A Proposal of Android Programming Learning Assistant System with Implementation of Basic Application Learning

Yan Watequlis Syaifudin, Nobuo Funabiki, Minoru Kuribayashi
Department of Information and Communication Systems
Okayama University
Okayama, Japan
funabiki@okayama-u.ac.jp

Wen-Chung Kao
Department of Electrical Engineering
Taiwan Normal University
Taipei, Taiwan
jungkao@ntnu.edu.tw

***Abstract*—Purpose - With rapid increase of *Android* devices and application systems, *Android application programmers* have been strongly demanded. A lot of schools are offering *Android programming* courses to meet this demand. However, Android programming can be different from the conventional one, because it needs interactive functions through interfaces with users, which makes the study more difficult.**

Design/methodology/approach - We propose an *Android Programming Learning Assistance System* named *APLAS*, to assist the Java-based Android programming study and education.

Findings - Using *JUnit* and *Robolectric*, the answers from the students are automatically marked in APLAS, to achieve independent learning without the presence of teachers. To cover extensive topics in Android programming, APLAS offers four stages where each stage involves several topics.

Originality/value - To evaluate the effectiveness of APLAS, we implemented the *Unit Converter* assignment that covers the first two topics, *Basic UI* in the first stage and *Basic Activity* in the second stage. Through solving the assignment, it is expected to learn a *Basic Application* development. We asked 40 novice students of an IT department in Indonesia to solve the both topics separately. The results show that APLAS is useful and helpful to learn Android programming, as they could complete codes with good execution performances.

***Index Terms*—APLAS; Android application; Learning system; Java; Test-driven development method; JUnit; Robolectric**

Paper type Research paper

I. Introduction

Nowadays, the use of mobile devices, especially smartphones, continues to increase steadily. According to Statista, the number of mobile device users has reached more than five billion around the world, and will continue to grow in the following years. Particularly, smartphone users reach 2.7 billion, which occupies around 52 percent of the mobile device users.

Google Android and *Apple iOS* are the two most popular operating systems for smartphones currently available. Among them, *Android* occupies 80 percent of the smartphone market. Based upon StatCounter GlobalStats, *Android* is the most popular operating system for mobile devices. At the beginning of 2019, *Android* reached the market share of 37.6 percent and was followed by *Microsoft Windows* with 35.6 percent, which reveals the increasing number of *Android* applications. On the other hand, the AppBrain indicates that the number of applications in the *Google Play Store* has reached 2.6 millions.

Under this trend, the demand for Android application programmers has become one of the highest in the IT field [Statista, 2019]. A lot of schools have offered *Android programming* courses to meet this demand, with a large number of students are studying it. However, Android programming seems different from the conventional one for implementing logic functions, because it additionally needs interactive functions through user interfaces with users, which makes the study more difficult. Therefore, how to teach Android programming effectively is an issue that has been widely discussed among lecturers, teachers, and trainers in order to produce high quality programmers.

In this paper, we propose an *Android Programming Learning Assistance System* named *APLAS* to assist the Java-based Android programming study and education. APLAS assumes the use of *Android Studio* IDE to develop an Android application with the combination of *Java* and *XML*. *Java* is used to implement the logic functions and *XML* is to describe the user interfaces.

In APLAS, adopting the *Test-Driven Development (TDD)* method [Farcic & Garcia, 2015], any answer from a student will be automatically marked, so that it can guide a student to have independent learning without the help from a teacher. To achieve it, *JUnit* and *Robolectric* are used in APLAS, which provide the unit testing capability [Koskela, 2008] for the student's answer code. That is, *JUnit* tests the logic functions and *Robolectric* tests the interfaces.

It has been observed that Android applications covering a variety of topics. Then, they could be explored through the four *stages*: *User Interface*, *Interactive Application*, *Content Provider*, and *Service Interaction*. Each stage contains several *topics* separately. When a student starts solving a new topic in a stage, he/she needs to obtain the package of the necessary files for the topic, including the *task guide* describing the set of tasks to do, and the detailed specifications.

In this paper, we implement the *Unit Converter* assignment to build a *Basic Application*, which can convert units in temperature, distance, and weight. This assignment covers the first two topics, *Basic UI* in the first stage and *Basic Activity* in the second stage. Through solving the assignment, it is expected for a student to learn a *Basic Application* development [Syaifudin et al, 2019].

As the first topic, this assignment involves the *Basic UI* topic that contains:

1) starting an android project,
2) configuring the resources, and
3) building the layout in the user interface, which are composed of nine tasks.

By completing these tasks sequentially, a student can build a user interface for a *Unit Converter* application.

As the second topic, it covers the *Basic Activity* topic that encompasses:

1) creating Java classes,
2) understanding *Activity Lifecycle*,
3) creating methods accessing of resources, and
4) creating event listeners, which are also made up of nine tasks.

Then, we asked 40 novice students of an IT department in Indonesia to solve the assignment. First, they solved the tasks for *Basic UI*, and then, solved the tasks for *Basic Activity*. The results show that 90% of the students successfully completed all the tasks in the assignment and built the *Unit Converter* application. Thus, it is confirmed that APLAS is useful and helpful for novice students to learn Android programming by building a simple user interface. In addition, the performance evaluations of unit and integration tests in APLAS using *JUnit* and *Robolectric* showed that they both have satisfied execution speeds.

The rest of this paper is organized as follows: Section II reviews related works on programming learning and Android application testing. Section III explains TDD, unit testing, and integration testing for Android application. Section IV shows the details of APLAS and learning model. Section V presents the implementation strategy of *Basic Application* development. Section VI explains the process of validating the codes. Section VII presents evaluations for the *Basic UI* topic implementation. Section VIII presents evaluations for the *Basic Activity* topic implementation. Finally, Section IX concludes this paper with future works.

II. Related Works

In this section, we survey some related works in literature.

Sadeh & Gopalakrishnan [2011] conducted an evaluation of the unit test use for Android applications. They showed that *Robolectric* will produce the fast testing process by providing relevant instruments to test a user interface code.

Chandra & Liem [2013] combined a source code for editing an evaluator with *Oddysseus*, a Web-based integrated system for learning programming with the *auto-grader* ability. They developed a source code editing evaluator named *Doppel and Ganger (D&G)* that can evaluate the typing process in a text editor, where *Oddysseus* can compile and grade the produced source code. They evaluated *D&G* by testing source codes of *Pascal*, *C*, and *C++* from students.

Funabiki et al [2013] proposed a Web-based *Java Programming Learning Assistant System (JPLAS)* for self-learning of Java programming using the TDD method. JPLAS uses *JUnit* for unit test of source codes from students. It can enhance educational effects in Java programming courses by allowing self-studies of students while reducing teacher loads. They evaluated the proposed system and showed the effectiveness of the system implementation in university students.

Ortiz et al [2015] presented an experience based on the use of *m-learning* in higher education. They determined the degree of the acceptance and usability of an educational application for mobile devices. The application allows teachers to monitor the tasks of students and detect the structures of written codes where they found difficulties. They concluded that teaching tools should be incorporated automatic assessment methods.

Kang & Cho [2015] studied the Android programming education using *Multi Android Development Tools* on *MIT App Inventor* and *Eclipse*. For novice students, it is easy to use puzzle models on *MIT App Inventor*. However, they have to use *Eclipse* to make real applications. Also, the teacher needs to manually check the validity of the code made by the student.

Vásquez et al [2017] conducted a survey of Android developers about their experiences when performing testing. This study aimed at gathering information about the practices of testing tools in Android application developments. The result showed that the most used tool for automated testing is *JUnit*, followed by *Roboelectic*, and then, *Robotium*. This result confirmed the previous study of Kochhar et al [2015] that showed the three tools are actually the favorite ones in Android application developments.

García & Rosa [2016] have developed a Web application interface for programming learning by children and youth, namely *RoBlock*. They designed the Web Application to satisfy the autodidactic programming learning by using *Visual Blocks Programming*, which includes the materials and concepts to be learned by them. This tool has the similar idea with *Multi Android Development Tools* where the block model is referred on the both works.

Rekhawi & Naser [2018] have developed a Web-based intelligent tutoring system for teaching Android application developments. This system provides the lessons of the Android programming overview, the basic user interface, and the application design. Besides, it offers questions in each lesson to be answered by students, and evaluations of the submitted answers to them. Unfortunately, this system fails to support learning coding for Android application developments.

Luccio [2019] proposed a case to use robots to learn distributed algorithms by simulations. The proposed system used robots that are used for educational purposes to increase the intellectual abilities of students, especially in distributed algorithms. The author presented a project-based learning approach that refers to the advanced algorithms course at the

University Ca Foscari of Venice, Italy, in 2017 and 2018. The result showed that the students participating this project obtained the excellent final grades, compared to the ones with traditional written tests.

III. Test-Driven Development Method in Android Application

In this section we review the TDD method in Android application developments.

