DOCTORAL THESIS

Mobile Application Testing Using Behavior-Driven Development Based Specifications for Android Platform

Zulfiqar Ali

Graz, 2019

Institute for Software Technology
Graz University of Technology

Supervisor/First reviewer: Univ.-Prof. Dipl.-Ing. Dr. techn. Wolfgang Slany
Second reviewer: Univ.-Prof. Dipl.-Ing. Dr. techn. Thomas Grechenig

Abstract (English)

Smartphones play an important role in our daily lives and are replacing regular desktop computers. The rapid advancement of mobile computing technology and the rising usage of mobile apps made our daily life more productive. Mobile apps should operate all the time without bugs in order to improve user satisfaction and offer great business value to the end user. In addition, mobile applications are essential globally and should be available in all languages including bidirectional languages. At the same time, smartphones are full of special features that make testing of apps more challenging. Automatic tests are needed for durable and high-quality software to check normal bugs and localization bugs in the application. Actually, the quality is a must for successful applications and cannot be achieved without testing and verification. Reliable and high-quality mobile applications require different effective testing approaches.

Over the past few years, mobile applications testing techniques, methodologies, and methods attract the attention of software engineers. Among them, Behavior-Driven Development (BDD) has become a popular agile software development methodology. The main success behind BDD lies in the executable acceptance tests that describe the behavior of features, its acceptance standard used in the form of scenarios and a syntax readable by business people. The Catrobat project developed a free and open source visual programming environment for smartphones. It was developed for educational and learning purposes to help students. It enables students/users to learn to build their own animations, games, stories, and many types of other apps solely on smartphones. The visual elements of Catrobat are arranged on-screen in order to build scripts. In this thesis, we present the BDD methodology and Cucumber framework to automate regression testing for an Android app (Catroid). Furthermore, BDD scenarios are used to identify the system behavior and to document system tests that will be executed through the app.

Furthermore, the goal of our study is to present multi-threading related issues and challenges of visual programming languages (VPL) in Catroid. Catrobat program scripts communicate via a broadcast mechanism. The objective is to test the broadcast mechanism from different angles and track regression errors as well as specify and diagnose bugs with the help of executable specifications. This thesis also utilizes existing methods to test issues of bidirectional Android app (Pocket Code), especially for the right-to-left (RTL) languages such as Urdu, Arabic, or

Pashto. We also present some critical issues and challenges in the Catrobat project w.r.t. bidirectional languages. The results show that the methods are able to effectively reveal deficiencies in broadcast mechanism/bidirectional language issues and ensure that the app meets end-users expectations and needs, and discuss our experience.

Abstract (German)

Mobiltelefone spielen eine wichtige Rolle in unserem tglichen Leben und ersetzen langsam herkmmliche Desktop-Computer. Die rasante Weiterentwicklung der mobilen Computertechnologie und die zunehmende Verwendung mobiler Apps gestallten unseren Alltag produktiver. Um die Benutzerzufriedenheit zu verbessern sollten die mobilen Apps fehlerfrei funktionieren, und dem Endbenutzer einen hohen Unternehmenswert bieten. Darber hinaus sind mobile Anwendungen von zentraler Bedeutung und sollten in allen Sprachen verfgbar sein, einschlielich bidirektionaler Sprachen. Gleichzeitig sind Smartphones mit speziellen Funktionen ausgestattet, die das Testen von Apps schwieriger machen. Fr die dauerhafte und qualitativ hochwertige Software sind automatische Tests erforderlich, um die normalen Fehler und Lokalisierungsfehler in der Anwendung zu berprfen. Eigentlich ist die Qualitt ein Muss fr erfolgreiche Anwendungen und kann nicht ohne Testen und Verifizieren erreicht werden.

Zuverlssige und qualitativ hochwertige mobile Anwendungen erfordern verschiedene effektive Testanstze. In den letzten Jahren haben die Testtechniken, -methoden und -verfahren fr mobile Anwendungen die Aufmerksamkeit von Software-Ingenieuren auf sich gezogen. Unter ihnen ist Behavior-Driven Development (BDD) zu einer beliebten agilen Softwareentwicklungsmethodik geworden. Der Haupterfolg hinter BDD liegt in den ausfhrbaren Abnahmetests, die das Verhalten von Features und deren Annahmestandard beschreiben, die in Form von Szenarien und einer von Geschftsleuten lesbaren Syntax beschrieben werden. Das Catrobat-Projekt entwickelte eine freie visuelle Programmierumgebung fr Smartphones. Es wurde fr Bildungs- und Lernzwecke entwickelt, um den Schlern zu helfen. Benutzerinnen und Benutzer knnen eigenen Animationen, Spiele, Geschichten und viele andere Apps nur auf den Smartphones erstellen. Die visuellen Elemente von Catrobat werden auf dem Bildschirm angeordnet, um Skripts zu erstellen.

In dieser Arbeit stellen wir die BDD-Methodik und das Cucumber-Framework vor, um Regressionstests fr eine Android-App (Catroid) zu automatisieren. Darber hinaus werden BDD-Szenarien verwendet, um das Systemverhalten zu identifizieren und Systemtests zu dokumentieren, die ber die App ausgefhrt werden. Ein weiteres Ziel unserer Studie ist es, die Probleme und Herausforderungen von Multi-Threading Mechanismen visueller Programmiersprachen (VPL) in Catroid aufzuzeigen. Die Catrobat-Programm-Skripte kommunizieren ber einen Broadcast-

Mechanismus. Ziel ist es, den Broadcast-Mechanismus aus verschiedenen Blickwinkeln zu testen und Regressionsfehler zu entdecken, sowie Fehler anhand ausfhrbarer Spezifikationen zu spezifizieren und zu diagnostizieren.

In dieser Arbeit werden auch vorhandene Methoden verwendet, um Probleme der bidirektionalen Android-App (Pocket Code) zu testen, insbesondere fr die RTL-Sprachen (Urdu, Arabisch oder Pashto). Wir stellen hier auch einige kritische Probleme und Herausforderungen im Catrobat-Projekt hinsichtlich der bidirektionale Sprachen dar. Die Ergebnisse zeigen, dass die Methoden in der Lage sind, Mngel im Broadcast-Mechanismus / in bidirektionalen Sprachproblemen effektiv aufzudecken, um sicherzustellen, dass die App die Erwartungen und Bedrfnisse der Endbenutzer erfllt. Schlielich diskutieren wir unsere Erfahrungen.

I dedicate this work to the memory of my sister,
Raida Bano (M.Sc Physics)
May Allah rest her souls in Paradise.

Publication

Accepted Papers.

- Zulfiqar Ali, Behavior-Driven Development as an Error-Reduction Practice for Mobile Application Testing, bearing paper id 'IJCSI-2019-16-2-12403' has been accepted for publication in IJCSI Volume 16, Issue 1, March 2019.

- Zulfiqar Ali, Aiman Awwad, Wolfgang Slany, Using Executable Specification and Regression Testing for Broad-cast Mechanism of Visual Programming Language on Smartphones" International Journal of Interactive Mobile Technologies. "International Journal of Interactive Mobile Technologies (ijim), ISSN: 1865-7923.

- Aiman A, Christian S, KK Luhana, Zulfiqar Ali, Improving Pocket Paint Usability via Material Design Compliance and Internationalization & Localization Support On Application Level" 19th International Conference on Human-Computer Interaction with Mobile Devices and Services, 4th-7th September, 2017, Vienna, Austria. doi:10.1145/3098279.3122142.

Under Review.

- Zulfiqar Ali, Bidirectional Languages Issues, Challenges and Testing for Android Apps with Behavior Driven Development. Under review (Manuscript Number SQJO-18-00112) Springer (Software Quality Journal) (https://www.springer.com/computer/swe/journal/11219).

- Zulfiqar Ali and Wolfgang Slany, Broadcast Mechanism Challenges of Visual Programming Language for Android, Using Behavior-Driven Development. Under review (MS number 5482471.vl) Hindawi journals for Mobile Information Systems (https://www.hindawi.com/journals/misy

Under progress.

- Aiman Awwad, Zulfiqar Ali, Behavior-Driven Development for Android Location Based Services, 2nd International Conference on Applied Mathematics and Computer Science. Lisbon, Portugal, April 12-14 2019.

- Zulfiqar Ali, Thomas Hirsch, Wolfgang Slany, The Catrobat Visual Programming Language Tests

Acknowledgment

In the name of Almighty who has guided and helped me to achieve this great goal of my life. Deepest appreciations are extended to my beloved parents and my elder brother Professor Javid Hussain, who have always inspired me towards my studies. It was their consistent support which has enabled me to avail the opportunity to study in Austria. A special thanks, beyond my family members, is extended to a number of people whose continuous support and motivation is worthy of appreciation at this auspicious occasion.

First and foremost, I feel much pleasure in expressing my heartiest gratitude to my ever-affectionate supervisor, Prof. Wolfgang Slany, for his dynamic and friendly supervision, consistent support, valuable advice and perceptive suggestions throughout the development of this thesis. It was only through his guidance that my efforts were fruitful. I can never forget Prof. Slany's support in all my life. I find no words of thanks for my family, especially for my supportive wife and my little stars Jeyia Ali and Anzalna Ali. Their share in my studies is beyond words, and my achievement is more than everything for them. They have been with me from the very beginning of this work, and their love and kindness have helped me to attain my goals. I thank them for their great support and encouragement in my Ph.D. journey.

I am also particularly grateful to my external reviewer, Prof. Thomas Grechenig, for his invaluable suggestions for improving my work. Sincere thanks to all my friends, especially Ali Qureshi, Muhammad Abdul Wahid, Muslim Khan, Sami Khan, Shaban AliJunaid, Shakeel Khan, Zahid Khan, Numan Khan and Muntazir Abbas among others, for their kindness and moral support during the course my studies. I consider myself a lucky person to have friends like them. The moments I spent with them will remain memorable in my life.

I am really grateful for the support of Catrobat team, especially of Vesna Krnjic, Bernadette Spieler and Christian Schindler. I would like to thank all my collegaues at the Graz University of Technology, especially Martin Kandlhofer for his best wishes, help, and support. I would also like to extend a special thank to the Afro Asia Institute for their extraordinary services in terms of my accommodation. Special thanks to Petra Imre, Johannes Mindler-Steiner, Claudia Unger, and Evelyn Tschernko for their best wishes and support.

Zulfiqar Ali Graz, 2019

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, ____________________ __________________________

Place, Date Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ____________________ __________________________

Ort, Datum Unterschrift

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Overview

This chapter presents and brings together the research, scope of new work, needs and challenges, and highlights the new research contributions, which is explored within the study. It ends with the structure of the thesis outline.

1.2. Background and Motivation of the Thesis

Mobiles and its applications are very common everywhere in the world such as banking, health, and education. The ratio between the mobile phones are ten times more than PCs. Especially for the children, this ratio is much more marked. So the quality, standard, and testing of the applications are very important [1, 2].

According to ABI Research, the revenue generating from mobile application testing tools excels the automation part. By the end of 2017, this change in revenue is noted nearer to $800 million[1]. According to Sogeti World Quality Report, over the last three years tests for mobile applications have increased from 36% in 2015 to 52% of projects in 2017. In this report, approximately half of the respondents (47%) do not have the right process/method and (46%) do not have the right tools to test the mobile applications[2]. Mobile application testing is very costly, time-consuming and a difficult process [3].

Developing mobile apps is challenged by the demand to keep moving with matching user-needs and short release cycles, while still providing high-quality software [4]. Basically, mobile applications are mostly bugs prone because of the developers unfamiliarity with mobile platforms. However, the increasing complexity of mobile application can arise many challenges in the testing process in order to make sure the app operates and meets the users expectations. Smartphones

[1] https://www.abiresearch.com/press/200-million-mobile application-testing-market-boos/

[2] https://www.sogeti.com/globalassets/global/downloads/testing/wqr-2017-2018/wqr_2017_v9_secure.pdf

are becoming common; this exposes the necessity for effective techniques for testing their applications. Mobile application testing plays a vital role in making mobile applications more reliable and bug-free [5, 6]. Particularly, the complexity of the mobile application development and testing them in a mobile platform need a change in the traditional testing process. For these reasons, testing, and especially regression testing, is one of the most essential activities during app development [4]. Smartphones have different platforms such as Android and iOS. Mobile app development companies have to develop applications for each platform. Specifically , the Android market fragment is large and the several sets of scenarios, in which a mobile application can be used, makes the testing of a new application more costly, time-consuming and a complex task [6].

Mobile apps are developed to address more and more critical areas, which is not only complex to develop but difficult to test and validate. The difficulty, diversity and functional richness of smartphone apps are increasing and the demand for mobile apps, offering even more complex, rich, and usable functionalities, are going to grow more and more in the coming future. Unluckily, the quality of smartphone apps is often poor just because of the very fast growth and development processes in which testing activity is ignored [7]. In this thesis, we describe several open research issues in mobile applications. To resolve these issues, one needs specialized testing techniques and methods [8, 7]. However, the correctness, quality and testing of mobile application is essential. This why, our developers and designers are always focused on new technological challenges and end-user requirements, which are very important for the business [2].

During this continuous changing environment, the agile methodologies is one of the most important software development practice. It is implemented by users at certain points. This happen when the requirement of a user is not exact. It is also required when the closing date, time limit and funds are tight. On the other hand some extra requirements are shown and presented by the mobile apps. However, traditional software apps are rarely found with these requirement. It is also important to mention that there are some other less important extra requirements along with traditional apps such as interface with other apps, different kinds of mobile apps, compatibility factor in cross-platform, changing hardware complications, risks pertaining to security, confidentiality, user interfaces, testing difficulty, development process and mobile phone screen size [9].

The agile practice is realized as naturally suitable for smartphone app development process. Therefore, agile practice is considered an excellent option, which assures developmental processes of a softwares life-cycle at different stages as well as resolves issues more professionally [9]. Agile based methodology is becoming prominent in any software production for its higher chances of being successful. The Acceptance Test Driven Development (ATDD), Test Driven Development (TDD) and BDD are examples of widely used agile development processes[10]. TDD is a short development-cycle approach that depends on the agile practice. Mainly it is also used for writing automatic software tests before the functional codes are written and refactoring as well as continuous integration [11].

BDD is the evolution of TDD and ATDD, which is focused on the specifications of the system behavior that can be automated. The core objective of BDD is to get executable test scenarios. These scenarios are simply understandable and clearly written in a common language. BDD relies on ATDD, and provides an ubiquitous language for every participant to specify their tests as well as supporting several toolkits [11] such as Cucumber[12] and JBehave [13]. For BDD scenarios automation, we need frameworks to convert the natural language scenarios in actual code that runs the tests, i.e. Cucumber. These frameworks use Gherkin language as an interpreter, which accepts plain text. Practically, in scenarios, each statement is transformed into step-definition, which has the actual code that executes the parts of the software project being tested. All the keywords in Gherkin base scenarios automatically call methods/functions in the code (glue code) to run the tests [14]. BDD methodology uses a business readable language called Gherkin language to describe softwares behaviors while hiding its implementation details. In BDD, acceptance test scenarios are explicitly written down with the following Gherkin keywords, which is shown in Figure 1.1

Given Pre-condition,
When while event occurs,
Then make sure some outcome is achieved.

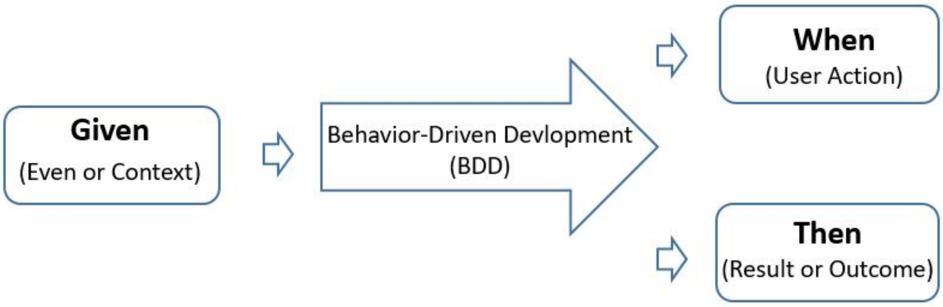


Figure 1.1.: BDD base Given-When-Then pattern

With the help of the above mentioned BDD base format (“Given”, “When” and “Then”) Cucumber scenarios are used specifically for collecting behavior. The Scenario is the combination of steps, in which BDD framework analyzes and executes the verification of software expectation. These specifications serve as a strong medium among all the stakeholders, i.e. developers, testers, and business analysts to increase quality assurance. All these specifications are also working as living documentations of the expected behaviors of the software [10].

In order to fascinate and attract unskilled developers to develop a simple software app, a Visual Programming Language (VPL) could ideally be used. It is defined that the programming language in which the end-user build/create programs elements graphically rather than textually. With VPL, programs are written visually, spatially (3-D or 4-D) or through arranging textual and graphical symbols. These could, however, be used as component/elements of syntax or as notation. There are more or less hundred different VPLs available with different motives. Most of them are frequently used in educational progression to support students to learn, visualize and understand common principles of algorithms [15].

In recent years, there are numerous countries, which are promoting programming education for school kids while some countries have already adopted programming as a formal subject in primary education. There are many Educational Programming Languages that have been developed for educational determinations and objective i.e., Scratch[3], Kodu[4], Snap![5] and Catrobat[6]. These VPLs are not only simple but also provide rich and attractive visual outcomes. In reality, most of them are visual programming languages, which are very famous in programming education for school children like primary school children and secondary school children, for the reason that they do not require knowledge of programming syntax and provide an environment where compile-time errors are nonexistent/absent. It is not only easy to learn but also introduces rich and charming visual interfaces [16]. On several circumstances, teenagers were being educated to program their own applications as well as design games for their classes in academic subjects, i.e, science, maths, and languages. There are a lot of excitement and fun using a visual programming language to build and create a simple game or app without any earlier/previous knowledge about programming, so therefore, not just being users but being developers as well.

Meanwhile, the free, open source visual programing languages (Catrobat) provide an easy possibility for the schoolchildren to learn the program without any programming knowledge and motivate them to create and share their own mobile apps. Children can easily learn how to program without having to think about drawbacks like compile-time errors or complicated workflows. Inspired by Scratch, Catrobat also defines blocks which can be snapped together in order to form a program. Unlike Scratch, Catrobat programs can be created and executed by entirely using smartphones [17, 18]. The version of Catrobat which is developed for Android smartphones is named Catroid, and is available on Google Play Store under the name Pocket Code' is a learning application for mobile devices that has been developed in Austria at the Graz University of Technology [19].

1.2.1. Why Behavior-Driven Development?

Many developers are confused using TDD and ATDD in their software projects. They wanted to know, what to test and what not to test, why a test fails, where to start, how much to test in one go, what to call their tests and how to understand the behavior[7] . In the early designs in TDD, the testers and developers work together. The project in which developers used TDD methodology, 40% reduction is noted in deficiencies. McConnell clarifies that in the requirement phase the defect injection rate can be 56% while in the design phase the rate can be 27%. Commonly,

[3] https://scratch.mit.edu/

[4] https://www.kodugamelab.com/

[5] http://snap.berkeley.edu/

[6] https://www.catrobat.org/

[7] https://dannorth.net/introducing-bdd/

these flaws are established at the later stage of the project, especially when the final product is available to the end user. With the help of technology to identify, the defects found earlier in the project will also reduce the cost. If these defects are not detected, they are shipped to the customers. Therefore, for the BDD approach, the quality control and cost reduction will be the main motivations [14].

During the requirement phase, BDD methodology combines the concept of TDD with the idea of writing test scenarios. The testers and developers well-defined functions and requirement description of the application. Inversely to TDD, the BDD test scenarios are written in natural languages, which facilitate the discussion with all project stakeholders including marketing and business people. These scenarios represent that how the end user will use the final product [14, 11]

Following the agile principles of Extreme Programming (XP) concerning testing and continuous integration (CI), the Catroid tests are run at least once a day on a Continuous Integration server. The tests cover functionality as well as UI design. Every Catroid developer is encouraged to follow the TDD principles and write tests before writing or changing any code. What is currently missing in the Catroid testing framework is a possibility to easily test behavior. Although it is possible to test the behavior with unit tests, they are often hard to maintain and hard to understand for people who are not proficient with technical knowledge. For this reason, Cucumber is established in Catroid as a behavior testing framework [20].

1.2.2. Behavior-Driven Testing with Cucumber in Catrobat

This section describes benefits using Cucumber as a behavior-driven framework in Catroid. The Cucumber testing framework is already introduced into Catrobat. The rest of the chapters discuss many aspects of using Cucumber as a behavior-driven framework of Catrobat project. One of the main benefits is that Cucumber scenarios can be written and read by every single person involved with the Catrobat. Every stockholder, such as developer, tester, designer, and manager can easily maintain existing scenarios as well as create new ones. The scenarios/specifications are written in a pure natural/ubiquitous language that could be understood by every member of the Catrobat team. New team members are able to understand the system easily and faster by reading the scenarios. The necessity of writing new step-definitions will decrease over time because existing step-definitions can be reused in further scenarios. The scenarios steps are defined once and can be used over again and again.

The external quality of Catrobat can he held upright by using Cucumber scenarios as test cases. This means that the behavior of Catrobat and theirs bricks can be adjusted through executable specification accordingly. The executable specification, described one by one in the upcoming chapters, shows that how bricks behave in certain situations? It also describes the specification of the broadcast mechanism of Catrobat by means of examples. Each scenario describes some particular situation and the expected behavior of Catrobat bricks. The Cucumber features represent Catrobat programs. The visual programming language Catrobat and the behavior-driven testing framework Cucumber were used to demonstrate some of the key ideas of BDD specifications. The broadcast system of Catrobat was systematically specified by examples. Some misbehavior in the existent system could be uncovered/exposed and corrected. The behavior of the broadcast system of the Catrobat language is easily specifiable by the help of Cucumber features. The other parts of Catrobat might also be straightforwardly specifiable by the means of Cucumber features. The Cucumber features could be written in Gherkin language and tested with the Cucumber testing framework.

Limitations/Shortcomings: There are also shortcomings to introduce the Cucumber into Catrobat project. First, the Cucumber framework can coexist with the already existent unit tests. But this requires a strict policy about what to test through unit tests and what to test via Cucumber scenarios. It is also boring to maintain two testing systems in parallel. Another option would be to establish Cucumber solely as a testing system in Catroid. Yet this would mean rewriting all the unit tests into Cucumber scenarios. Due to a large number of existent tests, this would require many hours of developers time.

Recommendation/Suggestion: To sum up, a practical solution would be to keep and maintain the unit tests, to guarantee a stable internal quality level of the Catrobat, and establish the Cucumber testing-framework additionally. This would also guarantee a high external quality level of Catrobat Project. With the passage of time, if the Cucumber tests and the unit tests overlap more and more, the number of unit tests could be reduced and, finally, the Cucumber framework could establish an exclusive testing framework in Catrobat Project.

1.2.3. Thesis Challenges and Contributions

This thesis introduces the concept of BDD methodology as well as Cucumber specifications. It also includes how the executable specifications were developed and used throughout the Catrobat project. With the help of these executable specifications, we specified the issues/bugs and described incorrect behavior regarding “Broadcast”, “Broadcast and wait”, “When you receive”, “Screen tap”, and “set variable” bricks in the Catrobat project for Android mobile application (Catroid) in the form of executable regression tests. We extended our testing and automation process with the help of BDD and Cucumber framework for Android mobile apps by integrating executable specifications to prove our approach.

With the help of this advanced agile software methodology (BDD), the approach aims to test and diagnose the bugs in Android mobile application (Pocket Code), which is tested with the help of BDD methodology specifications. Our work focuses on mobile applications and their regression testing with the help of BDD specifications.

As the visual programming environment (Catrobat) is localized into bidirectional in general and specifically to the Arabic language (Arabization) and Urdu language. Furthermore, in this thesis, we also summarized some challenging aspects of the RTL languages faced by the Catrobat programming languages with the help of BDD methodology, and using Cucumber specifications.

These challenges in RTL languages differ tremendously in terms of their character and morphology from other languages. However, our work contributes to the observed result of the various bugs we found. The result shows that test automation allows BDD base testing for the localized version. It supports finding localization (RTL language) bugs earlier and it helps to speed-up release cycles and to save human effort in testing.

This thesis also focuses on autonomous test scripting architecture for mobile apps, which step forward towards executable specification to become a living documentation. The BDD tries to address the interaction among the team members, aligning to the common understanding of the desired needs, functionalities and eliminate misunderstanding. The executable Cucumber specifications with their reusable steps, the Gherkin ubiquitous language with its concise structure, are efficient for testing and reduce the level of ambiguity. These specifications can be edited without recompiling the code base. Once a step definition is stored in your project, the new feature can be added without adding new glue code every time. Therefore, our proposed automated acceptance testing architecture improves BDD steps, reusability of their implementations, and provides a common platform among the developers, testers, and business analysts.

So, therefore, the core benefits of using Cucumber base specifications in our project would be an ideal cross-platform mobile UI test automation for other platform. The key outcome of the thesis is BDD methodology and Cucumber development tool that overcomes the limitations of native Android UI Automation for Android base application (Catroid). With the help of this approach, it influences the testing and increases the suitability, performance, efficiency, compatibility, reliability, maintainability, and portability of Android native business application to fill the existing gap in the current academic literature.

Chapter 2

Mobile Application and Testing

This chapter describes the mobile application and its history. It is important to know the background of the mobile world and have comprehensive knowledge of earlier mobile technology. We briefly describe the mobile application and the most common mobile operating systems, i.e., Android, iOS, and Windows Phone. Further, we discuss software testing, objectives and the types of testing.

2.1. Mobile Application

A mobile application is a software program, which is specifically developed and aimed for operation on mobile devices, tablet computers, and a number of other devices, related to smartphones, to obtain appropriate data for input. So, therefore, in this discussion, an application is meant to be playing electronic devices such as MP3 digital readers, digital cameras, and mobile phones. There are four restrictions to put in so as to determine and demarcate the rareness and the relative conceptual difference of mobile computing. These constraints include factors such as limited resources, security and weakness, and performance and reliability. An application is aware of the computing environment when it is running and adopting according to its context-aware computing [21].

2.2. Types of Mobile Apps

Three types of mobile applications are described in this section. They are named as native applications, hybrid applications, and web applications. It is the users input data, with limited resources, which is run by the application in a mobile phone. An extension to application is its operation on a server, which is made possible through internet called web application [21, 22].

Native applications: Developed for specific mobile devices, native applications are particularly designed for single mobile operating systems. The application is built for specific platform i.e.

iOS, Android, and Windows phone, which cannot be used on another platform. This simply means that Android applications can not be used on iPhone. Normally, factors such as performativity and the experience of users, during the period when the developers use native device user interface, are counted to gauge the benefit it carries out. Accessible specifically from the app stores of its kind, native applications are made available to the target users. A disadvantage of native application is that it is very expensive when it is compared to other kind of applications. This is because it is to create copies of the application for other platforms, render separate support as well as maintaining of different applications at the same time.

Hybrid applications: - This type of applications are created using different platform web technologies i.e. Javascript and HTML5. The hybrid applications are mostly web applications masked in a native wrapper. Hybrid application are very fast as well as comparatively easy to develop. This is because of perfect benefit for a single code-base to all platforms that confirms maintaining low budget and the subsequent easing of updates. Furthermore, the hybrid app also has some disadvantages such as lack in performance, speed as well as design problems because of application incompetence to oversee simultaneously the two or more operating systems.

