Using Executable Specification and Regression Testing for Broadcast Mechanism of Visual Programming Language on Smartphones

https://doi.org/10.3991/ijim.v13i02.9851

Zulfiqar Ali(✉)
Graz University of Technology, Graz, Austria
zulfiqar.ali@student.tugraz.at

Aiman M. Ayyal Awwad
Tafila Technical University, Tafila, Jordan

Wolfgang Slany
Graz University of Technology, Graz, Austria

Abstract—The rapid advancement of mobile computing technology and the rising usage of mobile apps made our daily life more productive. The mobile app should operate all the time bug-free in order to improve user satisfaction and offers great business value to the end user. At the same time, smartphones are full of special features that make testing of apps more challenging. Actually, the quality is a must for successful applications and it cannot be achieved without testing and verification. In this paper, we present the behavior driven development methodology and Cucumber framework to automate regression testing for Android apps. Particularly, the proposed methods use the visual programming language for smartphones (Catrobat) as a reference. The 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. The results show that the methods are able to effectively reveal deficiencies in the broadcast mechanism, and ensure that the app meets end users' expectations and needs.

Keywords—Mobile Application, Regression Testing, Behavior Driven Development, Visual Programming Language, Catrobat.

1 Introduction

Smartphone and its applications now become a key component in our everyday routine jobs. For the popularity of smartphones, many apps are developed and deployed every day [1, 2]. Such applications have changed totally the style we perform every day's activities, interact with each other, and complete important tasks [2, 3]. However, to introduce new features to users in any mobile apps requires the mobile software to be highly reliable [4]. Developing mobile apps is challenged by the demand to keep moving with matching user's needs and short release cycles, while still providing high-quality software – especially as poor quality is immediately visible in app stores and can have an influence on the app development companies' reputation [5].

Basically, mobile apps are mostly bugs prone because of developers' unfamiliarity with mobile platforms. The increasing complexity of mobile app can arise many challenges in the testing process in order to make sure the app will operate and meet the user's expectations. Smartphones are becoming common; this exposes the necessity for effective techniques for testing their apps. Mobile app testing plays a vital role in making mobile applications more reliable and bug-free [4, 6]. In particular, the complexity of the mobile applications 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 [5].

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, as well as the several sets of scenarios in which a mobile app can be used makes the testing of a new app is costly, time-consuming, and complex task [6].

According to a survey performed by the senior IT-management executives worldwide in 2017, 39 percent of respondents identified certification of their apps as one of the focus domains for mobile app regression testing[i]. Regression testing is an effective way to declare that the final version of the product remains behaving correctly in accordance with the new additions. In this type of testing, the manual process is time-consuming. Thus, to secure the inspection of requirements and promote testing, automated tests are a key factor to support testing in the software environment [7]. The regression testing must be executed after modifications and changes to a given app have been made. It is usually performed by re-running previously run tests and checking whether new faults have emerged.

Mainly, regression testing is used to assert that the software modification did not break previously working functionality. For a large number of tests, regression testing is costly. However, some studies estimate that the testing budget regarding regression testing can take up to 80% as well as 50% cost for the software maintenance. Therefore, as the software application grows the cost of regression testing increases. For instance, Google has examined that their regression testing 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 [8].

Regression testing 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 regression testing might be increased due to the growth in an iterative development and reusability of different software application features [9].

In Android app testing, the developer can test many features with unit tests. New descendant shows the commitment that the developers would want to test with Behavior-Driven Development (BDD) methodology, which focuses on the behavior of the users. BDD presents some new concepts, such as ubiquitous language to express the tests and the involvement of business stakeholders in the software development process.

The objective of the BDD is to bridge the gap between customers and developers with the help of Gherkin language [5].

Behavior-driven development is the evolution and advancement of Test Driven Development (TDD) and Acceptance Test Driven Development (ATDD). Among these approaches, there has been a growing interest for BDD due to its ability to engage business analyst requirement and to easily convert those requirements into business people readable specifications that should work as automated acceptance tests [10, 11].

Therefore, in this paper, an automated regression testing approach for smartphone apps is introduced. We explore the concept and the practice of BDD in the Catrobat programming language. The case studies testing for broadcast mechanism are presented to demonstrate the feasibility of this approach in early phases of the development process, provide a constant quality assurance of requirements, and help customers and development teams to identify possible problems before the publishing process for the mobile app. The results show that the methods are able to effectively expose deficiencies in the app under test, and ensure that the quality of the broadcast mechanism is increased.