A. *Overview of Test-driven Development Method*

The TDD method is the software development process that relies on the repetition of a very short development cycle. The requirements in the source code are turned into the specific *test cases* in the *test code* [Wikipedia, 2019]. The source code is improved by passing the tests [Blundell & Milano, 2015]. The *test code* is written in advance or parallel with the source code development process, and guides the programmer at developing the source code. The TDD method is adopted in APLAS to automatically validate the answer codes of the students. Testing automation allows improving efficiency and coverage of application for better validation [Kaur, 2015].

B. *TDD in Android Application Developments*

To implement TDD method, Android application testing has to perform automated testing process. This process means using source code or a design model as predefined requirement to create a test case and tests the application automatically [Kim, 2013]. Android application testingshould include three categories of tests, namely small, medium, and large [Google Developers, 2019]. To be specific, below the details of each type of testing:

1) **Small tests** are unit tests that validate Android appplications behavior one class at a time and cover big portion of entire Android application testing. The tools that can perform a small test are *JUnit*, *Mockito*, or *Powermock*. *JUnit* is the most popular, easy to use, and light tool to perform unit testing in Java platform. *Mockito* is an open-source testing framework for Java that allows the creation of mock objects testing in automated unit tests. *Powermock* a unit testing framework that extends others mock libraries with more powerful capabilities.
2) **Medium tests** are integration tests that integrate several components and provided by *Robolectric* that can be run on the local *Java Virtual Machine* (JVM).
3) **Large tests** are UI tests that run by completing a UI workflow on an emulator or real device using a platform like *Espresso*, *UI Automator*, or *Robotium*. *Espresso* is an open-source testing framework created by Google which provides an API that allows to create user interface tests. *UI Automator* is a testing framework that provides a set of APIs to perform UI tests automatically. *Robotium* is an open-source test framework that can perform automatic gray box testing cases and allows developers to write function and acceptance test scenarios.

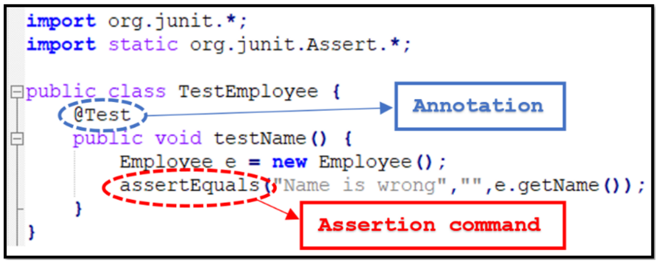
In this paper, we adopt *unit test* using *JUnit* and *integration test* using *Robolectric* in APLAS. Both types of testing allow us to define application specifications and validating them automatically without running the applications on smartphones or emulators. In addition, most of the functionality of an android application can be tested with both types of testing.

C. *Unit Test*

Unit test is a practice to test a small, individual, and isolated unit of a whole source code, such as a method and a class in Java. It is useful to prove the validity of the unit code. *JUnit* has been the *de-facto* standard for the automated unit test on Android application.

D. *JUnit*

JUnit is considered the most popular unit test framework for Java. It is fully supported by *Android Studio* that helps us create a test process for an Android application project. The unit test needs a test code that contains one or more *test methods* indicated by the *@Test* annotation. Each test method contains an *assertion method* to execute a single function in the source code and determine the pass or fail of this test case. The assertion method is provided by the class library *org.junit.Assert* extending *java.lang.Object class*. Figure 1 shows a simple test code for the unit test on *JUnit*.



```
import org.junit.*;
import static org.junit.Assert.*;

public class TestEmployee {
    @Test
    public void testName() {
        Employee e = new Employee();
        assertEquals("Name is wrong","",e.getName());
    }
}
```

Fig. 1. Unit test on JUnit.

E. *Integration Test*

Integration test is another practice to test combinations of individual components that work together. The components have been tested by *unit tests* independently. Then, they are combined together to test the integration of them. In general, an Android application project is composed of several components, such as *Manifest*, *Resources*, *Layout*, *Activity*, *Unit*, and *Gradle*. All or part of the components must be integrated to form a complete Android application that can contain a user interface. *User interface (UI) test* is important, because numerous Android applications offer UI as the main function. *Robolectric* is the framework that brings fast and reliable *integration tests*.

F. *Robolectric API*

Robolectric provides a testing framework that allows to test an Android application on *Java Virtual Machine (JVM)* without an emulator or a device [Hussain et al, 2017]. Then, it can perform the unit test on *JUnit* by integrating all the components in an Android application, as shown in Figure 2. The test can run fast, because it converts a Java code to a *Dalvik* code [Dalvik bytecode, 2019] that deploys an emulator, which rewrites the Android core libraries using shadow classes. *Robolectric* redefines Android methods so that they return default values and forwards the method calls to the shadow objects, mimicking the Android behaviors [Blundell & Milano, 2015].

IV. Proposal of Android Programming Learning Assistance System

In this section, we present the *Android Programming Learning Assistance System (APLAS)* for assisting self-learning of Android programming.

A. System Architecture

APLAS adopts *Android Studio* as the most popular integrated development environment (IDE) for the platform to develop Android applications. *Android Studio* is packaged with the *Android SDK* that is a set of tools to facilitate Android developments, allows users to work on tasks using a combination of Java and XML, and supports main operating systems of *Windows*, *Linux*, and *Mac OS*. As a system, the APLAS architecture consists of several components that run on Android Studio which runs on *Java Virtual Machine* with *JDK 8*, as revealed in Figure 3.

In *Android Studio*, there are Android components and test codes. Android components involve *Activity*, *Layout*, *Resources*, and other components like *Java Class*, *Fragments*, *Service*, *Intents*, etc. To illustrate, *Activity* is an Android component that contains the logic definitions of a *User Interface* using Java. *Layout* is a definition to build a *User Interface* and written in XML. *Resources* are the additional files and the static content that need in an Android project, such as bitmaps, strings, drawables, styles, animation instructions, and more.

A *test code* is written using Java. The test uses *JUnit 4* and *Robolectric 4.2.1* as API to run the unit test and the integration test. Then, the Android components and the test code are compiled using the *Gradle Build Tool* and run on *Java Virtual Machine*.

B. Four Learning Stages

Learning materials for Android programming contain a variety of topics from basic levels to advanced ones. In APLAS, the topics of learning will start in basic level and specific topic. It will be useful to focus in particular part on the enabling step of functioning as an novice student [Robins et al, 2003]. In this paper, we divide the topics into the following four stages, referring to the learning courses in [Udacity, 2019], [Google Training, 2019], [Android Developer Fundamentals, 2019], and [Horton, 2015]:

1) **User Interface** focuses on creating an Android application interface. Besides designing the user interface with XML, this stage covers configuring project properties and managing project resources.
2) **Interactive Application** focuses on implementing the activity in the project to build an Android interactive application. It is requested to create Java codes to utilize widgets, events, fragments, intents, and multimedia.
3) **Content Provider** focuses on utilizing shared sets of application data stored in *SQLite* database on the Web, or on any other persistent storage location.
4) **Service Interaction** emphasizes utilizing existing services provided by the Android operating system, where a service indicates an application component that can perform in the background without a user interface.

C. Learning Topics in Four Stages

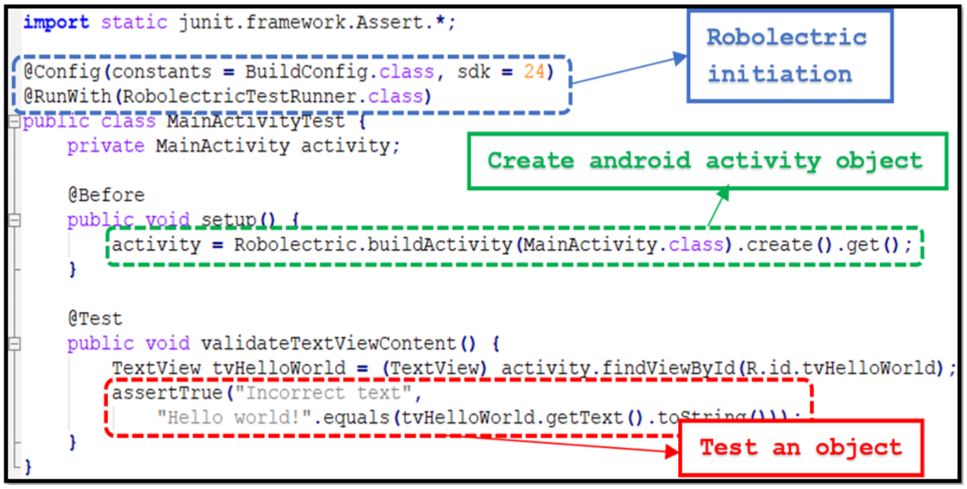
Every stage has one or several topics with different contents for learning. Each topic focuses on a specific case for building an Android application. The following is a list of learning topics for each stage on the APLAS.

1) User Interface: This stage ony has one topic named ***Basic UI***. This topic contains several material lessons including *project properties*, *layout design* using XML, and *definition and management of resources* like *drawable*, *colors*, *strings*, and *styles*. This is an important step in learning Android, where users will start creating Android applications by designing interfaces with XML.

