider. Usually, a company will still do some testing in-house. TaaS is most suitable for specialized testing efforts that don't require a lot of in-depth knowledge of the design or the system. Services that are well-suited for the TaaS model includes automated regression testing, performance testing, security testing, testing of major ERP (enterprise resource planning) software, and monitoring testing of Cloud-based applications. TaaS is also sometimes known as on-demand testing. www.keynotedeviceanywhere.com, for example, has already launched a service to verify mobile applications on a variety of devices [67].

2.3.6 Testing mobile apps in Emulators, Simulators, real mobile device

Mobile emulators and simulators can help developers to test their apps on different devices without the need to actually have them on hand. The testing tools can make testing easier, particularly where there are lots of variations in device types, screen sizes and operating systems. But they also face some limitations. The terms emulator and simulator are often used interchangeably. An emulator is an application that emulates real mobile device software, hardware and operating systems, allowing us to test and debug our applications. It is usually provided by device manufacturer, emulators are written in machine level assembly languages, they are more suitable for debugging. Often an emulator comes as a complete re-implementation of the original software. eg: Android (SDK) Emulator . A simulator is a less complex application that simulates internal behavior of a device, but does not emulate hardware and does not work over the real operating system. It may be created by the device manufacturer or by some other company. They are written in high level languages, they can be difficult for debugging purpose, simulator is just a partial re-implementation of the original software eg: iOS simulator [54].

2.3.7 Paradigms of mobile applications testing

Today a relevant family of techniques and tools for automated testing of mobile apps focus on their GUI to find bugs. They were recently classified into Random testing, Model-based testing, and Model-learning testing techniques.

2.3.7.1 Random Testing technique (RT)

The Testing of mobile applications in industry is still far from fully automatic. The testing engineers generally use two approaches to automate their time consuming work. The first one is the capture and replay tool with which the testers can automatically record test scripts by manipulating the mobile application under test. Then the testers can replay the recorded test scripts repeatedly afterwards. The problem with record-replay technique is that the quality of the generated test scripts depends on the testers' familiarity and understanding of the application under test, which can fluctuate greatly across different testers and applications. For example, there is a built-in Monkey application within the Android mobile OS. Testers can ask the Monkey to send random event sequences targeted at a specific application. However, a random testing, although fully automatic, may not be effective for detecting a fault. Researchers have proposed cleverly designed random testing technique to improve its effectiveness. Adaptive Random Testing is one such cleverly designed random testing technique for test case generation. Based on the observation, the input failure regions of an application tend to cluster together; it tries to spread the randomly generated test cases as evenly as possible. Liu et al. [43] presented Adaptive Random Testing (ART). Where they used much less test cases than random to detect the first fault. This shows that evenly spread the event sequences can increase the fault detection ability

of the generated test cases. Although ART can reduce the number of test cases required to detect first fault, it takes more time than random technique to generate the test cases.

2.3.7.2 Model Based Testing (MBT) for mobile apps

A set of techniques and tools to automate the creation of test cases based on a model of a system. Model typically is a UML representation or a finite state machine that describes the behavior of a system (or part of a system). And it is a description of a system that helps us understand and predict its behavior [70].

A Graphical User Interface (GUI) is a hierarchical, graphical front-end to a software system that accepts as input user-generated and system-generated events, from a fixed set of events and produces deterministic graphical output. A GUI contains graphical widgets; each widget has fixed set of properties. At any time during the execution of the GUI, these properties have discrete values, the set of which constitutes the state of the GUI [46]. Figure 2.4 shows that a GUI is made up of interfaces linked to each other by a Transition relationship. Each interface is characterized by the Activity instance that is responsible for drawing it and is composed by a set of Widgets. A Widget is defined as a visual item of the Interface. It can be implemented in the Android framework by an instance of a view class, a dialog class or a menu Item class [3].

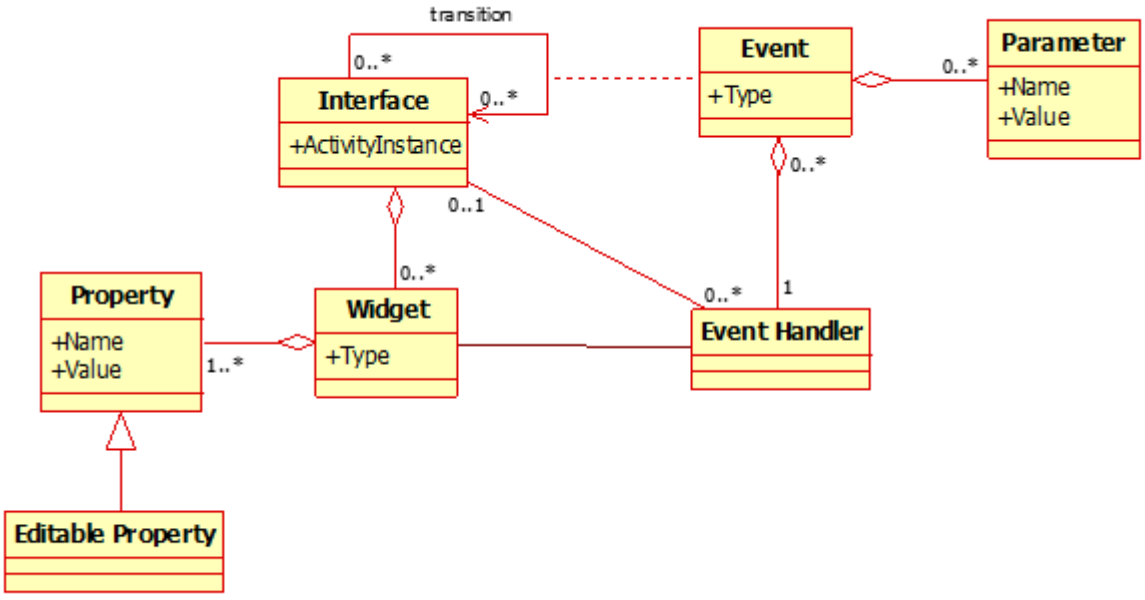

Figure 2.4: Conceptual Model of an Android Application. Figure reproduced from [3].

A.Memon et al. [10] presented an automated model-based testing technique based on reverse engineering that generates test cases to execute directly on the software's GUI. After that

D.Amalfitano et al. [3] presented AndroidRipper, an automated technique implemented in a tool that tests Android apps via their GUI. They leverage results of recent work on model based GUI testing. Some of these models include Event Sequence Graphs [10], Event Interaction Graphs [49], Event Flow Graphs [45], and Finite State Machines [1], [44]. Testing techniques based on these models perform test generation as a post-model creation step. The biggest obstacle to adopting these techniques for the Android platform is model development [44]. While researchers have developed techniques to reverse engineer some models from the subject system by fully or partially automated analysis techniques [1], [3] fully automatic analysis remains challenging for Android GUIs. In [2] inspiring by other Event Driven Software (EDS) testing techniques proposed in the literature a GUI crawling based automatic technique for crash and regression testing of Android applications was proposed by Domenico Amalfitano et al.[2]. This technique is supported by a tool named Android Automatic Testing Tool (A2T2), a tool producing test cases that can be automatically executed, the framework has been mostly proposed to carry out assertion based unit testing and random testing of activities. D.Amalfitano et al. [5] presented new contributions were they realized a conceptual framework called MobiGUITAR for automated mobile app testing. It 's a new fully automatic technique to test GUI-based Android apps. It is based on the observation, extraction, and abstraction of the run-time state of GUI widgets. The abstraction is used to create a scalable state-machine model that, together with event-based test coverage criteria provide a way to automatically generate test cases.

2.3.7.3 Model learning testing

Choi et al. [18], proposed an automated technique, called Swift Hand, for generating sequences of test inputs for Android apps. The technique uses machine learning to learn a model of the app during testing, uses the learned model to generate user inputs that visit unexplored states of the app, and uses the execution of the app on the generated inputs to refine the model. A key feature of the testing algorithm is that it avoids restarting the app, which is a significantly more expensive operation than executing the app on a sequence of inputs [32]. Model-based testing approaches are often applied to testing graphical user interfaces. Such approaches model the behavior of the GUI abstractly using a suitable formalism such as event-flow graphs finite state machines or Petri nets [66]. The models are then used to automatically generate a set of sequences of inputs (or events), called a test-suite. Hu et al. [35] have proposed two-staged automatic model-based GUI testing idea. In the first stage, a model of the target application is extracted by dynamically analyzing the target program. In the second stage, test cases are created from the inferred model.

2.3.7.4 Pattern based (GUI) for mobile testing

Pedro Costa et al. [19] apply a new methodology named Pattern-Based GUI Testing (PBGT) where they started by fault seeding mutants into an open-source Android application named

Tomdroid, this methodology intends to test mobile applications based on models built upon User Interface Test Patterns (UITPs) that were originally applied to test web applications. The PBGT approach is all about reusability and the models could be used to test different applications in different OSs. It's an approach based on mutation analysis. But what is a mutation analysis?

Mutation Analysis

Mutation testing is a structural testing method aimed at assessing/improving the adequacy of test suites, and estimating the number of faults present in systems under test. The process, given program P and test suite T, is as follows: We systematically apply mutations to the program P to obtain a sequence P1, P2, ..., Pn of mutants of P. Each mutant is derived by applying a single mutation operation to Pj. We run the test suite T on each of the mutants, T is said to kill mutant Pj if it detects an error. If we kill k out of n mutants the adequacy of T is measured by the quotient k/n. T is mutation adequate if k = n. One of the benefits of the approach is that it can be almost completely automated [6].

2.4 Conclusion

From the above discussion we conclude that there are many testing techniques and each one, we consider has pros and cons associated with it, and we likely find that there is not a single testing method that is completely satisfying. Rather, we will need to consider a testing strategy that combines different testing options that as a whole provide the best overall testing result especially during the evolution of our mobile apps, balancing the trade offs between cost, quality, and time-to-market. Chapter 3 takes a closer look to this point.

CHAPTER 3

EXISTING STUDIES:THE EVOLUTION OF MOBILE APPLICATIONS AND THEIR TEST CASES

3.1 Introduction

In this chapter we discuss software evolution generally and mobile application specifically, by answering to specific questions, first. What is a software evolution? Are there any laws for software evolution? Are they applicable on mobile apps, if no why mobile apps evolution is different? then we present different types of evolution that concerns mobile apps, the tools used in analyzing this evolution, in the second part of this chapter we see whether the generated test cases get influence by the evolution of mobile apps?

3.2 Software evolution definitions

In 1969 Meir M. Lehman did an empirical study (originally confidential, later published) within IBM, with the idea of improving the company's programming effectiveness. The study received little attention in the company and had no impact on its development practices. This study, however, started a new and prolific field of research: software evolution. M.Lehman and J.Ramil (2000) [39] defined it as: all programming activity that is intended to generate a new software version from an earlier operational version. [34]. Ned Chapin et al. [16] defines software evolution as: the application of software maintenance activities and processes that generate a new operational software version with a changed customer-experienced functionality or properties from a prior operational version together with the associated quality assurance activities and processes, and with the management of the activities and processes. [39].
The Research Institute in Software Evolution(RISE). [65], [71] defined software evolution as:

N	Breif Name	Law
I 1974	Continuing Change	E-type systems must be continually adapted else they become progressively less satisfactory
II 1974	Increasing Complexity	As an E-type system evolves its complexity increases unless work is done to maintain or reduce it.
III 1974	Self Regulation	E-type system evolution process is self regulating with distribution of product and process measures close to normal.
IV 1980	Conservation of Organizational Stability (invariant work rate)	The average effective global activity rate in an evolving E-type system is an invariant over product lifetime.
V 1980	Conservation of Familiarity	As an E-type system evolves all associated with it, developers, sales, personnel, users, for example must maintain mastery of its content and behavior to achieve satisfactory evolution. Excessive growth diminishes that mastery. Hence the average incremental growth remains invariant as the system evolves.
VI 1980	Continuing Growth	The functional content of E-type systems must be continually increased to maintain user satisfaction over their lifetime.
VII 1996	Declining Quality	The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes.
VIII 1996	Feedback System (first stated 1974, formalized as law 1996)	E- type evolution must be processes constitue multi-level, multi loop, multi agent, feedback systems and must be treated as such to achieve significant improvement over any reasonable base.

Table 3.1: Laws of software evolution.Table reproduced from [40].

« The set of activities, both technical and managerial, that ensures that software continues to meet organizational and business objectives in a cost effective way ».