Web applications: Called as browser application or web application, it downloads all or some parts of software by the time when it is running. In order to access it, mobile devices have to be web capable. These types of applications act and behave like native apps. It is always run by a web browser, which are commonly written in HTML5 or JavaScript. It requires the lowest and minimum of device memory. When internet is available, users can get access from anywhere and any device.

2.2.1. History

The mobile phones have been around since the middle of the 1970s. The devices have of course changed strongly since then but the biggest change came in 2007. From that movement on, the mobile smartphone market has been boosted. There are so many apps for every aspect of our lives, extending from music and photos to office use applications and games, fitness and health. However, what about the Quality of those applications? Are they reliable, trustworthy, easy to use, well developed, and tested? Before we can communicate with any kind of mobile device, a communication infrastructure must be available. The mobile infrastructure is currently in its fifth generation, known rightly as 5G. The zero generation, the early predecessors, included just analog radio communication and was mainly used in the 1960s [23].

In the real sense, a smartphone is a mobile phone with advanced aspects, operating system, and functionality besides basic phone features like making calls and sending text messages etc. The smartphones are equipped with the features to display images, play videos, play games, navigation, built-in camera, audio/video playback, and recording, send/receive e-mail, engaging in social networking, web surfing, wireless Internet and much more. In the first place the smartphones were only considered for business use due to their cost and application, but unlike that, today we are in a rapidly populated smartphone society with the smartphones from several vendors/dealers providing a variety of advanced functionalities and features on a small device [23, 24].

A smartphone has recently achieved to strike a long-expected milestone, which is appreciable and spectacular growth in the industry. Obviously, in the global shipments, smartphone overtook personal computers. Due to its unlimited spreading and social acceptance, we can find smartphone in schools, educational institutes, hospitals, public places, and shopping centers etc [24].

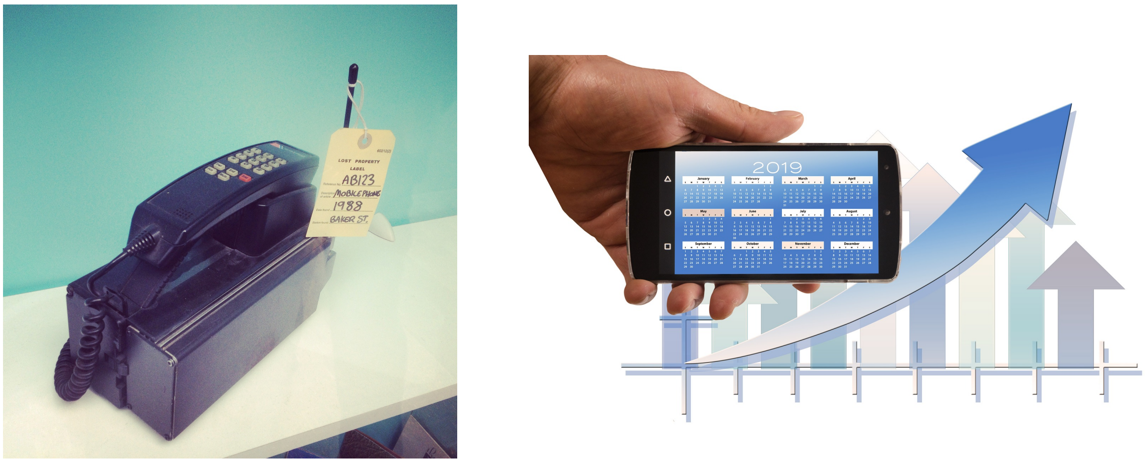
In 2007, Apple Inc released the first smartphone called iPhone, the smartphone has a feature of a multi-touch interface but in reality, the smartphone has been on the market since 1993. The difference between todays smartphone and early smartphone is that early smartphones were designed to be used by corporate/ commercial customers and for enterprises/business communication objectives, and in addition, those phones were too costly especially for the public customers [25].

The smartphone period is divided into three major phases. The first phase was focused on enterprises. During this phase, all the smartphones considered the corporations as their target and the features and functions were to follow the corporate requirements. This period began with the first release of the smartphone called The Simon from IBM in 1993. The revolutionary device of this period is Blackberry, it is supplied with many features like email, fax, web surfing, navigation, and built-in camera. The smartphones of this phase were totally built according to the enterprises objectives [24, 25].

The second phase of smartphone period started with the release of the iPhone. Steve Jobs of Apple exposed the iPhone, which he referred to as a revolutionary and magical product. The iPhone, the great leap/rise in the smartphone market, was first exposed in January 2007. This was the period when for the first time a company ever revealed the smartphone targeting the public customers market [24]. The third phase of smartphones focused on closing the gap between enterprise view and general customers view. Hence, the two views were mainly united to improve the display resolution, display system technology, mobile operating system, create more long life batteries and enhance the graphical user interface and many more services within these smart devices. In 2008, this phase started with the revolutionary upgrades in the mobile operating system and during the last eight years, there have been many upgrades in Apple iOS, Android, and Blackberry OS. The most common mobile operating systems (Android, iOS, Blackberry OS, Windows Mobile) and key smartphone vendors/dealers (Apple, Samsung, HTC, Motorola, Nokia, LG, Sony etc.) are focusing on developing new features in operating systems and devices which will introduce an exciting useful feature to enterprises and global customers [24].

The Apple device introduced numerous concepts for a design that have been adopted by modern smartphone platforms such as the use of multi-touch gestures for browsing. At the end of 2007, Google introduced the Android operating system with the aim to dominate the customer smartphone market. However, Google announced that they will introduce the Android operating system for free and anyone will be able to use and customize it. The companys intention during this period of time was to develop features that the customers need and expect. These include email, social networking, audio/video playing, internet navigation, and chatting. At the same time, Google keeps the cost at a lower level to attract more and more customers [23, 25]. The role of Android has been enormous during this time period since it is an open source operating system. It introduces a huge opportunity to all vendors to build new devices. Therefore, nowadays, Android is the best-selling smartphone platform [24].

The main difference between iOS and Android is the domain of devices. The iOS from Apple is strictly biased to products from Apple. It is used only on iPhone, iPad, and iPod. At the same time, Android runs on many smartphones on the market. The great players like HTC, Samsung, Sony, LG use Android on their smartphone. All in all, you have the ability to reach more users by developing apps for Android. Therefore, the role of the Android operating system has been huge from the time when it is an open source.



(a) Mobile Phone in 1988

(b) Mobile phone in 2019

Figure 2.1.: Evolution of mobile phone

2.3. Mobile Operating Systems

Mobile Operating System (OS) is a software platform that allows i.e mobile devise, smartphones, tablets to run applications and programs. Actually, a mobile operation system is a platform in which other applications can run on mobile devices i.e., smartphones, tablets, and so on. Since long, the mobile operating system design has experienced three stages in advancement. The first one belongs to the PC-based operating system, the second to an embedded operating system, the last and the third one belong to the current smartphone oriented operating system in the last 10 years. During the whole process, some major changes are made in which smartphone OS structural design has switched from complicated to simple ways. The advanced mobile OS merges different characteristics with the features of other OS, such as a touch screen, Bluetooth and video camera etc[26].

There are three popular Operation System i.e. Android, iOS, Windows Phone. Currently, Android and iOS mobile apps dominate the app market. Figure 2.2 shows and contains the number of applications downloadable in worlds famous known app stores since July 2018. Since then, the number of apps available to Android users ranged between 2.1 million. Similarly, the Apples App Store was found the second largest in terms of the availability of apps. Although, the number of these apps were 2 million. Additionally, other companies such as Blackberry and Windows were also noted with an enormous number of application[1].

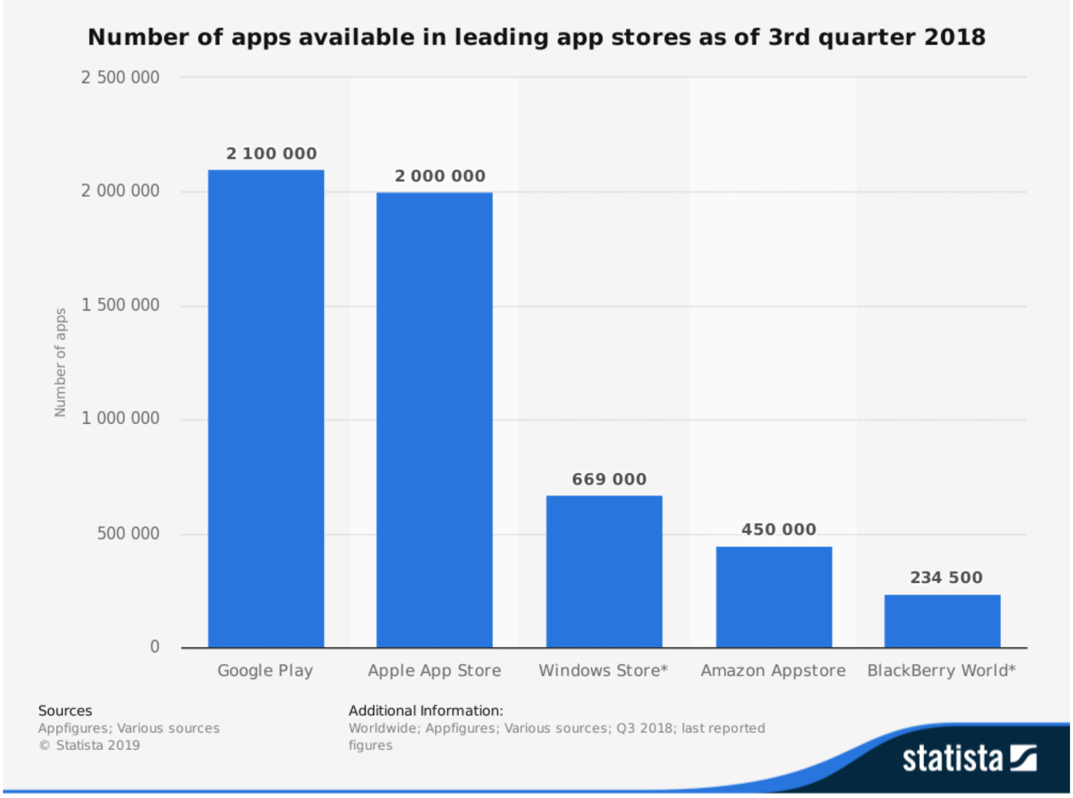
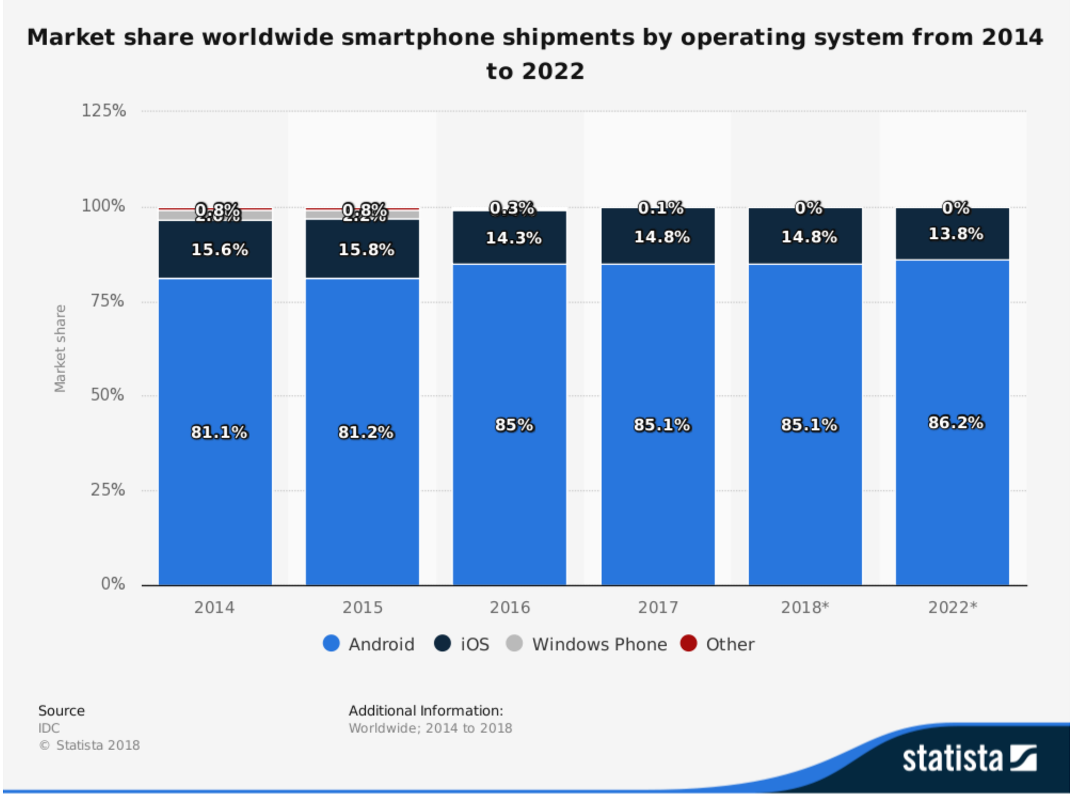


Figure 2.2.: Number of apps available for download with different platform

2.3.1. Android

There are many operating systems for mobile devices, however, the Android operating system developed by Google is the most popular one. It has a worldwide market share of over 80% which is shown in Figure 2.3. In 2018, the most popular platform of Android version was Marshmallow. The second most popular operating system for mobile devices is iOS which is developed by Apple and its worldwide market share is over 15% [27].

[1] http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/


Market share worldwide smartphone shipments by operating system from 2014 to 2022

Source
IDC
© Statista 2018

Additional Information:
Worldwide; 2014 to 2018

Figure 2.3.: Forecasting for different operating systems

Android Inc. was established by Andy Rubin, Rich Miner, Nick Sears, and Chris White in 2003, but was later acquired by Google Inc. The initial purpose of the company was to create an advanced OS for digital cameras. However, when it was perceived that the market for the intended devices was not huge enough, the company diverted its plans toward releasing a smartphone operating system that would compete with Symbian and Microsoft Windows Mobile [28].

The Android OS is an open source and is based on the Linux kernel. For that reason, it is possible for anyone to customize the operating system for his or her own desires and requirements. For that motivation, it marked a popular operating system with several technology companies [29].

The Open Handset Alliance supports the development of the Android operating system, and it consists of different technology companies, such as Google, Intel, Motorola, and Sprint. It is not publicly open but companies join them by a closed process managed by Google [30].

Android applications can also be developed on platforms that support the Android SDK such as Windows, Mac OS, and Linux. Different integrated development environments (IDE) can be used for developing applications for Android. Applications for Android are developed using Java, which is compiled into bytecode and translated into the Android platforms own byte code [31].

Android's major release versions are identified using code names. Each Android platform version supports one API level and each application has a minimum required API level. The higher API levels are designed that are compatible with all earlier versions [32].

The below mentioned Figure 2.4 shows the number of available apps on Google play store. The current Android apps figure on Google Play is 2,533,007. The AppBrain low quality app detection filter automatically detects which apps are unlikely to be useful. The Google team looks like to remove applications from the entire market approximately once a quarter, in which case the total number of available Android apps goes down. The removed apps are usually classified by their system by means of low quality apps [33].

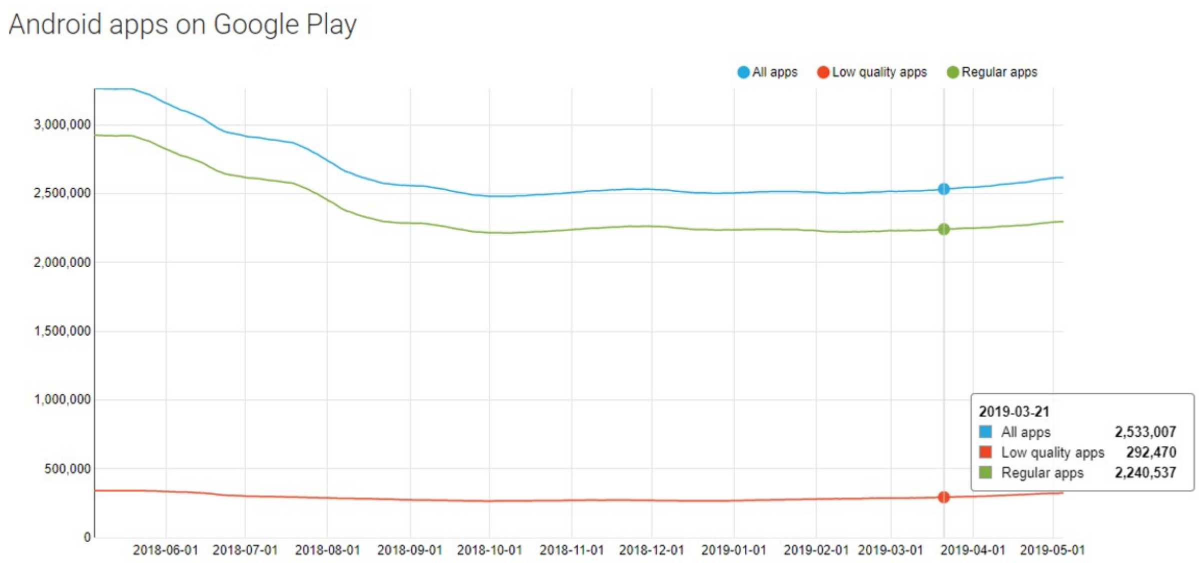


Figure 2.4.: Android apps on Google Play

Activities

Activities are one of the primary building blocks of applications on the Android platform. They act as the entry point for a user interaction with an application and are also acting as a means for user navigation within an application or between applications. Activities are used to place and display graphical elements for the user to interact with. Android applications usually consist of multiple or at least one activity that is bound to each other, and the activities can switch between each other to display various fragments of the application or execute different actions. Even though activities are often displayed in a full-screen window, they can also be displayed in other modes: as floating windows or embedded inside of another activity[34]. By using an intent object, activities can invoke other activities, which is like a glue between the activities in the application. Intents are basically messaging objects used to request an action from another application component. The intent is also used to start a new instance of an activity, which enables the user to send extra data to the started activity. It can also invoke a bundle, which acts as intermediate storage when transferring data between activities. Activities in the Android application are managed using the stack. When a new activity is started, it is pushed to the stack and is placed on the top with running state. The previous activity is always placed below it in the stack, and will not come to the front again until the new activity exit far [35].

Lifecycle of an Android

Activities are used to place and display graphical elements for the user. Android applications usually consist of multiple or at least one activity that is bound to each other, and the activities can switch between each other to display various fragments of the application or execute different actions. The activity lifecycle of an Android app describes how an activity can switch between different states. As for as Android app is concerned, all that a user understands is managed by activity elements. The activity handles the users input. Software application commonly contains a number of activities that are used for a single view. The lifecycle of an activity in Android is shown in Figure 2.5 [35]. Usually a main activity is created by an app when it is started. Afterwards, three methods are executed before the interaction with the end-user. In addition, the mobile activities are creating other activities, and the one already created turn into running activity. The previous activity moves to the background activity, which no longer remains active. Similarly, the activity, already stopped, keeps in the same state, so as this could be resumed whenever a user wants to return [35].

onCreate()- When an activity move in the creation state method it is called OnCreate() method. It is created once only for the entire life of activity and could be made subjected to app startup reason. **onStart()-** A callback is invoked when the activity is switched to the state of beginning. This same method bring visibility in the activity for the users. The app, in the meantime, let the activity move into the foreground in order to be interactive. **onResume()-** Upon resuming state of the activity, first it goes into the foreground after which the system invokes the onResume() callback. During this stage, interaction between the user and the app in formed. **onPause()-** When the activity enters into the pause state i.e. the onPause() method, the Activity is in the paused state, and that you expect to resume shortly. **onStop()-** When the activity enters to stopped state, anywhere the lifecycle components can stop any functionality that does not need to run while the component is not visible on the screen. **onDestroy()-** When the activity moves to the destroyed state, anywhere the lifecycle components can clean up anything it needs before the Activity is destroyed.

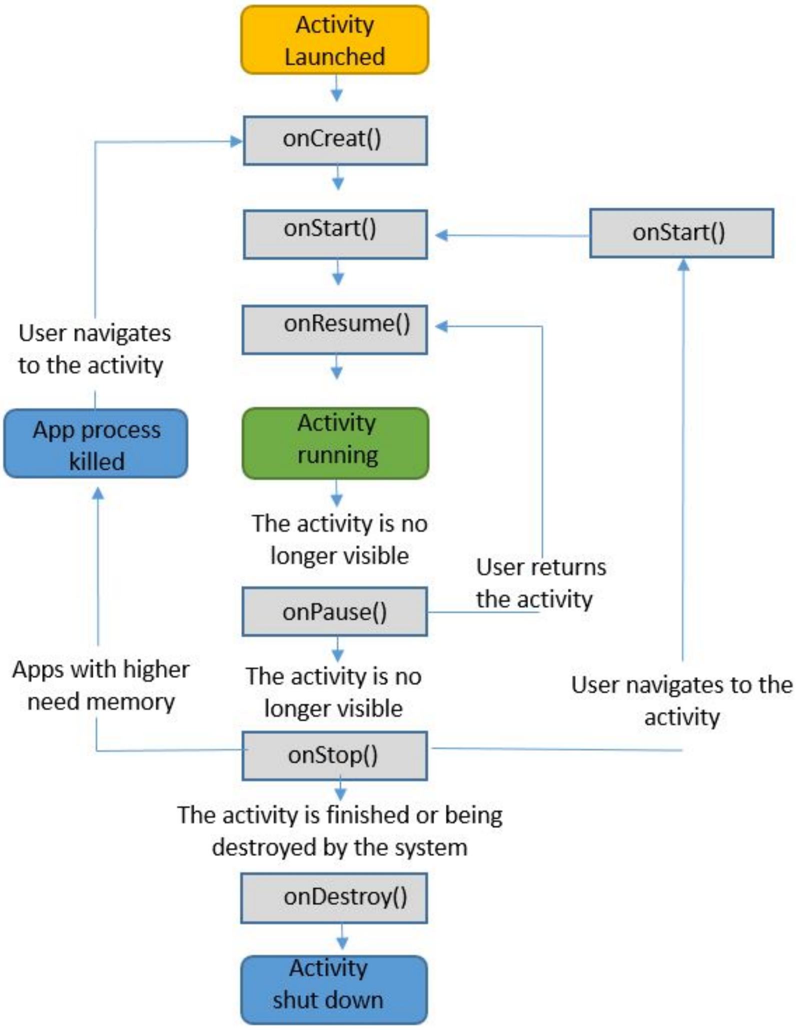

Activity Launched
onCreat()
onStart()
onStart()
onResume()
User navigates to the activity
App process killed
Activity running
The activity is no longer visible
User returns the activity
onPause()
Apps with higher need memory
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finished or being destroyed by the system
onDestroy()
Activity shut down

Figure 2.5.: The lifecycle of an Android activity

2.3.2. iOS

The iOS abbreviated as *iPhone Operating System* is developed by Apple Inc. It is a mobile operating system that runs on mobile devices produced by Apple i.e. iPhone and iPad devices. However, Xcode, used as an IDE, during the development of iOS applications handles compiles, validates and sends the application to the concerned App Store. To analyzing the applications, memory usage and its performance, debugging and fixing is also carried out by the IDE. In order to install an application on iOS mobile, an important stage during the developmental process is to sign in the application. This is possible after a user passes through the credentials given by Apple. The developer is obliged to buy the above credentials from Apple. On the other hand, in Xcode, there is still a possibility for the developers to test the applications on the emulators without the credentials [36].

According to Statista website (See Figure 2.2), users of Apple could get access to around 2 million apps. Unlike the Android apps, the guidelines towards the design regarding iOS apps have been a more insisting demand. In cases where development guidelines are not followed, the app could be rejected by the Apple App Store. It is also essential, however, that applications need to be native applications, this would still be the same if the applications under discussion are hybrid, and do not use native components for the user interface [37].

Lifecycle of an iOS

The software application lifecycle is a framework-defining task at each state in the software process. However, being an iOS base application, it comprises five states i.e., (1) not running, (2) inactive, (3) active, (4) background, and (5) suspended. All these states are mentioned in Figure 2.6 [38]. When an app is in the first state (not running) either the application has been cancelled or not launched or for the reason that the application is not yet started. Usually apps are in the second state (inactive state) only when they are performing other functions. It cannot accept any events, although it is in the foreground. This could happen in case a call or message is received. During the third state (active state), the operation runs normally (normal mode) in the foreground and receives any event. The only way to go to active state is through the inactive state. The applications in the fourth state (background) are able to execute code. They can also be delivered directly into this stage. Suspension is also ready for termination when applications are in the background state. The suspended applications in the background do not execute any code. It maintains and stays in memory. However, when a low memory condition occurs, the system may remove the apps in the suspended state without notice. The termination of application happens through users or the operating systems [38].

2.3.3. Window Phone

Windows Phone is a mobile operating system developed and owned by Microsoft. It is especially for smartphones. There are two types of application projects developed by Microsoft - Windows applications for PCs or tablets and Windows Phone applications for smartphones. Particularly, a lot of APIs, tools, and design principles are shared by these two projects. Hence, they were partially integrated into the Windows Universal Applications, which allows sharing of code between Windows Store and Windows Phone application projects [39].

Windows Store and Windows Phone Store[2] are exclusive application digital platform distribution, which could also be integrated into Universal Windows Store in Windows 10. A single developer's account is required by Microsoft to both stores, as well as the developer's account can be individual or company. Microsoft applies slightly different approach for approval process than the others. During the approval process, the application content compliance is checked manually but other tests are automated (security, technical compliance) [39].

[2] www.windowsphone.com/en-us/store

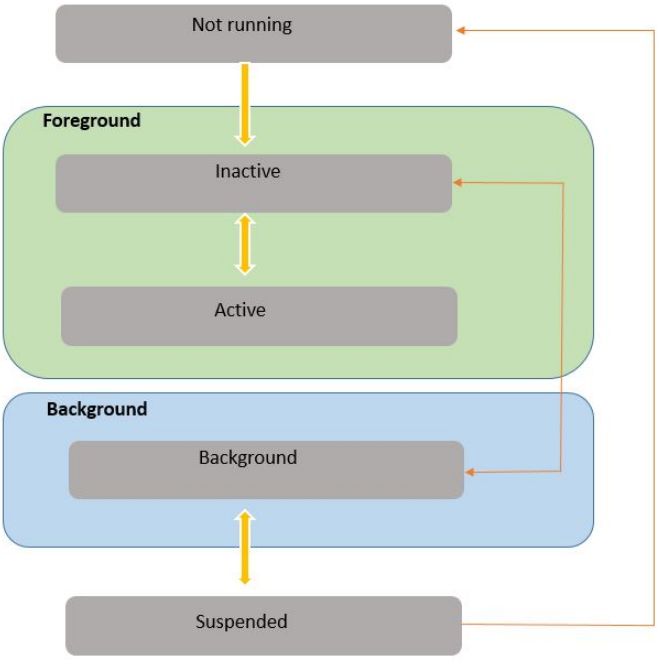


Figure 2.6.: Lifecycle of an iOS app

2.4. Software Testing

This section elaborates software testing as the process that exposes constraints, designing and coding of bugs in the software package. It serves to detect the accuracy, safety, completeness and renders quality to software products. Software testing is the practice of measuring good quality of the softwares that are being developed. It also detects and shows bugs, faults as well as the defects in the software [40].
The key objectives of software testing is assuring the quality, durability, verification, reliability, and validity. It is also an essential part to ensure software quality and characterizes an overview of description, the design as well as the coding [41, 42].