2 Mobile Regression Test Automation and Tools

This section presents an overview of the tools which are used by 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.

2.1 Behavior-driven development

Behavior-driven development was originally developed by Dan North[ii]. It is a software development process that evolved from TDD, which is invented by Kent Beck in the early days of the agile approach and usually used in the Extreme Programming approach [8, 10].

Behavior-driven development focuses on defining the fine-grained specification of the targeting system's behavior, in a way that they can be automated. In particular, it describes and observes the behavior of the system as executable specifications and focuses on how the system behaves and interacts with end users. The primary goal of BDD methodology is to promote communication amongst the stakeholders of the project so that all members of the team can understand correctly each feature before the development process begins. This serves to identify key scenarios for each story and also eliminate ambiguities from business requirements [12, 13].

Primarily, scenarios in BDD are used as acceptance criteria, which are written with the help of Gherkin language. The scenarios describe how a particular feature should act in different situations with different input parameters. BDD scenarios are clearly written and easily understandable for stakeholders because it provides natural languages that help stakeholders to specify their tests [10]

In the course of BDD, the executable specifications are automated tests, which show and verify that how the app can provide and deliver a specific business requirement. Whenever a change is made in the app, these specifications (tests) run as a part of the build process. However, they are serving as acceptance tests, determining which new features are complete, and as regression tests, ensuring that new changes have not damaged any existing features of the app. Therefore, the mobile tester can automate an executable specification by writing test code corresponding to each step. The BDD tool, i.e., Cucumber will match the text in each step of the executable scenario to the appropriate and suitable test code.

In addition, BDD specialists implement specifications with a top-down or stepwise approach using the acceptance criteria as goals. It describes the behavior of each component with unit tests which are written in the form of scenarios. With the help of the BDD process which is fully focused on the executable specification, the mobile testers can reduce the cost and effort as well as they can speed up the release process and easily make the changes [12].

2.2 Regression testing

Regression testing is re-testing of previously developed and tested software after a code's changes to ensure that the software is still working in the same way as before the changes. Changes may include software improvements, patches, configuration adjustments, etc.

Regression testing was recommended to test the app's efficiency and improving the transparency in the large-scale software development process. Indeed, 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 process gets unnecessary costly [14].

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 test cases 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. Such testing type is an essential testing to demonstrate that the system of the product and its features remains working and behave correctly in accordance with the new requirements [7].

Fig. 1 shows the automated regression testing for different app's states that can help in making the practice much more competent for the future of the app under test. In every version, the mobile tester can 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 to predict which fragments of the app a change will affect. 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's 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 the availability of resources is one of the common challenges [15].
- Maintaining a balance between the ever-growing test suite size and limited constraints is the biggest challenge in regression testing. A regression test suite can run after a group of bug fixes, every new build, or every modification [15].
- The mobile platforms and fragmentation in use. Regression testing is performed on existing software to ensure previous tests still pass after modifications have been made. Ideally, all parts of the software would be re-tested, but time and budget require prioritizing the features to re-test. Thus, with the aforementioned landscape of mobile platforms and fragmentation, regression testing can be particularly a challenge [16]. However, fragmentation is a critical issue in the mobile world and particularly in the Android world.
- 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 how to identify User Interface (UI) elements across apps on different platforms. [16].