3.2.1 Laws of software evolution

When the laws were first presented in the literature widespread criticism of the use of the term laws was voiced. It was suggested, for example, that the observed phenomena reflected the behavior of human designers, implementer, managers and users. Thus they could not be laws in the normal sense of the word. Others felt that the phenomenology they reflect could

be altered at will, by education for example. Still others observed that the behavior described was intimately bound up with a particular organization (IBM) and/or the software system (development technology of the 70s). As such the observed phenomena lacked the generality that use of the term law implies. The refutation of the first view was based on the facts that the laws reflect the cooperative activity of much individual and organizational behavior. Table 3.1 summarizes Lehman 's laws of software evolution. These laws calls attention to the difficulties of maintaining a deployed software system since maintenance is considered to be costly, risky and time consuming; Lehman has proposed a new view to maintenance to include also the adaptation and reshaping of the system responding to the reasons of evolving instead of only fixing bugs in order not to worry about managing the changes of the system resulting from the change of the user's requirements. [40].

3.2.2 Model driven approach for software evolution

In Model-Driven Engineering (MDE) [11], meta-model and domain-specific languages are key artifacts as they are used to define syntax and semantics of domain models. Since in MDE models are not created once and never changed again, but are in continuous evolution, different versions of the same model are created and must be managed [28]. Bart Depoortere in his thesis describes taxonomy where he determines what kind of evolutionary information (i.e. what program entities can be changed), for example: (PackageChange, ClassChange (AddClass, RemoveClass), MethodChange (AddMethod, RemoveMethod), FunctionChange (AddFunction, RemoveFunction), StructuralEntityChange, AttributeChange (AddAttribute, RemoveAttribute), GlobalVariableChange (AddGlobalVariable, RemoveGlobalVariable), localVariableChange(AddLocalVariable, RemoveLocalVariable) [22].

3.3 Evolution of mobile applications

The main question that we should answer what makes mobile evolution different?
In many aspects, developing mobile applications is similar to other software systems.
Common issues include integration with device hardware, as well as traditional issues of security, performance, reliability, and storage limitations. However, mobile applications present some additional requirements that are less commonly found with traditional software applications. Including first, the potential interactions with other applications because most embedded devices only have factory-installed software, but mobile devices may have numerous applications from varied sources, with the possibility of interactions among them. Second reason is sensor handling, most modern mobile devices, e.g., «smart phones», include an accelerometer that responds to device movement, a touch screen that responds to numerous gestures, along with real and/or virtual keyboards, a global positioning system, for example a microphone usable by applications other than voice calls, one or more cameras, and multiple networking protocols. Third point is

native and hybrid (mobile Web) applications, most embedded devices use only software installed directly on the device, but mobile devices often include applications that invoke services over the telephone network or the Internet via a web browser and affect data and displays on the device; fourthly families of hardware and software platforms, most embedded devices execute code that is custom-built for the properties of that device, but mobile devices may have to support applications that were written for all of the varied devices supporting the operating system, fifth point is the different versions of the operating system. An Android developer, for example, must decide whether to build a single application or multiple versions to run on the broad range of Android devices and operating system releases [30]. Six Security: most embedded devices are «closed», in the sense that there is no straightforward way to attack the embedded software and affect its operation, but mobile platforms are open, allowing the installation of new «malware» applications that can affect the overall operation of the device, including the surreptitious transmission of local data by such an application. In addition to User interfaces with a custom-built embedded application, the developer can control all aspects of the user experience, but a mobile application must share common elements of the user interface with other applications and must adhere to externally developed user interface guidelines, many of which are implemented in the software development kits (SDKs) [74]. Next the Complexity of testing while native applications can be tested in a traditional manner or via a PC-based emulator, mobile Web applications are particularly challenging to test. Not only do they have many of the same issues found in testing web applications, but they have the added issues associated with transmission through gateways and the telephone network and finally power consumption many aspects of an application affect its use of the device 's power and thus the battery life of the device. Dedicated devices can be optimized for maximum battery life, but mobile applications may inadvertently make extensive use of battery-draining resources. We categorized the evolution of mobile apps in two classes: external evolution and internal one [74].

3.3.1 External evolution of mobile applications

Before mobile operating systems, applications had to be specific to various phone models.
Figure 3.1 shows the external evolution of mobile applications. One of the drivers of third-party apps was open source OSs such as Symbian and Android. A fourth generation of apps is Cloud-based and runs on the mobile devices' browser. Utilizing the best-of-breed Web 2.0 technology and creating rich Internet applications for mobile browsers, these applications turn smart phones into mobile thin clients, where minimal code and data resides locally on the device. Interestingly, evolution of Web 2.0 promises to be one of the biggest technology champions for mobility. This is true if enterprises want to extend their applications to the latest portable devices by enabling them to deploy applications on portable personal devices using app stores and widget stores. Such use of Web 2.0 technology allows organizations to extend Enterprise 2.0 concepts to employees devices. Also, by adding context-awareness to such applications, enterprise could use the device

's GPS to pinpoint an employee 's exact location. Accessed through the enterprise portal, such applications could use augmented reality features to direct the employee to a required area. One of the biggest advantages of these types of Cloud-based mobile applications is that irrespective of device change or loss, the user enjoys seamless application usage on their next device. Context awareness can also aid autonomous applications to initiate machine-to-machine transactions on their own [41].

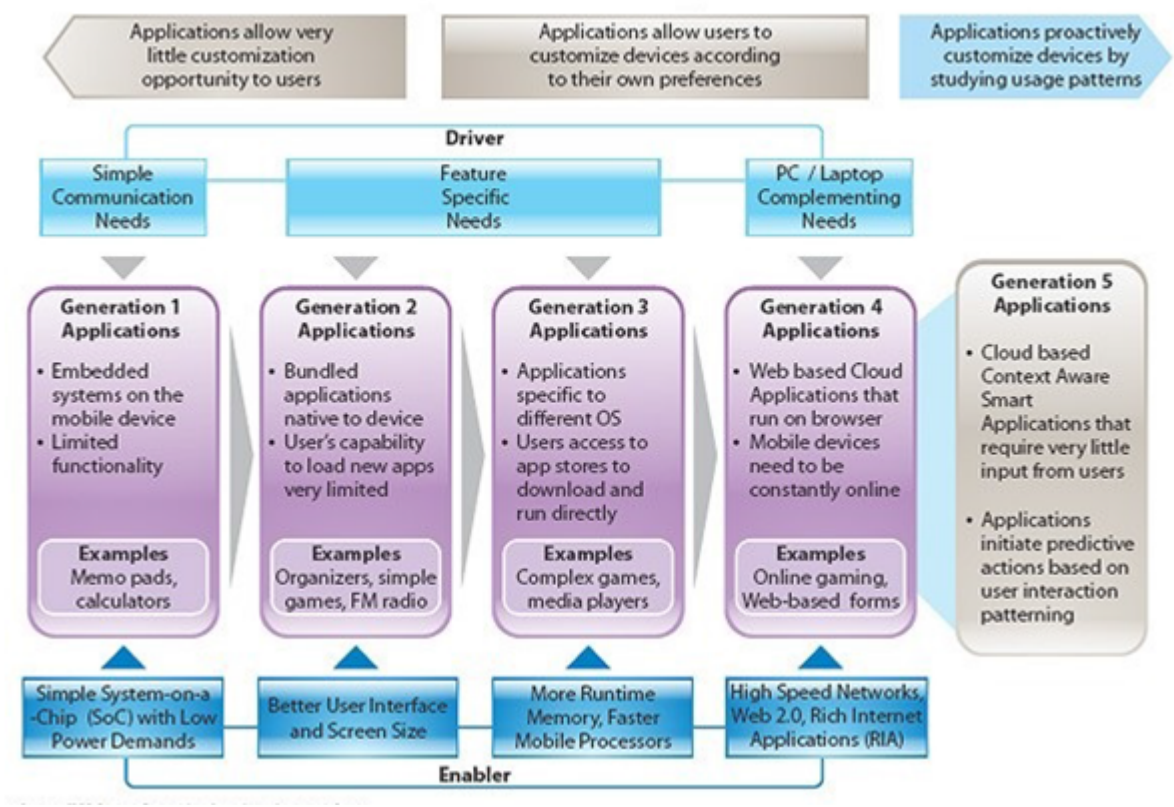

Figure 3.1: Evolution of mobile Application. Figure reproduced from [41].

3.3.2 Internal evolution of mobile applications

Elizabeth Phillips [63], said that apps themselves continue to undergo dramatic change and development. More personalized optimized for local use and offer social interaction to customer, the evolution of mobile apps focuses nowadays on their design for example: before Many apps utilized a simple list or icon system to display features. Now, apps are able to incorporate high-quality photos and mobility throughout the app. Instead of just scrolling from top to bottom or clicking on one-destination icons. The Facebook app incorporates a more interactive user experience with a «swipe-over menu to the side for accessing our profile, News Feed, events,

etc. instead of that home screen grid.» [63], also we find API evolution, API is an abbreviation of application program interface, is a set of routines, protocols, and tools for building software applications. The API specifies how software components should interact and are used when programming graphical user interface (GUI) components. A good API makes it easier to develop a program by providing all the building blocks.A programmer then puts the blocks together [8]. Another kind of internal evolution is LTE evolution: Long term evolution it is in services

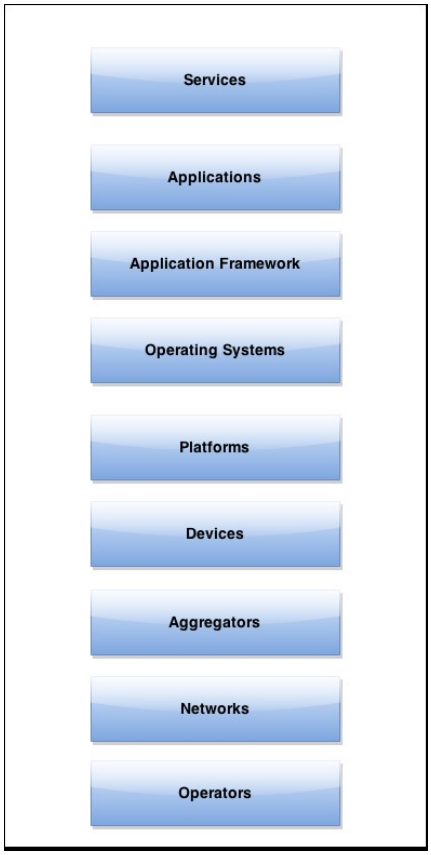

Figure 3.2: The Mobile ecosystem. Figure reproduced from [4].

provided by mobile apps [62], besides to Requirement evolution for example In 2002, a powerful smart phone BlackBerry released by RIM which increased the requirement of applications [15]. Furthermore ecosystem evolution where the application evolve according to changes in its surrounding environment Figure 3.2 shows the ecosystem of mobile devices [4].

3.3.3 Analyzing mobile application evolution

Roberto Minelli et al. [50] used first SAMOA which is an interactive web-based visual software analytics platform to analyse apps from a structural and historical perspectives, it uses visualizations to present data and it is available at http://samoa.inf.usi.ch, secondly F-Droid corpus which

is a FOSS (Free and Open Source apps) catalogue of Android, SAMOA is a platform that uses visualizations to represent data; this software is divided into five main parts: First a Selection Panel: it is a panel that helps to choose an app among the available apps and also chose the visualization perspective, Second Metrics Panel: it shows the list of metrics to be used in the app revision. Third Revision Info Panel: provides information and snapshots about the app 's revision results. Fourth Entity Panel: shows data and information about the on focus entity. Fifth Main view: this is the main interface to display the interactive visualization. some of the very important metrics used in SAMOA are, Number of packages in the project (NOP), Number of classes defined by the user (NOC), Number of methods defined by the user (NOM), The number of non empty lines of code (LOC), Number of method calls (CALLS),The average number of derived classes (ANDC) etc. Each metric is cited with its own scope. SAMOA contains three different views for visualization Snapshot view: used for specific revision of a single app, History view: used to explore the evolution of a single app over time, Ecosystem view: used to see the visualization of several apps at once [50]. Jack Zhang et al. [78] investigated weather Lehman 's law of continuing change, increasing complexity, and declining quality applicable to mobile apps?

(1): Continuing Change

For the first law, they used three different metrics Code Churn: the number of added, modified or deleted lines of code. Feature Commits: the number of commits made by each developer that add features and not fix bugs and Number of hunks, a hunk is the piece of code being changed between two commits.

(2): Increasing Complexity

For the second law, they used two metrics: Lines of Code: the number of non empty lines of code and commits per file: knowing the average number of commits per file helps knowing the rate of change being happened to the app.

(3): Declining Quality

Finally for the third law, they used only one metric: Bug Fixing Commits: the more we have the more the quality improves. And they concluded that in first, second laws all the three metrics prove that mobile apps accept the law of continuing change to be applied. And complexity is increasing overtime. But the third law it is difficult to analyze the quality declining law using small number of mobile apps they recommended further study to be done [48]. apps evolve over time and they require testing and maintenance activities to be performed, after we talked about how mobile apps evolve and how their evolution process is analyzed in the next section we will talk about the evolution of test cases. And how can we adapt them?