2.5. Software Testing Objectives

The software testing is a set of activities conducted with the intent of finding errors in any software app. With the help of software, testing is to ensure, verifies and validates whether the program is behaving correctly without bugs. Actually, it evaluates the software for finding bugs. It is not only used for searching and fixing of bugs, but to make sure that the software is working according to the requirements. It is a chain of procedure, which is designed to ensure that the system code does what it was designed for. The main purpose of testing can be quality assurance, reliability estimation, validation or verification. It is well-defined very simply that software testing is an activity to check whether the actual test results match the expected test results and

make sure that the software app is bug free.
Expected result: The final result defined is totally based on requirements description of the test implementation.
Actual result: The actual result expected after applying the test data to the software as per steps defined in test case under controlled test situation. [42, 41]. The other objectives or software testing includes are as fallow:

- Deliver a bug free software application.
- Software testing to make sure that the software works according to the requirements and customer satisfaction.
- Always make sure that error has done charmingly in the software specially at that situation when end-user entered inappropriate data and the software shows user friendly messages.
- A successful test is the one that uncovers an undiscovered error and provides a high quality software to the customers.
- Testing is a process to detect the accuracy, completeness and evaluate the overall performance of the software application.
- The common objective of software testing is to confirm the quality of software by scientifically practicing the software insensibly with skillful circumstances.

2.6. Mobile Application Testing

The process of mobile application testing by which application software is developed for mobile devices which mainly focuses on function, features, usability and consistency. Therefore, mobile apps also come pre-installed (i.e., camera message, etc) either installed from the concern mobile application distribution platform/play store [43].

2.6.1. Why Mobile Application Testing is Necessary?

Testing a bug free mobile application early is the right choice to develop. Actually, testing app is to see whether it is working with all functions, features and the app is running correctly. When the development process of mobile app is finished, you need to look up and consult the mobile app development company to check and fix developed mobile app for all the likely bugs a customer can face. All these companies will provide you a bug-free mobile application that supports the up-to-date and latest hardware and guarantee maximum hardware compatibility. Although, the entire mobile app testing will get maximum return on investment when there are positive reviews from the end-user or customers on uploaded app store [43, 44].

2.7. Types of Testing

In this section of the chapter, we present testing types as there are many different types of testing that you can use to make sure that changes to your code are working as expected i.e., stable usable and bugs free. Listed below are common types of software testing and also mentioned/-summarized in Figure 2.7 (Unit, Function, Regression, System, GUI, Performance, Usability, Security, Localization, Integration, Acceptance)

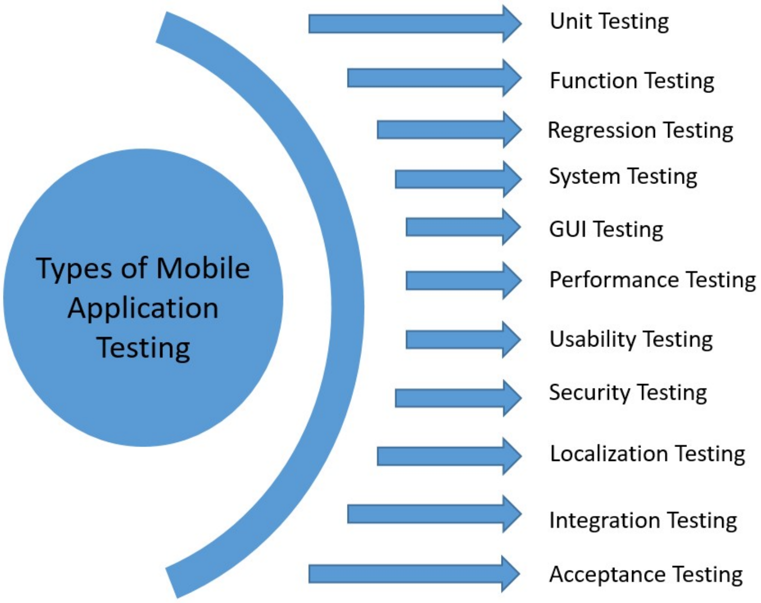


Figure 2.7.: Type of testing

2.7.1. Unit Testing

The unit testing is the minimum and smallest collection of software code, which can be tested. In software application the smallest testable parts called units are individually and independently analyzed for proper operation in a software development process. It can be done manually, however it is often automated. It is a line of code or method and the component of test-driven development. TDD requires that developers or testers first write failing unit tests, then they write code to pass the test and refactor the application until the test passes in a stable position [45, 46, 41].

2.7.2. Functional Testing

The first thing that mobile tester has to do is functional testing of the app. It is a type of testing and is one of the essential features of every software product. Functional testing ensures that the application features and requirements are implemented and to make sure all the functions are correctly performed i.e. inputs, outputs, sliding, buttons, navigation, and data processing. Furthermore, the functionality should be tested in different mobile user scenarios and environments. There are some factor including, appropriate input, the verification of the outcome and the comparison of the results - both actual and expected - which test the functionality of the system. [47, 48]. If functional testing is performed on mobile devices manually, not automatically, it is going to be a very difficult, exhaustive and a time-consuming task due to different mobile-specific challenges.

2.7.3. Regression Testing

Regression testing is re-testing of previously developed and tested software after a code change to ensure that the software is still working in the same way as before the change. Changes may include software improvements, patches, configuration adjustments, etc. Regression testing was recommended to test the efficiency and improving the transparency in the large-scale software development process. Regression testing depends on users who have a good experience on the app. However, it is possible for the mobile testers to improve and add additional test cases to the system just to be on the safe side and therefore the testing gets unnecessarily costly [49].

2.7.4. System Testing

System testing of the application should check the overall compliance of the product with its specified requirements (functional and non-functional). It also tests a totally integrated system in order to verify its end-to-end functionality. Furthermore, it makes sure that the integrated system does not damage or corrupt its operating environment, or any other processes that the application communicate with. Concerning mobile applications, it is also important to test for functioning for each carrier. Due to simulators and emulators drawbacks, which will be discussed in section 8.3.2, these tests should be achieved on the real devices in the real user environment in order to give the most realistic results (features of real hardware, targeted OS versions) [50, 41]. In practice, system testing should be carried out in two phases. The first phase should utilize the test automation tools (Robotium and Espresso). This enables access to a large variety of devices on which the app can be simulated. The second phase is testing the application on real devices. This phase of testing supports the final decision regarding the release of the app

2.7.5. GUI Testing

Defined as Graphical User Interface (GUI) testing, this practice ensures suitable functionality of GUI of a particular software application and to confirm the written requirements. Visual elements like font sizes, buttons, text formatting, links, layout, text boxes, content, colors, lists, fonts, icons, labels, and captions are evaluated and tested . This type of testing needs a lot of programming. It could be manual or automatic, and sometimes the members of the third party, instead of the developers or customers themselves, to implement it [45, 46, 51].

2.7.6. Performance Testing

Performance testing is one of the pivotal testing services in every software development project, especially for mobile apps. It determines how a system works in terms of responsiveness and stability under a specific workload. Mobile users expect an app to start/launch within two seconds;

otherwise, they are unsatisfied and may uninstall it. Performance testing process is conducted to inspect the performance and behavior of the application under certain circumstances, such as low battery, bad network connection, low memory, simultaneous requests to the applications server by several users and other such conditions. Furthermore, two sides might affect the performance of an application: applications server side and clients side [2, 48]. This type of testing decides what kind of performance is anticipated under such load, and evaluates the speed of response time for application under various network conditions (Wi-Fi speed, 4G connection etc.) Testers and developers can use performance tests to find out potential bottlenecks and deadlocks in their software application[52]. Normally, performance testing is carried out on servers or backend systems to inspect how the systems or software can handle huge numbers of requests and to match acceptable outputs for the users. The following list summarizes the simple mobile app performance tests:

- Compute the apps launch time.
- Compute the loading time of the content such as images, text, or animations.
- Check the delay time during operations or user interactions.
- Test on different architectures, especially on slower mobile phones.
- Compare the current app version with the new release.

2.7.7. Usability Testing

It is a good practice to conduct user experience testing very early in the test cycle. Most mobile testers plan to perform it in the last stage, but usability testing can reveal lots of defects with such UI elements as appearance, layout, content, fonts, graphics, text colors, background colors, etc. It is a good approach to find these bugs and fix them early in the cycle as possible. The domain of this testing depends on the guidelines of style, copy documents and content from the business requirements. Usability testing is performed to verify whether the application is fulfilling its objectives and getting a favorable response from customers. The testing process ensures that the mobile app is now easy to use and provides a suitable user experience to the customers. This is essential since the usability of an application is the key to commercial success. However, the app should be easy to use; otherwise, it might be with low ratings, which lead to the app damaging and negatively affect the companys reputation. Efficient mobile usability needs lots of refining, intensive user research, and even more testing with real clients. For instance, elements with the same type such as buttons or text views should have the same spaces, sizes, and colors. The mobile tester should inspect that all the UI elements are accessible, for example, that buttons can also be tapped by a user with thicker fingers and on different screen sizes and densities. The following list contains some aspects that should be checked during the usability testing such as: [53].

- If the app supports multi-languages, the text should fit into every UI element and that the translation is correct.
- App shows friendly and useful error messages.
- App should offer undo and redo functions in order to provide an easy way to correct errors.

- App should follow the same workflow in every section.
- All UI elements such as buttons, labels, and other elements are big or small enough to be used.
- The navigation style of the app is easy to use.
- Text used within the app should be clear and easy to understand.
- UI elements must have the same look-and-feel, the same text, spaces, colors, and images

2.7.8. Security Testing

Security testing is a complex testing type that requires a lot of background in many different areas, such as client-server communication and software and hardware architectures. It determines whether an information system protects data and maintains functionality as expected. Security testing can be carried out on both client-side mobile applications and the server-side software to address all the vulnerabilities [48]. The following list provides the most common security problems of mobile apps [52].

- Sensitive information such as passwords is cached on the device.
- Sensitive information such as passwords, tokens, or credit card details is accidentally stored.
- Sensitive information such as passwords is not encrypted on the mobile storage.
- Verification process for a password is accomplished only on the client side.
- Communication from app to the back end systems is not encrypted.
- Apps use permissions for device aspects and peripherals that they do not need or use.

2.7.9. Localization Testing

Nowadays, most of the apps are developed for global use and it is very essential to care about regional languages, cultures and time zones, etc. Localization testing is used to ensure quality testing. Primarily it evaluates the functionalities of the products, cosmetics and its quality. Moreover, geographically and culturally diverse regions are controlled through specific critical methods when employing localization to ensure quality of the products. The localization acts as a passport for the product that transfer it from country to another [54, 52]. Localization support has to be considered by the customer as one of the requirements in the initial stages of the application development lifecycle. It ensures that the local end-users expectations are met in terms of language, user experience in the traditional sense and features. Localization testing is usually performed on real devices and real user environment and is tested only on a handful of critical devices[54, 55].

2.7.10. Integration Testing

The integration testing in which program units are combined or integrated and test the interface between the units in multiple ways. However, in this perspective, a unit is well defined as the smallest testable part of a software application. With the help of integration testing you can expose bugs with the interfaces among program components before faults occur in real-world program execution. It is a component of extreme programming, a practical method of software development that takes a particular approach to build a product by means of continual testing and revision [45, 46, 41].

2.7.11. Acceptance Testing

The acceptance testing is usually conducted by the customers to ensure whether the product delivered the requirements and works as the customer or client expected and the systems satisfy its acceptance criteria. Acceptance testing is usually performed by the customer under the class of black box testing. Therefore, the customer accepts the software application while all the features and functionalities are working as estimated. Usually it is the last stage of software testing after which it goes into production [46, 41].

Chapter 3

Visual Programming Language

Visual programming language (VPL) is a paradigm of programming languages. In this language, a user creates his own programs, and the elements of the program are drawn graphically than textually. This language trend is particularly popular among children because of its easiness to create games and the visual charming appearances. Lately, there are many VPLs, namely Catrobat, Scratch, and Snap!, introduced in education sector [1, 56].

3.1. Catrobat Programming Language

Catrobat is a free, open source and cross-platform visual programming language for smartphones. It was developed for educational purposes to help students. It was encouraged and inspired by Scratch. From the very beginning, Catrobat project was aimed to be optimized for mobile devices, smartphones, and tablets. It gives users a solid and consistent experience. Catrobat was first implemented in Java in the Android application, i.e., Catroid. In the meantime, iOS and HTML5 version has also been in development [17, 56].

3.2. Catroid

Catroid is a free, open source visual programming system. Catrobat programs are written in a visual Lego-style program, where individual commands are combined by organizing them visually with ones finger [18]. Catroid is an IDE as well as an interpreter for Android. It is an Android application for creating and running programs in the Catrobat programming language. Catroid was the first visual programming environment for Catrobat. It is mainly aimed at teenagers with the goal to increase their interest in computer science and to promote logical thinking and programming skills. Currently, there are three flavors of Catroid, i.e., Luna & Cat, Phiro Code and Pocket Code, which are shown in Figure. 3.1.

3.2.1. Pocket Code

Pocket Code is a free and open source mobile visual programming software for the Catrobat programming language. It allows children and teenagers to create, play, share and execute their own game and animation on smartphones. Pocket Code provides the functionality to share program over all main platforms (Android and iOS)[57]. The version of Catrobat, developed for Android smartphones, is named Catroid and is available on Google Play Store under the name Pocket Code[1]. Therefore, Pocket Code is used by arranging command blocks which are called bricks'. These bricks are the atomic elements that represent a specific statement of the programming language. All the bricks are organized in scripts which can run in parallel, allowing simultaneous execution (see Figure. 3.2). The bricks are easy to understand and are used instead of traditional text-based code so that no previous knowledge is needed to start creating simple programs.

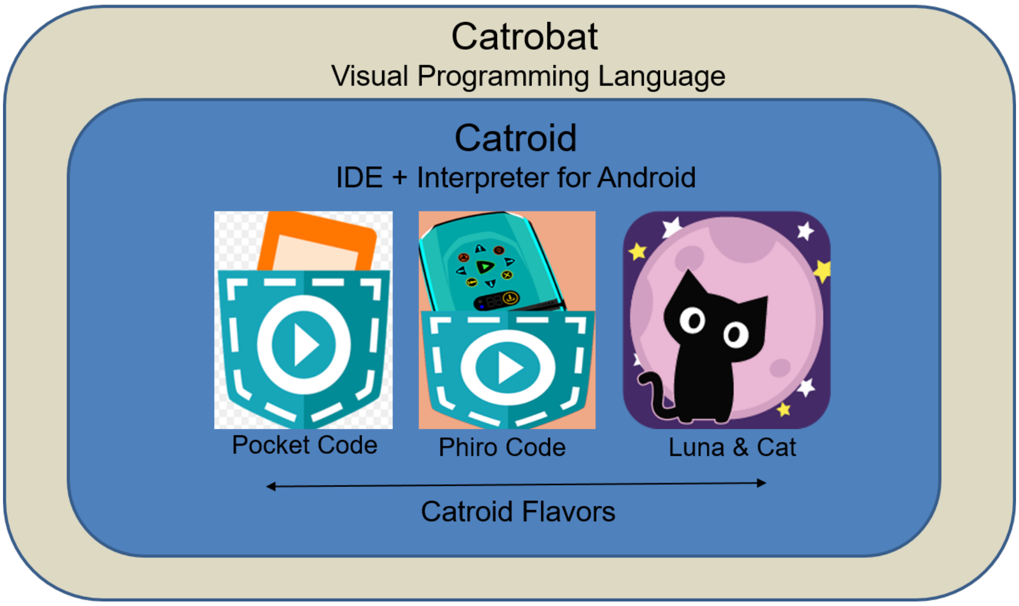


Figure 3.1.: The relation between Catrobat, Catroid and its flavors

Block, Bricks, Category. The smallest element of the VPL is called block or brick. Each of those elements has a special meaning and function. They are combined in certain ways to create a program.

3.2.2. Phiro Code

Phiro Code[2] is a free, open source visual programming mobile application in which you can build your own apps to control Phiro robots via Bluetooth.

[1] http://play.google.com/store/apps/details?id=org.catrobat.catroid

[3] http://robotixedu.com/phiroresources/introduction-to-pocket-code.html

3.2.3. Luna & Cat

Luna & Cate[3] is a new and modified version of Pocket Code, which is specific for female students (teenagers) to create and share their own apps. It is used for female viewers particular to study about the visual coding.

3.3. Catrobat Programming Language Elements

Catrobat has many types of Categories. Every category has its own related bricks which are explained below one by one (see Figure. 3.2) [56, 19].

Event: Elements of this category are needful for every project to start a program. In this category, broadcast related bricks can send and receive messages, and other event can be caught.

Control: Elements of this category are responsible for program flow bricks (e.g., loops and conditionals).

Motion: Elements of this category can change the position and the orientation of an on-screen object.

Sound: Elements of this category control the playback of audio files associated with an object or change the system volume level.

Looks: In this, category bricks change the appearance of the visual representation of objects. A different look can be chosen from the objects internal list, or the overall visibility, size, brightness and transparency, etc. of the object can be set and changed.

Pen: This category allows an object to draw shapes and color pixels. By choosing the number, you can change the size of the pen and draw a copy of an object on the stage.

Data: This category contains bricks to initialize variables or to change, show and hide their value.

Lego EV3: The elements of this category are used for program Lego Mindstorms EV3 robot.

Phiro: The elements of this category are used for program and control Phiro (educational robot) via Bluetooth.

Arduino: The elements of this category are used for program and control an Arduino via Bluetooth.

[2] https://play.google.com/store/apps/details?id=org.catrobat.catroid.lunaandcat&hl=en_US

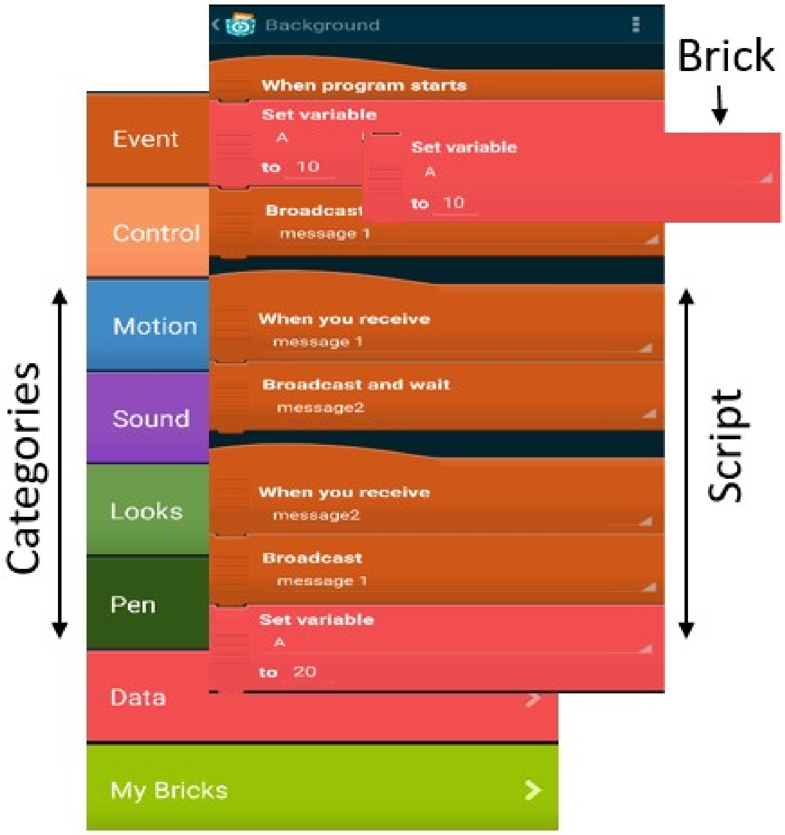


Figure 3.2.: Script view in Pocket Code

The visual elements are stated as bricks. Bricks are arranged in virtual containers that are called scripts. A brick is always part of a script and cannot exist stand-alone.

Event

- When program starts
- When tapped
- When screen is touched
- When you receive “new message”
- Broadcast “new message”
- Broadcast and wait “new message”
- When 1 <2 becomes true
- When physical collision with “anything”

- When background changes to “New”

Control

- Wait 1 second
- Note “add comment here...”
- Forever
- If 1 < 2 is true then... Else...
- If 1 < 2 is true then
- Wait until 1 < 2 is true
- Repeat 10 times
- Repeat until 1 < 2 is true
- Continue scene “New”
- Start scene “New”
- Stop “this script”
- Create clone of “myself”
- Delete this clone
- When I start as a clone
- When you start as a clone

Motion

- Place at X: 100 Y: 200
- Set X to 100
- Set Y to 200
- Change X by 10
- Change Y by 10
- Go to “Touch position”
- Move 10 steps
- Turn left 15 degrees

- Turn right 15 degrees
- Point in direction 90 degrees
- Point towards “New”
- Set rotation style “left-right only”
- Glide 1 second to X: 100 Y: 200
- Vibrate for 1 second
- Set motion type to “bouncing with gravity”
- Set velocity to X: 0.0 Y: 0.0 steps/second
- Rotate left 15.0 degrees/second
- Rotate right 15.0 degrees/second
- Set gravity for all objects to X: 0.0 Y: -10.0 steps/second2
- Set mass to 1.0 kilogram
- Set bounce factor to 80.0%
- Set friction to 20.0%

Sound

- Start sound “New”
- Start sound and wait “New”
- Stop all sounds
- Set volume to 60%
- Change volume by -10
- Speak “Hello!”
- Speak “Hello!” and wait.
- Ask Whats your name? and store spoken answer in “New”

Looks

- Next background
- Previous background

- Set size to 60%
- Change size by 10
- Hide
- Show
- Ask whats your name? and store written answer in “New”
- Set transparency to 50%
- Change transparency by 25
- Set brightness to 50%
- Change brightness by 25
- Set color to 0.0
- Change color by 25.0
- Clear graphic effects
- Switch to look “New”
- Switch to look “New” and wait
- Turn camera “on”
- Use camera “front”
- Turn flashlight “on”
- Say “”Hello!

Pen

- Pen down
- Pen up
- Set pen size to 4
- Set pen color to Red 0.0 Green 0.0 Blue 0.0
- Stamp
- Clear

Data

- Set variable “New” to 1.0
- Change variable “New” by 1.0
- Show variable “New” at X: 100 Y: 200
- Hide variable “New”
- Add 1.0 to list “New”
- Delete item from list “New” at position 1
- Insert 1.0 into list “New” at position 1
- Replace item in list “New” at position 1 with 1.0

Lego EV3

- Turn EV3 motor by 180 degrees
- Set EV3 motor to 100% speed
- Stop EV3 motor
- Play EV3 tone for 1.0 seconds frequency 2 *100Hz volume 100%
- Set EV3 LED status Green

Phiro

- Move Phiro motor forward “Left” Speed 100%
- Move Phiro motor backward “Left” speed 100%
- Stop Phiro motor Arduino
- Set Arduino digital pin 13 to 1
- Set Arduino PWM pin 3 to 5

Arduino

- Set Arduino digital pin 13 to 1
- Set Arduino PWM pin 3 to 255

Raspberry Pi

- When Rashpi Pin Change Brick
- RaspiSendDigitalValue Brick

- RaspiPwmBrick
- RaspiPwmBrick

Jumping Sumo

- Move Jumping Sumo forward 1 second with 80% power
- Move Jumping Sumo backward 1 second with 80% power
- Animations Jumping Sumo “Spin”
- Jumping Sumo sound “Normal” Volume 50%
- No Jumping Sumo sound
- Jump Jumping Sumo long
- Jump Jumping Sumo high
- Rotate Jumping Sumo left 90
- Rotate Jumping Sumo right 90
- Turn Jumping Sumo
- Taking picture Jumping Sumo

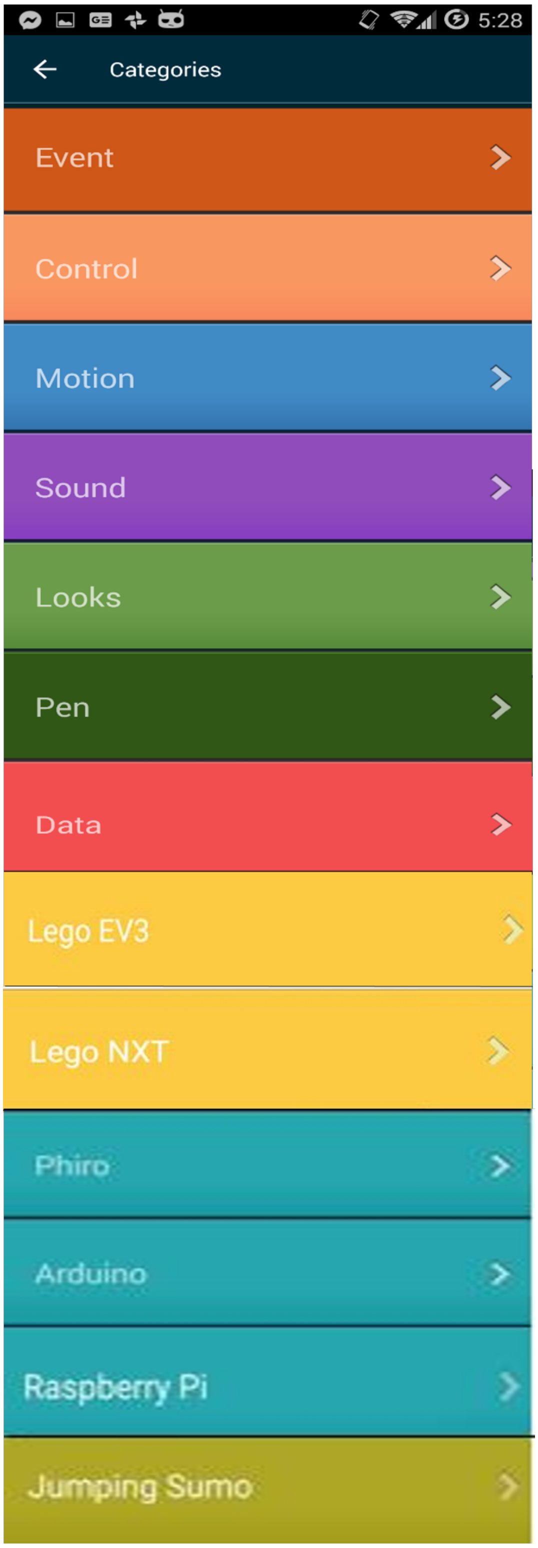


Figure 3.3.: Screenshot for category

Chapter 4

Agile Testing Methodologies

The software development methodologies came to surface in the late 1970s. The agile testing methodology is a software testing practice to provide better management and adjusts the dynamics of change business requirement for software development life-cycle. It delivers practices and aligns with iterative development to facilitate communication between all the stockholders, that's to say, developer, tester, customer, and to go through develop-deliver feedback cycles to have a more specific view of the requirements, and be ready for any change at any time. With the help of agile methodologies, the companies always build the right product as well as empower the software team to continuously redesign their release for improvement throughout the development process. With the methodology, new requirements are met in the rising software world market. As such, the aim is to deliver the need during the software development process. This approach is also described as successful when delivering the majority of circumstances. There are also some conflicting reports according to which the methodology is still too young and that it requires extensive academic proof of its being successful [9]. There are three testing techniques to improve testing practices.