2) Interactive Application: This stage includes four topics, named ***Basic Activity***, ***Advanced Widget***, ***Multiple Activity***, and ***Multimedia Resource***. *Basic Activity* aims to study the basic of programming in *Activity* of Android project. *Advanced Widget* explores utilizations of *widget*, *timer*, *style*, and *data structure*. *Multiple Activity* investigates how to create an application with multiple activities by utilizing *intent* and *fragment*. *Multimedia Resource* aims to study how to create animations with XML files and utilize multimedia resources.

3) Content Provider: This stage includes four topics, named ***Basic Data Storages***, ***SQLite Database***, ***Network Connection***, and ***Data Service***. *Basic Data Storages* aims to study the optios to save persistent application data, including *Shared Preferences*, *Internal Storage*, and *External Storage*. *SQLite Database* aims to study how to store structured data in a private database, namely SQLite. *Network Storage* aims to study how to store data in network including network server and cloud. *Data Service* emphasizes how to store data privately and make them available publicly.

4) Service Interaction: This stage includes four topics, named ***AsyncTask***, ***Web Contents***, ***Service Application***, and ***Notifications***. *AsyncTask* aims to study how to process some tasks in the background using *AsyncTask* class. *Web Contents* aims to study how to access data available on web service using the web API. Examples of data provided by web services are news article, weather, and contacts. *Service Application* highlights how to build application as a service. A service is an application component that performs long-running operations without a user interface. *Notifications* aims to study how to create, deliver, and reuse notifications. A notification is a message generated by application and displays it outside application's normal UI.



```
import static junit.framework.Assert.*;

@Config(constants = BuildConfig.class, sdk = 24)
@RunWith(RobolectricTestRunner.class)
public class MainActivityTest {
    private MainActivity activity;

    @Before
    public void setup() {
        activity = Robolectric.buildActivity(MainActivity.class).create().get();
    }

    @Test
    public void validateTextViewContent() {
        TextView tvHelloWorld = (TextView) activity.findViewById(R.id.tvHelloWorld);
        assertTrue("Incorrect text",
            "Hello world!".equals(tvHelloWorld.getText().toString()));
    }
}
```

Fig. 2. Integration test on Robolectric.

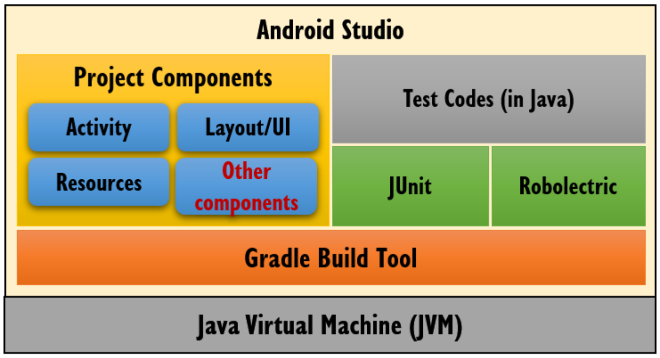


Fig. 3. APLAS Architecture

D. Learning Process in Each Topic

In APLAS, one *topic* consists of several *tasks*, which need to be finished by the students in sequence. Each task has the unique number and the objective, and is made up of the four steps, namely:

1) **start learning**,
2) **configure project**,
3) **do the task**, and
4) **test the code**

The process model of each topic is shown in Figure 4.

The detail of each step is described as follows:

1) Start Learning: This step is designed to start learning a task, by obtaining the corresponding package of files that consists of **guide document**, **supplement file(s)**, and **test code(s)**. The *guide document* will guide a student to handle a task, which involves the learning objective, the requirements of hardware and software, the resource details described by the list of files in the task package, the task description, the task guidance on how to do the task step by step, and the testing guide. The *supplement files* include the necessary files in the Android project, such as images, fonts, and animations. The *test code* is used to validate the result of the code in Android project produced by a student.

2) Configure Project: This step creates or opens a new Android project on *Android Studio*, and configures the project by following the *guide document*. It needs to be configured as below:

1) project properties: the project type, the target device, the application name, the package name, the file location, and the programming language,
2) Android manifest: the file that describes the essential information on project structures, the application to the Android build tools, the Android OS, and *Google Play*,
3) Gradle configuration: the advanced build toolkit to automate and manage the build process with flexible custom build configurations.

The student must synchronize the *Gradle* and all the Android components with one single click, and must ensure that the configuration results are correct and the *Gradle* synchronization process is successful in the next step.

3) Do the Task: This step will do the task described in the *guide document*. Each task contains copying the *supplement file(s)* to the project, and writing the codes to build an Android component. The codes may appear in *Activity*, *Layout*, *Resources*, *Java classes*, or the other components. After completing this task, students can run the application to review their achievement in each task.

4) Test the Code: This step validates the student task by running the *test code* to get the feedback. If all the tests in the test code are passed, the result appears with the *green icon*. If there is at least one failed test, the result appears with the *red icon*, as shown in Figure 5. If a student has passed the task, he/she may move to the next task.

E. Learning Environment

To use APLAS, every student must use a computer device that meets the minimum requirements of the hardware specifications listed in the *guide document*. *Android Studio*

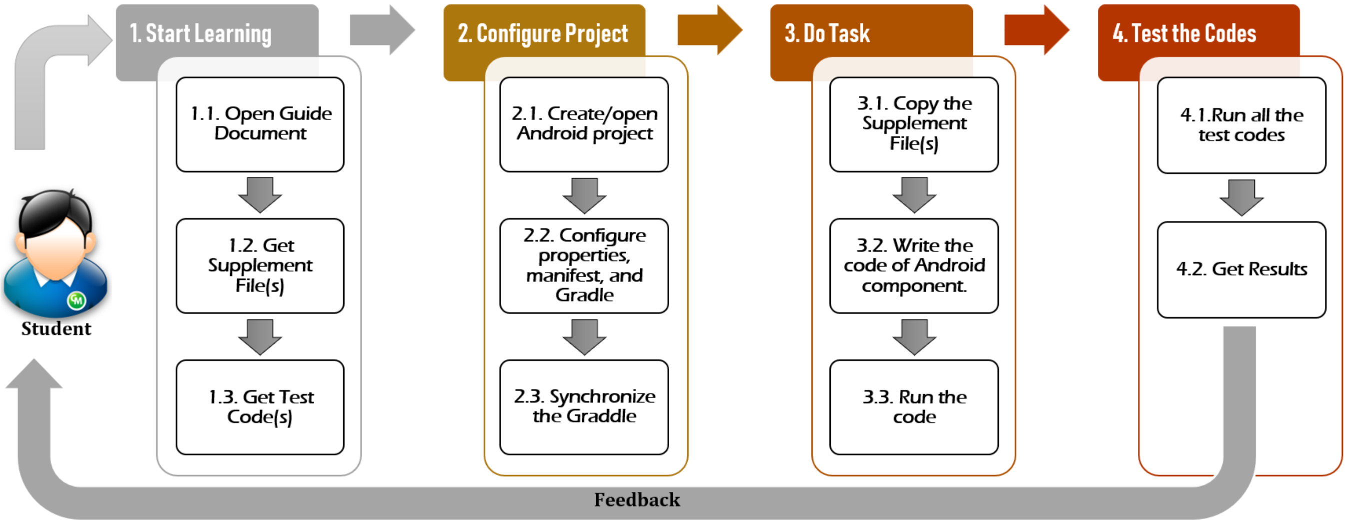

1. Start Learning
1.1. Open Guide Document
1.2. Get Supplement File(s)
1.3. Get Test Code(s)
2. Configure Project
2.1. Create/open Android project
2.2. Configure properties, manifest, and Gradle
2.3. Synchronize the Graddle
3. Do Task
3.1. Copy the Supplement File(s)
3.2. Write the code of Android component
3.3. Run the code
4. Test the Codes
4.1.Run all the test codes
4.2. Get Results
Student
Feedback

Fig. 4. Learning process for each topic.

Tests passed:

Tests failed: 5,

Fig. 5. Test result.

is installed in the device, which should be connected to the Internet, because *Android Studio* can be updated and install additional APIs to run APLAS, as indicated in Figure 6.

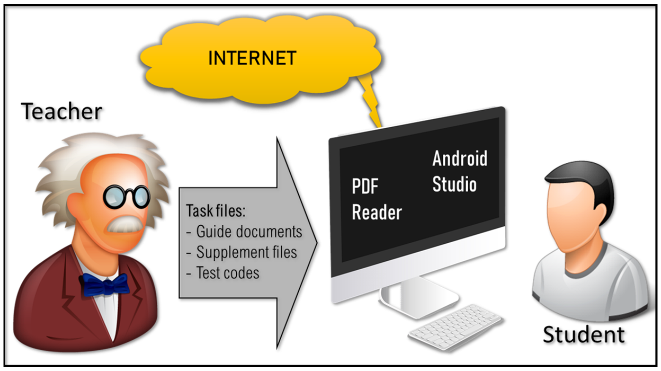


Fig. 6. Learning Environment of APLAS

The teacher will provide the students with a package of the files required for the assigned topic in a compressed file format. When a student decompresses it, several folders are generated. Each folder corresponds to one task with a specific number such that the students will accomplish the tasks sequentially. It contains the guide document, the supplement file(s), and the test code(s) for the task. By following the instructions in the document guide, the students could try to cope with the task.

