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

Zulfiqar Ali[1]

[1]**Institute of Software Technology, Graz University of Technology, Graz, 8010, Austria**

Abstract

With the rapid development of mobile technology, there is a significant increase of mobile's 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.

Keywords: *Behavior-driven development, testing, Cucumber, Localization, Visual programing language, Catrobat.*

1. 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 [1].

Mobile applications are mostly prone to errors because of the developer's 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 user's 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, 3].

In this advance era, the speed of delivering mobile applications to IT companies is a key challenge. However, in the past, a project ran over several years and the phases of a project would have been measured in months. Contrarily, currently, the projects have to be delivered over a minimum number of months and the project development phases are set for weeks or even days. Therefore, in this rapidity of changes, documenting the functionality is becoming a challenge and under these circumstances, testing the mobile applications takes place on two points: first the right development of the product, and second the development of the right product [4].

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 [5]. 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 [3].

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 up-front [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 [7].

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, 8]. 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 [8].

Test-Driven Development (TDD) is a short development-cycle approach that depends on the agile practices for writing automated tests before writing functional code [9]. Like TDD, Acceptance-Test Driven Development (ATDD) also involves creating tests before coding, and these tests represent the expected behavior of the software. Behavior-Driven Development (BDD) is a combination and enhancement of practices stemming from TDD and ATDD. BDD focuses on the behavioral aspect while the TDD focuses on the implementation aspect. Additionally, BDD is usually done in a very English-like language to help the domain experts to understand the implementation rather than exposing the code level tests. BDD encourages bridging the gap between the problem and the solution domain, providing a better understanding between both the development team and business stakeholders [10].

Meantime, a free open-source, the Catrobat [1] 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[2], 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 [11, 12].

[1]https://www.catrobat.org/
[2]http://scratch.mit.edu/

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.

2. 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.

2.1 Test-Driven Development

TDD (see Fig 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 [13, 14].

2.2 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 [15]. 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 [10].

2.3 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 [16]. It provides a common ubiquitous (pure natural language) language to facilitate the communication between the development team and the business stakeholders for better understanding [17]. 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 [16].

Fig 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.

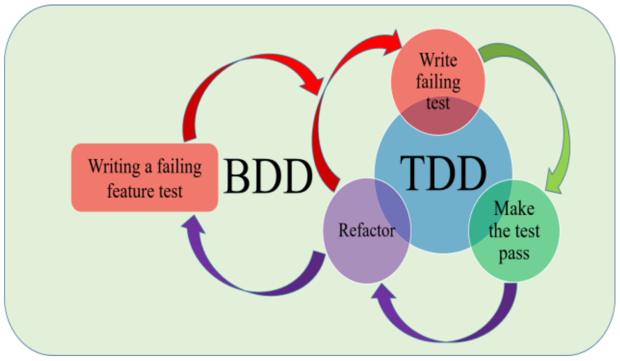


Fig. 1 TDD and BDD Process.

3. 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 [18]. The experts use Cucumber to generate and run user acceptance tests in three steps. First, the product stakeholders i.e., business analysts, developers, and testers work together to write Cucumber feature files. A feature file usually contains a list of scenarios and every scenario consists of steps as shown below (See Fig 2):

- **Given** steps: declare the system as in a known state.
- **When** steps: show the users actions.
- **Then** steps: verify the system outputs after the user actions.
- **And** steps: connect multiple Given, When, and Then steps to make the reading of written steps is more fluent.

In the second step, the testers write test codes for every step in the test scenarios using the Cucumber mapping mechanism, called step-definitions. Finally, in the third step, testers run the scenarios and get test reports using Cucumber tool in a continuous integration environment. [19]. Usually, all the stakeholders manually write BDD test scenarios that describe system behaviors of a system under test. Testers write an implementation for the BDD scenarios by hand and execute the Cucumber tests. Cucumber provides transparency about what test scenarios are covered and how the test scenarios are mapped. Then testers write Cucumber mappings for the generated scenarios [19, 20].

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 [18]. 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 1).

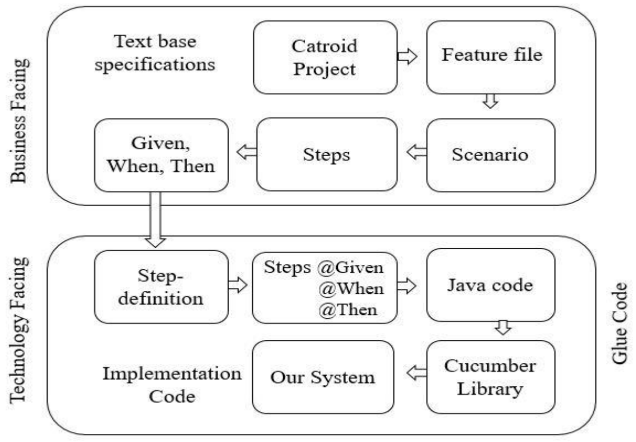


Fig. 2 Cucumber concept & work flow in Catroid.