1. Test Driven Development (TDD)
2. Acceptance Test Driven Development (ATDD)
3. Behavior Driven Development (BDD)

4.1. Test-Driven Development

Test-driven development is a programming practice to incrementally develop software, which was from time to time used for decades and re-emerged as a development pattern in the agile methodologies context. In numerous software projects, the increasing adoption of extreme programming with the identification of TDD is a key approach in agile software development process. TDD starts with designing and developing tests for every small functionality of a software application.

However, a number of studies have shown this tendency and claim the attention of various researchers working on the ability of TDD in diagnosing software bugs during the time when they are being developed. Using this methodology, unit tests are written which are taken from user requirements before writing the code itself. Afterworlds, the required code is implemented and needed to pass and succeeded. This practice fixes the bugs as soon as it is detected. Upon the passing of the tests, the developer resorts to refactoring of the code to acceptable standards. Later, they proceed to define a new set of test cases for other functionalities and implement a new piece of code to pass them [58].

The TDD methodology approach offers a fully opposite view of the traditional test approach, which is commonly used in software development, where production code is written according to design specifications. The most common example regarding TDD is implemented in the traffic light. A normal traffic light has **Red, Green, and Yellow**. Therefore, the red light represents the failing, or possibly non-compiling, test code and Green light is the conclusion result of writing the smallest amount of code to make that test pass. The yellow (refactoring) light is used to eliminate any code duplication or bad programming practice (see Figure. 4.1) [59].

Anyways, a number of books have been in black and white to educate the developers and testers with wide example, and to indicate enough interest in TDD to treat it as a serious practice. However, some software developers saw the advantages of the test first approach; numerous of the programmers had the opinion that it was very challenging as well as very different from what they normally do and make sure. One more research found that 56% of the software programmers involved in a TDD experiment had difficulty in adopting a TDD approach. So for 23% of these software developers, the lack of upfront design was acknowledged as a difficulty. While there is a third, research study collected answers starting from 2018 volunteer software programmer who contributed in an online analysis survey. In this survey the software programmers self reported how frequently they implemented particular TDD approach. This study also found that 25% programmers admitted to often, or every time, making faults in the traditional steps in TDD practice [59].

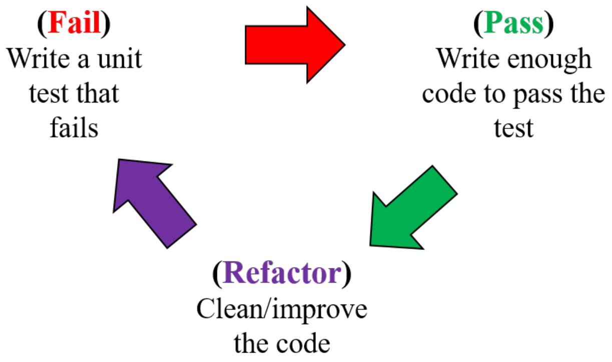

(Fail)
Write a unit test that fails
(Pass)
Write enough code to pass the test
(Refactor)
Clean/improve the code

Figure 4.1.: TDD Process

In larger and difficult systems, the advantages of the TDD are not very clear and understandable. In this practice the manual location and fixation of bugs presented in the iterative development steps is significant task. TDD practice is mostly suitable for comparatively small-scaled software project. Therefore, when the size of the project is large and complex, the regression bugs become a challenging job. When all regression tests reveal a failure, the debugger should track the bug advances in the propagation chain by repeatedly running the test step by step until they identify the bug source [58].

4.2. Acceptance Test Driven Development

While TDD is a developer technique, acceptance test-driven development (ATDD) is a whole team technique, where the entire team collaborates to define the acceptance criteria of a story before the implementation. Hence, they must not be confused. Both write tests first, but their goals are different [60]. Similarly, Koskela [61] elaborates that TDD is a technique for improving the softwares internal quality, whereas ATDD keeps a products external quality upright. McConnell [62] provides a definition for internal and external software quality.

Internal characteristics of software quality:

- Maintainability
- Flexibility
- Portability
- Reusability
- Readability
- Testability
- Understandability

External characteristics of software quality:

- Correctness
- Usability
- Efficiency
- Reliability
- Integrity
- Adaptability

- Accuracy
- Robustness

Some of the characteristics of internal and external software quality may influence each other. High internal quality makes software easy to understand and therefore easy to extend and to test. It states how well the needs of the developers and administrators are met [63]

TDD ensures that the internal quality stays upright. External characteristics of quality, however, measure how well the requirements of the stakeholders are held. ATDD just ensures that the external characteristics of quality are held upright. An external user of the system does not care about how well the source code or the tests can be read or how reusable some modules could be. From an external point of view, a system needs to be correct and be made easy as well as reliable for use. Adzic [64] describes the process of acceptance testing as a pattern where examples become acceptance tests. First, real-world examples are used to build a shared understanding of the domain. A set of these examples is then selected to be a specification and acceptance tests. The verification of the acceptance tests has to be automated. Software development can then focus on acceptance tests. Finally, the set of acceptance tests might be used as a base for discussion about future change requests, and a new cycle of development can start [64, 65].

Acceptance tests are a communication medium between several roles in a software development team. A test that passes is a specification of how the system works. ATDD is also called behavior-driven development (BDD) or specification by example (SbE).[66] The goal of ATDD is to specify executable requirements. Acceptance tests are written as Cucumber features which serve as executable specifications .

4.3. Behavior-Driven Development

BDD was introduced by Dan North and inspired by Kent Becks TDD. There has been a growing interest in BDD due to its capability to engage business analysts with formal requirement specification and to easily convert these requirements into executable scenarios in a way to make them readable by business people, with an added benefit to fruitfully employ them as automated acceptance tests [10]. BDD is a software engineering approach designed to help teams and improve the quality of software in a faster way. It is based on agile methodologies and some practices from TDD. It provides a common, ubiquitous language to facilitate communication between team members and the business stakeholders for better understanding among them [67].

Figure. 4.2 shows the relationship and process of two methodologies [10]. BDD follows the same red, green, and refactor cycle of TDD while emphasizing the behavior instead of testing. The automated acceptance test written in BDD fails initially but passes once the implementation fulfills the expected behavior. First, write a failing feature test, and then run the test. Initially, it should fail. Then write just enough code to pass the test. Afterward, during the refactoring phase, improve the code without changing its behavior, and run all the tests and succeed. The Behavior Driven Development approaches can turn into automated business-facing tests that guide development as well as the conversation with customers about an example of desired and undesired behavior for a new feature. BDD uses ubiquitous language to capture customers' specimen in a domain specific language. The goals are the customers discussion using natural language and create Given, When and Then, scenarios that express the behavior of a feature, including the condition and action in tests which everyone in the team understands [68].

Therefore, the project of the documentation and automated tests, stored in the project feature files, is an executable test. The testers will be able to launch tests, analyze and create new test scenarios. All the steps in Gherkin language syntax are reusable, and the keywords may be combined in many different ways to create different scenarios. The glue code plays an important role that links the ubiquitous language to the code under tests (see Figure How Cucumber works in Catrobat). These scenarios are made executable by providing glue code method annotated with regular expression matching the text of the scenario steps [69].

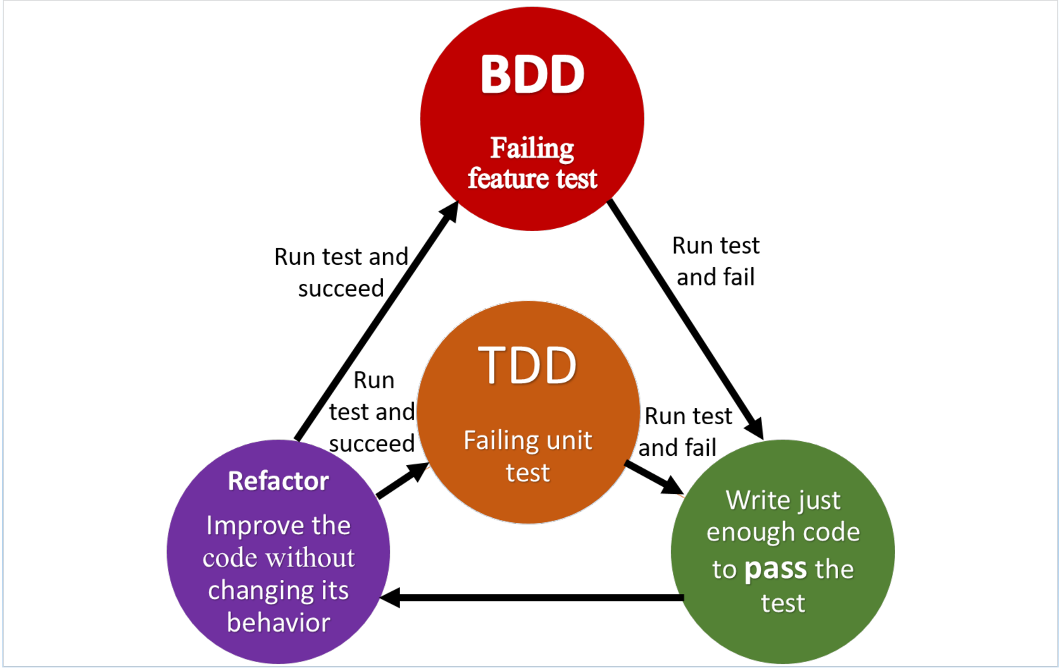


Figure 4.2.: TDD & BDD Process

4.3.1. Characteristics of BDD

From literature reviews and BDD software toolkits, the characteristics of BDD are as follows:

- Ubiquitous Language based on the business domain language, which enables developers and customers to speak the same language without ambiguity. This is the core of BDD [66, 11].

- User Story, plain text description, and scenario templates in BDD are pre-defined templates that specify them with ubiquitous languages.
- BDD proposes that the code should be part of the system documentation, which is in line with agile values. The entire codes should be readable and the specifications be part of the code [11].
- BDD approach is used in the initial planning phase, analysis phase and with the business outcomes and decomposed into a set of features which capture the behavior of the targeting system. Developers are encouraged to think of the behavior of components they are developing [11].
- An important characteristic of BDD is that the scenarios are easy to automate as well as easily understandable by the stakeholder [70].

Table 4.1.: TDD and BDD execution comparison

Stages	TDD	BDD
1	Write a test	Write a test scenario
2	Run the test and check if it fails	Execute the scenario and check if it fails
3	Write a code sufficient for the test to pass	Write a code sufficient to implement the expected behavior
4	Run the test and check if it passes	Execute the scenario and check if it passes
5	Refactor the code	Refactor the code

Behavior Driven Development (BDD) was first introduced as a solution to the issues that could be found in Test Driven Development. Today, BDD has evolved into an established agile practice. BDD is an emerging practice that has limited literature in comparison to other agile methods. Many software development companies are transitioning from the waterfall model to agile practices. These agile software development methods include Scrum, Kanban, DevOps, TDD, and BDD. In the late twentieth century, the elaboration of the concept of test first approach occurred into TDD. TDD is a programming method executed in short, repetitive cycles, wherein each cycle tests are written before the code. Years later, Dan North invented BDD to transform TDD into a more efficient software development process. The main goal of BDD is to get executable and well-defined specifications of software. In other words, BDD combines general techniques and principles of TDD with ideas from domain-driven design and object-oriented analysis and design. This combination is made in order to provide software development and management teams with shared tools and a shared process to collaborate on a software development. Table 4.1 presents a side-by-side comparison of the executions of TDD and BDD [71].

4.3.2. Why BDD?

Many developers are confused using TDD and ATDD in their software projects. They want to know what to test and what not to test, why a test fails, where to start, how much to test in one go, what to call their tests and how to understand the behavior. For early design in TDD, testers and developers work together. For projects in which developers use TDD, a reduction in deficiencies to 40% was observed. McConnell clarifies that in the requirement phase the defect injection rate can be 56% while in the design phase the rate can be 27%. Commonly, these flaws are found at a later stage of the project, especially when the final product is available to the end user. Identifying the defects earlier in the project will also reduce the cost. If these defects are not detected, they are shipped to the customer. Therefore, for the BDD approach the quality control and cost reduction will be the main motivations [14].

During the requirement phase, BDD methodology combines the concept of TDD with the idea of writing test scenarios. The testers and developers define functionality and requirement descriptions of the application. In contrast to TDD, the BDD test scenarios are written in natural languages, which facilitates the discussion with all project stakeholders, including marketing and business people. These scenarios describe how the end user will use the final product [14, 11].

4.3.3. Benefits of BDD

There are many key benefits for an organization adopting behavior-driven development practices. They are as fallows [67]:

Reduced waste

Behavior-driven development always focuses the development process of discovering and delivering the Cucumber specifications that will provide business importance and avoiding all those, which are not important. BDD supports to escape this sort of wasted effort by helping teams focus on Cucumber features that are aligned with business goals. BDD also reduces wasted effort by enabling faster, more useful feedback to consumers. This helps teams make changes sooner than later.

Reduced costs

The direct consequence of reduced waste is to reduce costs. Therefore, by improving the quality of the software code, you are actually reducing the number of bugs, costs of maintenance and the project risks.

Easier and safer changes

With the help of BDD, you can easily extend and change your specifications. As living documentations is generated from Cucumber features, the stakeholders are familiar with specifications that makes it easier for him to understand the whole application. BDD practices produce a comprehensive set of automated acceptance and unit tests, which reduce the risk of regressions caused by new changes to the application.

Faster releases

With the help of BDD base comprehensive automated-tests, the release cycle is also sped up significantly. However, the testers are not required to spend time on long manual testing sessions before every fresh release. As an alternative, they can use the automated acceptance tests as a starting point [67].

Natural Language

As discussed earlier, the natural or ubiquitous language allows all the stakeholders to write down Cucumber specifications (behavioral requirement) for better understanding to remove confusion. With the help of common language, it is easier for new participants to join the working process at any time.

The evolution of TDD

The BDD practice is an evolution of TDD, which supports the behavior of the system defined by the stakeholders. The TDD supports the quality of the code, which the developers write and implement.

Chapter 5

Cucumber

A Cucumber is a testing tool that supports Behavior Driven Development (BDD). Gherkin is an ubiquitous language for writing acceptance tests to be used with the Cucumber tool. BDD provides us with an opportunity to make test scripts from both the developers and the customers perspective. So in the beginning, developers, testers, business analyst, and product stakeholders get-together with mutual understanding to decide which test scenarios should be passed in order to call this application successful. Therefore, they write all these test scripts in a simple and common ubiquitous language.

5.1. Cucumber Environment

Usually, BDD acceptance tests are implemented using a standard BDD acceptance test framework, e.g., Cucumber [10]. Cucumber is an open source BDD test automation tool for textual specifications, automated checking of specifications, and correctness testing of an implementation. It is a framework that supports the Gherkin description language. Cucumber is available for Java as well as in many development environments. Cucumber reads a specification from a structured plain-language text, which is called feature file, and parses it for the scenarios to test. A scenario has a list of steps for Cucumber to work through.

For understanding these feature files, Cucumber must follow basic syntax rules defined through the Gherkin language (see Fig. 7). It includes a set of step definitions for each step in a code written in Java. In a very mature test case, the step definition should be one or two lines of code. Cucumber then continues to the next step in the scenario if the code in the step definition executes without error. Without raising any error in any step of the scenario, it marks the feature as passed. However, if any step in the scenario fails, Cucumber marks it as failed. Cucumber also prints out the whole test report on the console or the IDE [12].

The software application experts, testers and developers write Cucumber scenarios manually that express system behavior. Specialized testers write the implementation for these scenarios by hand, and then execute the Cucumber tests. Cucumber provides transparency about what test scenarios are mapped to which executable tests and what test scenarios are covered. Each Cucumber scenario has several steps that are written in ubiquitous language with Cucumber syntax. Testers write code for each step and then run Cucumber tests to generate test reports [72].

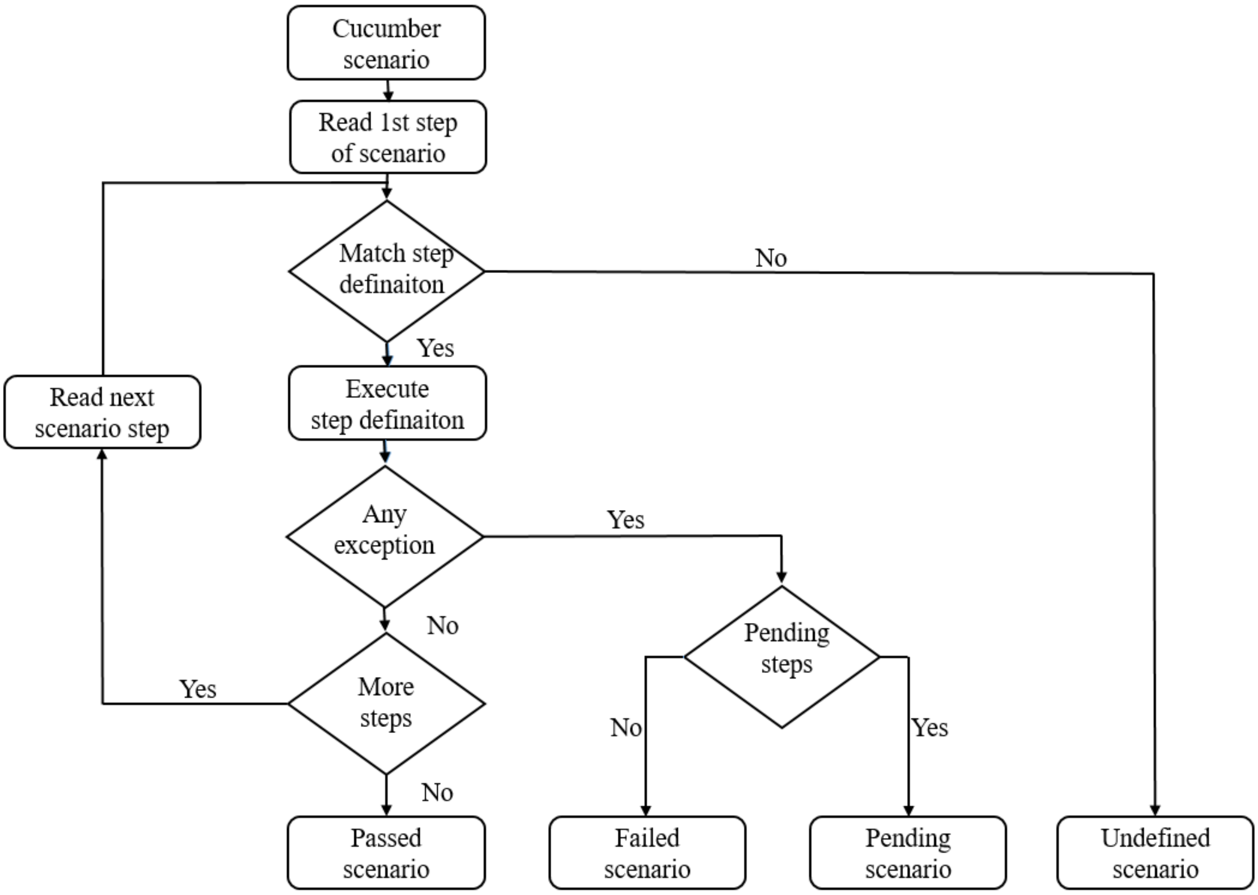


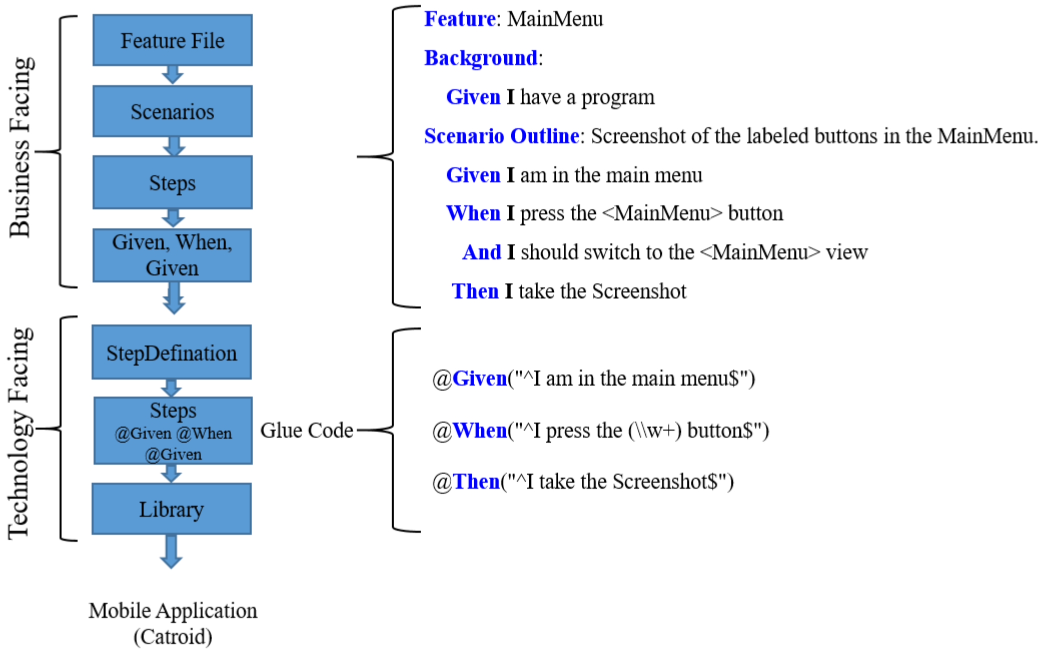
Figure 5.1.: Cucumber work flow in Catroid

Figure. 5.1 shows that when a scenario is executed, it should be in any of the following states: Undefined, Pending, Failed and Passed. This only indicates the progress of the test case[12].

- **Undefined**: The step is undefined and the scenario stops when Cucumber does not find any step definition that matches a step. The color should be yellow.
- **Pending**: The step is pending (yellow) and the scenario is stopped when Cucumber discovers a step definition that is not yet implemented. The rest of the steps will be skipped or marked as undefined.
- **Failed**: Cucumber marks steps as failed (red) and stops a scenario, as well as the rest of the steps will be skipped when the feature described in Gherkin syntax does not confirm with the implementation.

- **Passed**: When the block of code is executed and there is no exception then Cucumber will mark the steps as passed (green).

Figure 5.2 [12] illustrates that the overview of typical Cucumber test suites in the Catrobat project. There are two parts, one is business-facing side and the other is technology facing. The customer or business facing-part of a Cucumber test suite, which is stored in feature files and only Cucumber can read it. In the second part, step definitions translate from the business -facing language of each step into code, which is written in Java. When we run Cucumber, it reads specification from feature file as well as it inspects the entire scenarios to test and run the scenarios against your system. Individually every scenario has many steps (Given, When and Then), which Cucumber can understand. It follows some basic syntax rules, and this set of rules is called Gherkin language. Gherkin language files use the .feature file extension, which is saved as plain text. Besides, with the features file, Cucumber gives a set of step-definition for each step which is written probably one or two line of Java code [12]. Cucumber is an interpreter. It is executes and interprets Gherkin code in *.feature* files. The Gherkin keywords are translated into 35 different languages and you can write features in your own native languages. All you need to do is to have a header in your .feature file with the languages you are using. If you do not provide a language header, Cucumber will by default switch to English[70].



```
Feature: MainMenu
Background:
  Given I have a program
Scenario Outline: Screenshot of the labeled buttons in the MainMenu.
  Given I am in the main menu
  When I press the <MainMenu> button
    And I should switch to the <MainMenu> view
   Then I take the Screenshot
```

```
@Given("^I am in the main menu$")

@When("^I press the (\\w+) button$")

@Then("^I take the Screenshot$")
```

Figure 5.2.: How Cucumber works in Catroid

5.1.1. Cucumber-JVM

Cucumber-JVM is a more recent JVM, which allows you to write a step definition in Java and other JVM languages. It provides a Gherkin implementation of JVM-based languages [67]. Furthermore, recently support for Android was also added by the Cucumber team. This makes it comfortable for users to run Cucumber directly on their target devices, just like regular tests. The JVM version of Cucumber works with the same feature files, scenarios, and steps as the original Ruby implementation.

5.1.2. A Feature is Not a User Story

Figure 5.3 illustrated about user story and feature file.

- **A Feature** is basically entry point for acceptance tests. In this feature file you will write tests in natural language. It is an important portion of Cucumber tool by mean of automation test script and live documentations.It is a piece of functionality to describe your tests in ubiquitous language in order to achieve the business goals. It contains scenario or a list of scenarios [67].
- **A user story** is a planning tool in an agile software development process that helps you flesh out the details of what you need to deliver for a particular feature. It will help you in planning how we can deliver or implement the feature. It describes the essential planning of the software and the types of customer that what they actually want and why. When the feature is implemented, the user stories can be discarded. It is a short description of something a user or stakeholder would like to achieve and express in common language, which the entire stakeholder can understand.

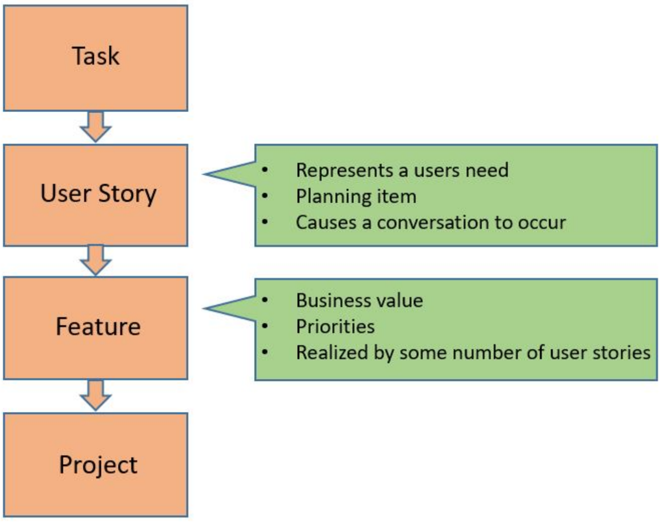

Task
User Story
Feature
Project
• Represents a users need
• Planning item
• Causes a conversation to occur
• Business value
• Priorities
• Realized by some number of user stories

Figure 5.3.: User Story & Feature file

5.1.3. User Stories and their Implementation in Catrobat

The user stories describe the interactions between users and a system. It also arranges the environment of the feature delivered by a system. BDD itself uses a simple ubiquitous language for the analysis process, which is domain independent and is used to structure the scenarios [11].

Implementation: to make your environment ready, a few dependencies need to be included in the project. In order for Junit to be aware of Cucumber and read feature files, when running the Cucumber class, it must be declared as the Runner.

```
@CucumberOptions (features = "features/bricks")
public final class Cucumber{
}
```

You can see the features element of *CucumberOption* locates the feature file created before another element *glue* provide a path to *step definitions*. Listing 1 shows how the *feature, Scenario Outline and steps* look like in English Gherkin syntax. So in this example, you can see the Given, When and Then keywords for verifying tests in Catroid.

```
Feature: MainMenu/Screenshot
       In order to give the screenshot to the user in respect of MainMenu.

Background:
Given I have a program

Scenario Outline: Screenshot of the labeled buttons in the MainMenu.
Given I am in the main menu
When I press the <MainMenu> button
And I should switch to the <MainMenu> view
Then I take the Screenshot
Examples:
       | MainMenu |
       | Continue |
       | New      |
       | Programs |
       | Help     |
       | Explore  |
       | Upload   |
-----------------------------------------------------------------------
       Listing 1. Example using Gherkin syntax in Catroid.
```

Setup the Catroid project as a typical Android project under the features folder, i.e., *src/androidTest/assets/features*, which hold Cucumber feature files. Inside of androidTest folder, you should have a Java folder by default, but not others. So, start by adding a new folder and call it assets. The assets folder is where you will put your Cucumber feature files. The step definition files are written in Java using Gherkin Tags. The .feature files have a corresponding Java class in the Steps file and place under the Step Definition folder, i.e., *src/androidTest/java/test/cucumber/ProgramStep.java*. To run the tests, we must specify the Cucumber options. These can be specified in a Java class. We can run all the implemented features, individual scenarios, or even a certain step from one scenario, which is a unit test. All the BDD related configuration and management of the Catroid is shown in the figure 5.4.