V. Basic Application Learning in APLAS

In this section, we present the design and the assignment implementation for learning *Basic Application* in APLAS.

A. *Overview*

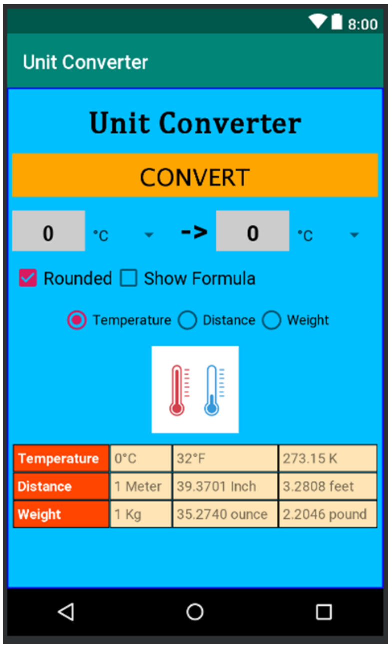
At the early stage of Android programming, students should learn how to create a simple application, which needs *User Interface* design using XML and *Activity* programming using Java. As the initial stage of implementing APLAS, a simple application is targeted, called *Basic Application*. First, the student will learn the basics of designing an application layout and defining the resources, which is packaged in the *Basic UI* topic. Next, the student will learn the fundamentals of *Activity* programming and class programming using *Java*, which is packaged in the *Basic Activity* topic.

B. *Outline of Assignment*

As an assignment of *Basic Application*, the *Unit Converter* application is implemented in APLAS. *Unit Converter* is a simple application to convert a value for temperature, weight, or length between different units. Figure 7 illustrates the user interface. This application uses *LinierLayout*, *RelativeLayout*, *TextView*, *EditText*, *Button*, *Spinner*, *Checkbox*, *RadioGroup*, *RadioButton*, *ImageView*, and *TableLayout* as the components. The resources defined are *strings*, *fonts*, *string-arrays*, *images*, and *drawables*. In the *Activity* programming, several procedures, such as changing the value of a unit to another unit, rounding the decimal value, displaying an image, and hiding an image should be implemented.

C. Basic UI *Topic*

First, the details of the *Basic UI* topic are described.



0 °C -> 0 °C

☑ Rounded □ Show Formula

● Temperature ○ Distance ○ Weight

Temperature	0°C	32°F	273.15 K
Distance	1 Meter	39.3701 Inch	3.2808 feet
Weight	1 Kg	35.2740 ounce	2.2046 pound

Fig. 7. *Unit Converter* application interface.

TABLE I
TASKS IN *Basic UI* TOPIC

No.	Task	Goal	Detail
1	project configuration	start	start new Android project, configure the properties, configure *the gradle*, and sync *the gradle*
2	resource configuration	resource	configure *Android Studio* project resources, like string and font
3	*Main layout*, *Textview*, *Button*	UI	design basic layout, add *TextView*, and add a *Button*
4	*Space* and *Child layout*	UI	create *Space* and add *Child Layout*
5	*String-array*, *EditText*, *Spinner*	resource, UI	configure *string-array* resources, make *EditText*, and *Spinner*
6	*Checkbox*	UI	create *Checkbox* and configure the properties
7	*RadioGroup*	UI	create a *RadioGroup* and add some *RadioButtons* inside
8	*Image resource* and *ImageView*	resource, UI	add image resource and make *ImageView* in layout
9	*Drawable resource* and *Table layout*	resource, UI	add *Drawable resource* and make *Table layout*

1) Learning Goals: Referring to *Horton's book* and *Udacity* site, the following three learning goals are set for students to start an Android application project:

- **How to start an Android project (start):** needs the knowledge of Android project paradigm, like the package, *Android SDK*, the application compatibility, and *Gradle Build Tool*.
- **How to configure the resources (resource):** requires the knowledge of Android project resource types.
- **How to build a user interface layout (UI):** needs the knowledge of the layout design paradigm using *XML*.

2) Tasks: The nine tasks in Table I must be completed in this order, then an application layout will be established. They include the three learning goals. When a student completes all of them, the *Unit Converter* application layout will become available.

D. Basic Activity *Topic*

Next, the details of the *Basic Activity* topic are described.

1) Learning Goals: Activity programming is the fundamental subject to develop an Android application. The following four goals allow students to understand the *Basic Activity* programming:

- **How to create *Java* class for unit (Java):** needs to learn how to make simple *Java* classes including fields and methods.
- **How to understand *Activity* lifecycle (lifecycle):** needs to learn that the *Activity* class in *Android* project provides callbacks to know the change of a state.
- **How to create method accessing to project resources (resource):** should gain the understanding of additional files and static contents beside codes, such as bitmaps, layout definitions, strings, and animation instructions.
- **How to create event listener (event):** needs to learn *Event* as an important way to make an Android application provide interactions between the user and the interactive components or widgets such as *Button click* or *Spinner select item*.

2) Tasks: The nine tasks in Table II must be completed in this order, to build a *Unit Converter* application. They include the four learning goals.

VI. AUTOMATIC CODE VALIDATION IN APLAS

In this section, we present the design and implementation of the automatic code validation for *Basic Application* in APLAS.

A. Overview

In APLAS, the final step in each task is the validation of the answer files of XML and Java source codes from the student. For self-learning of Android programming, the automatic code validation method using *Unit Testing* on *JUnit* and *Integration Testing* on *Robolectric*, is adopted in APLAS. This is regarded as the most important mission in APLAS. In this section, we will explain the validation model, the validation type, and the details of each type.

B. Validation Model

In APLAS, the student who has completed a task can validate the answer files using the unit testing method using *Unit Testing* and *Integration Testing*. As shown in Figure 8, the testing of *Java* source codes can be done on *JUnit* with *assertion* methods directly. On the other hand, the testing of specific features for the Android project, such as the application layout,

TABLE II
TASKS IN *Basic Activity* TOPIC.

No.	Task	Goal	Detail
1	make *Temperature* class	Java	make simple Java class to convert unit in *Temperature*
2	make *Distance* class	Java	make simple Java class to convert unit in *Distance*
3	make *Weight* class	Java	make simple Java class to convert unit in *Weight*
4	define fields and methods in *Activity*	Java	define fields of widgets and assign them with related resources in the project
5	make *Override method* in *Activity*	lifecycle	define methods to state *Activity* into *onCreate* and*onStart*
6	create *RadioGroup* event	event	create event listener method for *RadioGroup* that will influence contents in *Spinner*
7	make method to convert units	resource	create method to access values in *EditText*, *Spinner*, and *Checkbox*, run unit conversion, and put the result on the *EditText*
8	create *Widget* event listener	event	create event listener methods for action in *Button*, *Spinner*, and *Checkbox*
9	create *ImageView* and event	event	define image resource and show it when *Checkbox* is checked

the *Activity* lifecycle, the *event listener*, and the accessing resources, needs to use *Robolectric* to establish the unit testing of an Android project in the *Java* environment. *Robolectric* creates a new class to simulate an Android application. Then, the class will be tested with *assertion* methods on *JUnit*.

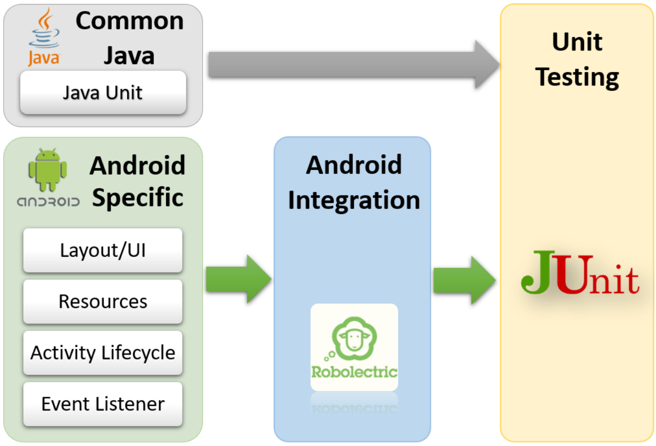


Fig. 8. Validation process of *Basic Activity*.

The validation of each component can be done through the following steps:

1) Initiation: This step runs the initial process, such as building the activity and assigning fields.

2) Validation of Components: The validation of the components are as follows:

- **Layout validation** using *Integration Test*,
- **Resource validation** using *Integration Test*,
- **Activity Lifecycle validation** using *Integration Test*,
- **Event listener validation** using *Integration Test*,
- **Java class validation** using *Unit Test*.

3) Fail Test Message: If there is a failed test, the corresponding message will appear in APLAS.