3.4 Evolution of test cases

Test repair can be an expensive activity, automating it, even if only partially could save a considerable amount of resources during maintenance and this is the motivation behind the development

of automated test-repair techniques , such as the ones targeted at unit test cases and those focused on GUI or system test cases . For example a change in the GUI may render some of the test cases useless, In practice, since a large number of the original test cases can not be reused, GUI regression testing requires redeveloping new test cases from scratch, Memon and Soffa [48] presented a repair technique that constructs models of the original and modified GUI components, where the models represent the possible flow of events among the components. Their technique compares the models to identify deleted event flows and attempts to repair the tests that traverse such (invalid) flows. The repair strategy deletes one or more events from a test sequence, or splices one or more events with a different event, so that the resulting sequence can be executed on the modified GUI. After that in 2004 Memon [47], compares models of GUI events to capture the differences between two software versions, and fix GUI test cases accordingly. The technique works well for GUI tests and in the presence of application models and runtime assertions. Test augmentation techniques use symbolic execution [61], concolic execution or genetic algorithms. [76] to derive test inputs that cover paths not executed yet. These approaches improve automatic test case generation techniques by taking advantage of the test cases manually written by developers, but suffer from the same limitations of classic automatic techniques, do not identify the setup actions necessary to execute the test cases and tend to generate a huge amount of test cases. ReAssert focuses on repairing test oracles, and fixes oracles broken by changes in the software specifications by re-executing the failing test cases using concrete or symbolic execution [21]. The suggestions to fix oracles proposed by ReAssert must be validated by software developers, as ReAssert modifies the test cases to make them pass, and thus, it could erroneously mask software failures. ReAssert repairs test oracles, but does not fix test inputs; thus, developers still need to manually correct the inputs. Approaches dealing with the identification of mismatches in parameter inputs exist, but they do not provide solutions to automatically repair the identified problems [33]. Some test cases can be corrected by means of automatic refactoring techniques that can prevent simple errors by automating some refactoring activities like moving or renaming methods unfortunately, common refactoring practices like adding new parameters to methods . [75] are only partially automated by existing tools and techniques. For example, ReBa. [25] and the Eclipse IDE. [57] can fix compilation errors caused by parameter changes, but only when the modified parameters can be replaced by default values, noting that test evolution scenarios can be categorized in two groups: repair and generation

Repair scenarios: identify situations in which programmers may reuse variables or values to repair test cases that do not compile because of changes in the method signature that derive from adding a parameter to a method, removing a parameter of a method, changing the type of a parameter, and changing a return type.

Generation scenarios: identify situations in which programmers can reuse test cases to validate new functionality: to create test cases for a class added to a hierarchy, to test the implementation of an interface, or to test new overridden or overloaded methods.

M.Mirzaaghaei et al. [52] presented a repair technique for fixing JUnit test cases that are broken because of changes in method signatures, that is, addition, deletion, or modification of parameters. The technique identifies the broken method call and attempts to create a modified call in which new parameters are initialized with suitable values. Using data-flow analysis, program differences, and run time monitoring, it searches for initialization values from existing data values generated during the execution of the test case against the original application. Although its general effectiveness is unclear, the technique may work well for specific types of changes (e.g., where a formal parameter is replaced with a type that wraps the parameter). This technique, by trying to handle changes in method signatures, is a right step in the direction of developing more widely applicable automated repairs, however, the effectiveness of the technique is still limited, and the development of more sophisticated approaches is required. In particular, the technique attempts to fix a broken method call by adding, deleting, and modifying parameters, but it does not synthesize new method calls. In 2014 Mirzaaghaei et al. [52] focus on the problem of updating test suites automatically and he presents a framework named: TCA Test Case Assistant that implements the algorithms to support the evolution of test suites written in java. It takes as inputs original software, modified software, test cases for the original software and displays the output which is test cases for the modified software. It is a first step towards the definition of techniques that take advantage of existing test cases to repair broken test cases or generate new ones; they used the code of the software not the GUI [47] One of the main reasons for test-suite evolution is test obsolescence: test cases cease to work because of changes in the code and must be suitably repaired. There are several reasons why it is important to achieve a thorough understanding of how test cases evolve in practice. In particular, researchers who investigate automated test repair an increasingly active research area can use such understanding to develop more effective repair techniques that can be successfully applied in real world scenarios. More generally, analyzing test suite evolution can help testers better understand how test cases are modified during maintenance and improve the test evolution process. To tackle this problem, Leandro Sales et al. [64] developed TestEvol. A tool that enables the systematic study of test-suite evolution for Java programs and JUnit test cases.

3.5 Conclusion

Successful mobile applications continue to evolve long after deployment, for example in response to requirements changes, and better understanding of user needs. Changes incur high maintenance costs not only for the Mobile app under test (MUT) but also for the regression test suites which can often be larger than the MUT itself , when requirements change and the MUT evolves, some existing tests break because they reflect the old behavior, and not the new intended behavior. Broken tests cause many problems. Updating broken tests takes time. As we discussed in previous sections of this chapter many works tackle this phenomenon in different ways by

different tools, and methods. The next chapter presents a new approach for repairing broken tests.

CHAPTER 4

PROPOSED APPROACH: MODEL DRIVEN APPROACH FOR TEST CASES EVOLUTION

4.1 Introduction

This chapter deals with the implementation of a model driven approach for test cases evolution. Where Section 4.2 presents basic concepts and challenges which we faced them, the technique adequate environment, tools, methodologies. Next in section 4.3 we present the global architecture of our proposal. Followed by a brief exploration of the chosen alternatives, how is the conceptual model implemented? How the changes are structured? Which kind of changes we used? Also this section reveals on how to analyze the changes implemented? And which model we used in studying the evolution between two versions?

4.2 Concepts and challenges

4.2.1 The Android operating system (OS)

Android is a mobile operating system (OS) based on the Linux kernel and currently developed by Google. With a user interface based on direct manipulation, Android is designed primarily for touch screen mobile devices such as smart phones and tablet computers, with specialized user interfaces for televisions (Android TV), cars (Android Auto), and wrist watches (Android Wear). The OS uses touch inputs that loosely correspond to real-world actions, like swiping, tapping, pinching, and reverse pinching to manipulate on screen objects, and a virtual keyboard. Despite being primarily designed for touch screen input, As of July 2013, the Google Play store has had over one million Android applications (apps) published, and over 50 billion applications downloaded. A developer survey conducted in April May 2013 found that 71% of mobile developers

develop for Android. At Google I/O 2014, the company revealed that there were over one billion active monthly Android users, up from 538 million in June 2013. As of 2015, Android has the largest installed base of all general-purpose operating systems. Android is popular with technology companies which require a ready-made, low-cost and customization operating system for high-tech devices. Android's open nature has encouraged a large community of developers and enthusiasts to use the open-source code as a foundation for community-driven projects, which add new features for advanced users or bring Android to devices which were officially, released running other operating systems. The operating system's success has made it a target for patent litigation as part of the so-called «smart phone war»" between technology companies [73]. The last version of android is android Lollipop API level 22 [14], and it has many important features such as ability to join Wi-Fi networks and control paired Bluetooth devices from quick settings, support for multiple SIM cards, device protection i.e. if a device is lost or stolen it will remain locked until the owner signs into their Google account, even if the device is reset to factory settings. High-definition voice calls, available between compatible devices running android 5.1, return of the silent mode, which was removed in Android 5.0.

4.2.2 Developing Android applications on Eclipse IDE

The development of android application on Eclipse IDE requires Android Software development kit (SDK) setup, Android Developer Tools (ADT) which offers the access to many features that help to develop Android applications. ADT provides GUI access to many of the command line SDK tools as well as a UI design tool for rapid prototyping, designing, and building of application's user interface. Android applications are primarily written in the Java programming language. During development the developer creates the Android specific configuration files and writes the application logic in the Java programming language. The ADT tools convert these application files, transparently to the user, into an Android application. When developers trigger the deployment in their IDE, the whole Android application is compiled, packaged, deployed and started. The Java source files are converted to Java class files by the Java compiler. The Android SDK contains a tool called dx which converts Java class files into a (.dex) Dalvik Executable file. All class files of the application are placed in this .dex file. During this conversion process redundant information in the class files are optimized in the.dex file. For example, if the same String is found in different class files, the .dex file contains only one reference of this string. These .dex files are therefore much smaller in size than the corresponding class files. The .dex file and the resources of an Android project, e.g., the images and XML files, are packed into an .apk (Android Package) file. The program aapt (Android Asset Packaging Tool) performs this step. The resulting .apk file contains all necessary data to run the Android application and can be deployed to an Android device via the adb tool[69]. Figure 4.1 presents the user interface of Eclipse IDE [23] during the development of Android Application.

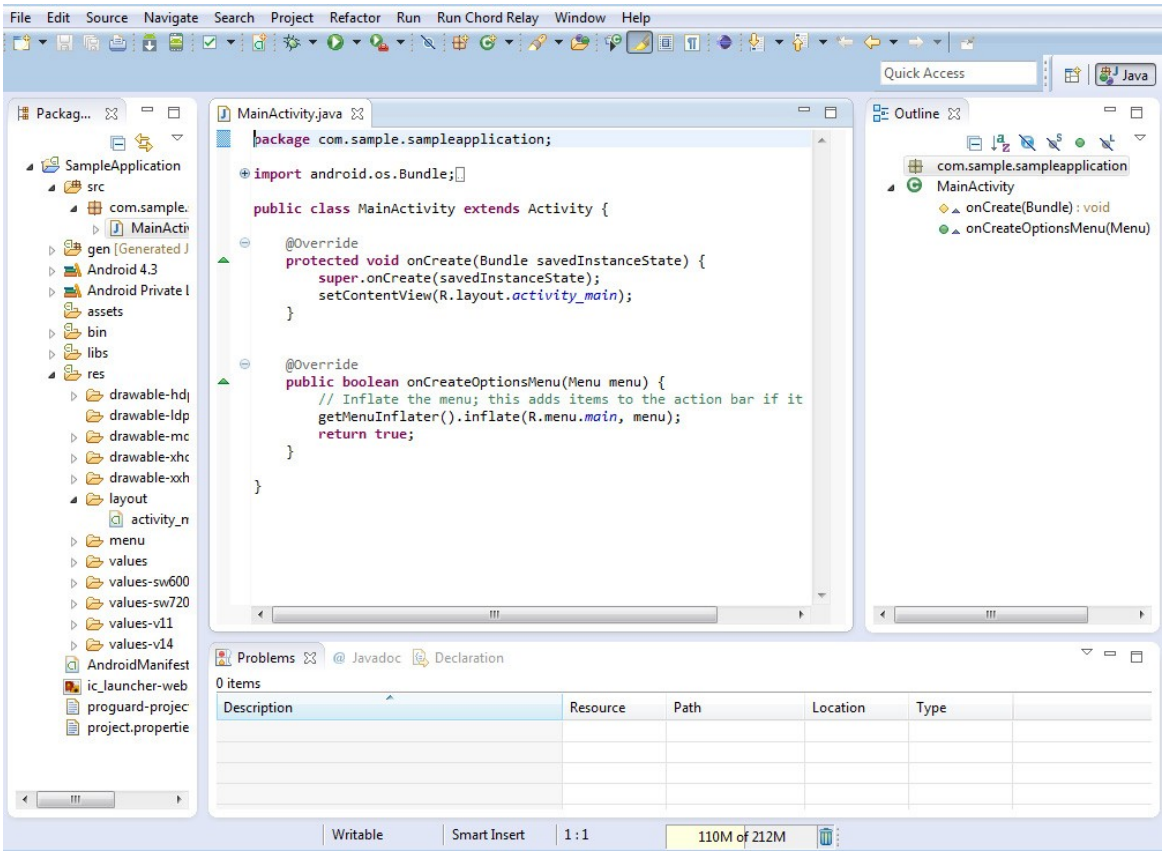

Figure 4.1: Eclipse user interface during the development of Android mobile App. Figure reproduced from [24].

4.2.3 Permission concept in Android

Android contains a permission system and predefined permissions for certain tasks. Every application can request required permissions and also define new permissions. For example, an application may declare that it requires access to the Internet. Permissions have different levels. Some permissions are automatically granted by the Android system, some are automatically rejected. In most cases the requested permissions are presented to the user before installing the application. The user needs to decide if these permissions shall be given to the application. If the user denies a required permission, the related application can not be installed. The check of the permission is only performed during installation; permissions can not be denied or granted after the installation. An Android application declares the required permissions in its Android Manifest.XML configuration file. It can also define additional permissions which it can use to restrict access to certain components [73].