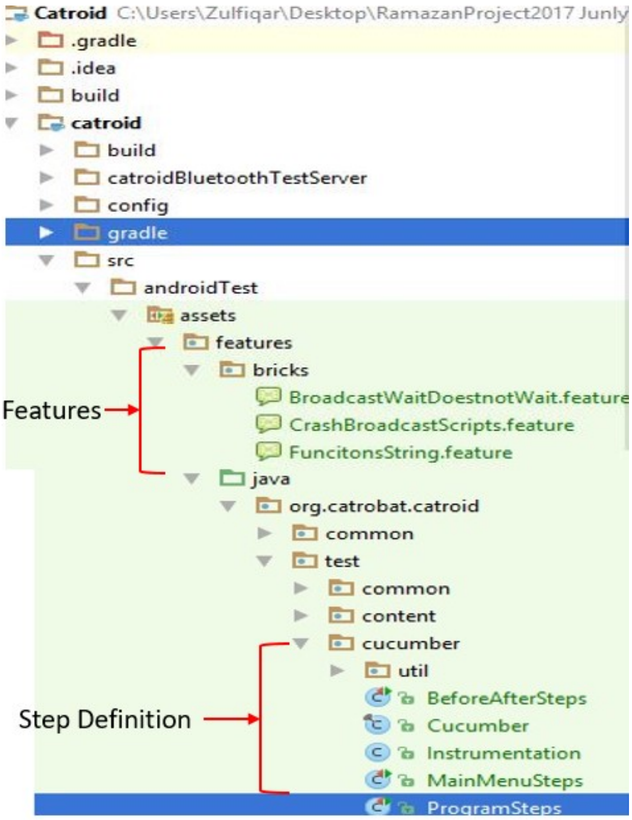


Figure 5.4.: Cucumber configuration in Catroid

5.1.4. Gherkin

Gherkin is a language for writing acceptance tests to be used with the Cucumber tool. The commercial and business facing parts of an acceptance test are always stored in Cucumber feature files. It should be written according to the syntax rules known as Gherkin language so that Cucumber can understand. It is an ubiquitous plain text language with little extra structure. It is easy to learn by non-programmers. Gherkin defines only a few mandatory keywords, and the rest of the feature is freeform text. The following Gherkin keywords are used [12, 73, 11].

- **Feature**: The stories in Cucumber are called features. A story written in Gherkin has a very well defined, but easily readable structure called feature. A file should contain one single feature, which can consist of one or more scenarios.
- **Background**: This section allows specifying steps, which are common to every scenario despite being repeated.
- **Scenario**: Each feature file contains several scenarios, and every scenario is a single actual test case. Every scenario consists of one or more steps, i.e. **Given** (Event or Context), **When** (User Action), **Then** (Result/Outcome), **And** (When you have many Given/When/Then, then you can use **And** and **But** for making scenarios easier to read) [11].

The basic concept and templates regarding the Gherkin language used in Cucumber, which allows writing specifications for software that can be both read and understood by stakeholders and tested by Cucumber. All the Gherkin keywords fit together to make Cucumber specifications readable and executable [12].

5.1.5. Step Definition

Step definition (SD) is the piece of code and glue that binds our Cucumber tests. We use Cucumber for conversion of feature file to step definitions. SD is always in between the business domain and programmer domain. The responsibility of SD is to translate Gherkin scenario steps into Java code. Gherkin Scenarios steps are only documentations that need step definitions to bring them to life. Special Cucumber annotation are used such as @Given, @When and @Then to create a step definition in Java (shown in the below code). In the annotation a regular expression is used to match the steps in between the double quotes [12]. Cucumber uses these regular expressions to match the scenarios with the names of the generating test methods [74]. The below mentioned code contains the three step-definition steps for creating *Broadcast* and *Broadcast wait* bricks as well as to test any variable.

```
@And("ˆbroadcast ’(\\w+)’$")
public void script_has_broadcast_brick(String message) {
       Sprite object = (Sprite) Cucumber.get(Cucumber.KEY_CURRENT_OBJECT);
       Script script = (Script) Cucumber.get(Cucumber.KEY_CURRENT_SCRIPT);
       BroadcastBrick brick = new BroadcastBrick( message);
       script.addBrick(brick);

@And("ˆbroadcastWait ’(\\w+)’$")
public void script_has_broadcast_wait_brick(String message) {
       Sprite object = (Sprite) Cucumber.get(Cucumber.KEY_CURRENT_OBJECT);
       Script script = (Script) Cucumber.get(Cucumber.KEY_CURRENT_SCRIPT);
       BroadcastWaitBrick brick = new BroadcastWaitBrick( message);
       script.addBrick(brick);}

@Then("ˆthe variable ’(\\w+)’ should be greater than or equal(\\d+.?\\d*)$")
public void var_should_greater_than_equal_float(String name, double expected) {
       Sprite object = (Sprite) Cucumber.get(Cucumber.KEY_CURRENT_OBJECT);
       Project project = ProjectManager.getInstance().getCurrentProject();
       UserVariable variable = project.getDefaultScene().getDataContainer().
       getUserVariable(object, name);
       assertNotNull("The variable does not exist.", variable);
       double actual = (double) variable.getValue();
       assertThat("The variable is < than the value.", actual,
       greaterThanOrEqualTo(expected));
```

Step Definition for broadcast and set variable bricks.

5.2. Implementation and Analysis of Results

The Catrobat visual programming language has different block elements or bricks. A program has one or more objects, and these objects contain a list of scripts, which is the code portion of the Catrobat program. The script is a set of bricks that consists of two or three bricks to combine the logic of the entire program. Here are issues/bugs faced by the Catrobat development team, which we tried to diagnose and detected with the help of executable specifications. The Android module of Cucumber JVM is successfully integrated with the existing code base of Catroid. The executable specification, which is written in the ubiquitous language *Gherkin*, can be used for documents and testing the visual programming language Catrobat bricks as implemented in the Android application Catroid.

5.2.1. BDD for Regression Testing: Case Study 1

Catrobat has different categories of bricks. The broadcast messages are used to ensure sequential execution of scripts in the same way as Scratch with respect to its broadcast mechanism. There are three types of broadcast bricks, “Broadcast”, “Broadcast and wait” and “When you receive”. The broadcast brick sends a message to all objects, and objects that wait for this message then immediately execute the corresponding scripts. The broadcast and wait brick send a message to all objects and wait until every responding script has finished execution.

In Listing 2, there is a scenario that specifies the objects correct behavior for a known bug in the implementation of the “Broadcast and wait” brick. It contains two scripts of an object. The value of the variable should be equal to 1. When we run the scenario, and an incorrect behavior is observed, the value of the variable is found not equal to 1. With the bug, the “Broadcast and wait” brick continues to wait forever when the scripts for whom it waits is terminated. Its waiting should actually be prematurely ended by a “Stop this script” brick. The “Broadcast and wait” brick should stop waiting and continue when all the corresponding “When you receive” broadcast scripts are stopped, even with a “Stop this script” brick.

```
Feature: Broadcast and wait

Correct behavior: The variable should be equal to variable 1.
Incorrect behavior: The variable is not equal to variable 1.
Background:
Given I have a program
And this program has an object 'Test'

Scenario: "Broadcast and Wait" brick should stop waiting and continue when
          all the corresponding "When you receive" broadcast scripts are
          stopped even with a "Stop this script" brick.

Given 'Test' has a start script
And broadcastWait 'hello'
And set 'var' to 1
Given 'Test' has a when 'hello' script
And stop this script
When I start the program
And I wait until the program has stopped
Then the variable 'var' should be equal 1
```

Listing 2. Cucumber feature for 'Broadcast and wait' brick.

5.2.2. BDD for Regression Testing: Case Study 2

Listing 3 specifies a regression test scenario for a known bug. In this example, the correct behavior of the program with respect to Catrobat bricks is that after 10 seconds the variable “timer” should have the value 10 or 9 during a period of 3 seconds. Then after 10 seconds, the variable “timer” again should have the value 10 or 9 during a period of 4 seconds, then after another 10 seconds, the variable “timer” again should have the value 10 or 9 in the same period continuously. In the Catroid implementation of the Catrobat programing language, the following incorrect behavior can be observed: After 10 seconds, the variable “timer” has the value 10 or 9 during the period of 3 seconds, which is still correct. Then, after 5 seconds, the variable “timer” has the value 5 or 4 during the period of 4 seconds, which is incorrect, then after another 5 seconds the variable “timer” again contains 5 or 4 continuously. Apparently, the receiving broadcast script incorrectly is executed with twice the speed when it is called the second or third time. The order of the first two “Broadcast” and “Broadcast and wait” script is important, otherwise the bug will not occur. If one reverses the order of these broadcast bricks or changes one of the types to another type, then the bug will disappear.
Note that the long delays given in the examples are indicatory only and need to be much reduced for the final feature definitions in order to not delay the execution of the tests.

```
Feature: Receive Broadcast Messages
Correct behavior: - correct behavior after 10 seconds, the variable
"timer" should have the value 10 or 9 during a period of 3 seconds, then
after 10 seconds, the variable "timer" again should have the value 10 or 9
during a period of 4 seconds, then after another 10 seconds, the variable
"timer" again should have the value 10 or 9 in the same period
continuously.
Incorrect behavior: - The Incorrect behavior, after 10 seconds the
variable "timer" has the value 10 or 9 during the period of 3 seconds.
Then after 5 seconds, the variable "timer" has the value 5 or 4 during the
period of 4 seconds, which is incorrect, then after another 5 seconds, the
variable "timer" again continue 5 or 4 continuously. Apparently, the
receiving broadcast script incorrectly is executed with twice the speed
when it is called the second or third time.

Background:
Given I have a program
And this program has an object 'Test'

Scenario: Once you receive script should not be executed with double speed
 on second call
```

```
       Given when program starts
       And forever
       And wait 1 seconds
       And change 'MySecond' by 1
       And forever end
       Given when program starts
       And broadcast 'hello'
       And wait 13 seconds
       And broadcastWait 'hello'
       Given 'Test' has a when 'hello' script
       And set 'starttime' to 'MySecond'
       And wait 10 seconds
       And set 'timer' to 'MySecond' - 'starttime'

       When I start the program
       And I wait for 24 seconds
       Then the variable 'timer' should be greater than or equal 9
-------------------------------------------------------------------------------
       Listing 3. Cucumber feature for 'Broadcast' and 'Broadcast and wait'bricks.
```

5.2.3. BDD for Regression Testing: Case Study 3

Listing 4 shows a complete example for a bug of the “When the screen is touched” brick. The expected and correct behavior is: Independently how often the screen is tapped, after the last tap, the variable gets an increase at the rate of 1 unit per second, because only a single thread instance runs at the same time. But we observe the following incorrect behavior if we tap the screen ten times and observe how quickly the variable increases: We see that it increases at the rate of 10 units per second. The reason is that as often as you tapped the screen, the script is executed in parallel thread instances. So normally this is the same behavior as Scratch, that all the scripts, including a “When Screen is touched” script run on their own single instance. When any event re-occurs that would trigger a script again, while it is still running from a previous event, the current run is aborted, and the script is restarted from its beginning.

```
       Feature: WhenScreen Tap

       Correct behavior: Independently how often the screen is tapped, after the
       last tap, the variable gets increased at the rate of 1 unit per second,
       because only a single thread instance runs at the same time.
```

```
        Background:
        Given I have a program
        And this program has an object 'Test'

        Scenario: Once screen is touched scripts can run only in one instance.

        Given 'Test' has a when screen is touched script
        And forever
        And change 'x' by 1
        And wait 1 seconds
        And forever end
        When I start the program
        And tap the screen 10 times
        And set 't' to 'x'
        And wait 1 seconds
        Then the variable 'x' - 't' should be less than or equal 2
------------------------------------------------------------------------
        Listing 4. Cucumber feature for when screen tap.
```

Chapter 6

Regression Testing for Broadcast Mechanism of Visual Programming Language on Smartphones

This chapter have been published in the [75].

This chapter presents an overview of the proposed automated regression testing approach. We describe the different concepts of mobile regression test automation tools and give a brief introduction on where and how to automate mobile regression tests for the target app and development environment.

Regression testing is used to assert that the software modification did not break the previously working functionality. For a large number of tests, regression testing is costly. However, some studies estimate that the testing budget regarding RT can take up to 80% as well as 50% cost for the software maintenance. Therefore, as the software application grows, the cost of RT increases. For instance, Google has examined that their RT system has a linear increase in both the number of software changes as well as the average test suite execution time; leading to a quadratic increase in the total test suite execution time [76]. RT is a repetitive process of software testing. It aims to ensure that new faults or defects will not become together or introduced into the extended code or modification of the app. The usage of RT might be increased due to the growth in an iterative development and re-usability of different software application features [77].

6.1. Regression Testing

Regression testing is re-testing of previously developed and tested software after a code is change to ensure that the software is still working in the same way as before the change. Changes may include software improvements, patches, configuration adjustments and so forth.
Regression testing was recommended to test the efficiency and to improve the transparency in large-scale software development process. Regression testing depends on users who have a good

experience on the app. However, it is possible for the mobile testers to improve and add additional test cases to the system to be on the safe side and not to get the testing unnecessarily costly [49].

The requirements are scattered in multiple artifacts with different features that describe them in different levels of concept, which is a big challenge. So all the tests have to run not only in the final version of the app but also in the entire set of the app to assure that they represent the same information in a non-ambiguous way. Moreover, along with the software development process, testing methods should be implemented, and at the same time, customers can introduce new demands and ideas or modify the existing ones along with every iteration. Regression testing is an essential testing to declare that the system of the product and its features remains working and behave correctly in accordance with the new requirements [78].

Figure 6.1 shows the automated regression testing for different apps state. This mechanism can help in making the practice much more competent for the future of the app under test. In every version, you perform regression testing by re-executing the same tests after each update. Regression testing is a good practice to run tests before the release of every new version of the app. In some situations, there is no way out to predict which fragments of the app will undergo change. In such cases, only full regression testing can guarantee that the system will perform well. With the help of this approach, we have to run all the tests cases after every amendment or change introduced to the project. A number of challenges are associated with regression testing. Some of them are listed below.

- The large size of the test suite after every successive regression run is a big challenge, which needs to be optimized using different tools and techniques. Choosing a right automation tool, based upon the nature of software application and availability of resources, is one of the common challenges [79].

- Maintaining a balance between the ever-growing test suite size and limited constraints, it is the biggest challenge in regression testing. A regression test suite can run after a group of bug fixes, after every new build or after every modification [79].

- For cross-platform, testing a loose coupling to the underlying platform is required to abstract the different platform's implementation away. This may be achieved by using a language that is not tied to a platform or programming language. A challenge in cross-platform testing is identifying User Interface (UI) elements across apps on different platforms [80].

6.2. Regression Testing Approach: Design and Implementation

In this section, the design and implementation of the proposed approach are presented in detail. In practice, the regression testing is usually performed by re-running the previously run tests and

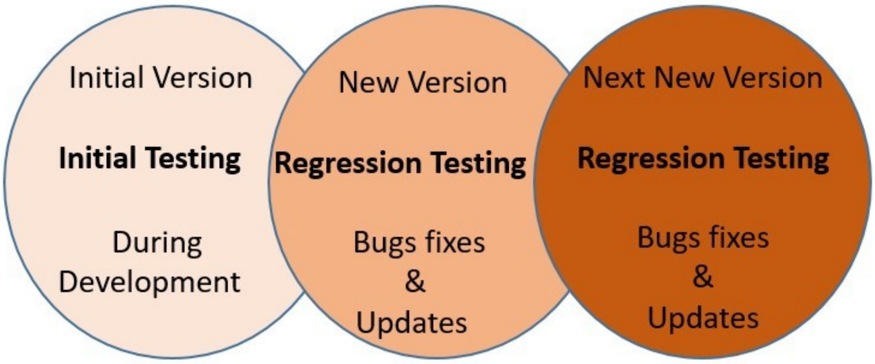


Figure 6.1.: Regression testing for different apps versions

verifying whether new faults have emerged. The regression testing objective for the new fresh version of Catrobat is to verify its correctness (especially for the broadcast mechanism) after a set of modifications and change. Therefore, such testing scenarios will be very helpful to establish a prominent role in the entire development progress. The test cases of Catrobat application require the reproduction of the actual conditions that are hard to reproduce without automated testing support.

6.2.1. Objectives of the Proposed Regression Testing Methods

The objective of this work is to improve the regression testing performance at the functionality test level in order to integrate the previously established regression testing into the updated version. Regression testing helps to increase the transparency of the test selection process and maintain the test efficiency. It guarantees that the bugs are detected earlier by using automation tools [81, 5]. Moreover, regression testing helps in improving the quality of the app as well as discovering the bugs that may be introduced by chance because of new modifications [81].

6.2.2. BDD for Regression Testing: Case Study 1

Catrobat is a visual programing language, which fully depends on bricks functions and their behavior. Catrobat faces some issues because of its incorrect behavior of the bricks, which is why, we discuss one issue of Catrobat programming language. Listing 1 shows the Cucumber specification for the deterministic crash with broadcast scripts. The following steps reproduce the issues in the Catrobat project.

- Create a new empty program.
- Add this script to the background.
- When the program starts.
- The program immediately and deterministically crashes and the Catrobat has stopped messages.

The expected and correct behavior act in way that the program should execute without crashing. However, we observe some incorrect behavior, which in this particular case the program would run in an infinite loop; the crash would occur already at the second time, the message “1” is sent. Unluckily, the program immediately crashes, and the broadcast brick has stopped message.

```
       Feature: Crash with Broadcast Scripts

       Correct Behavior: The program should execute without crashing.
       Incorrect Behavior: The program immediately crashes at the second time,
                            when the message "1" is sent.

       Background:
            Given I have a program
              And this program has an object ’Object’

         Scenario: Deterministic crash with Broadcast scripts
            Given ’Object’ has a start script
              And set ’var’ to 10
              And broadcastWait ’1’
            Given ’Object’ has a When ’1’ script
              And broadcastWait ’2’
            Given ’Object’ has a When ’2’ script
              And broadcast ’1’
              And set ’var’ to 20
             When I start the program
              And wait 1 seconds
             Then the variable ’var’ should be equal 20
-----------------------------------------------------------------------
       Listing 1. Deterministic crash with broadcast scripts
```

6.2.3. BDD for Regression Testing: Case Study 2

Listing 2 shows the Cucumber feature for “broadcast and wait” brick. It is mentioned in the attached specification of an object that has something simple like a variable changes its value. The expected correct behavior is that the variable “var” should change/increase its value one by one, with five second intervals.The incorrect behavior, the variable “var” first waits for 5 seconds, then incorrectly change/increase its values one by one without further waits.

```
       Feature: Broadcast and wait

       Correct Behavior: The variable should change/increase their value with
           five second intervals
       Incorrect behavior: The variable change/increase their value without
           further waits.

       Background:
              Given I have a program
               And this program has an object 'Object'

       Scenario: Broadcast and Wait brick does not wait.
              Given 'Object' has a start script
               And when receive 'hello'
               And change 'var' by 1
               And wait 5 seconds
               And when program starts
               And forever
               And broadcastWait 'hello'
               And forever end

              When I start the program
               And wait 1 seconds
              Then the variable 'var' should be less than or equal 4
----------------------------------------------------------------------
              Listing 2. "Broadcast and with" brick does not wait.
```

6.2.4. BDD for Regression Testing: Case Study 3

Listing 3 shows and specifies the Cucumber feature for the broadcast brick that incorrectly called two times. The following steps will reproduce the issue.

- Create a new program in landscape mode (the bug does not appear in portrait program).
- Add this script to the background (create the variable as a variable for all objects, i.e., as a global variable).
- When starting the program of the script activity (do not switch to another screen, otherwise bug less frequently occurs and thus it is more difficult to observe the issue).
- You have to observe whether the screen shows 1.0 or 2.0 after one second of execution. If it is 1.0, which would be the correct, go back to the script view activity (simply pressing

the “Restart” button in the Pause menu on the stage is not sufficient to observe the bug, as it never occurs after a simple “Restart”) and then start the program again. Repeat this step until the bug appears i.e., 2.0 is shown on the screen.

Note: In most cases, the bug occurs already during the first execution. Correct behavior: The correct value on the screen should be 1.0. Incorrect behavior: When the bug occurs, the incorrect value shown on the screen is 2.0.

- Additional observation / reservation that this is not just a calculation issue. This bug can also be observed with any other bricks e.g., replacing the “Change var by 1.0” by a “Move 10 steps” (with a look) will be executed as “Move 20 steps” the look will move twice as for, etc. Apparently, the whole “When I receive” broadcast script is incorrectly executed two times instead of just once.
- This bug also occurs if the “Broadcast” brick is replaced by a “Broadcast and wait” brick.
- This bug occurs much less frequently when the “Wait 1 second” brick is deleted.
- This bug makes it impossible to create complex programs in landscape mode, as there is no workaround (besides not using the landscape mode). Therefore, it should be considered as a critical bug.

```
Feature: Broadcast incorrectly called.

Correct behavior: The correct value of the variable should be equal to 1
Incorrect behavior: In Landscape mode, when the bug occurs, the incorrect
                      value will be 2
Background:
Given I have a program with landscape
And this program has an object 'Object'

Scenario: Broadcast incorrectly called two times
Given 'Object' has a start script
And wait 1 seconds
And broadcast 'hello'
Given 'Object' has a When 'hello' script
And change 'var' by 1

When I start the program
And wait 1 seconds
Then the variable 'var' should be equal 1
--------------------------------------------------------
Listing 3. The broadcast mechanism is incorrectly called two times
```

Chapter 7

Bidirectional Languages Issues, Challenges and Testing for Android Apps

With the growing mobile application market far beyond the English-speaking world, it is important for the application to support different scripts and data formats to reach all the potential customers. To make that happen, the most important step is to develop the application with the consideration of internationalization and localization in mind. The below sections provide some generic information about the internationalization elements and to test all the feature right-to-left language and diagnosed bugs in the localization and internationalization.

7.1. Internationalization and Localization in Mobile Application

Globalization (G11n) is a two-step process such as internationalization (l18n) and localization (L10n). The software internationalization brings up the process of re-engineering system that supports different languages. It deals mostly with the functionality of the software, enabling back-end technology with the backing target languages as well as the local languages in which the software is used. The software localization deals mostly with the linguistic and cosmetic or front-end aspects of a software application containing cultural, local-specific content, software design and translation. In the same way, testing completed in two stages, localization testing and internationalization testing. It brings up, the practice of familiarizing an internationalization software for a specific nation or culture by adding local-specific features and translated accordingly. For unique features of the application and bidirectional software, testing requires specific methods [56, 55]. Figure 7.1 shows the relation between Globalization, Internationalization, and localization.

Globalization (G11n) = Internationalization (l18n) + Localization (L10n)

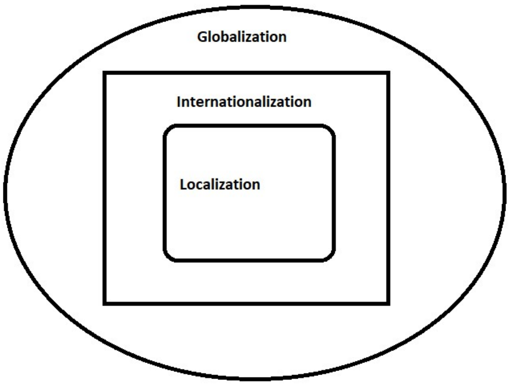


Figure 7.1.: Relation between Glln, 118n and L10n

7.1.1. Bidirectional Localization Testing

In recent years, the use of mobile applications has increased and played a vital role in our daily life. The software apps should be correct everywhere in the world. Thus, the software applications, before distributing into the markets, the internationalization and localization testing, whether manually or automatically, must be performed. The localization testing of mobile application faces many issues just because of the difficulty of testing and limited incomes. The Android operating system, supporting different GUI elements and a huge number of different desktop applications, is used by end-users and can freely choose display resolutions. This aspect ratio of devices, which make the development of the app GUI with all these combinations, is a difficult challenge [54]. In this advanced software localization area, the Arabic language is the challenging language. It is RTL, cursive, bidirectional (BiDi) and context dependent[56]. Bidirectional (BiDi) describes any software products manipulating and displaying text in both directions, i.e., LTR and RTL. Bidirectional script processing algorithm, described by Unicode such as Bidi Algorithm, confirms exact rendering and formatting of Arabic script. The requirement regarding BiDi affects the coding, design and testing of the internationalized app [82].

7.1.2. Characters Re-ordering

The Bidirectional (BiDi) languages such as Arabic, Farsi, Urdu, and Hebrew, where the text is read and written from RTL while numbers are read and written from left-to-right (LTR). It required layout customizations not only for text but also for all user interfaces [56, 83]. The user enters text with order of virtual-key inputs in logical sequence for specific scripts and the visual order in which the characters are rendered on the screen are different. To solve the important challenges, i.e., RTL scripts in BiDi context, the character positioning and a hat characters movement in which LTR and RTL character coexist. The bidirectional context could be a mixture of Latin, Hebrew or Arabic text with numbers that have an LTR characteristic [56].

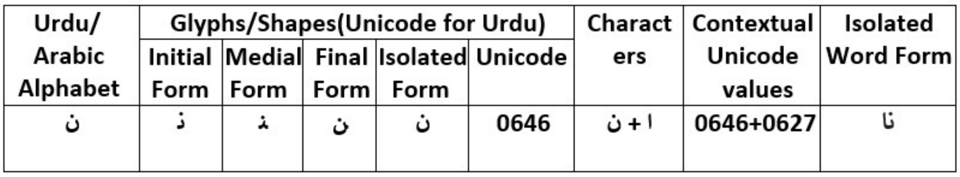
7.1.3. Contextual Shaping

One of the main source of vocabulary for many languages (e.g., Urdu, Hindi, and Bengali) are the Arabic language. The Spanish and Portuguese also have a huge amount of Arabic loaned words. In the area of software localization, the RTL languages such as Arabic, Urdu, etc., are still facing many challenges [83]. The RTL languages (Arabic, Urdu, Persian, etc.) characters are connected and their font is cursive. The characters glyph of these languages are four different shapes, which depend on glyphs position used in a word and the surrounding characters. The letters are *isolated*, *initial*, *medial* and *end* or *final* form and take a different shape [56]. The illustration of different shapes of the RTL languages character is shown in Table 7.1.

There is only one Unicode, defined in different encoding standards for all different glyphs, which is the challenging part of contextual shaping. At runtime layout and rendering mechanisms should conclude the suitable glyph used from the font tables depend on the context[56]. In RTL languages, such as Arabic or Urdu, all these shapes are known as glyphs in which each character has four shapes which depend on the context used. For instance, the alphabet Nun' in Urdu has four shapes/ glyphs, i.e., the *isolated* form of the alphabet in Urdu is ”ن”, *Initial* form, ”نـ” , *Medial* Form, ”ـنـ”, *Final* form ”ـن”. Urdu has a different glyph and is a context-sensitive language of an alphabet depend on the position of use [84].