C. Layout Validation

The Android UI is built using an *XML* code that contains the layout elements or widgets such as *Button*, *LinearLayout*, *Space*, and *EditText*. Each element has a unique number to represent the identity. Each element also has properties such as the width, the height, and the background color, where each property has the specific data type and value.

To validate the application layout, the following two steps are applied:

1) Check the completeness and sequence: the number of elements that must be right and the order of placement of each element must be correct.
2) Check the value of each property for all the elements:

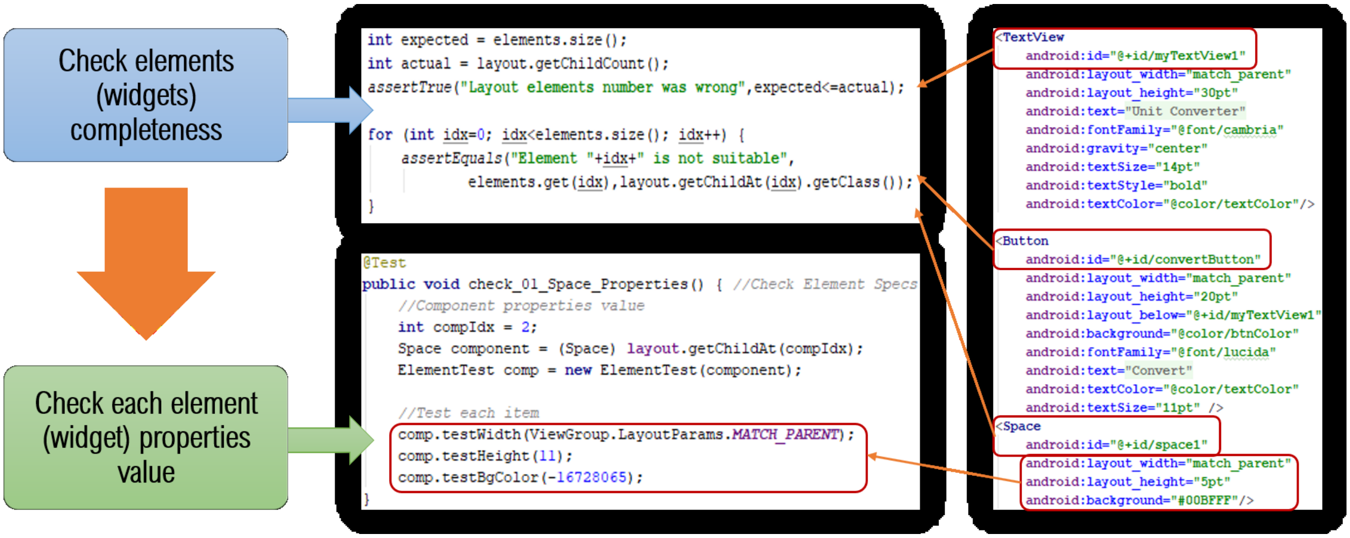
If the elements in the layout are correct in the number and the sequence, the value of each property in each element needs to be tested. Figure 9 shows the process and the sample test code to prove the accuracy of a layout.

D. Resource Validation

The use of resource files is a way of separating the static values in the application from the source codes Android Developer Fundamentals [2019]. The following six types of resources need to be checked in the *Unit Converter* application:

1) String: The string resource values are stored in the *strings.xml* file with XML. To validate them, it must confirm the availability of each string and the correctness of its value.
2) Color: The color resource values are stored in *colors.xml* file with XML. Likewise, it is essential to check the availability of each color and the correctness of its value.
3) String-array: The string-array resource values are stored in *strings.xml* file with XML. To validate them, it is necessary to check the availability of each string-array and the correctness of its value.
4) Font: The font resource files are stored in the *font* folder under the *res* folder. To validate them, it is necessary to check the availability of each font file.
5) Image: The image resource files are stored in the *drawable* folder under the *res* folder. Also, it is necessary to check the availability of each image file.
6) Drawable: The drawable files are stored in the *drawable* folder under the *res* folder. The drawable file is an XML file that contains the definition of the painting color or the style in the surface. Hence, it is critical to discern the availability of each drawable file.

Figure 10 shows the sample test code to validate resources.


Check elements (widgets) completeness
Check each element (widget) properties value

```java
int expected = elements.size();
int actual = layout.getChildCount();
assertTrue("Layout elements number was wrong",expected<=actual);

for (int idx=0; idx<elements.size(); idx++) {
    assertEquals("Element "+idx+" is not suitable",
            elements.get(idx),layout.getChildAt(idx).getClass());
}
```

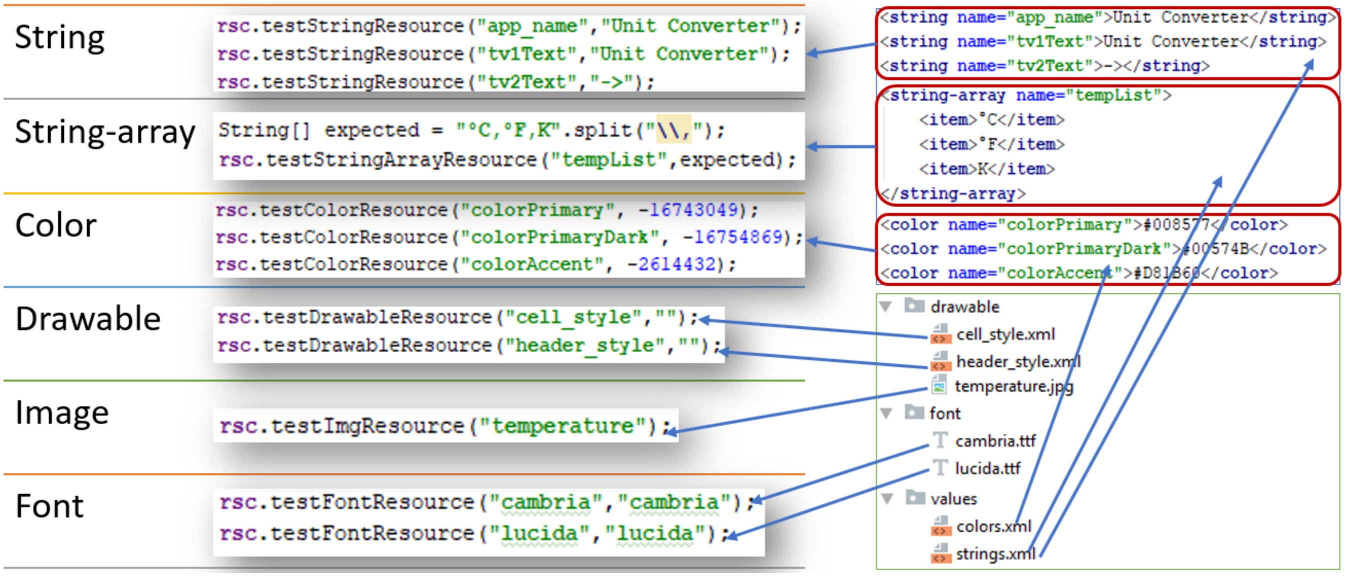
```java
@Test
public void check_01_Space_Properties() { //Check Element Specs
    //Component properties value
    int compIdx = 2;
    Space component = (Space) layout.getChildAt(compIdx);
    ElementTest comp = new ElementTest(component);

    //Test each item
    comp.testWidth(ViewGroup.LayoutParams.MATCH_PARENT);
    comp.testHeight(11);
    comp.testBgColor(-16728065);
}
```

```xml
<TextView
    android:id="@+id/myTextView1"
    android:layout_width="match_parent"
    android:layout_height="30pt"
    android:text="Unit Converter"
    android:fontFamily="@font/cambria"
    android:gravity="center"
    android:textSize="14pt"
    android:textStyle="bold"
    android:textColor="@color/textColor"/>

<Button
    android:id="@+id/convertButton"
    android:layout_width="match_parent"
    android:layout_height="20pt"
    android:layout_below="@+id/myTextView1"
    android:background="@color/btnColor"
    android:fontFamily="@font/lucida"
    android:text="Convert"
    android:textColor="@color/textColor"
    android:textSize="11pt" />
<Space
    android:id="@+id/space1"
    android:layout_width="match_parent"
    android:layout_height="5pt"
    android:background="#00BFFF"/>
```

Fig. 9. Sample test code to validate layout.



```java
rsc.testStringResource("app_name","Unit Converter");
rsc.testStringResource("tv1Text","Unit Converter");
rsc.testStringResource("tv2Text","->");

String[] expected = "°C,°F,K".split("\\,");
rsc.testStringArrayResource("tempList",expected);

rsc.testColorResource("colorPrimary", -16743049);
rsc.testColorResource("colorPrimaryDark", -16754869);
rsc.testColorResource("colorAccent", -2614432);

rsc.testDrawableResource("cell_style","");
rsc.testDrawableResource("header_style","");

rsc.testImgResource("temperature");

rsc.testFontResource("cambria","cambria");
rsc.testFontResource("lucida","lucida");
```

```xml
<string name="app_name">Unit Converter</string>
<string name="tv1Text">Unit Converter</string>
<string name="tv2Text">-></string>
<string-array name="tempList">
    <item>°C</item>
    <item>°F</item>
    <item>K</item>
</string-array>
<color name="colorPrimary">#008577</color>
<color name="colorPrimaryDark">#00574B</color>
<color name="colorAccent">#D81B60</color>
```

Fig. 10. Sample test code to validate resources.