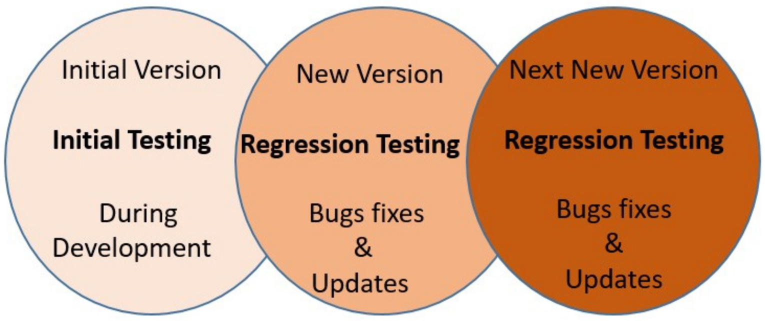


Fig. 1. Regression testing for different app's versions

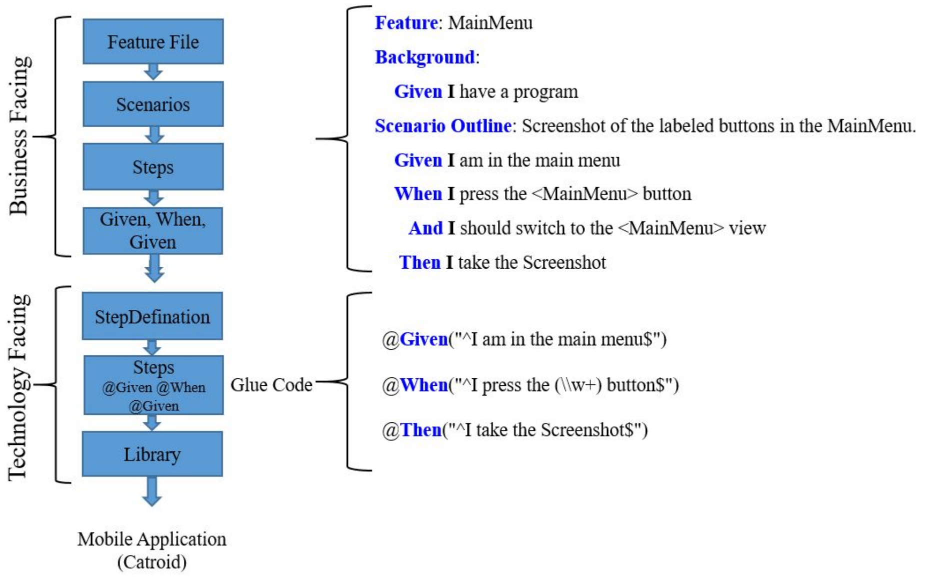
2.3 Cucumber testing tool

Cucumber is an open source testing framework that supports acceptance tests written in a BDD style. The Cucumber was initially written in Ruby and then developed to support the Java framework. Both the tools support native JUnit. Cucumber executes specifications, which are written in natural languages called features. In particular, the features are written by the business analyst, developers, and testers [17, 18].

Fig. 2 shows the Cucumber process, each feature has many scenarios and each scenario has a list of steps (Given, When, Then) for Cucumber to work through and run independently. Furthermore, it shows how the *feature*, *Scenario,* and *steps* look like in English Gherkin syntax. So in this example, we can see the *Given*, *When* and *Then* keywords for verifying tests in Catrobat product. Hence, Cucumber can understand these feature files, which must follow some basic syntax rules (i.e., Gherkin). Along with the features, when Cucumber executes features, it will look for a matching set of step definitions, which map the natural language of each step into a specific code written in Java.

Step definition is just being one or two lines of Java code, which is specific to the domain of the product. Step definitions are the representation of the specifications in code and directions for cucumber on what to do. However, if this code is executed without error, Cucumber would proceed to the next step in the scenario and if it gets to the end without any error, it marks the scenario as passed. If one or any of the steps fail in the scenario, the Cucumber marks this scenario as failed and move on to the next scenario and prints out the results. Some of the benefits the Cucumber tool introduces are: [17].

- It is useful to involve business stakeholders who cannot easily understand the code.
- Cucumber testing concentrates on the end-user experience.
- Writing the testing style provides easier code reusability in the tests.
- The setup and execution processes are quick and easy.
- Cucumber tool provides an efficient framework for testing.
- The mobile tester can write his specifications in more than forty different spoken languages.



```
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$")
```

Fig. 2. The Cucumber process

Cucumber-JVM: Cucumber-JVM is a pure Java implementation of Cucumber, which allows the mobile tester to write a step definition in Java and other JVM languages. It provides a Gherkin implementation of JVM-based languages [12]. Recently, it is added to Android support in the Catrobat Project. This makes it comfortable for users to run it directly on their target devices like regular tests.

Gherkin: Gherkin is a programming language that Cucumber understands and uses to define test cases. It is a ubiquitous plain text language with little extra structure. It is human-readable and easy to learn by non-programmers. Stories usually have a little, a narrative, and a number of scenarios. The Gherkin defines only a few mandatory keywords and the rest of the feature is free-form text [18]. The below mentioned are the Gherkin keywords (See Fig. 3) [7].