Ligatures. In Arabic and Indic families of languages, the groups of two or more characters linked together to form a new glyph and an independent word that is the complex and challenging script [83, 56]. The example of the ligature regarding “nun” and “alif” are explained in Table 7.1.

Table 7.1.: Four different shapes of Urdu character



Urdu/ Arabic Alphabet	Glyphs/Shapes(Unicode for Urdu)					Characters	Contextual Unicode values	Isolated Word Form
	Initial Form	Medial Form	Final Form	Isolated Form	Unicode			
ن	نـ	ـنـ	ـن	ن	0646	ا + ن	0646+0627	نا

7.1.4. Challenges of Bidirectional Languages in Mobile Apps

The important issues in bidirectional languages in mobile apps are, text string direction, character sets and encoding, Bidirectional text and sorting order, show variable, variable/object name with RTL languages, String concatenation and the joining of the string is also a challenge. In concatenated strings, placeholders contain data, which is also a challenge due to different word order. The localization testing is usually focused on checking the graphical user interface, for missing grammar, translations, spelling issues and layout problems [54, 85].

7.1.5. Why Localization Testing?

In March 2014, the data collected by Google and admob, that the mobile applications were neglected and stopped using apps just because of insufficient localization. The varies between 34% and 48%, depending on the origin of the data (United States, China, Japan, United Kingdom and South Korea) [53]. Therefore, in this situation, the business market cannot afford to distribute software that contain bugs or usability issues. It is related to that software, which do not support many languages. The users switched to any other product just because of bad experience and uninstall non-localized apps. The Arabic language is spoken in more than 23 countries, and there are more than 30 language scripts used which are derived from RTL languages. Therefore, multi-language and bi-directional script support is needed [53]. For quality assurance, localization testing is the type that mainly focuses on the quality of the localization and evaluation of the products functionality and cosmetics. The objective of the automatic localization testing for BiDi-languages are as under [54].

- The developers, who do not know about the language and cultural background, to documents the attributes of different localization issues.
- To make sure at a later stage the localization is stable even when the bugs and deficiencies are introduced.
- The localization defects should be reported and detected.

7.2. Catrobat Programming Languages Function

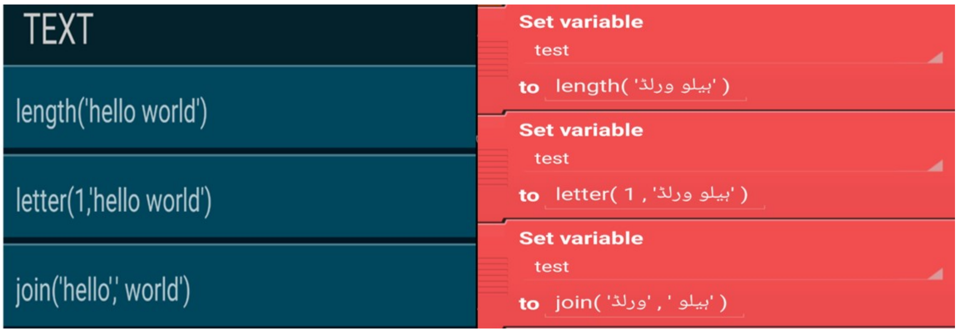
Catrobat has many features and functions. A function is a unique mapping from the domain (the input) to the range (the output). The function must return a value. Here we discuss two important types of function in Catrobat, one is STRING Function and other is LIST Function.

7.2.1. STRING Function

A function is a set of mathematical operations performed with one or more inputs (variable) that result in an output function will take one or more real numbers as an input and return a numerical output. However, a simple function might return any value. Such a function of LTR string looks like in Catrobat are as follow.

- length(hello world)
- letter(1, hello world)
- join(hello,world)

Figure 7.2 shows string function and the example for RTL language in Pocket Code. A string is a series of encoded letters, which is constructed with a specific decoder. Generally a string of text, ready by the computer as plain text. The string function is the sequences of binary encoded computer character including spaces. The letters and numbers of any length up to 10240 characters which include any type of character and put to work in the brick, mostly found in the operator section of the application [86].



```
TEXT
length('hello world')
letter(1,'hello world')
join('hello',' world')
```

```
Set variable
  test
to  length( 'ہیلو ورلڈ' )
Set variable
  test
to  letter( 1 , 'ہیلو ورلڈ' )
Set variable
  test
to  join( 'ورلڈ' , ' ہیلو' )
```

Figure 7.2.: Snapshot for (a) String Function in Pocket Code (b) Example for RTL language.

7.2.2. LIST Function

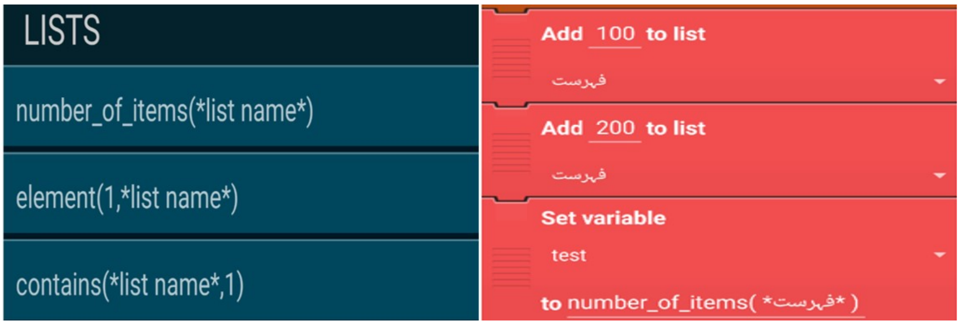
In list function, items are stored with the values. There are three List functions in Pocket Code are as follow.

- number_of_items(*list name*)
- elements(1,*list name*)
- contains(*list name*, 1)

Figure 7.3 shows a list function and the example of RTL language in Pocket Code. List is a tool, which can be used to create, save and store several pieces of information at once. Therefore, it can be defined as a variable having many other variables and consists of numbers paired with items. Its paired number can retrieve each item, in the list function. The list, function items can be added or deleted manually or by programming [86].

7.3. Approaches to Specification Based Testing in Catrobat Project

Android base Cucumber JVM has already been integrated with the existing code of Catrobat and specified in a behavior-driven way. Many executable specifications have been created for visual programming language so far. These Cucumber specifications are business readable even for those people who are not familiar with the programming languages. Listing 1 introduces a Cucumber feature for variable names with different language (LTR, RTL) words with the help



LISTS

number_of_items(*list name*)

element(1,*list name*)

contains(*list name*,1)

Add 100 to list

فهرست

Add 200 to list

فهرست

Set variable

test

to number_of_items(*فهرست*)

Figure 7.3.: Snapshot for (a) List Function in Pocket Code (b) Example of RTL language

of Gherkin language. For example, we add two variable with different names, one variable name is in RTL language, and the other is in LTR language. These are the test cases in the form of scenario and pass a variable value. In this specification, the correct behavior of the program in Catrobat bricks should be equal to 14. Therefore, the localization of the Catrobat project and their bricks are working correctly otherwise, bugs will be diagnosed.

```
        Feature: Variables with different name.

        Background:

        Given I have a program
          And this program has an object 'Object'

        Scenario: Add two variable with different name(RTL & LTR) Languages.
        Given 'Object' has a start script
          And set "متغير" to 4
          And set 'Variable' to 10
          And set 'test' to "متغير" + 'Variable'
         When I start the program
         Then the 'test' should be equal to 14
------------------------------------------------------------
        Listing 1. Cucumber feature for variable (RTL & LTR)
```

7.3.1. Functions List with Cucumber Specification

The localization relating testing with Cucumber specification for verifying each function with RTL languages. Listing 2 shows and specify the basic functionality of List function, especially with RTL language in Catrobat.

```
Feature: Functions list.

Behavior:- In the List function using RTL(Urdu/Arabic), we test number of
           items, exact element as well as the element in the list in the form
           of true and false
 Background:
 Given I have a program
 And this program has an object 'Object'

 Scenario: In this function, we test number of items in the list.
 Given 'Object' has a start script
 And add 100 to list "پہلی فہرست"
 And add 200 to list "پہلی فہرست"
 And set 'test' with the no of items in the list "پہلی فہرست"
 When I start the program
 Then the 'test' should be equal to 2

 Scenario: In this function, we test the exact element in the list.
 Given 'Object' has a start script
 And add 10 to list "دوسری فہرست"
 And add 20 to list "دوسری فہرست"
 And set 'test' with the element 2 of "دوسری فہرست"
 When I start the program
 Then the 'test' should be equal to 20

 Scenario: In this Function, We test the element in the list in the form of
           true and false.
 Given 'Object' has a start script
 And add 10 to list "تسری فہرست"
 And add 20 to list "تسری فہرست"
 And set 'test' with the "تسری فہرست" contain 20
 When I start the program
 Then the 'test' should be true
-----------------------------------------------------
 Listing 2. Cucumber feature for function list (RTL)
```

There are three scenarios in the above executable specification for the list function in Catrobat.

- **number_of_items (*list name*).** In this function the reporter block that reports, how many items are in a list. We test the number of items in a mentioned list with RTL language by using executable specification.
- **elements (1,*list name*).** In this scenario, a reporter block that reports the value of the specified entry in a specified list in Catrobat. It tests the exact element (RTL Languages) in the list function by using executable specification.
- **contains (*list name*, 1).** The function *contains ()* block is a boolean block. In this function, we test the element (RTL languages) in the list in the form of true and false using executable specification.

7.3.2. Functions Strings with Cucumber Specification

Listing 3 shows Cucumber feature for text related functions in Catrobat, especially for RTL. The example specifies the basic functionality of *string* functions one by one. In these functions, we test alphabets of any word in respect of RTL languages with the help of Gherkin languages using Cucumber specification.

```
Feature: Text related functions.

Behavior: - In this function we test alphabets of any word, join any two words
             as well as the length of any word in RTL languages.

Background:
Given I have a program
And this program has an object 'Object'

Scenario: In this function, we test the length of the words in RTL languages.
Given 'Object' has a start script
And set 'test' to length "ہیلوورلڈ"
When I start the program
Then the 'test' should be equal to 8

Scenario: In this function, we test alphabets of any words regarding
          RTL languages.
Given 'Object' has a start script
And set 'test' to letter no '1' of "سافٹ وئر"
When I start the program
Then the 'test' should be equal to "س"
```

```
  Scenario: In this function, we test and join any words RTL languages.
  Given 'Object' has a start script
  And set 'test' with join "ہیلو" and "ورلڈ"
  When I start the program
  Then the 'test' should be equal to "ہیلوورلڈ"
---------------------------------------------------------
  Listing 3. Cucumber feature for function text (RTL)
```

Feature 3 shows three scenarios, first is about the **length(hello world)** function to test the length of the *word/string* in RTL languages with the help of Cucumber specification. The *length of ()* block is an operator block and reporter block. The block reports that how many characters, words/string contains. The spaces count as characters in the string. To test and calculate the number of RTL characters in *word/string* in Catrobat function in between the parenthesis ”ہیلو ورلڈ” i.e. **length(”ہیلو ورلڈ”)**. The output value should be equal to 8. Therefore, by putting any RTL text/string in between parenthesis, the scenario counts the alphabets of the string and give the exact result.

The second scenario is about l**etter(1, hello world)** function to test and show character on a specific location in a mentioned string. In this scenario, we test a single character of any word/string regarding RTL languages in the Catrobat function i.e. **letter(1, ”سافٹ ویئر”)** using Cucumber specification such as ”سافٹ ویئر” the first character of the string should be equal to ”س”, the test pass otherwise fail. The block reports the specified character of the given *string*; it reports all characters, including letters, numbers, symbols and even spaces.

The third scenario is about **join(hello,' world')** function test and join any RTL language strings with the help of Cucumber specifications. The joining of two strings connected without space. The block concatenates the two values together and reports the result.

Correct behavior: For example LTR string “hello” and “world” are placed in the block, it reports the result, i.e., “helloworld”. In this scenario, we test and join any words in respect of RTL languages in the parenthesis. It should be reversed and the appending is from RTL, i.e. **join(‘ہیلو‘, ‘ورلڈ‘)** the test result should be equal to ”ہیلو ورلڈ”.

Incorrect behavior: We tested and diagnosed with the help of Cucumber specification that the Join () function is not working properly in Catrobat. It reads/concatenates both parameters from LTR. When we put RTL language parameters in a block, it reads and concatenate from RTL direction instead from LTR, i.e., **join(‘ہیلو‘, ‘ورلڈ‘)**, the automatically or manually test result gave incorrect result/report i.e. ”ورلڈ ہیلو” which is the incorrect behavior of the Join () function. For said issues/bug, we reported our Catrobat developer team for correction.

7.3.3. Set variable and Show variable with Cucumber Specification

Listing 4 shows Cucumber feature file for the *set variable*. The correct behavior that the variable text, RTL language should be shown on the stage. The incorrect behavior, the variable text RTL language is not shown on the stage. The purpose of this automatic test case is to test the language character rendering, especially RTL languages such as Arabic, Urdu, Persian, etc. In order to complete this type of testing, a small program is created using Catroid, and this program contains two bricks one brick to *set variable* and the other to *show variable*.

Issue: - When the value of a variable is Latin character “LTR character” the value of this variable is completely displayed on the Screen/stage. However, when the value of a variable contains RTL character, then these characters are not displayed on the screen/stage. Variable is a name set in a storage area that our programs can manipulate and can use this variable, in a global variable accordingly. Therefore, it can set a value in this variable and get value on any activity/broadcast receiver in the application environment. A *set variable brick* designation expresses the compiler where and how much storage to create for the variable while the show variable brick display the (integer or string) on the stage/screen.

```
Feature: SetVariable

Correct behavior: The set variable text, RTL languages i.e. Urdu/Arabic should
                  be shown in the stage.
Incorrect behavior: The set variable text, RTL languages are not shown in
                  the stage.
Background:
Given I have a program
And this program has an object 'Object'

Scenario: Set Variable & Show Variable then take screenshot of the stage.
Given 'Object' has a start script
And set 'var' to "متغير"
And Show Variable 'var' at X 100 Y 200
When I start the program
And wait 2 seconds
And take a screenshot of the stage
Then the stage/screenshot is not empty
--------------------------------------------------
```

Listing 4. Cucumber feature for variables

Screenshot/Image/Pixels

The screenshot is made up of pixels having a horizontal and vertical position in a two-dimensional coordinate. The screenshot having width and height then the pixel array has a total number of elements equaling Width * Height for the horizontal and vertical point in the screenshot. Now we discuss pixel-by-pixel searching technique, which is very simple. This technique gets and search the color of each pixel one by one in the selected screenshot. Pixel-by-pixel searching is considered successful only if the color of all the pixels are white then the selected screenshot will be considered white/empty. If the color of one pixel is not white then the selected screenshot will not be considered white/empty. To enhance the quality of the screenshot, the capturing mechanisms must have the same configuring. Some factors affect screenshot rendering such as Screen Resolution, Color Depth, Font Smoothing, and Image Format.

RTL Languages Characters Rendering Test

To make sure that RTL character is rendering correctly on the stage. We need a fast method to check and detect that the screenshot pixels are white, transparent and empty. For testing objective, we take a screenshot with the same configuration, the one which has a set variable brick containing RTL characters. The proposed test case check all the pixel values of the screenshot, If all pixels are with white color, this means that the characters of RTL languages are not rendering correctly on the stage and the test should fail otherwise, the test should pass as illustrated in the below mentioned code.

```
@Then("^the stage/screenshot is not empty$")
   public void theStageScreenshotIsEmptyOrNot() throws Throwable {
   File desktop = Environment.getExternalStorageDirectory();
   File directory = new File(desktop.getAbsolutePath() + "/Pocket Code/My
   first program/Scene 1");
   File screenShotFileA = new File(directory, "automatic_screenshot.png");
   FileInputStream streamInA = new FileInputStream(screenShotFileA);
   Bitmap ScreenshotA = BitmapFactory.decodeStream(streamInA);
   streamInA.close();
   assertFalse("manual_screenshot is White and Transparent",
   screenshotIsEmpty(ScreenshotA));
   }
```

In order to complete the test case the (screenshotIsEmpty) method is implemented.

Chapter 8

Behavior-Driven Development for Android Location-Based Services

In this chapter, we discuss the behavior-driven development practices for location-based mobile apps and show how the behavior-driven development identifies business goals and look for features that will help deliver these goals. Moreover, such features are automated in the form of executable specifications in order to validate the location-based apps and provide automatically updated technical and functional documentation.

We have recently been observing the increase in popularity of Location Based Services (LBS). One of the reasons why LBS are so popular in our days is the ability of LBS to identify the users needs for services and offer appropriate services to meet such needs, based upon a users current location. According to data collected by Statista in July 2015, the ratio of smartphones users aged 18 to 29 years who accessed LBS reaches 95%[1]. On the other hand, IDATE DigiWorld published in 2017, that the percentage of mobile users in selected European countries using LBS, varies between 35% and 47.9% depending on the origin of the data (United Kingdom, France, Germany, Italy, and Spain)[2]

Location-based function services in mobile apps not only improve mobile user experience but also bring new challenges and issues in software testing. Location-based apps use large volumes of location data and require scalable network and data services to deliver users with high-quality and easily maintainable LBS. Mobile application testing is a vital and significant research subject due to the explosive increase in mobile app downloads and mobile users. Therefore, testing is required for quality validation of these apps [87]. Nowadays, huge research is focusing on LBS and many of these services have been implemented and should be tested efficiently [88]. Testing such apps is a challenging task because of numerous reasons: the mobile device fragmentation and limitations, a large volume of location information, different techniques employed including

[1] https://www.statista.com/statistics/191842/percentage-of-location-based-service-users-in-the-us-by-age-group/

[2] https://www.statista.com/statistics/294314/share-of-mobile-subscribers-using-location-based-services/

geospatial and location techniques, and Internet and wireless communication technologies [87].

Certainly, there have been many types of research on the development and testing of mobile apps, but unfortunately, less consideration has been paid to the quality assurance of location-based mobile apps [87]. Therefore, this thesis focuses on location-based mobile apps issues and proposes a new testing approach to address such issues. The aim of this thesis is to show how we can integrate and perform BDD practices in mobile apps. For a better understanding of acceptance testing, the BDD provides a common natural language to facilitate communication between team members and business stakeholders. To promote and implement BDD based specifications for LBS we need the help of Cucumber tool.

8.1. Android Location-Based Services

Location-Based Service (LBS) is a platform, which produces geographical location information based on existing or an identified position, supported by the electronic map platform. LBS is able to find the current geographical location of the mobile phone and then provides services based on this location information [88].

The location base information, such as latitude and longitude coordinates of the smartphone subscribers, could be received through the Global Positioning Satellites (GPS), cell identification (Cell ID), a broadband satellite network, or Wireless Local Area Networks (WLAN). For example, a mobile user can request for information about the closest pharmacy store based on her location in the town. Thus, it could be stated that LBS transfers location information between mobile and/or static users via the wireless network [89].

Similarly, It is necessary to integrate the mobile computing technology and the location provider technologies in order to match the demands of location base services. It is considered as one of the most promising apps of the mobile world. LBS provides services to end-user with information modified by the mobile current physical location, i.e. the closest place, school, park or hospitals, which are retrieved from a spatial (3-D or 2-D) database stored remotely in the location-based services server. This service has not only helped single smartphone customers, but also play a great part in public community protection, public transportation, emergency response, and disaster management [87, 88].

LBS offers the modern mobile world with tools for effective mechanism, management and constant control. So, many users employ LBS in their business, industry and life to better accomplish their aims. The growing interest in marketable LBS has forced scientists to concentrate on more accurate positioning solutions. It employs accurate, real-time positioning to connect mobiles subscribers to points of their concern and informs them of the current locations conditions such as traffic and weather conditions, or introduces routing and tracking information using wireless communication technologies [88].

8.2. Why the Behavior-Driven Development is Important for Location-Based Apps

The importance and the key benefits for using the BDD approach in the location-based Android apps are as follows.

8.2.1. The Code is Itself Documentation

The executable specifications act as living code documentation. The documentation uses terms that stakeholders are familiar with. However, it is very easy for testers, developers and customers to understand the common language that what the software application actually does. Also, the technical documentation for developers is generated from the low-level executable specifications, making it simpler for them to understand the app's code and to perform their own changes. This documentation is always up to date, is inexpensive to maintain, includes working code samples, and shows the purpose behind each specification.

8.2.2. Eliminate Misunderstandings

There are often misunderstandings between customers, developers, and testers that are eliminated by writing stories in a ubiquitous language. While programmers can also write and integrate feature files into their development process.

8.2.3. Efficient Features and Reusable Steps

Cucumber specifications with their reusable steps, the Gherkin ubiquitous language with its concise structure, are efficient for testing. These specifications can be edited without recompiling the code base. Once a step definition is stored in a software project, the new feature can be added without adding new glue code every time.

8.2.4. Multiplatform Independent Specification

The Cucumber tool and Gherkin syntax are the base for platform-independent testing and specification. Cucumber supports different software platforms, such as Android, iOS, Windows Phone, and HTML5. Every Cucumber implementation provides the same functionality, and its processes of execution are similar. As soon as Cucumber is available on other platforms, all the different development teams can share the complete set of feature files instead of relying on their native tests. They will only need to implement step definitions for automatically executable specifications.

8.2.5. Faster Feedback

Cucumber scenarios give faster feedback even in the verification and testing of complicated scenarios. The features are easy to understand and write for non-technical stakeholders.

8.2.6. Right Direction

BDD is useful for getting help from the right people to discuss the right amount of issues at the right moment. TDD has too much focus on testing. On the other hand, BDD helps the team to focus on system behavior rather than on testing. Usually, there is a gap, misunderstanding, and frustration between team members. BDD does not solve this problem alone, but at least it encourages the team members to work more closely and reduce the gap.

8.2.7. Effective Testing Methods

In the mobile world, there are different mobile vendors and mobile platforms. Accordingly, different LBS techniques are used at different platforms for the development and deployment of a mobile app. Also, for some mobile platforms, there is more than one mobile device manufacturer. However, fragmentation is a huge issue in the mobile world and particularly in the Android platform. Android is not the only platform that is affected by this problem; other mobile platforms such as iOS, Windows Phone, and BlackBerry as well. The possible hardware and software combinations on those platforms can also be a problem. Testing of LBS apps, especially, manual testing is time-consuming and is not sufficient. The LBS apps will most likely contain lots of bugs and issues that potential users will find. However, proposing new automated testing methods could enhance the evaluation process and provide a fast and efficient approach.

8.2.8. Huge Information Quantity

One of the most common apps of big data is LBS. However, location is used in many of various kinds of apps such as map apps, camera apps, and social media apps. Users can share their current location with apps and send their current location. If the app under test uses the location sensor to find the devices current location, a huge amount of location information the app will receive.

8.3. Behavior-Driven Development Practices for Location Based Services

In this section, the design and implementation of testing LBS using BDD are presented. BDD is a light of hope and is more appropriate to test this type of app because it seems to concentrate more on the functionality of the app rather than on how it should be built. When a team practicing BDD wants to describe LBS requirements and request in a more testable method, in a manner that both the developers and business stakeholders can easily understand, they should implement a Cucumber feature. Further, they collaborate with customers and other stakeholders to define stories and scenarios of what customers need this feature to present. In Gherkin language, the requirements of a specific feature are combined and added to a separate text file called a feature file. A feature file includes a brief explanation of the specification, followed by a number of scenarios, or examples of how a feature acts. In specific, the customers help explain a set of concrete examples that demonstrate the key outcomes of the LBS feature. These executable examples use a common language vocabulary and can be easily understood by all stakeholders and development team. It is usually expressed are as fallows:

- Given: Condition to allow the user to start the use case
- When: The first action of the use case
- Then: The final action of the use case

A Cucumber base executable feature file is an automated test that shows and verifies how the app presents a particular business requirement. In addition, these automated and acceptance tests run as a part of the build process and run after a codes changes to ensure that the software is still working in the same way as before the changes. Usually, LBS specifications are written in a natural language which is a poor way of communicating requirements, especially for nontechnical stakeholders and users. Thus, examples are an excellent way to eliminate the space of ambiguity, assumptions, and misunderstandings and clarify the requirements (see Figure 8.1).

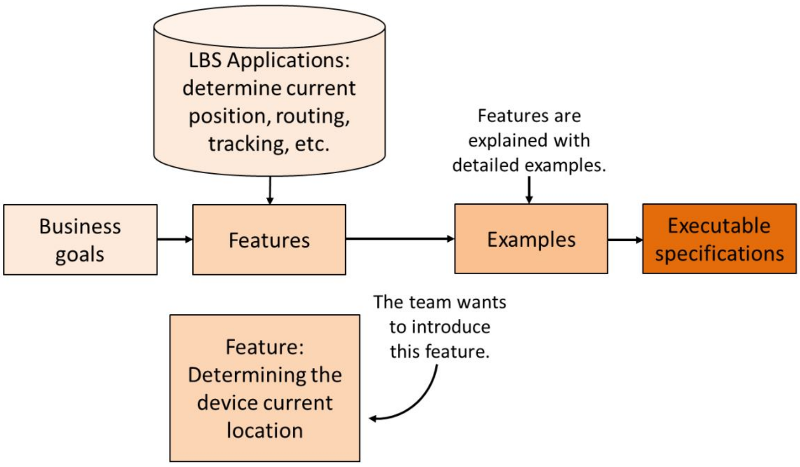

LBS Applications: determine current position, routing, tracking, etc.
Features are explained with detailed examples.
Business goals
Features
Examples
Executable specifications
The team wants to introduce this feature.
Feature: Determining the device current location

Figure 8.1.: The sequence of activities the team should practice for each LBS feature in BDD

8.3.1. Routing Service (Cucumber base model for LBS)

In addition to determining the location of different destinations, LBS can also be employed to provide routing and tracking information to users and guide them along the best routes. The tracking system obtains the users latitude and longitude and then converts them as the location. The system monitors the movement of the user from anywhere in real time as illustrated in the below Cucumber base model specification in Listing 1.

```
Scenario: Routing service (Latitude and Longitude).
Given the area location
| Area | Latitude | Longitude |
| A | X1 | Y1 |
| B | X2 | Y2 |

When evaluate the direction
| From | Toward | Direction | Distance |
| A | B | AB | -- |

Then I should get the routing information
-------------------------------------------------------
Listing 1 . Cucumber base model for LBS
```

8.3.2. Determining the Device Current Location

The LBS can be employed to determine the devices geographical location. The app translates latitude and longitude coordinates into location name. The latitude and longitude represent respectively the vertical and horizontal location on the Earth. However, in below stated BDD based examples play a basic role, helping everyone understand the requirements more readily and precisely. For instance, a simple example that illustrates the “Determining the device current location” feature might look like this in the blow mentioned Listing 2.

```
Scenario Outline: Determining the device current location
Given I have a Latitude value with <lat-value>
And I have a Longitude value with <long-value>
When evaluate the location information
Then I should see <new-location-name>

Examples:
| lat-value | long-value | new-location-name |
| 48.210033 | 16.363449 | Vienna, Austria |
--------------------------------------------------------
Listing 2 . Example using Gherkin syntax for LBS
```

Chapter 9

Behavior-Driven Development as an Error-Reduction Practice for Mobile Application Testing

This chapter have been published in the [90].