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 executed in an emulator or a real mobile device. The Espresso framework is implemented on Android Instrumentation framework. It is a small API and is fairly a simple automation tool. It synchronizes with UI thread and hence this makes it more reliable and fast. Furthermore, the framework does not require any type of sleeping methods [21].

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 [22]. There are plenty of easy to use methods that extend the Junit that can be used for Android testing. The black-box test cases that are executed are effective and robust. The community has good support and there are intermediate releases for this automation tool. The functions, system, and acceptance test scenarios can be written with the availability of the source code [21].

Table 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 [18]. 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 [20, 4].

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 [20]. 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 Fig 3) [16].

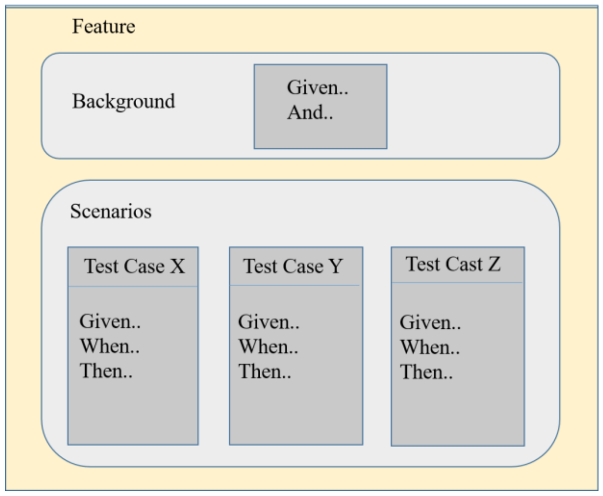


Fig. 3 BDD base Given-When-Then Pattern.

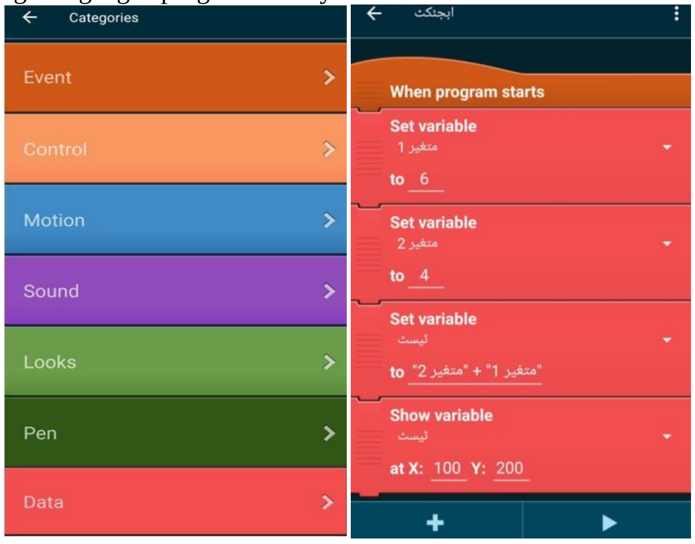
Step-definition is the part where the natural language is converted into the actual working code based on the mapping of the constructs of the natural language. A specific regular expression is used to determine the code, which is to be executed on reading the sentence [20]. With the help of Java code, 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 step definitions are created, testers use annotations to specify the feature files to execute in a test. Then *Cucumber* will look for the step-definitions, execute the test scenarios, and generate test reports [19]. **Glue** (See Fig 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.

4. 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 [11, 12]. 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 Fig 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 system’s 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 ٹیسٹ to "متغیر 2" + "متغیر 1"
Show variable ٹیسٹ at X: 100 Y: 200

(A) (B)

Fig. 4. Snapshot of (A) Categories (B) Script view for RTL

5. 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, some kind of internationalization and localization testing - whether manually or automatically - must be performed. The localization testing of mobile app faces many problems merely because of the difficulty of testing and limited resources. The Android operating system, supporting different GUI elements and a huge number of different keyboard applications, can be chosen freely by users. As well as they can use a hugely varying display resolutions, and aspect ratios of devices with all combinations, which is a difficult challenge [23]. 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 [25].

Similarly, the tests must have the ability to confirm the functionality and performance of localized software and the components according to the original product as well as to detect linguistic and functional problems. It is essential that the correctness of translations is verified and the consideration of cultural issues is guaranteed. However, the localization process often introduces severe issues, such as, 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 [23, 27]. 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 [24].

Software applications always need to be developed in different regions of the world. However, the local version of application helps local customers for better understanding, and to attract more customers, and maximize its sales [26]. 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 [27, 21].

6. 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 Fig 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.

6.1 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

6.2. 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)

6.3. 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: "متغیر1" and "متغیر2". 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 "متغیر1" to 6
And set "متغیر2" to 4
And set "ٹیسٹ" have set "متغیر1" + set "متغیر2"
When I start the program
And I wait until the program has stopped
Then the "ٹیسٹ" should be equal to 10

Scenario: To test "Set variable" and "Broadcast" brick message with RTL language.
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)

7. Conclusions

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.

Acknowledgments

The author is grateful for the support of Catrobat team.

References

[1] Sogeti, World Quality Report 2018-19 https://www.sogeti.com/globalassets/global/wqr201819/wqr-2018-19_secured.pdf.