- **Feature**: A story written in Gherkin base readable structure. A file should contain one single feature, which contains one or more scenarios.
- **Background**: This section of the feature file allows specifying steps, which are common to every scenario instead of having to repeat similar steps.
- **Scenario**: A feature file contains several scenarios and every scenario is a single actual test case and consists of one or more steps i.e., Given (event or context), When (user action), Then (result/outcome), And, But (when you have many Given/When/Then, then you can use (And and But) to enhance the scenarios' reading process).

Step definition (SD): Step definition is the piece of code and glue that binds our Cucumber tests. The Cucumber is used to convert the feature file to step definitions. The responsibility of step definition is to translate Gherkin scenario steps into Java code and always in between the business and programmer domain. The scenarios steps are only documentation that needs step definitions to bring them to life. Special Cucumber annotation is used i.e., @Given, @When, and @Then to create a step definition in Java. A regular expression is used to match the steps in between the double quotes [17]. Cucumber uses these regular expressions to match the scenarios with the names of generating test methods.

Implementation: To make the testing environment ready, a few dependencies need to be included in the project. In order for Junit to be aware of Cucumber and reads feature files when running the Cucumber class, it must be declared as the Runner. We can see the features element of CucumberOption how it locates the feature file which is created before element glue and provides a path to step definitions (see code snippet).

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

3 Visual Programming Language for Smartphones: Catrobat

Catrobat is a free and open source software project started in Austria at the Graz University of Technology [19]. Catrobat is a visual programming language developed for smartphones and it is inspired by the Scratch project of the Lifelong Kindergarten Group at the MIT Media Lab. Due to the dynamic nature of the Catrobat; children can easily learn how to program without having a previous knowledge of programming syntax [20, 21]. Catrobat defines command bricks (see Fig. 3), which can be snapped together and run in parallel in order to build a program. Unlike Scratch, Catrobat programs can be created and executed solely by using smartphones [20].

The broadcast mechanism is used to communicate between objects or to trigger execution of scripts. By using such a mechanism, sequential or parallel execution of scripts is feasible. Furthermore, Catrobat introduces the capability to include graphics, animations, and sounds. The drag and drop feature offers a variety of bricks that can be snapped together to create complete programs. The app is available free for Android on Google's Play Store i.e., "Pocket Code" [21, 22]. The Catrobat bricks are organized into the categories and every category contains a group of bricks as illustrated below [19]:

- **Event**: Without this category bricks, a program would not be able to start. For example, a "When screen is touched" event can run a script when the screen is touched. Events bricks are necessary for every program.
- **Control**: The elements of this category are used to control the program's flow. For example, a "Forever" brick can run the scripts infinitely.
- **Motion**: Elements of this category are responsible for the sprite movement, by using such elements we can adjust the position and the direction of a sprite on the stage. For example, a "Set X to" brick can be used to set the sprites X coordinate.
- **Sound**: The category's elements are used to control sound such as play or stop a sound file. For example, a "Start sound" brick can be used to play a sound and continues with the next brick immediately.
- **Looks**: The appearance of objects can be controlled by category's elements. The visual effect such as transparency and brightness can be adjusted using this category. For example, a "Set transparency to" brick can be used to set the sprite's transparency to a specific value.
- **Pen**: This category controls the pen aspect of the Catrobat program. It allows a sprite to draw shapes, and plot colored pixels. For example, a "Pen down" brick can be used to put down the sprite's pen, so the sprite will draw as it moves.
- **Data**: The elements of this category allow you to create and manipulate data in your project. Two types of data can be created: variables and lists. For example, a "Set variable to" brick can be used to set the variable to a certain value.
- **Lego EV3**: Elements of this category are used to program Lego Mindstorms EV3 robot.
- **Phiro**: Elements of this category can be used to program and control Phiro (educational robot) via Bluetooth.