4.2.4 Migrating to Android Studio

Android Studio is the official IDE for Android application development, based on IntelliJ IDEA. It offers a flexible Gradle-based build system, build variants and multiple apk file generation, Code templates to help the building of common app features, Rich layout editor with support for drag and drop theme editing, lint tools to catch performance, usability, version compatibility, and other problems, ProGuard and app-signing capabilities, Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and app Engine, And much more [24], Figure 4.2 presents the android studio user interface for developing android application [23].

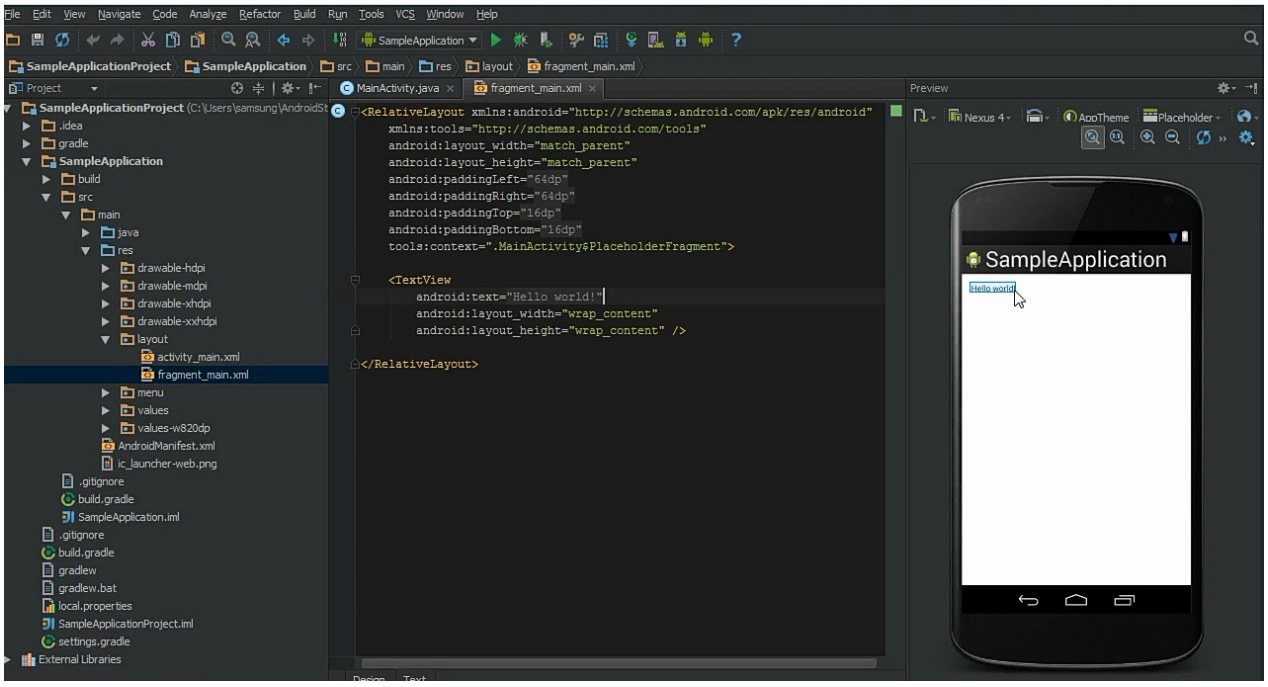

Figure 4.2: Android Studio user interface during the development of Android mobile App. Figure reproduced from [24].

Android Studio is an IntelliJ-based platform that offers better UI for designing and coding. It has ProGuard and app-signing capabilities, as well as gradle-based support. The Live Layout feature allows the developer to view the user interface as the codes are written and modified. It also has the Presentation View aside from the Drag and Drop UI layout editor. An improved intelligence feature through added annotation is also available. Still in its initial release, the future versions are geared towards a more efficient handling of multiple projects . Table 4.1 summarizes basic differences between Eclipse and Android Studio IDE [69].

Category/Feature	Android Studio	Eclipse
Build	Based on the IntelliJ platform; thus, IntelliJ plug-in and Android Studio are built from the same code	Serves as an ADT plug-in, through Google provides an IDE bundled with it
Installation	Requires frequent,updates	Stable does not require frequent update
Cost	FREE	FREE
Project structure	Contains a .idea,folder, gradle files, and other folders like src, gen, res,generated by Eclipse	Contains src, gen and res folders by default
Dependency,declaration	Dependencies are,declared in both gradle and XML files	Dependencies are declared only in the XML files
Terminologies	Examples :,Android studio, Term= eclipse Term, Workspace/Projects=Project,Project=module	
Visual editor	live Layout	Offers a review of the layout but it is not rendered at real time
View	provides Drag and Drop UI Layout Editor,Presentation View, and Live Layout	Provides Drag and Drop UI layout editor
File editor	uses annotated based intellisense in addition to the default code completion options	Supports java based code completion
Testing	AVDs are compatible	
Debugging	offers support for Lint Tools, Ant Tools, Thread Monitoring, Groovy, Gradle, and more	Offers support,for lint tools, ANT tools, thread monitoring and more

Table 4.1: Android Studio via Eclipse. Table reproduced from [69].

4.2.5 Monkey talk for testing mobile applications

Testing using Monkey Talk requires to installation AJDT (for AspectJ conversion) on eclipse and addition of monkeytalk.jar library to the Application under test (AUT) , Monkey talk is the world's greatest mobile app testing tool. It automates real, functional interactive tests for iOS and Android apps - everything from simple "smoke tests" to sophisticated data-driven test suites. The Monkey Talk Community Edition 2.0 is the culmination of Gorilla Logic's five years of creating open-source automated testing tools. Downloaded over 70,000 times, MonkeyTalk is enabling teams all over the world to achieve five star mobile app quality at agile speed. Figure 4.3 presents its features such us the robust cross i.e. platform Recording /Play back , also the use of the simple Monkey Talk command language or powerful JavaScript API, run the tests interactively or from continuous integration environments, and it is Free and open Source tool [53],[69].

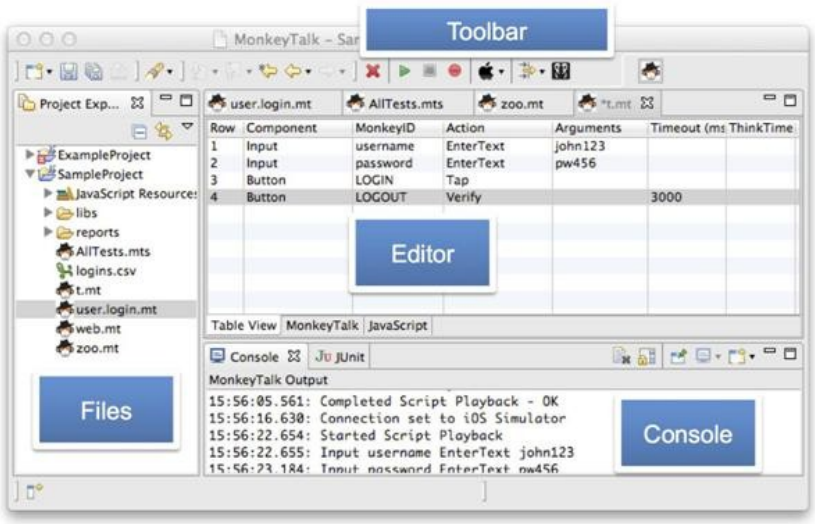

Figure 4.3: Recording through the Monkey Talk IDE. Figure reproduced from [53].

4.3 Proposed Approach

To ease the adaptation process of test cases in mobile applications, there needs to be a solid protocol between developers and testers involved before and after deployment stages of a mobile application this is in order to capture the set of changes that might even be quickly. TCA [51], ReAssert [20], TESTEVOL [64] have been reported to be useful for studying test suite evolution and repairing broken test cases, However, such tools are proprietary and therefore can not be extended to repair test cases at all testing levels for example for Integration testing , Component interface testing or System testing in addition to the fact that those tools are used only for repairing broken Unit Tests for software evolution generally. In this research studies, we propose a model driven approach for test cases evolution that can be used for repairing broken test cases for android mobile applications starting by analyzing the evolution of mobile apps and therefore the possibility of automated the process of repairing old test cases and generate new once then verify if the latter covers the new evolved version of mobile app. An overview of the proposed approach is being illustrated in Figure 4.4 initially we have an original version of a mobile app that evolves by adding new functions, or changing in its design at the API level, either to adapt with the requirements of the end user, or market trends, or its ecosystem. The models based mobile apps contains all essential components of a native android mobile app, which is serialized to an XMl file complying with a pre-defined java MoDisco meta model and XML MoDisco meta model [12]. It is worth noting that the MOF is using the XMI specification (standing for XML Metadata Interchange). Our process analysis the differences by taking the two models the ancient and the new one and makes a comparison between them using EMF compare Figure 4.5 shows the architecture of the framework [27], the two models are considered as inputs of the framework

Models are represented with graphical notations; After that from this given starting point EMF compare makes a model resolving which means finding all other fragments required for the comparison of the whole logical model. For example to compare two XML models they are not a standalone models but generally they are loaded as embedded within their XML meta-model.

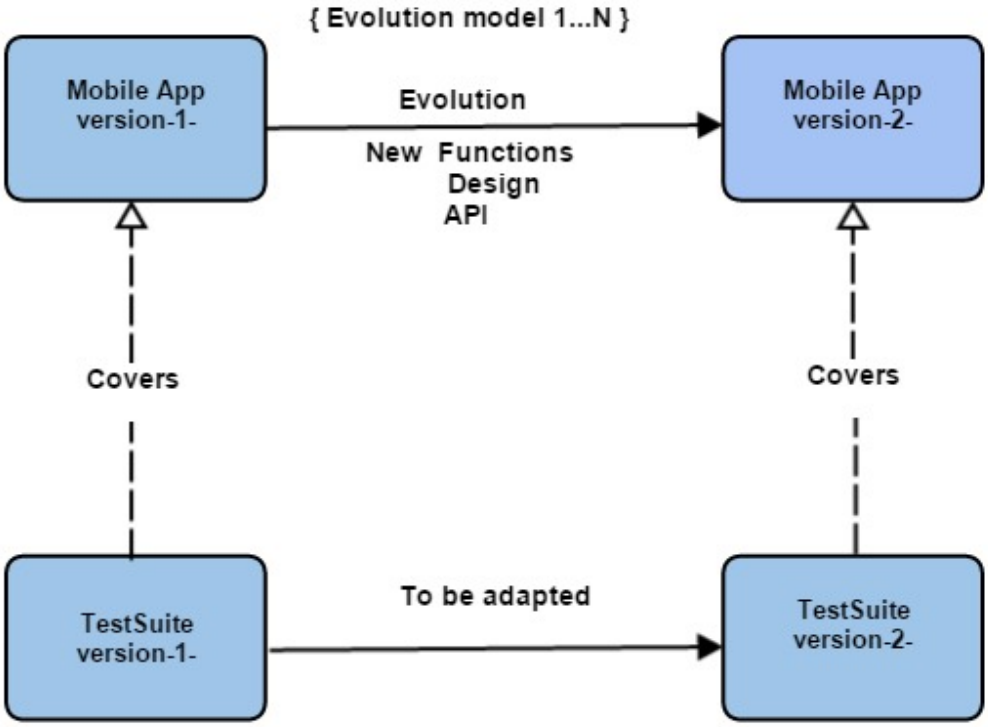

Figure 4.4: Overview of the Model-driven Process for test cases evolution.

Matching will be done on the two models in order to map elements together two-by-two or three-by-three. For example, determine that class Class1 from the first model corresponds to class Class1' from the second model. In the Comparison model we find differences, equivalences, The differences phase will browse through the mappings elements and determine whether the two (or three) elements are equal or if they present differences (for example, the name of the class changed from Class1 to Class1'). However, two distinct differences might actually represent the same change meanwhile an equivalences phase will browse through all differences and link them together when they can be seen as equivalent (for example, differences on opposite references). Requirements, for the purpose of merging differences, there might be dependencies between them. For example, the addition of a class C1 in package P1 depends on the addition of package P1 itself. During this phase, we'll browse through all detected differences and link them together when we determine that one can not be merged without the other. Conflicts, comparing files with one from a Version Control System (CVS, SVN, Git...), there might actually be conflicts between

the changes made locally, and the changes that were made to the file on the remote repository. This phase will browse through all detected differences and detect these conflicts [27].