9.1. Abstract

With the rapid development of mobile technology, there is a significant increase of mobiles impact in our daily life. This brings new business requirements and demands in mobile application testing, introduces new issues, and challenges in their automation. We introduce the Behaviour-Driven Development methodology for developing the Catrobat project. With Behaviour-Driven Development base tool (Cucumber), we develop executable feature files to express business requirements, which can be read and understood by the whole team. The purpose of this study is to present the critical issues and challenges of Catrobat. In particular, we test the broadcast mechanism for right-to-left languages from different angles and track regression errors as well as specify and diagnose localization issues. The results show that the proposed approach is able to expose the application deficiencies in the Catrobat script mechanism, ensures that the app meets bi-directional requirements, and guarantees that the app is more reliable and better documented.

9.2. Introduction

Currently, the software development process should ensure system availability, functionality, and cost reduction. It is also expected to contribute to business goals. According to the World Quality Report 2018, the importance of ensuring end-user satisfaction is a key objective of the quality assurance and testing strategy. This survey also reveals that the digital transformation creates higher demands on quality assurance and testing approaches, and that a large proportion of enterprises have some serious challenges. While doing this survey, when the mobile testers were asked the possible challenges and testing their applications, they responded differently. 52% of the respondents pointed that they did not have enough time to test an issue, followed by 43% who said that we do not have the right tools to test. 28%, among them, believed that they do not have in-house testing environment while 34% said that we do not have the right testing practice [91].

Mobile applications are mostly prone to errors because of the developers unfamiliarity with mobile platforms. However, the increasing complexity of mobile applications can arise many challenges in the testing process in order to make sure the app will operate and meet the end users expectations. Smartphones are becoming common; this exposes the necessity for effective techniques for testing their applications. Mobile application testing plays a vital role in making mobile applications more reliable and bug-free [2, 6].

In this advanced era, the speed of delivering mobile applications for the IT companies is a key challenge. However, in the past, the stages of a project is measured in months and a project is competed over a number of years. Contrarily, currently, the projects have been delivered over a least number of times and the stages of a projects development are measured and setup in weeks or sometime even in days. Therefore, in this rapidity of changes the project functionality documentations are becoming a challenge for the software community. Under these environments, testing the mobile applications has been taking place on two points: first the right development of the product, and second the development of the right product [74].

Mobile apps developed to address more and more critical areas, which is not only complex to develop, but also difficult to test and validate. The difficulty, diversity, and functional richness of smartphone apps are increasing and the demand for mobile apps is offering even more complex, rich, and usable functionalities, which are going to grow more and more in the coming future. Unluckily, the quality of smartphone apps is often poor just because of the very fast growth and development processes in which testing activity is ignored [7]. Moreover, the Android market fragment is large, as well as the several sets of scenarios in which a mobile application can be used and makes the testing of a new application more costly, time-consuming and a complex task [6].

In addition, the localization of mobile applications is still infrequent because of the shortage of research. There are many applications in the market and, at the same time, companies are not willing to pay attention and not enough consideration for the future expansion of the products. The fixing of internationalization bugs cost around 30 times more than handling these bugs upfront [6][]. The quality of localizing app for right-to-left (RTL) languages is often still inadequate and cannot be compared with the standards and quality of other localized products. Even software companies like Microsoft and IBM still find it challenging to achieve a sufficient quality level. The RTL languages use non-Latin based alphabets. The glyph type of character in the Arabic language depends on the position of the character within a word because the letters are connected to each other [83].

Currently, the software community has focused on technical practices for high quality and to build the product right. Therefore, it is equally important to build the right product. However, it needs a different technique like specification (Behavior Based Testing and Black Box Testing). The specification-based tool is very helpful, which supports the development process and solves many problems outright. The precise specification helps to reduce extra work initiated by ambiguities and provide many advantages for the overall progress automatically. However, mobile applications quality is a must and therefore their testing is essential. Meanwhile, the adoption of agile software development has been growing. The percentage of teams using agile base practices in their organizations is 52% [2, 9]. Agile practice is suitable for the mobile application development process. Many studies have shown that agile practices are the best choice that assures different phases of software development life cycle and solve the mobile app development issues more efficiently [9].

Test-Driven Development (TDD) is an agile base and a short development-cycle approach for writing software automatic tests before writing functional code [11]. Like TDD, ATDD also consists of creating tests before writing code and all these tests represent the expected behavior of the software. The Behavior-Driven Development practice is a combination as well as enhancement from TDD and ATDD. This practice focuses on the behavioral aspect while the test driven development emphasizes the implementation aspect. Additionally, BDD usually writes acceptance tests in an English-like language (Common language) to help all the stockholders to understand the implementation rather than exposing the code level tests. BDD encourages and bridges the gap between the problems and the solution domain, providing a better understanding between both the development team and business stakeholders [92].

Meantime, a free open-source, the Catrobat visual programing language (CVPL) provides an easy opportunity for the young children to build their own animations and games without any programming awareness and encourage them to generate and share their own mobile apps. The teenagers can simply learn how to program without having to think about the drawbacks like compile-time errors or complicated workflows. Motivated and inspired by Scratch , the Catrobat project also defines visual blocks, which can be snapped together in order to form a single program. Like Scratch programming, the Catrobat base programs can be generated and implemented entirely by using mobile devices [17, 18].

In this paper, the practices of BDD methodology are employed to solve the recurrent testing issues and improve the quality of the Catrobat project. We describe how the proposed executable specifications can test the issues of Catrobat elements (i.e. bricks). Furthermore, we summarize the challenging aspects of the RTL languages, which are facing by the Catrobat developers. With the help of Cucumber, we automate concrete examples and build executable specifications to evaluate the broadcast mechanism, specifically for RTL languages.

9.3. Agile-based Methodologies

This section provides an overview of agile-based methodologies practices. The agile practice focuses and requires less planning, and divides the task into small increments. In this practice, the customer satisfaction is of higher priority with an effort of faster development teamwork with mutual understanding of stakeholders.

9.3.1. Test-Driven Development

TDD (see Figure 9.1) is an agile based technique that incrementally develops software apps. On the other hand, a number of ongoing studies on the capability of TDD identify the software app bugs earlier in the software development practice. In this practice, the software developer writes unit tests from end-user requirements before writing the code itself. Afterwards, the software developer implements the program code needed to pass the tests. During the developing process, when a fault is detected, it is punctually fixed accordingly. As soon as the tests are passed, the software developer implements the refactoring by rereading that what already has been completed to improve the code and design. In TDD lifecycle, while the tests are passed, the code for new functionalities can easily be accepted for the old applied ones, which is called regression errors. Similarly, to recognize the possible regression errors/faults in the developing process, the software developer when implemented functionalities, implements regression tests before proceeding to implement new functionalities in the software [58, 93].

9.3.2. Acceptance-Test Driven Development

ATDD is also an agile based practice, which is deeply interconnected to BDD practice, and both are derived from test-driven development, acceptance tests and unit test from user stories. The ATDD process drives on the specification level in the same way like TDD in code level with unit tests. The acceptance testing performs as specifications for the required behavior and functionality of a software development process. So, when the desires and requirements are expressed by natural language base example, rather than by complex formulas, code or ambiguous descriptions, the required acceptance tests case are expressed with actual examples are easier to read, understand, validate, and write. However, in ATDD an end-user requirement is converted into a set of executable scenario tests for practical implementation, which is legalized against the TDD practice for writing automated unit tests for low-level program creations based on simple user stories [94]. Therefore, in ATDD, the software team creates one or more acceptance tests for the required specification before their implementation. ATDD practice changes the purpose of testing by creating concrete examples of business base rules for clarifying and documenting requirements [92].

9.3.3. Behavior-Driven Development

The BDD is a software development practice based on an agile methodology as well as the advanced form of TDD and ATDD originally developed by Dan North [11]. It provides a common ubiquitous (pure natural language) language to facilitate the communication between the development team and the business stakeholders for better understanding [67]. The key objective of this practice is to build the executable specification of a system. In addition, it always trusts on ATDD and its scenarios are clearly written in plain language, which is easily understandable by the whole team. All the scenarios are easier to maintain and reflect the end-user perspective as well as improves the documentation of the system [11].

Figure 9.1, shows the principles and practices of the two methodologies. Practically, BDD suggests an outside-in approach, which is starting with an acceptance test to begin writing scenarios and work through the model. This approach helps us to effectively implement our feature earlier, and make the right design based on it. When we start with a new feature file, before we write it, we make sure to analyze and understand the problem. At this point, we need to know how the user interface allows a user to do a job and do not worry about the implementation of scenario steps.

The BDD uses the red-green-refactor cycle with Cucumber tool to make sure that the step-definition steps are assigned and the actions respond correctly. Next, to run the scenario, initially it should fail. Therefore, we need to write a step-definition for the first failing or pending steps. Once the first scenario step passes, the tester should move onto the next one and follow the same steps, and then the entire scenarios have to be implemented accordingly. Thus, the scenario passes along with all the underlying specifications. If something goes wrong, the tester refactors further.

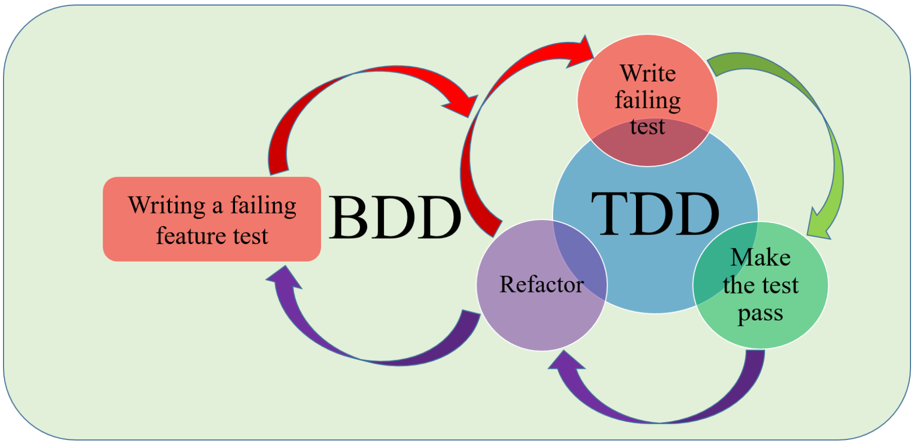


Figure 9.1.: TDD and BDD Process

9.4. The Automation Test Tools

Cucumber is an open source tool used for BDD test automation. It is specifically built for textual specification and its implementation. Cucumber executes specification, which is written

in natural language called features. Feature and scenario are written by the business analyst, developer, and tester [13].
The software experts uses Cucumber framework to execute and run user acceptance tests in the three basic stages. In the first stage, all the product stakeholders work together to write Cucumber specifications (feature files). Usually, a Cucumber feature file contains a list of scenarios and every scenario contains a list of steps as shown below (See Figure 9.2):

- Given: In this step it announce the system as in a well-known state.
- When: In this step, it shows some actions.
- Then: In this step, it confirm and prove the system outputs after actions.
- And: In this step it connect multiple Given, When, and Then steps to make the reading of written steps is more fluent.

In the second stage, the testers write test codes for every step in the test scenarios using the Cucumber mapping mechanism called step-definitions. Finally, in the third stage, testers run the scenarios and get test reports using Cucumber tool in a continuous integration environment [12]. Usually, all the stakeholders manually write Cucumber scenarios that describe system behaviors of the app. Moreover, the testers write implementation codes for the Cucumber scenarios steps and execute them. The Cucumber framework provides transparency about what test scenarios are covered and how the test scenarios are mapped [12, 72].

Cucumber test results are more sophisticated than a simple test case. A scenario that has been executed can end up in any of the following states. These states are designed to indicate the progress as you make your tests. Undefined, Pending, Failed or Passed [13]. For test implementation, we picked Espresso and Robotium as a testing framework, but cucumber is not dependent on any specific testing framework. This means you can work both of them and with other libraries (See Table 9.1).

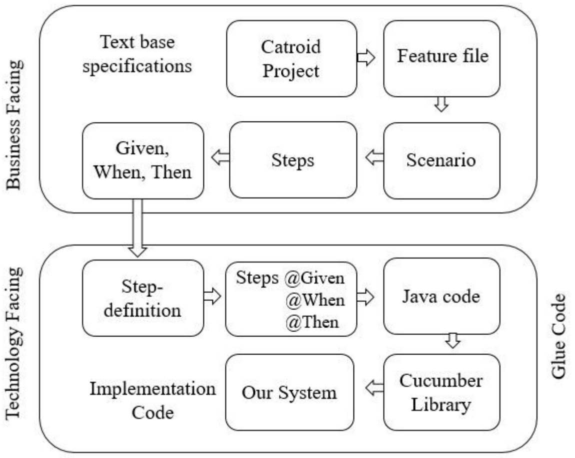


Figure 9.2.: Cucumber concept & work flow in Catroid

Espresso: The Espresso testing framework was launched by Google. It provides testing support library for the Android platform. It supports APIs to write User Interface (UI) testing. The purpose of this framework is to simulate user interactions within a target application as well as it can be run and executed in on emulator or a real mobile device. The Espresso framework is implemented on Android Instrumentation framework. It is a simple automation-testing tool and a small API that is synchronizes with UI thread, which makes it more quick and reliable. Furthermore, the framework does not require any type of sleeping methods [95].

Robotium: The Android-based Robotium is an open source framework. It is developed to facilitate and enable automated testing for software development. It also supports the building of acceptance test scenarios for test cases using GUI components in both emulators and mobile device [96]. Many easy methods are used, which extend the Junit that can be used specific for Android platform base testing. Moreover, the black box test cases, which are performed are real, strong and productive. There are intermediate releases for this automation tool and has good support in the software community. Functions of the system and acceptance test scenarios can be written with the availability of the source code [95].

Table 9.1.: BDD based tool

Platform	Cucumber	Robotium	Espresso
Android	Yes	Yes	Yes
License	Open source	Open source	Open source

Gherkin:- Gherkin is a business readable and domain specific language that Cucumber understands. It is a programming language, specific for the test cases for Cucumber [13]. It does not have a very complex and detailed syntax. The syntax is available in 60 languages, including right-to-left languages in which few keywords are required to use Gherkin as a language. When we run the Gherkin scripts in cucumber, it generates a report based on the keywords. After that, the related information is sent to the mobile test generator for execution [72, 74].

Gherkin files use the .feature file extension and is saved as plain text, and their stories usually have a little, narrative, and a number of scenarios [72]. A story written in Gherkin has a very well defined, but easily readable structure, called Feature. In the Background section, feature file allows to specify steps, which is common to every scenario in the Feature file instead of having to repeat the same steps. Each feature contains several scenarios, and every single scenario is a single concrete example and every scenario consists of one or more steps (see Figure 9.3) [11].
Step-definition is the part where the natural language is converted into the actual working code based on the mapping of the constructs of the natural language. A specific regular expression is used to determine the code, which is to be executed on reading the sentence [72]. With the help of Java code, you can write step-definitions for the rest of the lines. The step definitions are written with Java annotations for methods and those methods implement tests. When Cucumber tests are execute, the step-definitions are created. The testers use annotations to specify the feature files to

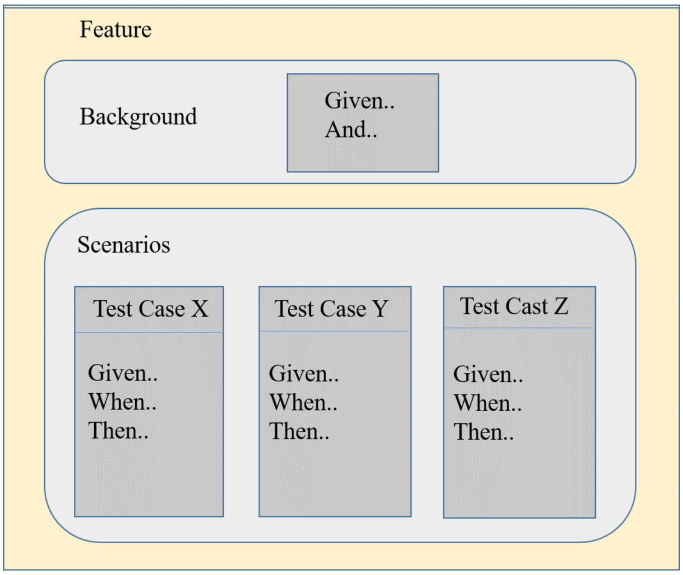


Figure 9.3.: BDD base Given-When-Then Pattern

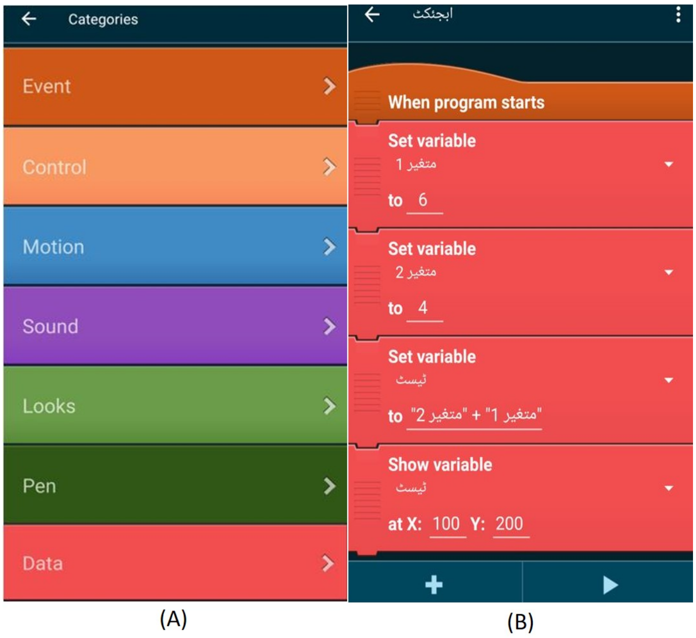
execute in a test. Then Cucumber will look all the step-definitions steps with the scenarios steps and generate test reports [12].
Glue (See Figure 9.2) is the path to step-definitions format and for report outputs. Features is the path to the Cucumber feature files through which we write .feature files under our test project's assets folder. Additionally, we also write our step-definitions Java files under the package name specified in glue. Cucumber test can be implemented and broken down into three easy steps i.e. first one is to write feature scenarios, the second one is to write its step-definitions, and the third one is to write the actual test implementation.

9.5. Visual Programming Language: Catrobat

Catrobat project concept is derived and inspired from the Scratch programming system, which is specific for desktop computers. The Catrobat is a free and open source project, particularly for smartphones. With the help of Catrobat programming language, school children can intuitively create their own apps, games and animations in a very simple way on mobile phones and tablets. Similarly, the notion of bricks used are an atomic element to represent specific statement of the Catrobat programming language. There are control flow bricks for structured programming, but also more specialized bricks, which directly adjust a graphical object on the screen. Although it is implemented in a different programming language and with a different architecture, Catrobat also maintains the principal visual language concepts and program composition from Scratch.

A program contains one or more objects and possibly a set of global variables. An object can possess local variables and typically also has two sets of specialized attributes, namely looks (images) and sounds (audio files), for the audiovisual animations. Most importantly, an object contains a list of scripts. The script is the code portion of a Catrobat program and contains the list of bricks, which in turn incorporates the logic of the entire program. Scripts essentially behave like subroutines because they are triggered by different external or internal events [17, 18]. The Catrobat programming language program always writes with a visual Lego style program. Therefore, the Catrobat base version, which is developed specifically for Android devices is named Catroid and is available on Google Play Store under the name Pocket Code (Pocket Code: https://catrob.at/pc). The product is a learning application for smartphones, which is developed in Austria at Graz University of Technology.

Elements:- Catrobat (VPL) has many kinds of group categories (See Figure 9.4) and every category has particular associated bricks, which are clarified as follow. Event is the important category for every single project to start a program. The elements of Control category is accountable for control flow bricks i.e., conditional and loop. Motion is the category in which the elements are adjusted with the position of an object on the screen either directly or by using a pre-defined animation. Sound category includes elements, which control the recording of audio files related with an object or change the systems volume level. In Looks category, the bricks change the appearance of the visual representation of any object. Hence, different looks can be selected from the object internal list, or the overall visibility, size, transparency. The bricks of Pen category allow an object to draw shapes and color pixels. In doing so, you can also change the size of the pen. Data is the important category, which contains bricks to initialize and show variables as well as to change their values.



When program starts
Set variable متغیر 1 to 6
Set variable متغیر 2 to 4
Set variable تیست to "متغیر 2" + "متغیر 1"
Show variable تیست at X: 100 Y: 200

(A)

(B)

Figure 9.4.: Snapshot of (A) Categories (B) Script view for RTL

9.6. Bidirectional Localization Testing

The software applications should be bug free and accurate all over the world. Hence, every software application before distribution into the global markets must perform testing i.e. internationalization and localization. This could be manual or automatic. The localization testing of mobile app faces many problems merely due to the difficulty of testing and compact capitals. The Android operating system, supporting different GUI elements and a large amount of diverse keyboard applications, can be chosen easily by users. As well as they can use immensely variant display resolutions along with aspect rations of mobile devices with all their combinations, which is a difficult challenge [54]. The Bidirectional (BiDi) describes any software applications, manipulating and displaying text in both directions i.e. LTR and RTL. For bidirectional script processing algorithm, described by Unicode such as Bidi Algorithm, confirms the exact rendering and formatting of Arabic script. The requirement regarding BiDi, affect the coding, design and testing of the internationalized app [82].

Similarly, the tests must be capable to check the functions and the performance of the localized software as well as to detect linguistic and functional problems. The components and features of the software should be same according to the original product. For every software product, the verification of correct translations could be made possible and assurances are made that cultural issues are tested. Furthermore, the localization testing frequently identifies and brings severe issues to the forefront. For instance, clipped strings or strings that overlap the edge of UI elements on the screen, inappropriate layout or text direction, incorrect alphabetical sorting and untranslated strings [54, 56]. For misplaced translations, grammar and spelling issues and layout problems, localization testing usually emphasizes checking of the graphical user interface, (GUI). Usually, these belongings are tested manually, which is time consuming and a resource-intensive task. In this case, the automation support for localization testing helps to save the time as well as allows running the localization test more frequently [85].

Comparatively, the Software applications always need to be developed in different regions of the world. Therefore, the local version of the application helps local customers for better understanding, and to attract more customers as well as to maximize its sales [97]. For quality assurance testing of a software application, localization testing is the type that mainly focuses on the quality of the localization and evaluation of the products functionality and cosmetics. The objective of the automatic localization testing for BiDi-languages are as under:

- To document the attributes of different localization issues to the developers who do not know about the language and cultural background.
- To make sure at a later stage that the localization is stable even though when the bugs and deficiencies are introduced.
- The localization defects should be reported and detected.

The challenges encountered when localizing apps into bidirectional languages include, Character encoding, Right-to-left and vertical text, Mobile Phone Screen Size, Font Style for Mobile

Applications, Text Expansion, Regional standards, Search and replace [56, 95].

9.7. The Proposed Behavior-Development Practices for Catrobat

The objective of the Catrobat programming language is to deliver dependable functionality and stable experience and to ensure that the program script is behaving exactly as expected. The Catrobat project has different functionality bricks. Every program has one or more objects, and these objects contain a list of scripts, which is the code portion of the Catrobat program (See Figure 9.4). The script is a set of many bricks that combine the logic of the program. Scripts essentially behave like subroutines because they are triggered by different external or internal events. In these cases, a script is constructed to execute automatically when the whole program is started. The following examples show how some of the primary features of Catrobat have been specified in a behavior-driven way, using Cucumber scenarios. The below mentioned specifications are plain domain-specific language Gherkin, which does not associate with the so-called Java code. Step-definitions are used to map the Gherkin language to Java code, and to reside in Java code, which are written in a regular expression to match the Cucumber feature scenario steps.

9.7.1. Case Study 1

In the Catrobat programming language, scripts begin to run in response to an event, which is the same behavior as in Scratch. The event can be at the starting of the whole program, an external input event on the hardware or some kind of internal event. The Cucumber feature starts with the keyword Feature followed by a short description. The keyword Background tells Cucumber to execute the following steps before every scenario. The below mentioned scenarios contain two common steps. In the existing version of the Catrobat project, the Set variable brick must display the variable correctly on the mobile screen/stage. Therefore, in a script when you are using more than one variable, it displays always the last initialized variable.

This Cucumber feature relies completely on native features of the Catrobat programming language to specify the expected behavior of a broadcast and set variable brick. The scenario involves two scripts, which start running at the same time and continue to run concurrently. One of the scripts contains a set variable 10 and broadcast message hello. The other script specifies where When scripts (When you received hello) react to the same message, wait for two seconds and then check the value of the variable. The correct behavior of the set variable should be equal to 20. The incorrect behavior is that the variable should not be equal to set variable. However, in the second scenarios, the correct behavior of the script with the change brick, the variable should be equal to 3.

```
Feature: Catrobat bricks

The Correct Behavior: Test the different bricks in Catrobat. The variable should
    be equal to their values in different scenarios.

Background:
Given I have a program
And this program has an object 'Object'

Scenario: To test the "Set variable" and "Broadcast" brick.
Given 'Object' has a start script
And set 'var' to 10
And broadcast 'hello'
Given 'Object' has a When 'hello' script
And wait 2 seconds
And set 'var' to 20
When I start the program
And I wait until the program has stopped
Then the variable 'var' should be equal 20

Scenario: To test the "set variable","change variable" and "broadcast" brick.
Given 'Object' has a start script
And set 'var' to 1
And broadcast 'hello'
Given 'Object' has a When 'hello' script
And wait 2 seconds
And change 'var' by 2
When I start the program
And I wait until the program has stopped
Then the variable 'var' should be equal 3
----------------------------------------------------------------------------------------------
Listing. 1. Cucumber specification for set variable, change variable and
    broadcast brick
```

9.7.2. Case Study 2

For an object, the user can add some images taken from the gallery of his own device or can draw the image in Pocket Paint App. The user must assign a name to the new object (LTR or RTL languages). Then, by tapping on the element, he can assign some script, background or some sounds to the object. This item is treated as the background object of the program script. We can say that the item background is the default object and then the user can customize his application by adding custom elements in the objects activity. In the below-mentioned Listing 2, we tested the object name with one of the RTL languages, i.e., Urdu language. The program has an object name ”آبجیکٹ” The correct behavior should be equal to ”آبجیکٹ”

```
Feature: Object

Scenario: To test the object name with RTL
Language (Urdu)

Given I have a program
And this program has an object "آبجیکٹ"
When I start the program
And I wait until the program has stopped
Then the object should be equal to "آبجیکٹ"
---------------------------------------------------------------------
Listing. 2. Cucumber specification for object (RTL)
```

9.7.3. Case Study 3

In the below mentioned Listing 3, the first scenario with object name ”آبجیکٹ”. This is to test the name of the variable with RTL language (i.e., Arabic/Urdu) ”متغیر” and make sure that the variable is initialized with RTL characters/words correctly. We need a fast method to check the exact behavior of the bricks, which is used in the program script. Hence, we are using Cucumber specification for the same configuration, the one that has a set variable brick. The proposed test case checks that the variable name should be equal to ”متغیر”, otherwise, the name of the variable is not set and the localization issues are revealed. The second scenario introduces two variables name with RTL language (i.e., Urdu and Arabic). For example, we add two variables with RTL language names: ”١متغیر” and ”٢متغیر”. These are the test cases in the form of a scenario and pass a variable value. In this specification, the correct behavior of the program in Catrobat bricks should be equal to 10.