[2] H. Muccini, A. D. Francesco, and P. Esposito, “Software testing of mobile applications: Challenges and future research directions”, In proceedings of the 7th International Workshop on Automation of Software Test, 2012, pp.29–35.

[3] 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, volume1, pages 171–174. IEEE, 2017.

[4] A. Contan, C. Dehelean, and L. Miclea “Automated Testing Framework Development based on Social Interaction and Communication Principles” 14th International Conference on Engineering of Modern Electric System(EMES), 2017, pp.136-139.

[5] 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 SoftwareTesting, Verification and Validation Workshops, ., March 2011, pp.252–261.

[6] Net-Translators, Localization for Mobile Apps. Available at: https://www.net-translators.com/blog/localization-for mobile-apps ; [Last accessed February 2019].

[7] S. Abufardeh, and M. Kenneth, “Software localization: the challenging aspects of Arabic to the localization process (Arabization)”. In Proceedings of the IASTED International Conference on Software Engineering 2008, pp275–279.

[8] Explore, Collab, versionone Agile report https://explore.versionone.com/stateofagile/version one-12th-annual-state-of-agile-report

[8] H. Flora, and S. Chande, “A REVIEW AND ANALYSIS ON MOBILE APPLICATION DEVELOPMENT PROCESSES USING AGILE METHODLOGIES”, In

International Journal of Research in Computer Science, volume 3, July 2013, pp.9–18.

[9] C. Solis, and X. Wang, "A Study of the Characteristics of Behaviour Driven Development", In Proceedings of the 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications, 2011, pp.383–387.

[10] J. Medeiros, A Vasconcelos, M Goulao, and C Silva, "Approach based on design practices to specify requirements in agile projects" In Proceedings of the Symposium on Applied Computing (SAC '17), Morocco, 2017.

[11] W. 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, 2014, pp.35–36.

[12] 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), 2012, pp265–266.

[13] M. Ficco, R. Pietrantuono, and S. Russo, "Bug Localization in Test-Driven Development" Hindawi Publishing Corporation Advances in Software Engineering , 2011.

[14] N. Nagappan, E.M. Maximilien, and T Bhat, "Realizing quality improvement through test-driven development: results and experiences of four industrial teams" Empirical Software Engineering 13.3, 2008, pp.289-302.

[15] A. M. Braga, D.C. Schwab, and A.L. Vannucci, "The Use of Acceptance Test-Driven Development in the Construction of Cryptographic Software", The Ninth International Conference on Emerging Security Information, Systems and Technologies, 2015.

[16] C. Solis, and X. Wang, "A Study of the Characteristics of Behaviour Driven Development," 37th EUROMICRO Conference on Software Engineering and Advanced Applications, 2011, pp.383-387.

[17] J.F Smart, BDD in Action, Behavior-driven development for the whole software life cycle, 2014

[18] C. Tao, and T, Wang, "An Approach to Mobile Application Testing Based on Natural Language Scripting", SEKE, 2017, pp.260-265.

[19] The Cucumber for Java Book by Seb Rose, Matt Waynne, Aslak Hellesoy.

[20] N. Li, A. Escalona, T. Kamal, "Skyfire: Model-Based Testing with Cucumber", IEEE International Conference on Software Testing, Verification and Validation (ICST), 2016

[21] D. B. Silva, A. T. Endo, and M. M. Eler, "An analysis of automated tests for mobile Android applications ", XLII Latin American Computing Conference (CLEI), 2016, pp.1-9.

[22] M. K. Kulkarni, and P.Soumya A, "Deployment of Calabash Automation Framework to Analyze the Performance of an Android application" Journal for Research, 2016.

[23] A. M. A. Awwad, and W. Slany, "Automated Bidirectional Languages Localization Testing for Android Apps with Rich GUI" Mobile Information Systems, 2016.

[24] R. Ramler, R. Hoschek, "Process and Tool Support for Internationalization and Localization Testing in Software Product Development", Springer International Publishing 2017, pp.385-393.

[25] S. Abufardeh and K. Magel, "QA/Testing Bi-directional Languages Software: Issues and Challenges" 32nd Annual IEEE International Computer Software and Applications Conference, 2008, pp. 172-175.

[26] X. Xia, D. Lo, F. Zhu, X. Wang and B. Zhou, "Software Internationalization and Localization: An Industrial Experience", 18th International Conference on Engineering of Complex Computer Systems, 2013, pp. 222-231.

[27] A. M. A. Awwad, "Localization to Bidirectional Language for a Visual Programming Environment on Smartphones," IJCSI International Journal of Computer Science Issues, Volume 14, Issue 3, May 2017, doi:10.20943/01201703.113.

Zulfiqar Ali was born in Pakistan. He received his Master Degree (MCS) from Kohat University of Science and Technology (KUST). Currently, he is doing Ph.D in Computer Science under the supervision of Prof. Wolfgang Slany from Graz University of Technology, Austria. His main area of research interest related to software engineering, mobile applications and testing using Behavior Driven Development, methodology and Cucumber framework.