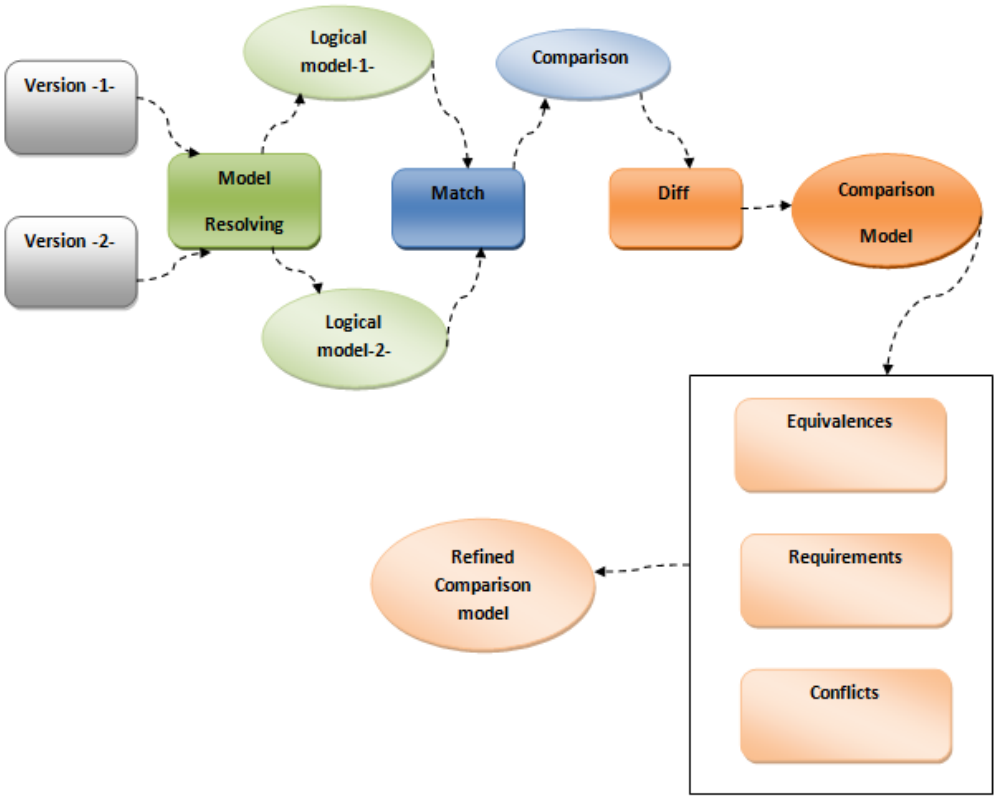

Figure 4.5: Architecture of EMF Compare framework used for the analyzing process. Figure reproduced from [27].

The comparison between different models is triggered through a graphical view on eclipse environment; graphical differences are not necessary model differences. Models may be non-ordered resources. Order is a possible difference between two elements in a list. Models are constituted of elements. Elements are linked to each other by relationships that are defined by their meta model, they can then be browsed as trees. Elements are composed of attributes. Attributes are objects identified by a unique key (name, is Abstract) assimilated to an attribute and a value. The value itself is typed to a simple type, a text, a number, a Boolean. To further illustrate the use of EMF compare graphically and make a visually comparison.

The result of a graphical comparison using EMF compare is an XMI model containing equivalences and differences. Figure 4.6 presents the work flow of the global process of the proposed approach. Generating test cases from the original mobile application, using the comparison results of EMF compare as inputs in order to obtain the list of deleted test cases for example the ones concern a deleted element from version 1 to version 2, kept test cases which are the common

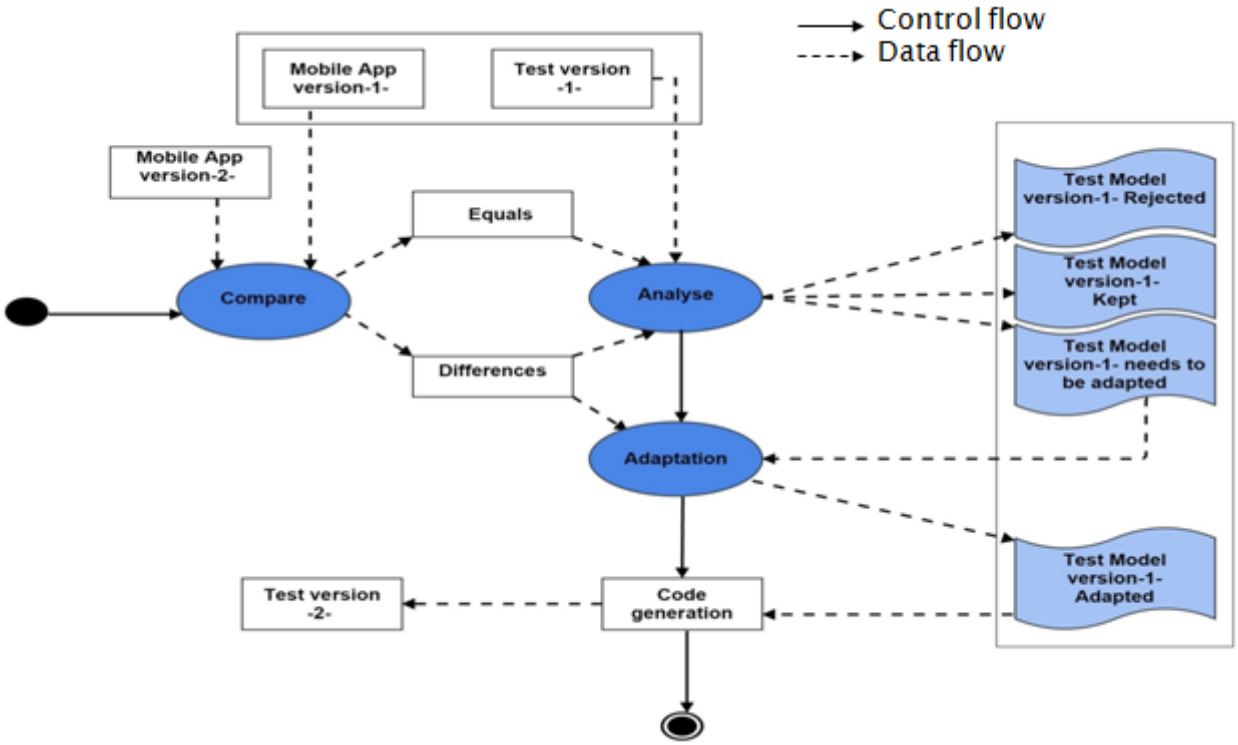

Figure 4.6: Work flow diagram of model driven approach for test cases evolution.

test cases between two versions, and the adapted test cases which are the most important one, because by adapting them new test cases that cover the new version of a mobile application will be generated automatically as an XMI model, then it can be transformed to java code, or java script test suite [56].

4.4 Conclusion

After having see the principle of Model driven approach for test cases evolution of mobile application and its theoretical sound. Next chapter presents case study for the evolution of mobile apps, generation of test cases and their adaptation.

CHAPTER 5

CASE STUDY: NATIVE ANDROID MOBILE APPLICATIONS

5.1 Introduction

In previous chapter we presented the theoretical part of our approach, in this chapter we are going to present illustrative examples of our proposition. Worth noting that this train ship was made in collaboration with the enterprise BeApp [72], Nantes City, France. Our main purpose is treating the common phenomenon «evolution of tests» in both research and industry. The evaluation of work progress was made by a monthly meeting in LINA laboratory or in the company BeApp with the presentation of supervisors of the project Doctor jean Marie Mottu, Professor Christian Attiogbe and from the enterprise the Technical Director Mr. Cedric Guinoiseau and a leader in Android development Mr. Damien Villeneuve .

5.2 Description of our case study

5.2.1 HandicApp

Is an open source application Figure 5.1, it allows to easily interacting by voice with the interviewer. Transcribing words, HandicApp listen and when the interviewer finished, it displays what has been pronounced. HandicApp can also vocally synthesize a written text on the smart phone. Available in French and English (as the language of the phone). This is done through voice. It is available free on the Play Store from Android 2.3.3 for smart phones and tablets [59].

5.2.2 Toile 2 vert

Toile 2 vert is android mobile app developed to recharge electric vehicle, finding a Bike Point, Marguerite station, Geolocate quickly, Toile 2 Vert as represents Figure 5.2 helps to become easily

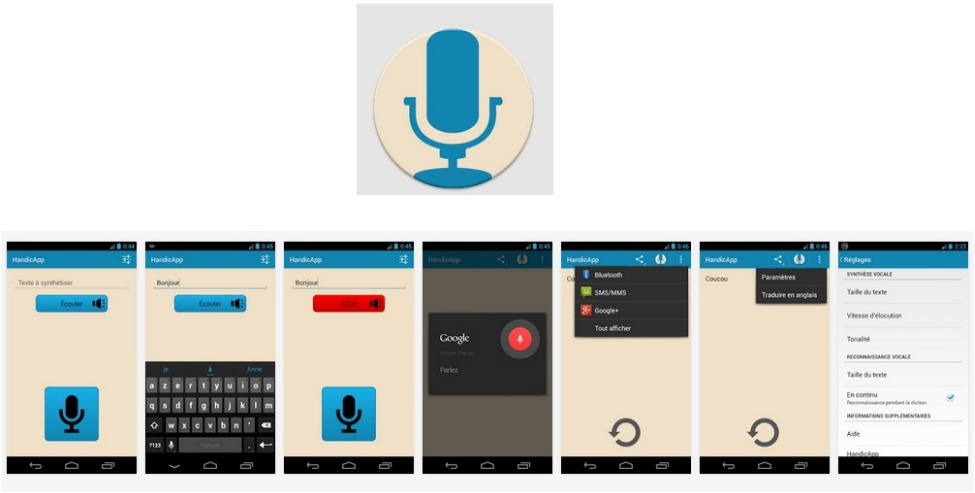

Figure 5.1: HandicApp native Android mobile application. Figure reproduced from [59].

an eco-citizen. The junior room of Nantes Metropole South Loire, with the support of BeApp, developed this project [9].

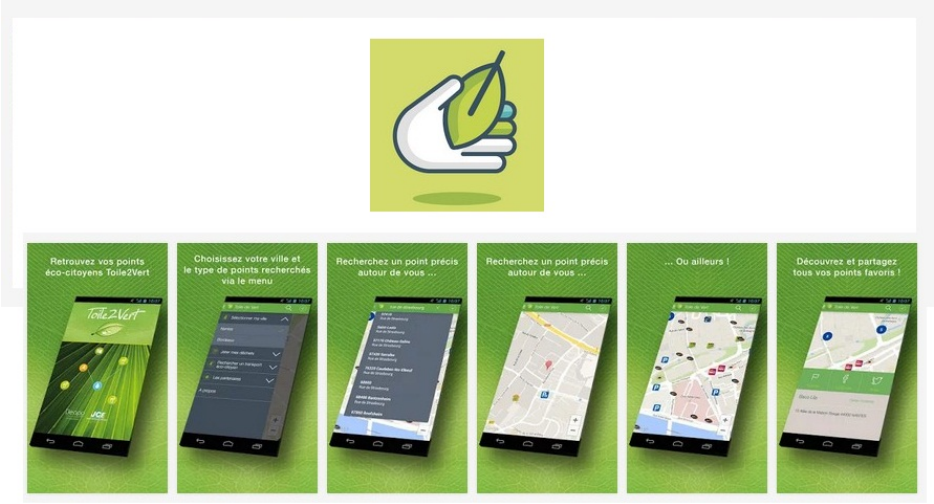

Figure 5.2: Toile 2 vert native android mobile application. Figure reproduced from [9].

5.3 Testing of android application

5.3.1 Manual Testing

For manual testing we have used the eclipse emulators using AVD manager but we have found many disadvantages for example no support for placing or receiving actual phone calls , No support for USB connections, no support for camera /video capture , no support for determining battery charge level and AC charging state , no support for Bluetooth, Sometimes lower versions will not support completely, no support for orientation testing. Figure 5.3 presents android Acer smart phone version 4.2.2 and Google tablet that we used them in our experimentation study.

Figure 5.3: Smart devices using in the testing of android mobile

5.3.2 Activating the developer mode on smart devices

Before linking the smart device to the machine by USB cable in order to test the application by its code written in java on Eclipse or Android Studio we have to activate the developer mode by using such Android device choose Settings > About phone > Build number Get the build number section of the settings, Tap on the section 7 times. After two taps, a small pop up notification should appear saying "you are now X steps away from being a developer" with a number that counts down with every additional tap. After the 7th tap, the Developer options will be unlocked and available. They can usually be found in the main settings menu. Dive into that menu to do things like enable USB debugging (a frequent prerequisite to lots of hacks).
After that for example for Stock Android:
Settings > Developer options> Stay awake,
Stock Android: Settings > USB debugging
Stock Android: Settings > Allow mock locations

5.3.3 Automation testing Using Monkey Talk

For interface testing we have choose Monkey Talk, but how can we connect Monkeytalk to Eclipse IDE? How can we test using Monkeytalk IDE? What's the relation between two environments?