E. Java Class Validation

A Java class can contain several methods such that each method has the specific purpose and the functionality. To test the validity of each Java class, the methods are adopted through the following two steps:

1) Method structure check: The number of the methods, the name of each method, and the input/output parameters of each method must be correct.
2) Method functionality check: The source code must pass every test method in the test code.

Figure 11 shows the sample test code to validate a Java class.

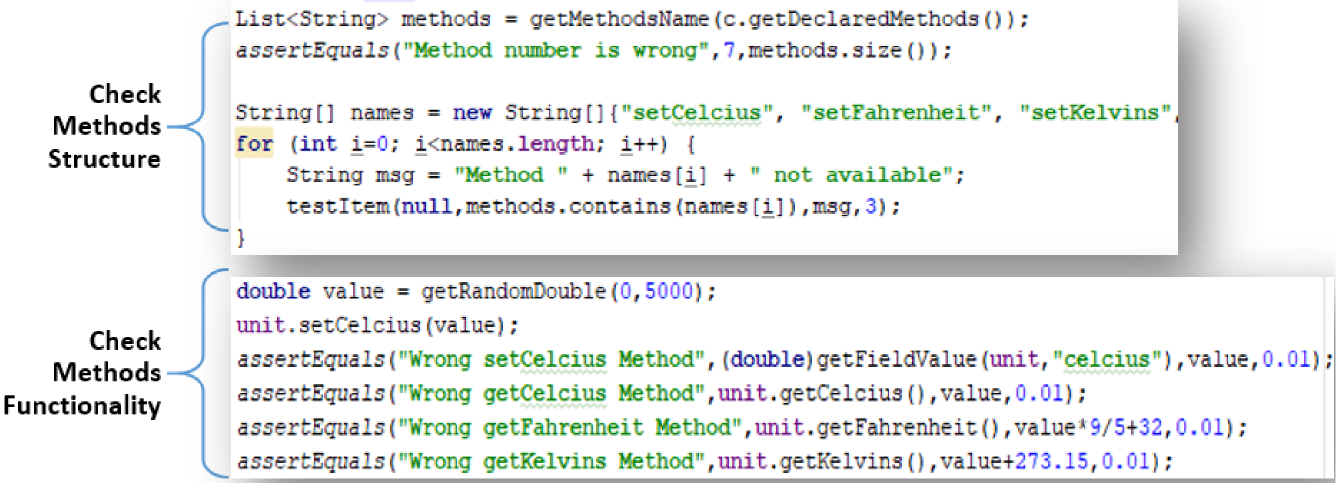
F. Activity Lifecycle Validation

Activity has a lifecycle represented by a set of states, during its entire lifetime from the initially created time until the destroyed one Android Developer Fundamentals [2019]. This process is known as the *Activity Lifecycle* on Android applications. The stages in *Activity* lifecycle consist of *onCreate*, *onStart*, *onResume*, *onPause*, *onStop*, and *onDestroy*. *Robolectric* can test whether an application enters in one step in the cycle and moves to the next step, using the code in Figure 12.

On top of that, *Activity* is a Java class making an application to be interactive. The *Activity* class has the defined methods and fields. To validate *Activity* in the project, it is necessary to check the methods and fields that are defined.

G. Event Listener Validation

Event listener is an interface in the *View* class that contains

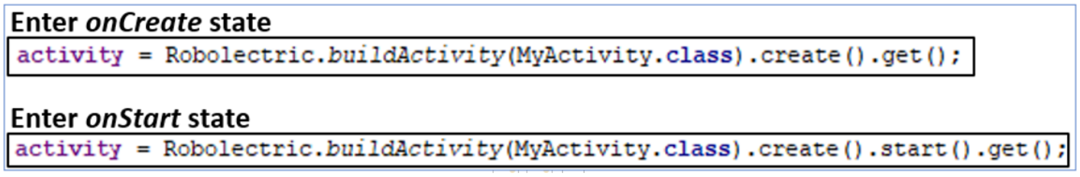


```
List<String> methods = getMethodsName(c.getDeclaredMethods());
assertEquals("Method number is wrong",7,methods.size());

String[] names = new String[]{"setCelcius", "setFahrenheit", "setKelvins",
for (int i=0; i<names.length; i++) {
    String msg = "Method " + names[i] + " not available";
    testItem(null,methods.contains(names[i]),msg,3);
}
```

```
double value = getRandomDouble(0,5000);
unit.setCelcius(value);
assertEquals("Wrong setCelcius Method",(double)getFieldValue(unit,"celcius"),value,0.01);
assertEquals("Wrong getCelcius Method",unit.getCelcius(),value,0.01);
assertEquals("Wrong getFahrenheit Method",unit.getFahrenheit(),value*9/5+32,0.01);
assertEquals("Wrong getKelvins Method",unit.getKelvins(),value+273.15,0.01);
```

Fig. 11. Sample test code to validate Java class.



```
activity = Robolectric.buildActivity(MyActivity.class).create().get();
```

```
activity = Robolectric.buildActivity(MyActivity.class).create().start().get();
```

Fig. 12. Sample test code to validate Activity lifecycle.

a single callback method Udacity [2019]. This method will be called by the Android framework if the registered *View* with the listener is triggered by a user interaction of the *User Interface*. The elements in the layout that have *event listener* are *Button*, *Spinner*, *RadioButton*, and *Checkbox*. The methods in the *event listener* interface, such as*onClick()*, *onItemSelected()*, *onFocusChange()*, and *onCheckedChange()*, are used here.

The validation of *Event listener* is processed through the following steps:

1) Make an event for a widget: A number of widgets, *Button*, *Spinner*, and *RadioButton*, can generate an event. In the test code, *Button* can call the *performClick()* method to generate an *onClick()* event, *Spinner* can call the *setSelection()* method to generate an *onItemSelected()* event, and *RadioButton* can call the *setChecked()* method to generate an *onCheckedChange()* event.
2) Check the Result: After an event is generated, the result of the event is compared with the correct value using the test code, as shown in Figure 13.

VII. Evaluation for *Basic UI*

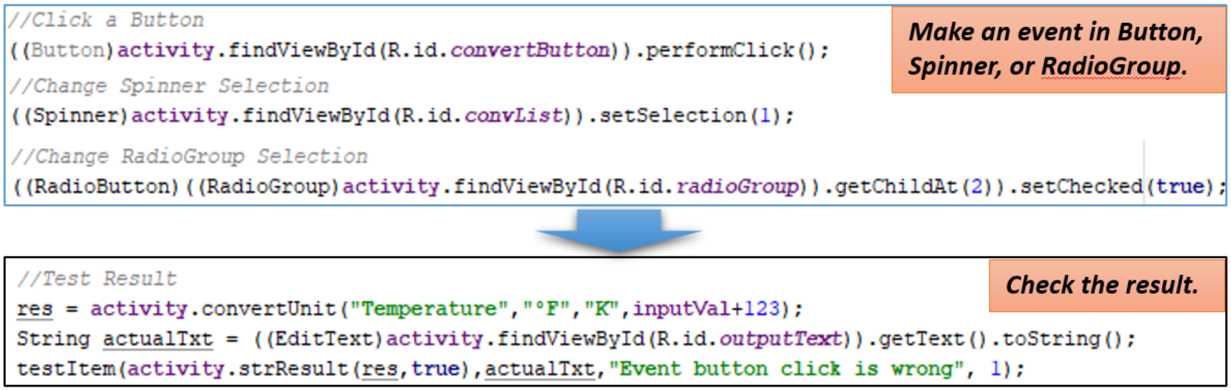
In this paper, we evaluate the APLAS implementation for *Basic Application* learning through applications to 40 undergraduate students majoring in IT, with *Basic UI* and *Basic Activity*) topics. In this section, we will discuss the evaluation results for the *Basic UI* topic. In the next section, we will discuss the evaluation results for the *Basic Activity* topic.

A. *Solving Activity Results*

For evaluations, we have appointed 40 undergraduate students majoring in IT to solve the assignment using APLAS on *Android Studio* Android Studio [2019]. Each student used their own computer in the Android programming class. The computer adopted *Windows 10* OS, *Intel Core I5-3470 3.2GHz* CPU, and *12GB* RAM, where *JUnit* spends $1-2$ milliseconds and *Robolectric* does $6-8$ seconds to mark one answer. During the class, the required time and the feedback from APLAS for each task were recorded. Table III summarizes the results.