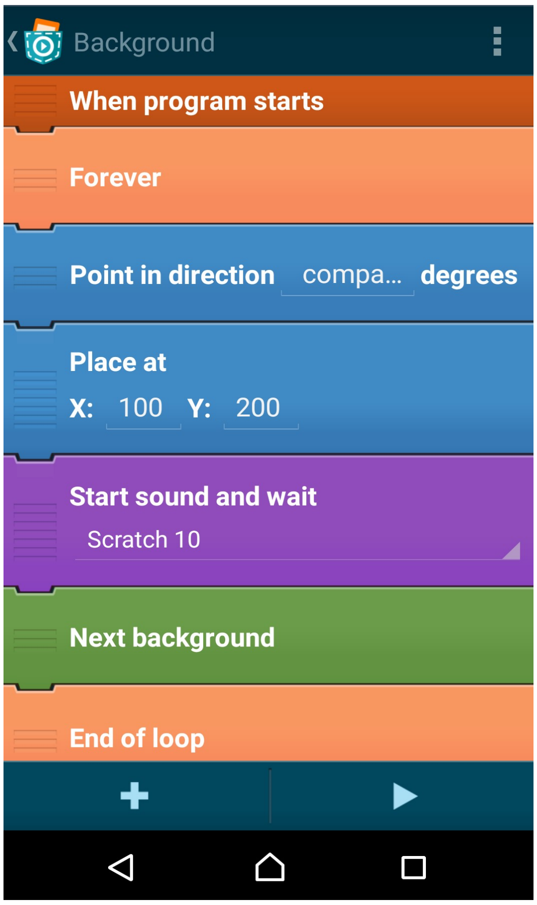


Fig. 3. Script's view in Pocket Code

4 Automated Regression Testing for Catrobat: 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 previously run tests and 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 changes. 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.

4.1 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. The proposed test cases help to increase the transparency of the test selection process and to maintain the test efficiency. They guarantee that the bugs are detected earlier by using automation tools [23, 4]. Moreover, they help in improving the quality of the app as well as discovering the bugs that may be introduced by chance just because of new modifications [23].

4.2 BDD for Regression Testing: Case Studies

The case studies testing for broadcast mechanism are presented to illustrate the proposed approach's feasibility in early phases of the development process and help stakeholders and development teams to locate possible problems before the publishing process for the mobile app.

Case Study 1*:* Catrobat is a visual programing language, which fully depends on bricks functions and their behaviors. Catrobat faces some issues just because of its incorrect bricks' behaviors. So here, 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 a script which is shown in Listing 1 to the background.
- When the program starts.
- The program immediately and deterministically crashes and the Catrobat has stopped messages.

The expected behavior like the following: the program should execute without crashing. When the Catrobat's program is executed, we observe some incorrect behavior in this case. Particularly, the program should run in an infinite loop, but the crash occurs already at the second time and the message "1" is sent. Unluckily, the program immediately crashes and the broadcast brick has stopped message.

```
  Feature: Crash with Broadcast Scripts

  Expected Behavior: The program should execute with-
out crashing.
Actual 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 10.0
       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 20.0

     When I start the program
       And wait 1 second
      Then the variable 'var' should equal to 20.0
```

Listing 1. Deterministic crash with broadcast scripts

Case Study 2: Listing 2 shows the Cucumber feature for "*broadcast and wait*" brick. The expected behavior is: the variable "var" should increase its value one by one with five-second intervals. Whereas, the actual behavior is: the variable "*var*" first wait 5 seconds, then incorrectly increase its values one by one without any further waits.

```
  Feature: Broadcast and wait

  Expected Behavior: The variable should change/in-
crease its value with five-second intervals.
Actual Behavior: The variables change/increase its
value without any 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 broadcast and Wait 'hello'
       And forever end

    When I start the program
       And wait 1 second
     Then the variable 'var' should be less than or
equal to 4.0
```

Listing 2. "Broadcast and wait" brick test feature

Case Study 3*:* Listing 3 shows and specifies the Cucumber feature for the broadcast brick that incorrectly invoked two times. The following steps will generate the critical issue.

- Create a new program in the *landscape* mode (the bug does not appear in the portrait mode).
- Add this script to the background (create the variable as a variable for all objects i.e., global variable).
- When starting the program of the script activity, the mobile tester must not switch to another screen, otherwise bug less frequently occurs, and thus it is more difficult to observe the issue.
- We have to observe whether the screen shows 1.0 or 2.0 after one second of execution. If it is 1.0, which is the expected value, go back to the *script view activity* (simply by pressing the "Restart" button). However, pressing the "Pause" button on the stage is not sufficient to let mobile tester to observe the bug as it never occurs after a simple "Restart"). 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.
Expected behavior: The correct value on the screen should be 1.0.
Actual behavior: When the bug occurs, the incorrect value shown on the screen is 2.0.