At first right click on a java project, choose Configure to Aspectj Project after that create libs folder if it is not created yet, and add both libraries: android-support-v4.jar Monkeytalk-agent-2.0.10.jar. Next in the monkeytalk.jar right click, AspectJ tools, Add to Aspect path, after that in the Android Manifest.xml add two lines:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.GET_TASKS" />
```

Figure 5.4: Permission concept used during the testing of android mobile app

By right click on the project properties Java Build Path Order tick monkeytalk-agent-2.10.Jar, AspectJ Runtime Libraray, Android Dependencies, Android Private Libraries, and tick also the tested project. After that click OK. Launching the emulator: Either we use: the Android Virtual Device Manager by going to window android Virtual Device Manager create a new emulator Fill the information of the application's emulator in the table for example: AVD Name: My emulator, Device: Nexus One,... etc, OK. Device Manager create a new emulator the selected emulator must be en adequate to the API level : by checking in the Android Manifest.XML we can find this information. Finally in the application (java project) right click run as android application after a while the application will be in run time. When the application is running in eclipse or android studio after lunching monkey talk, and by clicking on the red button for start recording traces of the end user And start using the application by clicking and in the end we need to stop recording and executing the script, if everything goes well then we obtain «A completed script plays back, OK.» If there is an/many error(s) in the application we obtain «A completed script plays back, FAILURE.»

5.4 Evolution of the applications

In Handicapp we have started by proposing that the developer makes minor syntactic changes on the interface of the application such as modify a button, add a button, delete a button these changes in the application generate many differences between the different versions comparing to the original one. For example modify button content After using monkey talk generate two different test cases Figure 5.4.

Figure 5.5 (left) presents the execution of first test cases on the first version, Figure 5.6 (right) presents the execution of test cases on the second version, we always get positive results, however. The execution of first test suite on the second version causes run time errors Figure 5.6, the first

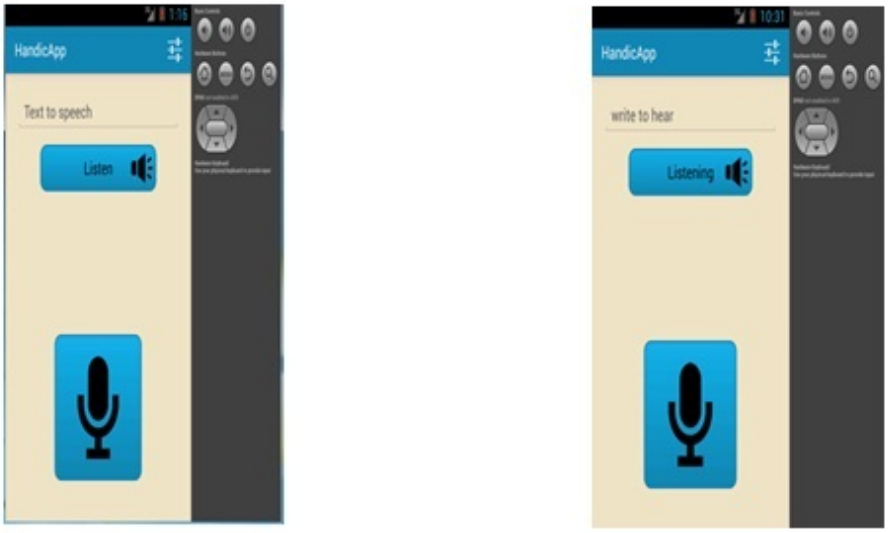

Figure 5.5: Syntactic minor Changes during Handicapp Application evolution.

```
load("libs/Handicapp.js");


Handicapp.FirstTest.prototype.run = function() {
        /**
         * @type MT.Application
         */
        var app = this.app;


        app.textArea("Text to speech").tap();
        app.textArea("Text to speech ").enterText("Hello");
        app.button("Listen").tap();
        app.image("Speech recognition").tap();
        app.device().back();
};
```

```
load("libs/Handicapp.js");


Handicapp.SecondTest.prototype.run = function() {

        /**
         * @type MT.Application
         */
        var app = this.app;

        app.textArea("write to hear").tap();
        app.textArea("write to  hear").enterText("Hello");
        app.button("Listening").tap();
        app.image("Speech recognition").tap();
        app.device().back();
};
```

Figure 5.6: Java script test cases generated using Monkeytalk.

test cases are broken, our main purpose is to adapt them; to repair them according to the minor evolution of the application. In order to cover the new version of the app, better than testing from scratch especially in the case of applications with rich interfaces. And huge complexity. In the case of deleting a button from Handicapp application,

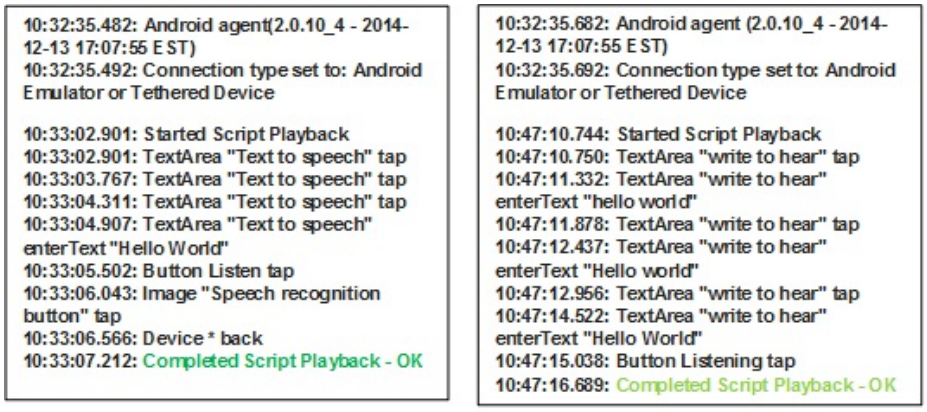

Figure 5.7: The execution of the first test cases on the first version and second test cases on the second version.

Figure 5.8: The execution of first test cases on the second version.

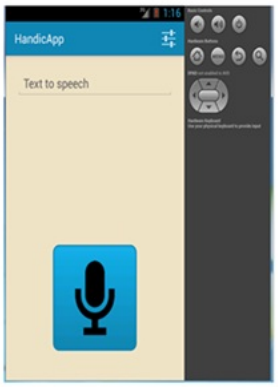

Figure 5.9: Handicapp application without listening button.

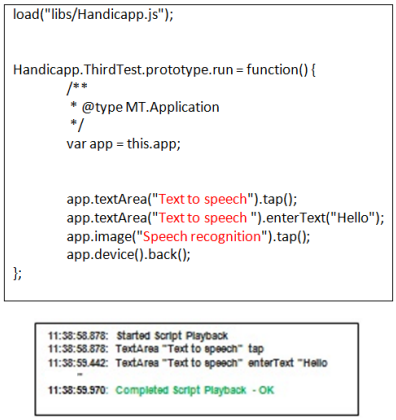

Figure 5.10: The execution of the third test cases on the third version.

Figure 5.7, from a visual comparison between test cases (FirstTest.js) that concern original version and test cases (ThirdTest.js) that concern third version Figure 5.8 we can see that there are deleted test cases Figure 5.9. How can we go from these three inputs (the first version of mobile app and its test cases and the third version of HandicApp) to generate the new version of test cases?

5.5 Analyzing the evolution (Calling EMF Compare graphically/ programmatically)

In order to analyze the evolution of different applications we have tried to use EMF compare graphically but first we have made a reverse engineering from java to XMI models using MoDisco, a framework in Eclipse added by Eclipse Marketplace. Calling EMF compare graphically or programmatically requires a meta-model of a mobile application. We have focused on rec and src components it means the java and XML files that require a java Meta model and XML Meta model.

5.5.1 Using EMF Compare Graphically

First we have started by making a reverse engineering of res files (strings.xml to strings.xmi and strigns2.xml to strings2.xmi) and we have used EMF compare graphically we obtained this results For the example of changing the content of a button listen to Lina as shows Figure 5.10, text to speech to Write to hear,text Restart to Text Repeat, text speech recognition button to speech recognition.

We have saved these results in a comparison xmi model, the format of this model is not useful. For the example of deleting a button listen besides the XML files we should make a

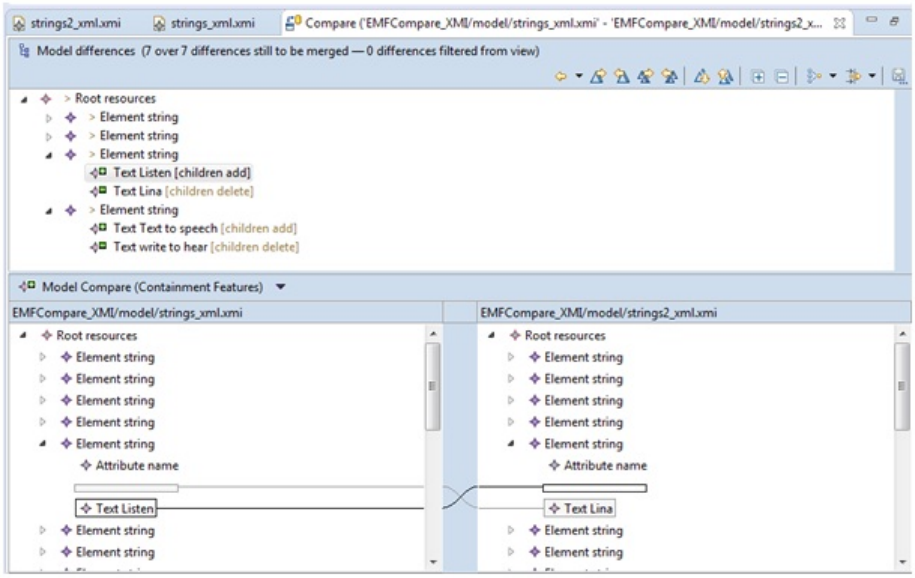

Figure 5.11: The content of a button listen to Lina

reverse engineering of source files i.e. java classes (HandicApp-java.java and HandicAppdelete-java.java) to (HandicApp-java.xmi and HandicAppdelete.xmi) graphically always and we get those differences, here the delete affects the two folders rec and src because any method calls this button will be deleted. As a results EMF compare displays 548 differences between the two java.xmi models, It is a little strange for one deleted button, the investigations have prooved that in graphical comparison EMF compare makes an error which is taking the order of elements into consideration when doing comparison and MoDisco when makes a reverse engineering changes the order. i.e by this strategy for the same version we can find many obsolete, useless differences.

Toile 2 Vert

In order to valorize the proposed approach we wanted to use a concrete example from the industry but unfortunately we have found a problem in the type of internet of Nantes University that prevents us from exploiting the application.

5.5.2 Using EMF compare programmatically

Because of many problems in EMF compare graphical comparison we have decided to use the programmatical concept of this Framework. Because here the developer is the leader and he controls the whole comparison process.

5.6 Conclusion

Until now the obtained results encouraging us to continue this research project but many results can be contemplated such us: the analysis and adaptation processes, development of a tool of automatic repairing test cases according the evolution of software systems will be a great revolution in the industry, other perspectives require more investigation especially in the domain of mobile apps that has never been dealt with before.

CHAPTER 6

CONCLUSION AND FUTURE WORKS

6.1 Conclusion

The main goal of this research studies was to support reasoning about mobile application evolution; in order to reason about the adaptation of test cases; researchers require evolutionary information about systems evolution i.e. different commits made by developers. One way to acquire that information is by analyzing the artifacts stored at the repository of change management systems such as: Git, GitHub, SVN, ...etc. During five months of this train ship we tried to answer to many research questions such as how mobile applications evolve, what kind of evolution they can do? did their evolution affects on the generated test cases? how can we test them? are there any testing paradigms or techniques?, what is the best one? what is the difference between our proposition and other related works? This thesis provides four contributions which are discussed bellow:

In chapter 2 we have defined taxonomy of Mobile apps in the context of software testing. The categorization was based on two sections; in the first one we have presented a general overview of mobile application and in the second section we have focused on different testing techniques, we have concluded that there are many testing techniques, each one has pros and cons associated with it, and there is not a single testing method that is completely satisfying, and the best way is to consider a testing strategy that combines different testing options that as a whole provide the best overall testing result especially during the evolution of mobile apps. In chapter 3 we have established a taxonomy for the evolution of both mobile application and their test cases which is supported by Lehman laws for software evolution, different kind of changes, the analyzing tools for software evolution and the factors influences on the mobile apps evolution, the ecosystem, context of an application, the second part of this chapter treat the evolution of test cases and how

to repair, adapt obsolete test cases to be useful. In Chapter 4 we have resulted the architecture of the proposed approach based on many tools, frameworks, IDE, Testing techniques, many languages. Chapter 5 we have experimented our approach on one small case study by using android native mobile applications, where we have used the serialization of different versions to an XMI, the study of this language has highlighted some problems first of all the fact that the same model could be represented by different XMI productions, we have different pattern of serialization and different versions which means that we can't compare a set of any XMI files. This consideration forced us to define some restrictions which have to be satisfied in order to consider this work valid. For our case we have serialized the java classes and XML files (res and src) to XMI model that conforms to XMI MoDisco meta model , then we have used EMF compare graphically, the outputs of graphical comparison (which is verified by a set of examples), we can see how this XMI approach suffers from the lack of semantic information we find that the XMI of MoDisco Pattern generates for the same original version many XMI files where the difference between them is the order of elements and EMF compare by his turn during comparison process consider this disorder as a change from version to another, all of this problems forced us to migrate to use EMF compare programmatically, this program works deeply because we can see the kind of change and the matched elements and the state of change.

6.2 Future Works

The results presented until now and comparing to proposed approaches of previous works encourages us to :

- Continue the second part of the global process: adaptation of test cases using java or constraint programming with kermeta language in order to get the new version of test cases that covers the new evolved mobile application.
- Developing a tool that adapt and repair old test cases based on Model Driven Engineering for android native mobile apps.
- Generalizing the approach and apply it on other types of mobile apps such as web, hybrid applications.
- Generalizing the approach and apply it on other operating Systems: iOS, windows phone,...etc

APPENDIX

Reverse Engineering: Java—> UML using Visual paradigm

1. Handicapp

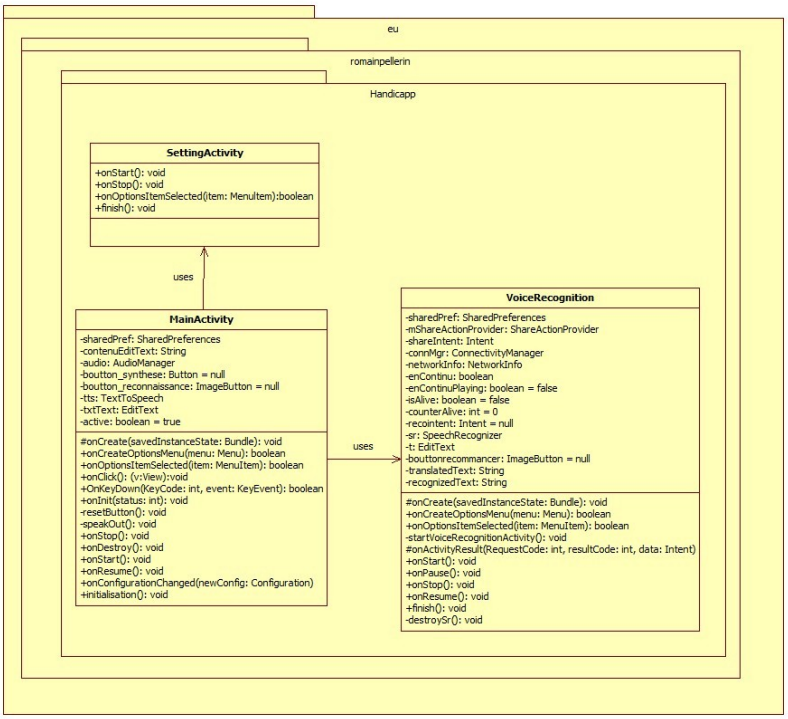

Figure A.1: UML class diagram of HandicApp Application..

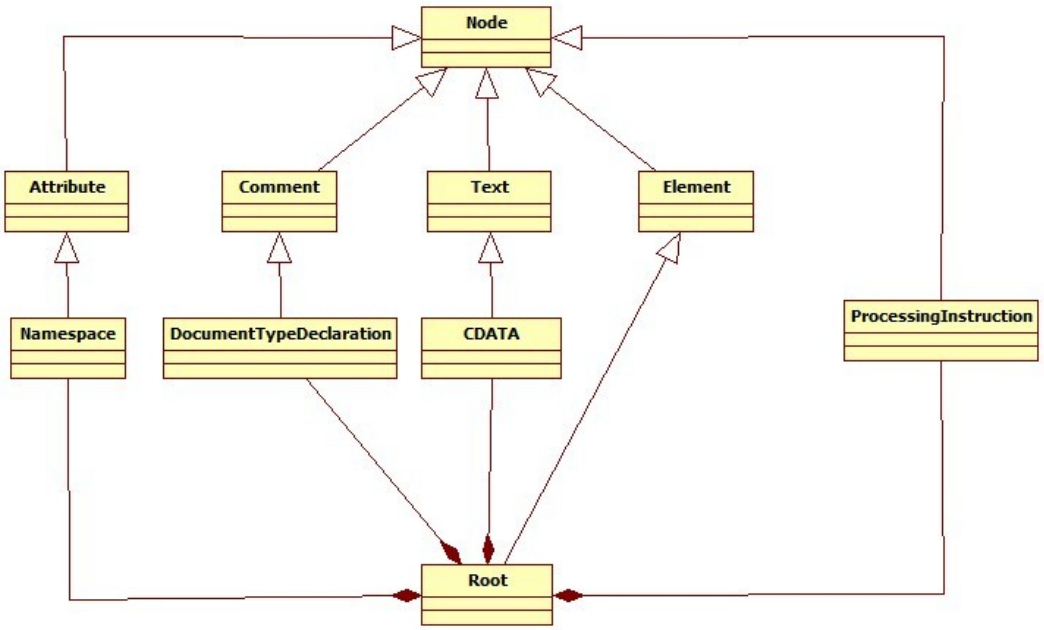

Figure A.2: XML Meta-model.

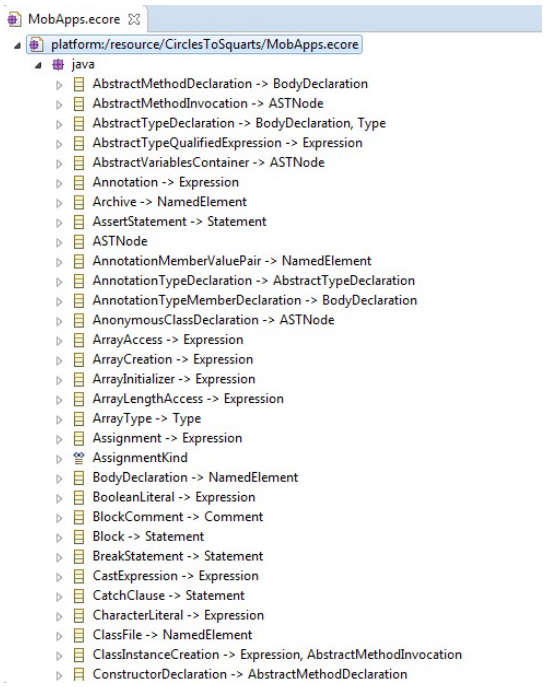

Figure A.3: Portion of Java Meta-model.