TABLE III
SOLVING ACTIVITY RESULTS FOR *Basic UI*

task no.	1	2	3	4	5	6	7	8	9
# of passed	39	39	40	38	36	36	36	36	35
# of failed	1	0	0	1	2	0	0	0	1
# of skipped	0	1	1	1	2	4	4	4	4
# of passed under 15 minutes	3	3	4	7	6	8	6	7	2
# of passed under 30 minutes	20	24	24	30	27	28	28	29	17
average time (minutes)	26	26	29	19	23	21	20	20	31
shortest time (minutes)	10	11	8	4	7	3	5	3	4
longest time (minutes)	55	85	95	40	45	45	40	35	55



```
//Click a Button
((Button)activity.findViewById(R.id.convertButton)).performClick();
//Change Spinner Selection
((Spinner)activity.findViewById(R.id.convList)).setSelection(1);
//Change RadioGroup Selection
((RadioButton)((RadioGroup)activity.findViewById(R.id.radioGroup)).getChildAt(2)).setChecked(true);
```

Make an event in Button, Spinner, or RadioGroup.

```
//Test Result
res = activity.convertUnit("Temperature","°F","K",inputVal+123);
String actualTxt = ((EditText)activity.findViewById(R.id.outputText)).getText().toString();
testItem(activity.strResult(res,true),actualTxt,"Event button click is wrong", 1);
```

Check the result.

Fig. 13. Sample test code to validate event.

B. *Discussions on Solving Activity Results*

Table III indicates that 35 among the 40 students (87.5%) successfully completed the given nine tasks. On the other hand, five students (12.5%) could not complete any task. One student among them failed *task 1*, by failing to synchronize *gradle* due to the Internet connection failure. Because all the tasks must be done sequentially, the failed student needs to skip all the remaining tasks. One student failed *task 4* and two students failed *task 5*, which are related to the UI design. One student failed *task 9*, which is related to making the table layout.

The required time to complete one task is distributed from 19 to 31 minutes (24 minutes on average). The shortest time is at $3 - 11$ minutes. The longest time is from 35 to 95 minutes. The improvement in the completion time was observed at *task 4*, where the longest time was reduced to 40 minutes.

C. *Task Difficulty Analysis*

The difficulty level of each task can be analyzed depending on the number of students who could complete each task in less than 30 minutes and in less than 15 minutes respectively. Afterwards, the difficulty level of the tasks was categorized into four levels in Table IV.

TABLE IV
TASK DIFFICULTY LEVEL CLASSIFICATION FOR *Basic UI*

level	# of passed under 15 minutes	# of passed under 30 minutes	tasks no.
very hard	0-2	0-19	9
hard	3-4	20-24	1,2,3
medium	5-6	25-27	5
easy	7-8	28-30	4,6,7,8

D. *Feedbacks from Students*

After solving the assignment, we requested each student to give an opinion on the problems for each task. Figure 14 shows the distribution of their opinions.

First, most of the students gave the positive opinion *Easy to do* on this topic. However, five students gave the negative opinion for any task including *task 9* that was classified as *very hard*. Then, we analyzed the reasons of the failures in the four tasks. *task 1* failed due to the Internet connection error. *Android Studio* could not synchronize *gradle*. *task 4* and *task 5* failed due to the insufficient knowledge of XML. As well, *task 9* failed as a result of the insufficient knowledge on the table structure in the database. In addition, most of the students suffered from the specification insufficiency of hardware, which caused problems in loading and executions.

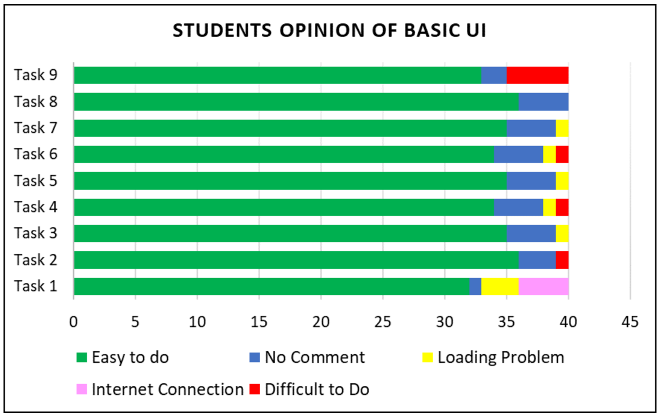


Fig. 14. Opinions for *Basic UI* topic.

VIII. EVALUATION FOR *Basic Activity*

In this section, we discuss the evaluation results for the *Basic Activity* topic.

A. *Result of* Basic Activity

For evaluations of this topic, we have also appointed 40 undergraduate students majoring in IT. Once more, each student used the own computer that has the same specifications in the Android programming class, and the required time and the feedback from APLAS for each task were recorded. Table V summarizes the results.

B. *Discussions on Solving Activity Results*

Table V indicates that 36 among the 40 students (90%) successfully completed all the tasks in sequence. Beyond that,

TABLE V
SOLVING ACTIVITY RESULTS FOR *Basic Activity*

task no.	1	2	3	4	5	6	7	8	9
# of passed	40	40	40	38	37	37	37	37	36
# of failed	0	0	0	1	2	0	0	0	1
# of skipped	0	0	0	0	1	3	3	3	3
# of passed under 5 minutes	8	7	8	7	6	4	6	4	4
# of passed under 20 minutes	28	28	28	29	23	27	24	27	19
average time (minutes)	22	22	28	21	17	16	16	16	19
shortest time (minutes)	3	4	4	4	4	3	4	4	4
longest time (minutes)	90	105	105	100	55	52	45	57	61

there are four students who could not complete any task. One student failed *task 4*, a stage which a student begins programming the *Activity* with defining several fields and methods. Two students failed *task 5*, which is for learning *Activity Lifecycle*. One student failed *task 9*, which is for learning how to process *ImageView*.

The required time to complete one task is distributed from 16 to 28 minutes (20 minutes on average). The shortest time is from 3 to 4 minutes. The longest time is at $90 - 105$ minutes. The improvement in the completion time was observed at *task 5*, where the longest time was reduced to 55 minutes. It is noted that three students did not pass *task 4* and *task 5*, and their time was not recorded.

C. Task Difficulty Analysis

For *Basic Activity*, the difficulty level of each task can be analyzed depending on the number of students who could complete each task in less than 20 minutes or less than 5 minutes. The number of students who completed a task under 5 minutes appears to be fewer, which indicates the level of difficulty of the task becomes higher. Also, the difficulty level of the tasks can be categorized into the three levels in Table VI.

TABLE VI
TASK DIFFICULTY LEVEL CLASSIFICATION FOR *Basic Activity*

level	# of passed under 5 minutes	# of passed under 20 minutes	tasks no.
hard	0-4	0-20	9
medium	4-6	21-27	5,6,7,8
easy	7-8	28-30	1,2,3,4

D. Feedbacks from Students

Figure 15 shows the distribution of the opinions from the students. Most of the students gave positive opinions on learning *Basic Activity* topic with APLAS. Only five students expressed negative feedback that this learning was not easy. All of them failed on *task 4*, *task 5*, and *task 9*, where they faced the most difficult task of creating *event listener* with an image file. The improvements will be in future studies.

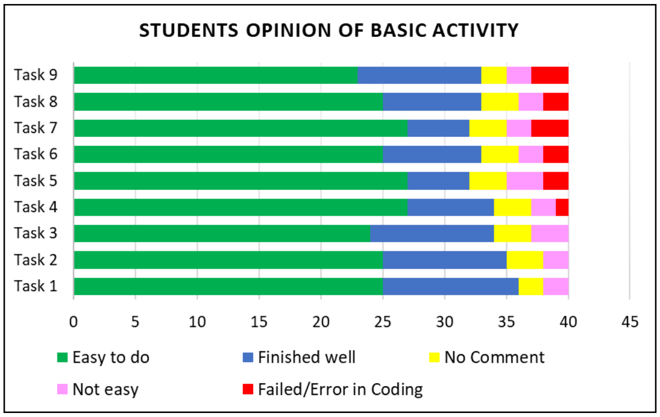


Fig. 15. Opinions for *Basic Activity* topic.

IX. CONCLUSION

In this paper, we proposed the *Android Programming Learning Assistance System (APLAS)* to assist the Java-based Android programming study and education through four stages. Based on the *Test-Driven Development (TDD)* method using *JUnit* and *Robolectric*, the answers from the students are automatically marked in APLAS. The design and implementation for learning *Basic Application* were presented to cover the first two topics, where the application results in the *Unit Converter* assignment confirms the effectiveness of APLAS. In future works, we will continue the implementation of APLAS for the remaining stages, including the functions for tracking failed tests and online learning, and the handbook for teachers to guide students to use APLAS.

ACKNOWLEDGMENT

We are very grateful to Mr. Pramana Yoga Saputra and Mr. Rudy Ariyanto from State Polytechnic of Malang, and Mrs. Inggriani Liem from Del Institute of Technology, for great helps in directing the students to solve the first APLAS assignment for evaluations.

REFERENCES

Android Open Source Project, (2019). Dalvik bytecode. *Android Open Source Project (AOSP) repository (Online).* Retrieved from https://source.android.com/devices/tech/dalvik/dalvik-bytecode.

AppBrain (2019, March). *Number of Android apps on Google Play.* Retrieved from https://www.appbrain.com/stats/number-of-android-apps.