- Additional observation or reservation is 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” or by a “Move 10 steps” (with a look) that will be executed as “Move 20 steps.” 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

  Expected Behavior: The correct value of the variable
should be equal to 1.0.
Actual Behavior: In the Landscape mode, when the bug
occurs, the incorrect value will be 2.0.

  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 1.0

     When I start the program
     And wait 1 second
        Then the variable 'var' should equal to 1.0
```

Listing 3. The Broadcast mechanism is incorrectly called two times

5 Conclusion and Future Work

In this paper, we have introduced an advanced agile software methodology (BDD) for a visual programming environment. The approach aims to test and diagnose the bugs in Android mobile app (Pocket Code) which is tested with the help of BDD methodology specifications. Our work focuses on mobile apps and their regression testing. The results show that the proposed approach has the ability to effectively expose deficiencies and bugs in the broadcast mechanism, and it guarantees that the app under test meets the end users expectations. The cross-platform aspect of the Catrobat specifications could not yet be fully implemented. Our challenge is to implement consistent common feature files across different devices as well as with different mobile operating systems i.e., iOS.

6 References

[1] M. Sahrir, M. F. Yahaya, T. Ismail, M. A. Zubir, W. R. Wan Ahmad. (2018). Development and Evaluation of i-Mutawwif: A Mobile Language Traveller Guide in Arabic for Mutawwif (Umrah Tour Guide). International Journal of Interactive Mobile Technologies (iJIM), Vol. 12, No. 2. https://doi.org/10.3991/ijim.v12i2.7708

[2] Amir Dirin, Marko Nieminen. (2015). mLUX: Usability and User Experience Development Framework for M-Learning. International Journal of Interactive Mobile Technologies (iJIM), Vol. 9, No. 3. https://doi.org/10.3991/ijim.v9i3.4446

[3] T. Monahan, M. Bertolotto, G. McArdle. (2009).Usability Testing of a Collaborative and Interactive University on a Mobile Device. International Journal of Interactive Mobile Technologies (iJIM), Vol. 3, No. 4.

[4] L. Nagowah, G. Sowamber. (2012). A Novel Approach of Automation Testing on Mobile Devices, International Conference on Computer & Information Science (ICCIS), Kuala Lumpeu, pp. 924-930. https://doi.org/10.1109/ICCISci.2012.6297158

[5] J. Calamé, P. Kulkarni, S. Euteneuer. (2014). Multi-Platform Mobile Test Automation for the Financial Sector. Testing Experience. No. 27, pp.63-65.

[6] P. Nidagundi, L. Novickis. (2017). New Method for Mobile Application Testing Using Lean Canvas To Improving The Test Strategy. 12th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT). Lviv, pp. 171-174. https://doi.org/10.1109/STC-CSIT.2017.8098761

[7] T. R. Silva, J. Hak, M. Winckler. (2016). Testing Prototypes and Final User Interfaces through an Ontological Perspective for Behavior-Driven Development. Human-Centered and Error-Resilient Systems Development. Published by Springer International Publishing Switzerland. pp. 86-107.https://doi.org/10.1007/978-3-319-44902-9_7

[8] M. Gligoric, L. Eloussi, D. Marinov. (2015). Practical Regression Test Selection with Dynamic File Dependencies. In Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA 2015), ACM, New York, NY, USA, pp. 211-222. https://doi.org/10.1145/2771783.2771784

[9] R. Kazmi, D. N. A. Jawawi, R. Mohamad, I. Ghani. (2017). Effective Regression Test Case Selection: A Systematic Literature Review. ACM Comput. Surv, Vol. 50, No. 2. https://doi.org/10.1145/3057269

[10] C. Solis, X. Wang. (2011). A Study of the Characteristics of Behavior Driven Development. 37th EUROMICRO Conference on Software Engineering and Advanced Applications, Oulu, 2011, pp. 383-387.

[11] M. Rahman, J. Gao. (2015). A Reusable Automated Acceptance Testing Architecture for Micro services in Behavior-Driven Development. IEEE Symposium on Service-Oriented System Engineering, San Francisco Bay, CA, pp. 321-325. https://doi.org/10.1109/SOSE.2015.55

[12] J. Ferguson Smart. (2014). BDD in Action. Behavior Driven Development for the Whole Software Lifecycle. Manning Publications.

[13] M. Alhaj, G. Arbez, L. Peyton. (2017). Using Behavior-Driven Development with Hardware-Software Co-Design for Autonomous Load Management. 8th International Conference on Information and Communication Systems (ICICS).IEEE, Irbid, pp. 46-51. https://doi.org/10.1109/IACS.2017.7921944