```
import java.io.File;
import java.util.List;

import mobapps.JavaPackage;
import mobapps.impl.JavaPackageImpl;

import org.eclipse.emf.common.util.BasicMonitor;
import org.eclipse.emf.common.util.URI;
import org.eclipse.emf.compare.Comparison;
import org.eclipse.emf.compare.Diff;
import org.eclipse.emf.compare.EMFCompare;
import org.eclipse.emf.compare.extension.merge.IMerger;
import org.eclipse.emf.compare.match.DefaultComparisonFactory;
import org.eclipse.emf.compare.match.DefaultEqualityHelperFactory;
import org.eclipse.emf.compare.match.DefaultMatchEngine;
import org.eclipse.emf.compare.match.IComparisonFactory;
import org.eclipse.emf.compare.match.IMatchEngine;
import org.eclipse.emf.compare.match.eobject.IEObjectMatcher;
import org.eclipse.emf.compare.merge.BatchMerger;
import org.eclipse.emf.compare.merge.IBatchMerger;
import org.eclipse.emf.compare.merge.IMerger;
import org.eclipse.emf.compare.scope.DefaultComparisonScope;
import org.eclipse.emf.compare.match.impl.MatchEngineFactoryImpl;
import org.eclipse.emf.compare.match.impl.MatchEngineFactoryRegistryImpl;
import org.eclipse.emf.compare.scope.IComparisonScope;
import org.eclipse.emf.compare.utils.UseIdentifiers;
import org.eclipse.emf.ecore.resource.Resource;
import org.eclipse.emf.ecore.resource.ResourceSet;
import org.eclipse.emf.ecore.resource.impl.ResourceSetImpl;
import org.eclipse.emf.ecore.xmi.impl.XMIResourceFactoryImpl;
import org.eclipse.emf.ecore.xmi.impl.XMIResourceFactoryImpl;


public class Compare{

public void compare(File model1, File model2) {

URI uri1 = URI.createFileURI ("D:/Evolution/EMFCompare_MobApps/model/HandicApp_java.xmi");
URI uri2 = URI.createFileURI ("D:/Evolution/EMFCompare_MobApps/model/HandicAppdelete_java.xmi");


Resource.Factory.Registry.INSTANCE.getExtensionToFactoryMap ().put ("xmi", new
XMIResourceFactoryImpl());
ResourceSet resourceSet1  = new ResourceSetImpl();
ResourceSet resourceSet2  = new ResourceSetImpl();

  resourceSet1.getPackageRegistry().put(JavaPackage.eNS_URI, JavaPackage.eINSTANCE);

  resourceSet2.getPackageRegistry().put(JavaPackage.eNS_URI,JavaPackage.eINSTANCE);

  resourceSet1.getResource(uri1, true);
  resourceSet2.getResource(uri2, true);
```

Figure A.4: Calling EMF Compare Programmatically.

```
//IComparisonScope scope = new DefaultComparisonScope(resourceSet1, resourceSet2);
IComparisonScope scope = EMFCompare.createDefaultScope(resourceSet1, resourceSet2);
Comparison comparison = EMFCompare.builder().build().compare(scope);

List<Diff> differences = comparison.getDifferences();
 for
( Diff d : differences){


                System.err.println("d.getKind(): "+d.getKind());
                System.err.println("d.getMatch(): " + d.getMatch());
                System.err.println("State: " + d.getState());
                }

        }
//main method
        public static void main(String[] args) {
                Compare ct = new Compare();
                ct.compare(null, null);
        }

}
```

Figure A.5: Calling EMF Compare Programmatically.

Figure A.6: Comparison results part-1.

```
d.getMatch(): MatchSpec{left=SingleVariableDeclaration@b19b79 savedInstanceState,
right=SingleVariableDeclaration@1a54932 savedInstanceState, origin=<null>,
#differences=2, #submatches=3}
d.getKind(): ADD
State: UNRESOLVED
d.getKind(): DELETE
d.getMatch(): MatchSpec{left=SingleVariableDeclaration@b19b79 savedInstanceState,

right=SingleVariableDeclaration@1a54932 savedInstanceState, origin=<null>,
#differences=2, #submatches=3}
State: UNRESOLVED
d.getKind(): CHANGE
d.getMatch(): MatchSpec{left=TypeAccess@1b204a1, right=<null>, origin=<null>,
#differences=1, #submatches=0}
State: UNRESOLVED
d.getKind(): CHANGE
```

Figure A.7: Comparison results part-2.

```
<? xml version="1.0" encoding="ASCII"?>

<compare: Comparison xmi: version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:compare="http://www.eclipse.org/emf/compare">

 <matchedResources
leftURI="platform:/resource/EMFCompare_MobApps/model/HandicApp_java.
xmi"
rightURI="platform:/resource/EMFCompare_MobApps/model/HandicAppdelet
e_java.xmi"/>

 <matches>

  <submatches>

   <submatches>

          <left
href="platform:/resource/EMFCompare_MobApps/model/HandicApp_java.xm
i#//@ownedElements.0/@ownedPackages.0/@ownedPackages.0/@ownedEle
ments.2/@bodyDeclarations.13/@annotations.1/@values.0/@value"/>

          <right
href="platform:/resource/EMFCompare_MobApps/model/HandicAppdelete_ja
va.xmi#//@ownedElements.0/@ownedPackages.0/@ownedPackages.0/@own
edElements.1/@bodyDeclarations.12/@annotations.1/@values.0/@value"/>