Therefore, the Catrobat project is localized correctly and their bricks are working properly, otherwise, localization issues will be detected. In the third scenario, we need to test the variable and broadcast bricks with RTL language. In orders to complete this type of testing, a small program is created, and this program contains two bricks, one brick is to set a variable and the other to broadcast a message (RTL). The broadcast is signals or undirected messages, which are sent into the script at the app's runtime. A broadcast brick should send a message with (RTL or LTR) language and the scripts should react to it. We also set the variable to ”متغیر”, and it must show the last variable initialized on the stage.

```
Feature: RTL language

Background:

Given I have a program
And this program has an object "آبجیکٹ"

Scenario: To test the Variable name with RTL Language.
Given this "آبجیکٹ" has a start script
And set "متغیر" to 4
When I start the program
And I wait until the program has stopped
Then the name of the variable should be equal "متغیر"

Scenario: Test and add two variable name with RTL Language.
Given this "آبجیکٹ" has a start script
And set "١متغیر" to 6
And set "٢متغیر" to 4
And set "ٹسٹ" have set "١متغیر" + set "٢متغیر"
When I start the program
And I wait until the program has stopped
Then the "ٹسٹ" should be equal to 10
Scenario: To test "Set variable" and "Broadcast" brick message with RTL

Given this "آبجیکٹ" has a start script
And set "متغیر" to 10
And broadcast "نشر"
Given this "آبجیکٹ" has a When "نشر" script
And wait 2 seconds
And set "متغیر" to 20
When I start the program
And I wait until the program has stopped
Then the "متغیر" should be equal to 20
--------------------------------------------------------------------------------------------
```

Listing. 3. Cucumber specification for variable & Broadcast brick (RTL)

9.8. Conclusion

In this paper, we presented the BDD practice and Cucumber-base testing for the Catrobat project development. The Cucumber scenarios are used as acceptance testing in the project. With the help of BDD practice, we concise few challenging aspects regarding LTR and RTL languages, which are faced by the Catrobat development team. The acceptance tests results show that the test automation allows BDD base testing for localization issues, especially for RTL languages. The purpose is to develop a unified system that enables mobile testers to dynamically test the apps without dependence on any scripting language. The BDD approach enables testers to define the scenarios to be tested in a natural language that supports seamless and efficient testing of mobile apps. In this approach, we attempt to design a system capable of testing the properties of the app automatically once the scenarios are written for a set of features. This helps in defining key scenarios for each story and eliminates ambiguities from the requirements. The primary purpose of such methodology is to encourage communication amongst the stakeholders of the Catrobat project. The results show that the proposed approach examines the issues of RTL languages from different angles and track regression errors as well as diagnose localization issues of such languages. For future work, we are endeavoring to develop and improve the correctness of localization for Korean, Japanese, Hindi and Chinese languages.

Chapter 10

Conclusion and Future Work

10.1. Conclusion

In this thesis, we presented an advanced agile software methodology (BDD) and Cucumber framework for a visual programming environment to automate regression testing of Android apps. It also includes how the executable specifications were developed and used throughout the Catrobat project. With the help of these executable specifications, we specified the issues/bugs and described incorrect behavior regarding “Broadcast”, “Broadcast and wait”, “When you receive”, “Screen tap”, and “set variable” bricks in the Catrobat project for Android mobile application (Catroid) in the form of executable regression tests.

BDD is an advanced agile software methodology. This approach aims to test, identify and diagnose the bugs in Android mobile application (Pocket Code) which is tested with the help of Cucumber specifications. Therefore, our work focuses on mobile applications and their regression testing with the help of BDD specifications.

We also summarized some challenging aspects of the RTL languages facing the Catrobat programming languages with the help of BDD methodology using Cucumber specifications. These challenges in RTL languages differ tremendously in term of their character and morphology from other languages. However, our work contributes to observed result on the various bugs we found. The result shows that tests automation allows BDD base testing for the localized version of the Catrobat project. BDD base testing, supports and finds localization RTL languages bugs earlier as well as this accelerates the process of releasing cycles and minimizes the human struggle and efforts. This approach supports all BiDi-languages issues, which brings forth common localization issues in the software design. This has to be made sure that the localized app fulfills local end-user needs by Cucumber specifications in terms of the requirement of the traditional senses. These traditionally required elements are language, feature and user experience. While automation approach is helpful in saving time and efforts alongside bringing accuracy and maintaining the usability and portability of the app.

Moreover, it decreases effort and the time spent for regression testing. The main advantages to adopt behavior-driven development are better collaboration, automation and remove misunderstanding. Therefore, the autonomous test scripting architecture for mobile apps, which step forward towards executable specification, become a living documentation. The proposed approach work to address the intercommunication among all the team members, aligning to the common understanding of the wanted needs. The executable specifications with their reusable steps, the Gherkin language with its concise structure, are efficient for testing and reduce the level of ambiguity and quality assurance improves. Therefore, our proposed automated acceptance testing strategy/approach improve BDD steps, reusability of their implementations in the project, and provides common platform among all stockholders i.e. developers, testers, and business analyst.

10.2. Future Work

The current work focused on mobile applications and their testing. The BDD approach also has other advantages, but to realize its full potential for cross-platform application, there is still some work to be done. The cross-platform aspect of the Catrobat specifications could not yet be fully implemented. The reason is that Cucumber features must first be natively implemented on other mobile platforms, e.g. iOS and maybe in a future another interpreter or compiler for the Catrobat language. To implement the consistent common feature files across the different hardware as well as with different operating systems is the biggest challenge. Therefore, Gherkin scenarios should be implemented on other mobile platforms. Our challenge is to implement consistent common feature files across different devices as well as with different mobile operating systems e.g. iOS.

There are some interconnected challenging aspects in Catrobat project to improve the accuracy of the implemented localization not only for RTL languages but also for Korean, Japanese, Hindi and Chinese languages. While translating the Catrobat project, there are, however, some challenges faced by Japanese and Korean languages. Both the languages have different writing system, the number of lexical similarities and the same sentence structure.

Bibliography

[1] Wolfgang Slany. Catroid: A mobile visual programming system for children. In *Proceedings of the 11th International Conference on Interaction Design and Children*, IDC '12, pages 300–303, New York, NY, USA, 2012. ACM. (Cited on pages 1 and 27.)

[2] Henry Muccini, Antonio Di Francesco, and Patrizio Esposito. Software testing of mobile applications: Challenges and future research directions. In *Proceedings of the 7th International Workshop on Automation of Software Test*, pages 29–35. IEEE Press, 2012. (Cited on pages 1, 2, 23, 82, and 83.)

[3] Mohammed Akour, Ahmad A Al-Zyoud, Bouchaib Falah, Salwa Bouriat, and Khalid Alemerien. Mobile software testing: Thoughts, strategies, challenges, and experimental study. *International Journal of Advanced Computer Science and Applications*, 7(6):12–19, 2016. (Cited on page 1.)

[4] Kulkarni P. Euteneuer S Calam, J. Multi-platform mobile test automation for the financial sector. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, pages 63–65. ACM, 2014. (Cited on pages 1 and 2.)

[5] L. Nagowah and G. Sowamber. A novel approach of automation testing on mobile devices. In *2012 International Conference on Computer Information Science (ICCIS)*, volume 2, pages 924–930, June 2012. (Cited on pages 2 and 59.)

[6] Padmaraj Nidagundi and Leonids Novickis. New method for mobile application testing using lean canvas to improving the test strategy. In *Computer Sciences and Information Technologies (CSIT), 2017 12th International Scientific and Technical Conference on*, volume 1, pages 171–174. IEEE, 2017. (Cited on pages 2 and 82.)

[7] D. Amalfitano, A. R. Fasolino, and P. Tramontana. A gui crawling-based technique for android mobile application testing. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, pages 252–261, March 2011. (Cited on pages 2 and 82.)

[8] Samer Zein, Norsaremah Salleh, and John Grundy. A systematic mapping study of mobile application testing techniques. *Journal of Systems and Software*, 117:334–356, 2016. (Cited on page 2.)

[9] H. Flora and S. Chande. A review and anaysis on mobile application development processes using agile methodlogies. In *International Journal of Research in Computer Science*, volume 3, pages 9–18, July 2013. (Cited on pages 2, 37, and 83.)

[10] M. Rahman and J. Gao. A reusable automated acceptance testing architecture for microservices in behavior-driven development. In *2015 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, volume 00, pages 321–325, March 2015. (Cited on pages 2, 3, 40, and 45.)

[11] Carlos Solis and Xiaofeng Wang. A study of the characteristics of behaviour driven development. In *Proceedings of the 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, SEAA '11, pages 383–387, Washington, DC, USA, 2011. IEEE Computer Society. (Cited on pages 2, 3, 5, 41, 42, 43, 49, 51, 83, 84, 85, and 87.)

[12] S. Rose, M. , and A. Hellesoy. *The Cucumber for Java Book: Behaviour-Driven Development for Testers and Developers*. Pragmatic Bookshelf, 2015. (Cited on pages 3, 45, 46, 47, 51, 86, and 88.)

[13] Chuanqi Tao, Jerry Gao, and Tiexin Wang. An approach to mobile application testing based on natural language scripting. In *SEKE*, pages 260–265, 2017. (Cited on pages 3, 86, and 87.)

[14] A. Texas A. Elisa E. Using behavioral driven development (bdd) in capstone design projects. In *American Society for Engineering Education, 2014ASEE Annual Conference & Exposition*, June 2014. (Cited on pages 3, 5, and 43.)

[15] Pavel Smutny. Visual programming for smartphones. In *In 12th International Carpathian Control Conference (ICCC)*, pages 358–361, 2011. (Cited on page 4.)

[16] H. Tsukamoto, Y. Takemura, Y. Oomori, I. Ikeda, H. Nagumo, A. Monden, and K. Matsumoto. Textual vs. visual programming languages in programming education for primary schoolchildren. In *2016 IEEE Frontiers in Education Conference (FIE)*, pages 1–7, Oct 2016. (Cited on page 4.)

[17] Wolfgang Slany. Pocket code: A scratch-like integrated development environment for your phone. In *Proceedings of the Companion Publication of the 2014 ACM SIGPLAN Conference on Systems, Programming, and Applications: Software for Humanity*, SPLASH '14, pages 35–36, New York, NY, USA, 2014. ACM. (Cited on pages 4, 27, 83, and 89.)

[18] W. Slany. A mobile visual programming system for android smartphones and tablets. In *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 265–266, Sept 2012. (Cited on pages 4, 27, 83, and 89.)

[19] W. Slany. Catrobat, Education, TU Graz. `https://edu.catrob.at/brick-documentation`. last visited Jan. 5, 2019. (Cited on pages 4 and 29.)

[20] Manuel Wallner. Specification by example of the broadcast mechanism of catrobat. (Cited on page 5.)

[21] B. Kirubakaran and V. Karthikeyani. Mobile application testing challenges and solution approach through automation. In *2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering*, pages 79–84, Feb 2013. (Cited on page 9.)

[22] Aiman Mamdouh Ahmad Ayyal Awwad. Automated bidirectional languages localization testing for android apps development. *Ph.D Thesis*, 2017. (Cited on page 9.)

[23] Daniel Knott. *Hands-On Mobile App Testing: A Guide for Mobile Testers and Anyone Involved in the Mobile App Business*. The name of the publisher, 1 edition, 2015. (Cited on pages 10, 11, and 12.)

[24] Sarwar Muhammad and RahimSoomro Tariq. Impact of smartphones on society. volume 22, pages 2016–226, 2013. (Cited on pages 11 and 12.)

[25] Network World. `https://www.networkworld.com/article/2869645/network-security/a-brief-history-of-smartphones.html`. last visited Jan. 5, 2019. (Cited on pages 11 and 12.)

[26] OO Okediran, OT Arulogun, RA Ganiyu, and CA Oyeleye. Mobile operating systems and application development platforms: A survey. *International Journal of Advanced Networking and Applications*, 6(1):2195, 2014. (Cited on page 12.)

[27] IDC. Worldwide Smartphone Market Will See the First Single-Digit Growth Year on Record. `https://www.statista.com/statistics/272307/market-share-forecast-for-smartphone-operating-systems/`. last visited Jan. 5, 2019. (Cited on page 13.)

[28] Jerome DiMarzio. *ANDROID A PROGRAMMERS GUIDE*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 2009. (Cited on page 14.)

[29] Source. The Android Source Code, Android. `http://source.android.com/source/index.html`. last visited Jan. 5, 2019. (Cited on page 14.)

[30] Endre Grtnes. Standardization as open innovation: two cases from the mobile industry. *Information Technology & People*, 22(4):367–381, 2009. (Cited on page 14.)

[31] Android studio, Android. `http://developer.android.com/sdk`. last visited Jan. 5, 2019. (Cited on page 14.)

[32] Developer. The Android Source Code. `http://developer.android.com/guide/topics/manifest/uses-sdk-element.html`. last visited Jan. 5, 2019. (Cited on page 15.)

[33] Appbrain. Android and Statistics. `https://www.appbrain.com/stats/number-of-android-apps`. last visited Jan. 16, 2019. (Cited on page 15.)

[34] Developer. Starting an Activity, Android. `https://developer.android.com/guide/components/activities/index.html`. last visited Jan. 15, 2019. (Cited on page 15.)

[35] Developer. Starting an Activity, Android. `https://developer.android.com/guide/components/activities/activity-lifecycle`. last visited Jan. 15, 2019. (Cited on page 16.)

[36] Xcode, Apple. `https://developer.apple.com/support/xcode/`. last visited Jan. 5, 2019. (Cited on page 17.)

[37] Xcode, Apple. `https://developer.apple.com/app-store/review/guidelines/`. last visited Jan. 5, 2019. (Cited on page 18.)

[38] The App Life Cycle, Apple. https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/andiPhoneOSProgrammingGuide/TheAppLifeCycle/TheAppLifeCycle.html. last visited Jan. 5, 2019. (Cited on page 18.)

[39] Microsoft.com. Publish Windows apps. https://docs.microsoft.com/en-us/windows/uwp/publish/. last visited Jan. 5, 2019. (Cited on page 18.)

[40] Mohd. Ehmer Khan and FarmeenaKhan. A comparative study of white box, black box and grey box testing techniques. In *(IJACSA) International Journal of Advanced Computer Science and Applications*, volume 03, 2012. (Cited on page 19.)

[41] Gaurav Saini and Kestina Rai. An analysis on objectives, importance and types of software testing. *International Journal of Computer Science and Mobile Computing (IJCSMC)*, 2(9):18–23, 2013. (Cited on pages 19, 20, 21, 22, and 25.)

[42] Mohd. Ehmer Khan. Different forms of software testing techniques for finding errors. In *IJCSI International Journal of Computer Science Issues*, November. (Cited on pages 19 and 20.)

[43] Ravi Ramchandra Nimbalkar. Mobile application testing and challenges. In *International Journal of Science and Research (IJSR),*, July 2013. (Cited on page 20.)

[44] The Android Test. . http://developer.android.com/reference/android/test/AndroidTest-Case.html. last visited Jan. 5, 2019. (Cited on page 20.)

[45] M. E. Joorabchi, A. Mesbah, and P. Kruchten. Real challenges in mobile app development. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 15–24, Oct 2013. (Cited on pages 21, 22, and 25.)

[46] searchsoftwarequality, techtarget,. https://searchsoftwarequality.techtarget.com/definition. last visited Jan. 5, 2019. (Cited on pages 21, 22, and 25.)

[47] Jerry Gao, Xiaoying Bai, Wei-Tek Tsai, and Tadahiro Uehara. Mobile application testing: a tutorial. *Computer*, (2):46–55, 2014. (Cited on page 22.)

[48] Ilene Burnstein. *Practical software testing: a process-oriented approach.* Springer Science & Business Media, 2006. (Cited on pages 22, 23, and 24.)

[49] E. Engstrm, P. Runeson, and A. Ljung. Improving regression testing transparency and efficiency with history-based prioritization – an industrial case study. In *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, pages 367–376, March 2011. (Cited on pages 22 and 58.)

[50] Android Starting an Activity. . https://www.softwaretestinghelp.com/system-testing/. last visited Jan. 5, 2019. (Cited on page 22.)

[51] Cuixiong Hu and Iulian Neamtiu. Automating gui testing for android applications. In *Proceedings of the 6th International Workshop on Automation of Software Test*, AST '11, pages 77–83, New York, NY, USA, 2011. ACM. (Cited on page 22.)

[52] R Selvam and V Karthikeyani. Mobile software testing–automated test case design strategies. (Cited on pages 23 and 24.)

[53] Aiman M. Ayyal Awwad, Christian Schindler, Kirshan Kumar Luhana, Zulfiqar Ali, and Bernadette Spieler. Improving pocket paint usability via material design compliance and internationalization & localization support on application level. In *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI '17, pages 99:1–99:8, New York, NY, USA, 2017. ACM. (Cited on pages 23 and 66.)

[54] Aiman M. Ayyal Awwad and Wolfgang Slany. Automated bidirectional languages localization testing for android apps with rich gui. In *Journal Article, Mobile Information Systems*, volume 2016, page 13, Feb 2016. (Cited on pages 24, 64, 65, 66, and 90.)

[55] S. Abufardeh and K. Magel. Software internationalization: Testing methods for bidirectional software. In *2009 Fifth International Joint Conference on INC, IMS and IDC*, pages 226–231, Aug 2009. (Cited on pages 24 and 63.)

[56] A Ayyal Awwad. Localization to bidirectional language for a visual programming environment on smartphones. In . *IJCSI International Journal of Computer Science Issues*, volume 14 of *IDC '12*, 2017. (Cited on pages 27, 29, 63, 64, 65, 90, and 91.)

[57] Annemarie Harzl, Philipp Neidhoefer, Valentin Rock, Maximilian Schafzahl, and Wolfgang Slany. A scratch-like visual programming system for microsoft windows phone 8. *CoRR*, abs/1310.1390, 2013. (Cited on page 28.)

[58] Massimo Ficco, Roberto Pietrantuono, and Stefano Russo. Bug localization in test-driven development. *Adv. Soft. Eng.*, 2011:2:1–2:18, January 2011. (Cited on pages 38, 39, and 84.)

[59] Susan Hammond and David Umphress. Test driven development: The state of the practice. In *Proceedings of the 50th Annual Southeast Regional Conference*, ACM-SE '12, pages 158–163, New York, NY, USA, 2012. ACM. (Cited on page 38.)

[60] Craig Larman and Bas Vodde. *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*. Addison-Wesley Professional, 1st edition, 2010. (Cited on page 39.)

[61] Lasse Koskela. *Test Driven: Practical Tdd and Acceptance Tdd for Java Developers*. Manning Publications Co., Greenwich, CT, USA, 2007. (Cited on page 39.)

[62] Steve McConnell. *Code Complete*. Microsoft Press, 2nd edition, 2004. (Cited on page 39.)

[63] S. Freeman and N. Pryce. *Growing Object-Oriented Software, Guided by Tests*. Addison-Wesley Signature Series (Beck). Pearson Education, 2009. (Cited on page 40.)

[64] Gojko Adzic. *Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing*. Neuri Limited, United Kingdom, 2009. (Cited on page 40.)

[65] K. Pugh. *Lean-Agile Acceptance Test-Driven Development: Better Software Through Collaboration*. Net Objectives Lean-Agile Series. Pearson Education, 2010. (Cited on page 40.)

[66] Gojko Adzic. *Specification by Example: How Successful Teams Deliver the Right Software*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2011. (Cited on pages 40 and 41.)

[67] S ohn Ferguson. *BDD in Action. Behavior driven development for the whole software lifecycle*. Manning Publications, 2014. (Cited on pages 40, 43, 44, 48, and 85.)

[68] J. Gregory and L. Crispin. *More Agile Testing: Learning Journeys for the Whole Team*. Addison-Wesley Signature Series (Cohn). Pearson Education, 2014. (Cited on page 41.)

[69] L. P. Binamungu, S. M. Embury, and N. Konstantinou. Detecting duplicate examples in behaviour driven development specifications. In *2018 IEEE Workshop on Validation, Analysis and Evolution of Software Tests (VST)*, pages 6–10, March 2018. (Cited on page 41.)

[70] Z. Dennis A. Hellesoy B Helmkamp D. Chelimsky, D. Astels and D. North. *The RSpec book: Behavior Driven Development with RSpec*. First Edition Pragmatic Bookshelf, 2012. (Cited on pages 42 and 47.)

[71] Abigail Egbreghts. A literature review of behavior driven development using grounded theory. 2017. (Cited on page 42.)

[72] N. Li, A. Escalona, and T. Kamal. Skyfire: Model-based testing with cucumber. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 393–400, April 2016. (Cited on pages 46, 86, and 87.)

[73] Marc Hesenius, Tobias Griebe, and Volker Gruhn. Towards a behavior-oriented specification and testing language for multimodal applications. In *Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '14, pages 117–122, New York, NY, USA, 2014. ACM. (Cited on page 51.)

[74] A. Contan, L. Miclea, and C. Dehelean. Automated testing framework development based on social interaction and communication principles. In *2017 14th International Conference on Engineering of Modern Electric Systems (EMES)*, pages 136–139, June 2017. (Cited on pages 51, 82, and 87.)

[75] Zulfiqar Ali, Aiman M Ayyal Awwad, and Wolfgang Slany. Using executable specification and regression testing for broadcast mechanism of visual programming language on smartphones. *International Journal of Interactive Mobile Technologies*, 13(2), 2019. (Cited on page 57.)

[76] Milos Gligoric, Lamyaa Eloussi, and Darko Marinov. Practical regression test selection with dynamic file dependencies. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, pages 211–222. ACM, 2015. (Cited on page 57.)

[77] Rafaqut Kazmi, Dayang NA Jawawi, Radziah Mohamad, and Imran Ghani. Effective regression test case selection: a systematic literature review. *ACM Computing Surveys (CSUR)*, 50(2):29, 2017. (Cited on page 57.)

[78] Thiago Rocha Silva, Jean-Luc Hak, and Marco Winckler. Testing prototypes and final user interfaces through an ontological perspective for behavior-driven development. In

Cristian Bogdan, Jan Gulliksen, Stefan Sauer, Peter Forbrig, Marco Winckler, Chris Johnson, Philippe Palanque, Regina Bernhaupt, and Filip Kis, editors, *Human-Centered and Error-Resilient Systems Development*, pages 86–107, Cham, 2016. Springer International Publishing. (Cited on page 58.)

[79] Kamna Solanki, Sandeep Dalal, and Sudhir . Challenges of regression testing: A pragmatic perspective. *International Journal of Advanced Research in Computer Science*, 9(1):499–503, 2018. (Cited on page 58.)

[80] T. Grnli and G. Ghinea. Meeting quality standards for mobile application development in businesses: A framework for cross-platform testing. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, pages 5711–5720, Jan 2016. (Cited on page 58.)

[81] C. Bernaschina, R. Fedorov, D. Frajberg, and P. Fraternali. A framework for regression testing of outdoor mobile applications. In *2017 IEEE/ACM 4th International Conference on Engineering and Systems (MOBILESoft)*, pages 179–181, May 2017. (Cited on page 59.)

[82] K. Magel and S. Abufardeh. Qa/testing bi-directional languages software: Issues and challenges. In *2008 32nd Annual IEEE International Computer Software and Applications Conference(COMPSAC)*, volume 00, pages 172–175, 07 2008. (Cited on pages 64 and 90.)

[83] Sameer Abufardeh and Kenneth Magel. Software localization: The challenging aspects of arabic to the localization process (arabization). In *Proceedings of the IASTED International Conference on Software Engineering*, SE '08, pages 275–279, Anaheim, CA, USA, 2008. ACTA Press. (Cited on pages 64, 65, and 82.)

[84] Naqvi S.N.S. Khan A. Abbasi, A.T. and B Ahmad. Urdu text steganography: Utilizing isolated letters. In *13th Australian Information Security Management Conference*, volume 00, pages 37–46, November 2015. (Cited on page 65.)

[85] Rudolf Ramler and Robert Hoschek. Process and tool support for internationalization and localization testing in software product development. In Michael Felderer, Daniel Méndez Fernández, Burak Turhan, Marcos Kalinowski, Federica Sarro, and Dietmar Winkler, editors, *Product-Focused Software Process Improvement*, pages 385–393, Cham, 2017. Springer International Publishing. (Cited on pages 65 and 90.)

[86] Scratch. . `https://en.scratch-wiki.info/wiki/Stringandhttps://en.scratch-wiki.info/wiki/List`. last visited Jan. 5, 2019. (Cited on page 67.)

[87] Wajeh Addin Mohsen, Chuanqi Tao, Qian mu Li, and Jerry Gao. A viewpoint on location-based mobile apps testing. In *Big Data Computing Service and Applications (BigDataService), 2017 IEEE Third International Conference on*, pages 305–312. IEEE, 2017. (Cited on pages 75 and 76.)

[88] Sandeep Kumar, Mohammed Abdul Qadeer, and Archana Gupta. Location based services using android (lbsoid). In *Internet Multimedia Services Architecture and Applications (IMSAA), 2009 IEEE International Conference on*, pages 1–5. IEEE, 2009. (Cited on pages 75 and 76.)

[89] Ricardo O Mitchell, Hammad Rashid, Fakir Dawood, and Ali AlKhalidi. Hajj crowd management and navigation system: People tracking and location based services via integrated mobile and rfid systems. In *Computer Applications Technology (ICCAT), 2013 International Conference on*, pages 1–7. IEEE, 2013. (Cited on page 76.)

[90] Zulfiqar Ali. , behavior-driven development as an error-reduction practice for mobile application testing,. volume 16. IJCSI, March 2019. (Cited on page 81.)

[91] Sogeti. Sogeti, World Quality Report. `https://www.sogeti.com/globalassets/global/wqr201819/wqr-2018-19_secured.pdf`, 2018-19. dd. (Cited on page 82.)

[92] Juliana Medeiros, Alexandre Vasconcelos, Miguel Goulão, Carla Silva, and João Araújo. An approach based on design practices to specify requirements in agile projects. In *Proceedings of the Symposium on Applied Computing*, pages 1114–1121. ACM, 2017. (Cited on pages 83 and 84.)

[93] Nachiappan Nagappan, E Michael Maximilien, Thirumalesh Bhat, and Laurie Williams. Realizing quality improvement through test driven development: results and experiences of four industrial teams. *Empirical Software Engineering*, 13(3):289–302, 2008. (Cited on page 84.)

[94] Alexandre Melo Braga12, Daniela Castilho Schwab, and André Luiz Vannucci. The use of acceptance test-driven development in the construction of cryptographic software. (Cited on page 84.)

[95] Davi Bernardo Silva, Andre Takeshi Endo, Marcelo Medeiros Eler, and Vinicius HS Durelli. An analysis of automated tests for mobile android applications. In *2016 XLII Latin American Computing Conference (CLEI)*, pages 1–9. IEEE, 2016. (Cited on pages 87 and 91.)

[96] Madhuri Kishan Kulkarni and A Soumya. Deployment of calabash automation framework to analyze the performance of an android application. *Journal for Research— Volume*, 2(03), 2016. (Cited on page 87.)

[97] Xin Xia, David Lo, Feng Zhu, Xinyu Wang, and Bo Zhou. Software internationalization and localization: An industrial experience. In *2013 18th International Conference on Engineering of Complex Computer Systems*, pages 222–231. IEEE, 2013. (Cited on page 90.)