[14] E. Engström, P. Runeson, A. Ljung. (2011). Improving Regression Testing Transparency and Efficiency with History-Based Prioritization - An Industrial Case Study. Fourth IEEE International Conference on Software Testing, Verification and Validation, Berlin, pp. 367-376. https://doi.org/10.1109/ICST.2011.27

[15] S. Dalal, Sudhir, K. Solanki. (2018). Challenges of Regression Testing: A Pragmatic Perspective. International Journal of Advanced Research in Computer Science, Vol. 9, No. 1, pp. 499-503. https://doi.org/10.26483/ijarcs.v9i1.5424

[16] T. Grønli, G. Ghinea. (2016). Meeting Quality Standards for Mobile Application Development in Businesses: A Framework for Cross-Platform Testing. 49th Hawaii International Conference on System Sciences, Koloa, HI, pp. 5711-5720. https://doi.org/10.1109/HICSS.2016.706

[17] S. Rose, M. Wynne, A. Hellesoy. (2015). The Cucumber for Java Book: Behavior-Driven Development for Testers and Developers. Pragmatic Bookshelf.

[18] C. Tao, J. Gao. (2017). An Approach to Mobile Application Testing Based on Natural Language Scripting. The 29th International Conference on Software Engineering and Knowledge Engineering, Pittsburgh, PA 15238 USA. pp. 260-265. https://doi.org/10.18293/SEKE2017-170

[19] W. Slany. (2010-2018). Catrobat education. [Online]. Available: https://edu.catrob.at/

[20] W. Slany. (2012). A Mobile Visual Programming System for Android Smartphones and Tablets. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Innsbruck, pp. 265-266. https://doi.org/10.1109/VLHCC.2012.6344546

[21] W. Slany. (2012). Catroid: A Mobile Visual Programming System for Children. In proceedings of the 11th International Conference on Interaction Design and Children. pp. 300-303. https://doi.org/10.1145/2307096.2307151

[22] W. Slany. (2014). Pocket Code: a Scratch-Like Integrated Development Environment for Your Phone. ACM SIGPLAN Conference on Systems, Programming, and Applications: Software for Humanity, Portland, Oregon, USA, pp. 35-36. https://doi.org/10.1145/2660252.2664662

[23] C. Bernaschina, R. Fedorov, D. Frajberg, P. Fraternali. (2017). A Framework for Regression Testing Of Outdoor Mobile Applications. IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft), Buenos Aires, pp. 179-181. https://doi.org/10.1109/MOBILESoft.2017.13

7 Authors

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.

Aiman Mamdouh Ayyal Awwad is currently a full-time lecturer in the Department of Computer Science and IT at Tafila Technical University. He received his B.Sc in Computer Science from Mutah University in 2007 and his M.Sc in Computer Science from the University of Jordan in 2010. He obtained his Ph.D. in Computer Science from Graz University of Technology/ Austria in 2017 with research interests related to smartphone applications. From February 2010 to September 2014, he was a lecturer at Computer Science and IT Department / Tafila Technical University. He has more than 7 publications in various international journals and conferences. His research interests include mobile computing and applications, image processing, and cellular automata.

Professor Wolfgang Slany heads the Institute of Software Technology at Graz University of Technology and is the head and founder of the Catrobat non-profit free open source project, in which more than 1,000 pro-bono collaborators from around 100 countries are developing apps that allow kids to create their own games, animations, and other apps, directly on their phones. Wolfgang is passionate about poverty alleviation through coding education for teens, in particular girls, refugees, and teens in developing countries. Catrobat works in a sustainable way also for teens in less privileged regions that do not have access to PCs and laptops, by relying on the phones most teens everywhere on Earth already personally own, and by bypassing traditional school pedagogy, instead using a constructionist approach focusing on game app development and fun. Professionally, Wolfgang conducts research, teaches, and is consulting on sustainable large-scale agile software development and user experience topics for mobile platform projects.

Article submitted 12 October 2018. Resubmitted 13 November 2018. Final acceptance 07 January 2019. Final version published as submitted by the authors.

[i] https://www.statista.com/statistics/500605/worldwide-mobile-application-testing-focus-areas/
[ii] https://dannorth.net/introducing-bdd/