         </submatches>
```

Figure A.8: Part of the comparison XMI model produced from the graphical comparison.

BIBLIOGRAPHY

[1] D. AMALFITANO, A. R. FASOLINO, AND P. TRAMONTANA, *Reverse engineering finite state machines from rich internet applications*, in Reverse Engineering, 2008. WCRE'08. 15th Working Conference on, IEEE, 2008, pp. 69–73.

[2] D. AMALFITANO, A. R. FASOLINO, AND P. TRAMONTANA, *A gui crawling-based technique for android mobile application testing*, in Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on, IEEE, 2011, pp. 252–261.

[3] D. AMALFITANO, A. R. FASOLINO, P. TRAMONTANA, S. DE CARMINE, AND A. M. MEMON, *Using gui ripping for automated testing of android applications*, in Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, ACM, 2012, pp. 258–261.

[4] D. AMALFITANO, A. R. FASOLINO, P. TRAMONTANA, AND B. ROBBINS, *Testing android mobile applications: Challenges, strategies, and approaches.*, Advances in Computers, 89 (2013), pp. 1–52.

[5] D. AMALFITANO, A. R. FASOLINO, P. TRAMONTANA, B. TA, AND A. MEMON, *Mobiguitar-a tool for automated model-based testing of mobile apps*, (2014).

[6] S. ANDERSON, *Mutation testing*, 2011.
http://goo.gl/578c5K.

[7] P. ANDRE, J.-M. MOTTU, AND G. ARDOUREL, *Building test harness from service-based component models*, in MoDeVVa 2013 Workshop on Model Driven Engineering, Verification and Validation, vol. 1069, 2013, pp. pp–11.

[8] V. BEAL, *Api - application program interface*, 2015.
http://goo.gl/NlGDU8.

[9] BEAPP, *Toile2vert*, 2015.
https://goo.gl/zuARzc.

[10] F. BELLI, C. J. BUDNIK, AND L. WHITE, *Event-based modelling, analysis and testing of user interactions: approach and case study*, Software Testing, Verification and Reliability, 16 (2006), pp. 3–32.

[11] J. BÉZIVIN, *On the unification power of models*, Software & Systems Modeling, 4 (2005), pp. 171–188.

[12] H. BRUNELIERE, J. CABOT, G. DUPÉ, AND F. MADIOT, *Modisco: A model driven reverse engineering framework*, Information and Software Technology, 56 (2014), pp. 1012–1032.

[13] R. BUDIU, *Mobile: Native apps, web apps, and hybrid apps*, 2013. http://goo.gl/ZEsfGJ.

[14] BUSINESSINSIDER, *Android 5.0‚Äùlollipop‚Äù homescreen*, 2014. http://goo.gl/3yyeu1.

[15] CAMERONMCKENZIE, *Evolutionofmobileapps*, 2012. http://goo.gl/eNJl5k.

[16] N. CHAPIN, J. E. HALE, K. M. KHAN, J. F. RAMIL, AND W.-G. TAN, *Types of software evolution and software maintenance*, Journal of software maintenance and evolution: Research and Practice, 13 (2001), pp. 3–30.

[17] G. CHEN AND D. KOTZ, *A survey of context-aware mobile computing research (tech. rep.)*, Hanover, NH, USA, (2000).

[18] W. CHOI, G. NECULA, AND K. SEN, *Guided gui testing of android apps with minimal restart and approximate learning*, in ACM SIGPLAN Notices, vol. 48, ACM, 2013, pp. 623–640.

[19] P. COSTA, A. C. PAIVA, AND M. NABUCO, *Pattern based gui testing for mobile applications*, in Quality of Information and Communications Technology (QUATIC), 2014 9th International Conference on the, IEEE, 2014, pp. 66–74.

[20] B. DANIEL, D. DIG, T. GVERO, V. JAGANNATH, J. JIAA, D. MITCHELL, J. NOGIEC, S. H. TAN, AND D. MARINOV, *Reassert: a tool for repairing broken unit tests*, in Proceedings of the 33rd International Conference on Software Engineering, ACM, 2011, pp. 1010–1012.

[21] B. DANIEL, T. GVERO, AND D. MARINOV, *On test repair using symbolic execution*, in Proceedings of the 19th international symposium on Software testing and analysis, ACM, 2010, pp. 207–218.

[22] B. DEPOORTERE, *Reasoning about first-class changes for support in software evolution*, PhD thesis, 2007.

[23] DEVELOPERS, *Android studio overview*, 2013.
https://goo.gl/RCXoDB.

[24] S. DEVELOPERS, *Android studio overview*, 2014.
http://goo.gl/B1soC3.

[25] D. DIG, S. NEGARA, V. MOHINDRA, AND R. JOHNSON, *Reba: re factoring-aware b inary a daptation of evolving libraries*, in Proceedings of the 30th international conference on Software engineering, ACM, 2008, pp. 441–450.

[26] J. DURANDO, *Usa today network*, 2014.
http://goo.gl/acpzkt.

[27] ECLIPSE, *Emf compare ‚Äî developer guide*, 2011.
http://goo.gl/53XASj.

[28] J.-M. FAVRE, *Meta-model and model co-evolution within the 3d software space*, in ELISA: Workshop on Evolution of Large-scale Industrial Software Applications, 2003, pp. 98–109.

[29] D. FIRESMITH, *Using v models for testing*, 2013.
http://goo.gl/JUYTLW.

[30] B. FLING, *Mobile Design and Development: Practical concepts and techniques for creating mobile sites and web apps*, " O'Reilly Media, Inc.", 2009.

[31] M. GORNOI, *6 key challenges of mobile app testing*, 2014.
https://testlio.com/http://goo.gl/FQznk5/post/6-key-challenges-of-mobile-app- testing.

[32] R. GROZ, M.-N. IRFAN, AND C. ORIAT, *Algorithmic improvements on regular inference of software models and perspectives for security testing*, in Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change, Springer, 2012, pp. 444–457.

[33] W. G. HALFOND AND A. ORSO, *Automated identification of parameter mismatches in web applications*, in Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, ACM, 2008, pp. 181–191.

[34] I. HERRAIZ, D. RODRIGUEZ, G. ROBLES, AND J. M. GONZALEZ-BARAHONA, *The evolution of the laws of software evolution: a discussion based on a systematic literature review*, ACM Computing Surveys (CSUR), 46 (2013), p. 28.

[35] C. HU AND I. NEAMTIU, *A gui bug finding framework for android applications*, in Proceedings of the 2011 ACM Symposium on Applied Computing, ACM, 2011, pp. 1490–1491.

[36] JAYESH, *Mobile application lifecycle management*, 2012. http://goo.gl/uAV9kp.

[37] H. K. KIM, *Test driven mobile applications development*, in Proceedings of the World Congress on Engineering and Computer Science, vol. 2, 2013.

[38] B. KIRUBAKARAN AND V. KARTHIKEYANI, *Mobile application testing,Äîchallenges and solution approach through automation*, in Pattern Recognition, Informatics and Mobile Engineering (PRIME), 2013 International Conference on, IEEE, 2013, pp. 79–84.

[39] M. LEHMAN AND J. F. RAMIL, *Software evolution in the age of component-based software engineering*, IEE Proceedings-Software, 147 (2000), pp. 249–255.

[40] M. M. LEHMAN, J. F. RAMIL, P. D. WERNICK, D. E. PERRY, AND W. M. TURSKI, *Metrics and laws of software evolution-the nineties view*, in Software Metrics Symposium, 1997. Proceedings., Fourth International, IEEE, 1997, pp. 20–32.

[41] T. C. S. LIMITED, *Mobile telecommunications: Telecom technology evolution*, 2015. http://goo.gl/Xd6SQi.

[42] LINA, *Laboratoire d'informatique de nantes atlantique*, 2015. https://goo.gl/6mVTFf.

[43] Z. LIU, X. GAO, AND X. LONG, *Adaptive random testing of mobile application*, in Computer Engineering and Technology (ICCET), 2010 2nd International Conference on, vol. 2, IEEE, 2010, pp. V2–297.

[44] A. MARCHETTO, P. TONELLA, AND F. RICCA, *State-based testing of ajax web applications*, in Software Testing, Verification, and Validation, 2008 1st International Conference on, IEEE, 2008, pp. 121–130.

[45] A. MEMON, *An event-flow model of gui-based applications for testing*, Software testing, verification and reliability, 17 (2007), pp. 137–157.

[46] A. MEMON, I. BANERJEE, AND A. NAGARAJAN, *Gui ripping: Reverse engineering of graphical user interfaces for testing*, in null, IEEE, 2003, p. 260.

[47] A. M. MEMON, *Using tasks to automate regression testing of guis*, in International Conference on Artificial intelligence and Applications (AIA 2004), Innsbruck, Austria, 2004.

[48] A. M. MEMON AND M. L. SOFFA, *Regression testing of guis*, ACM SIGSOFT Software Engineering Notes, 28 (2003), pp. 118–127.

[49] A. M. MEMON AND Q. XIE, *Studying the fault-detection effectiveness of gui test cases for rapidly evolving software*, Software Engineering, IEEE Transactions on, 31 (2005), pp. 884–896.

[50] R. MINELLI AND M. LANZA, *Software analytics for mobile applications–insights & lessons learned*, in Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on, IEEE, 2013, pp. 144–153.

[51] M. MIRZAAGHAEI, F. PASTORE, AND M. PEZZE, *Supporting test suite evolution through test case adaptation*, in Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on, IEEE, 2012, pp. 231–240.

[52] M. MIRZAAGHAEI, F. PASTORE, AND M. PEZZÈ, *Automatic test case evolution*, Software Testing, Verification and Reliability, 24 (2014), pp. 386–411.

[53] MONKEYTALK, *Cloudmonkey*, 2015.
https://goo.gl/FrrXBY.

[54] J. MOORE, *mobile simulators and emulators an update*, 2015.
http://goo.gl/fK3myE.

[55] H. MUCCINI, A. D. FRANCESCO, AND P. ESPOSITO, *Software testing of mobile applications: Challenges and future research directions*, in Automation of Software Test (AST), 2012 7th International Workshop on, IEEE, 2012, pp. 29–35.

[56] OBEO, *Hereweareemfcompare2*, 2013.
http://goo.gl/Oi5jq0.

[57] T. E. F. OPEN SOURCE COMMUNITY, *Eclipse*, 2015.
http://goo.gl/MkIRN2.

[58] J. PAN, *Software testing*, 1999.
http://goo.gl/Wosf3X.

[59] R. PELLERIN, *Handicapp*, 2013.
https://goo.gl/UVVlOK.

[60] S. PEREZ, *Mobile app usage increases in 2014, as mobile web surfing declines*, 2014.
http://goo.gl/VjqUUW.

[61] S. PERSON, M. B. DWYER, S. ELBAUM, AND C. S. PASAREANU, *Differential symbolic execution*, in Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, ACM, 2008, pp. 226–237.

[62] R. M. . N. C.-H. PHILIP JAMES KITCHEN, *Long term evolution mobile services and intention to adopt: a malaysian perspective*, 2015.
http://goo.gl/6SFknQ.

[63] E. PHILLIPS, *The evolution of the mobile app landscape*, 2013.
http://goo.gl/TYuL1B.

[64] L. S. PINTO, S. SINHA, AND A. ORSO, *Testevol: a tool for analyzing test-suite evolution*, in Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, 2013, pp. 1303–1306.

[65] U. RESEARCH INSTITUTE IN SOFTWARE EVOLUTION (RISE). DURHAM, *Dept of computer science, university of durham.*, 1999.
http://goo.gl/eViU6u.

[66] H. REZA, S. ENDAPALLY, AND X. GRANT, *A model-based approach for testing gui using hierarchical predicate transition nets*, in Information Technology, 2007. ITNG'07. Fourth International Conference on, IEEE, 2007, pp. 366–370.

[67] M. ROUSE, *Testing as a service (saas)*, 2014.
http://goo.gl/vHr0KY.

[68] M. SATYANARAYANAN, *Fundamental challenges in mobile computing*, in Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing, ACM, 1996, pp. 1–7.

[69] T. SOLUTION, *Monkeytalk android user manual*, 2013.
http://goo.gl/pjXeJm.

[70] F. STORTONI, *More about model based testing*, 2012.
http://cqaa.org/Resources/Documents/Presentations

[71] M. TORCHIANO, F. RICCA, AND A. DE LUCIA, *Empirical studies in software maintenance and evolution*, in Software Maintenance, 2007. ICSM 2007. IEEE International Conference on, IEEE, 2007, pp. 491–494.

[72] A. VIVION, *Beapp*, 2011.
http://goo.gl/KpAcVO.

[73] L. VOGEL, *Introduction to android development with android studio*, 2015.
http://goo.gl/prSYLm.

[74] A. I. WASSERMAN, *Software engineering issues for mobile application development*, in Proceedings of the FSE/SDP workshop on Future of software engineering research, ACM, 2010, pp. 397–400.

[75] Z. XING AND E. STROULIA, *Refactoring practice: How it is and how it should be supported-an eclipse case study*, in Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on, IEEE, 2006, pp. 458–468.

[76] Z. XU, Y. KIM, M. KIM, G. ROTHERMEL, AND M. B. COHEN, *Directed test suite augmentation: techniques and tradeoffs*, in Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering, ACM, 2010, pp. 257–266.

[77] D. ZHANG AND B. ADIPAT, *Challenges, methodologies, and issues in the usability testing of mobile applications*, International Journal of Human-Computer Interaction, 18 (2005), pp. 293–308.

[78] J. ZHANG, S. SAGAR, AND E. SHIHAB, *The evolution of mobile apps: An exploratory study*, in Proceedings of the 2013 International Workshop on Software Development Lifecycle for Mobile, ACM, 2013, pp. 1–8.