Blundell, P., & Milano, D., T., 2015. *Learning android application testing*. Packt Publishing.

Chandra, T., N., & Liem, I. (2013). Source code editing evaluator for learning programming. *Proceeding of 4th International Conference on Electrical Engineering and Informatics (ICEEI 2013), Kuala Lumpur*, 169-175.

Eclipse Foundation. (2019). *Eclipse IDE*. Retrieved from https://www.eclipse.org.

Farcic, V., & Garcia, A. (2015). *Test-driven Java development*. Packt Publishing.

Funabiki, N., Matsushima, Y., Nakanishi, T., Watanabe, K., & Amano, N. (2013, February). A Java programming learning assistant system using test-driven development method. *IAENG International Journal of Computer Science*, vol. 40, no. 1, 38-46.

Google Developers (2019). *Android Studio*. Retrieved from https://developer.android.com/studio/.

Google Developers, (2019). Android Developer Fundamentals. *Google developers training course*. Retrieved from https://developer.android.com/courses/fundamentals-training/overview-v2.

García, P., G., F., & Rosa, F., D., (2016). RoBlock – Web App for Programming Learning. *International Journal of Emerging Technologies in Learning (iJET)*, Vol 11, no.12, 45-53.

Google Developers, (2019). Espresso. *Google Developers Documentation (Online)*. Retrieved from https://developer.android.com/training/testing/espresso.

Google Developers, (2019). Fundamental of testing. *Google Developers Documentation (Online)*. Retrieved from https://developer.android.com/training/testing/fundamentals.

Google Developers, (2019). Google training courses. *Google developers training course*. Retrieved from https://developer.android.com/courses/.

Google Developers, (2019). UI Automator. *Google Developers Documentation (Online)*. Retrieved from https://developer.android.com/training/testing/ui-automator.

Google Inc. (2019). *Google Play Store (online)*. Retrieved from https://play.google.com/store.

Gradle Inc. (2019). *Gradle Build Tool*. Retrieved from https://gradle.org/.

Horton, J., (2015). *Android programming for beginners*. Packt Publishing.

Hussain, A., Razak, H., A., & Mkpojiogu, E., O., (2017, April). The perceived usability of automated testing tools for mobile applications. *Journal of Engineering, Science, and Technology*, Special Issue on ISSC ' 2016, 86-93.

JUnit (2019). *JUnit*, Retrieved from https://junit.org/junit4/.

Kang, H., & Cho, J., (2015). Case study on efficient Android programming education using multi Android development tools. *Indian Journal of Science and Technology*, vol. 8, no. 19, 1-5.

Kaur, A., (2015). Review of Mobile Applications Testing with Automated Techniques. *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, no. 10, 503-507.

Kim, H., K., (2013, October). Test Driven Mobile Applications Development. *Proceedings of the World Congress on Engineering and Computer Science 2013, San Fransisco, USA*, vol. 2.

Kochhar, P., S., Thung, F., Nagappan, N., Zimmermann, T., & Lo, D., (2015, April). Understanding the Test Automation Culture of App Developers. *Proceeding of 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), Graz, Austria.*

Koskela, L. (2008). *Test Driven: Practical TDD and acceptance TDD for Java developer*. Manning Publications.

Luccio, F., L., (2019). Learning distributed algorithms by programming robots. *Journal of e-Learning and Knowledge Society*, Vol. 15, No. 2, 89–100.

Massachusetts Institute of Technolgy, (2019). *MIT App Inventor*. Retrieved from https://appinventor.mit.edu.

Mockito framework site (2019). *Mockito - Tasty mocking framework for unit tests in Java*. Retrieved from https://site.mockito.org/.

Ortiz, 0., Alcover, P., M., Sánchez, F., Pastor, J., A., & Herrero, R., (2015). M-Learning Tools: The Development of Programming Skills in Engineering Degrees. *IEEE Journal of Latin-American Learning Technologies*, Vol. 10, No. 3, 86–91.

Oracle Corporation (2019). *Java programming language*. Retrieved from https://docs.oracle.com/javase/7/docs/technotes/guides/language/.

JDK8 Oracle Corporation, 2019. *Open JDK 8*, Retrieved from https://openjdk.java.net/projects/jdk8u/.

Powermock framework site (2019). *Powermock*, Retrieved from https://powermock.github.io/.

Rekhawi, H., A., A., & Naser, S., S., A., (2018). An intelligent tutoring system for learning Android applications UI development. *International Journal of Engineering and Information System*, vol. 2, no. 1, 1-14.

Robins, A., Rountree, J., Rountree, N., (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, Vol. 13, No. 2, 137–172 .

Robolectric (2019). *Robolectric*. Retrieved from http://robolectric.org/.

Robotium (2019). *RobotiumTech/robotium: Android UI testing*. Retrieved from https://github.com/RobotiumTech/robotium/.

Sadeh, B. & Gopalakrishnan, S. (2011). A study on the evaluation of unit testing for Android systems. *International Journal on New Computer Architectures and Their Applications*, 1(4), 926-941.

SQLite Consortium (2019). *SQLite*. Retrieved from https://www.sqlite.org/index.html.

StatCounter GlobalStats (2019, February). *Operating system market share worldwide*. Retrieved from http://gs.statcounter.com/os-market-share.

Statista GmbH (2019, March). *Number of smartphone users worldwide from 2014 to 2020 (in billions)*. Retrieved from https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/.

Statista GmbH (2019). *App Developers - statistics & facts*. Retrieved from https://www.statista.com/topics/1694/app-developers/.

Syaifudin, Y., W., Funabiki, N., & Kuribayashi, M. (2019, March). Learning model for Android programming learning assistant system. *Proceeding of 2019 IEICE General*

Conference, Waseda University, Tokyo, S89-S90.

Udacity Inc., 2019. Android basics by Google. *Udacity nanodegree program*. Retrieved from https://www.udacity.com/course/android-basics-nanodegree-by-google–nd803.

W3C (2019). *Extensible Markup Language (XML)*. Retrieved from https://www.w3.org/XML/.

Vásquez, M., L., Cardenas, C., B., Moran, K., & Poshyvanyk, D., (2017, September). How do Developers Test Android Applications?. *Proceeding of 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, China*.

Wikimedia Foundation Inc., 2019. Test-driven development. *Wikipedia - The Free Encyclopedia (Online)*. Retrieved from https://en.wikipedia.org/wiki/Test-driven_development.

Yan Watequlis Syaifudin received the bachelor degree in Informatics from Bandung Institute of Technology, Indonesia, in 2003, and the master degree in Information Technology from Sepuluh Nopember Institute of Technology, Surabaya, Indonesia, in 2011, respectively. In 2005, he joined State Polytechnic of Malang, Indonesia as a lecturer. He is currently a Ph.D. candidate in Graduate School of Natural Science and Technology at Okayama University, Japan. His research interests include educational technology and database systems. He is a student member of IEICE.

Nobuo Funabiki received the B.S. and Ph.D. degrees in mathematical engineering and information physics from the University of Tokyo, Japan, in 1984 and 1993, respectively. He received the M.S. degree in electrical engineering from Case Western Reserve University, USA, in 1991. From 1984 to 1994, he was with Sumitomo Metal Industries, Ltd., Japan. In 1994, he joined the Department of Information and Computer Sciences at Osaka University, Japan, as an assistant professor, and became an associate professor in 1995. In 2001, he moved to the Department of Communication Network Engineering (currently, Department of Electrical and Communication Engineering) at Okayama University as a professor. He was the chairman at IEEE Hiroshima section in 2015 and 2016. His research interests include computer networks, optimization algorithms, educational technology, and Web technology. He is a member of IEEE, IEICE, and IPSJ.

Minoru Kuribayashi received his B.E., M.E., and D.E. degrees from Kobe University, Kobe, Japan, in 1999, 2001, and 2004. From 2002 to 2007, he was a research associate at the Department of Electrical and Electronic Engineering, Kobe University. In 2007, he was appointed as an assistant professor at the Division of Electrical and Electronic Engineering, Kobe University. Since 2015, he has been an associate professor in the Graduate School of Natural Science and Technology, Okayama University. His research interests include digital watermarking, information security, cryptography, and coding theory. He received the young professionals award from IEEE Kansai Section in 2014. He is a senior member of IEEE and IEICE.

Wen-Chung Kao received the M.S. and Ph.D. degrees in electrical engineering from National Taiwan University, Taiwan, in 1992 and 1996, respectively. From 1996 to 2000, he was a Department Manager at SoC Technology Center, ERSO, ITRI, Taiwan. From 2000 to 2004, he was an Assistant Vice President at NuCam Corporation in Foxlink Group, Taiwan. Since 2004, he has been with National Taiwan Normal University, Taipei, Taiwan, where he is currently a Professor at Department of Electrical Engineering and the Dean of School of Continuing Education. His current research interests include system-on-a-chip (SoC), flexible electrophoretic display, machine vision system, digital camera system, and color imaging science. He is a fellow of IEEE.