

**Implementation and evaluation of a tool for translating
Layout-based to Visual android tests**

Chaudhary Robin

Master Degree in
Communication and Computer Network Engineering



Politecnico di Torino
Academic Year: A.Y. 2019-2020

Supervisors:
ARDITO LUCA
COPPOLA RICCARDO

Contents

1	INTRODUCTION	5
1.1	Mobile Application Testing	5
1.1.1	Mobile Application Development Life cycle	5
1.1.2	Types of Mobile Application Testing	6
1.1.3	Types of Testing Tools	6
1.2	Motivation	7
1.3	Layout Based to Visual based Translation of Test Scripts	8
1.3.1	Creating Layout Based Test Cases:	8
1.3.2	Data Collection:	8
1.3.3	Translation Program Implementation:	8
1.3.4	Data Generation for Visual Test Cases:	8
1.3.5	Execution of Visual Test Cases:	9
2	BACKGROUND	9
2.1	ANDROID APP STRUCTURE	9
2.2	ANDROID APPLICATION TESTING	13
2.2.1	Layout Based Testing:	14
2.2.2	Visual GUI Based Testing	15
2.3	ANDROID TESTING ISSUES	16
3	TOOL ARCHITECTURE	20
3.0.1	Layout Based Test Script Creation:	21
3.0.2	Parser Program:	21
3.0.3	Update Layout Test Cases:	23
3.0.4	Capturing Screenshots:	23
3.0.5	Resize Screenshots:	24
3.0.6	Action List Extraction:	24
3.0.7	Convert to GUI based Keywords:	25
3.0.8	Creating Visual Test Cases:	26
3.0.9	Executing GUI Test Cases:	27
4	EVALUATION	27
4.1	Definition of the Experiment:	27
4.2	Interaction Log Data:	28
4.3	Attempts for Translation	30
4.4	Issues faced in Translation:	31
4.5	Time Taken For Translation:	34
4.6	Execution Attempts:	35
4.7	Test Suites and Apps Used:	35
4.8	Results:	39
4.8.1	Omni Notes Test Script Example	40
4.8.2	Calculator App Test Script Example	40
4.8.3	Unit Converter Test Script Example	41

5	Threats to Validity:	42
5.1	Threats to External Validity:	42
5.2	Threats to Internal Validity:	42
5.3	Threats to Construction Validity:	43
6	CONCLUSION	43
6.1	Conclusion of Experiment	43
6.2	Expected Future Work	44
	Appendices	45
A	Omni Notes Application	45
A.A	Omni Notes Text Script 1	45
A.B	Omni Notes Text Script 2	46
A.C	Omni Notes Text Script 3	46
A.D	Omni Notes Text Script 4	47
A.E	Omni Notes Text Script 5	48
A.F	Omni Notes Text Script 6	48
A.G	Omni Notes Text Script 7	49
A.H	Omni Notes Text Script 8	50
A.I	Omni Notes Text Script 9	51
A.J	Omni Notes Text Script 10	51
B	Calculator Application	52
B.A	Calculator App Test Script 1	52
B.B	Calculator App Test Script 2	53
B.C	Calculator App Test Script 3	54
B.D	Calculator App Test Script 4	55
B.E	Calculator App Test Script 5	56
B.F	Calculator App Test Script 6	57
B.G	Calculator App Test Script 7	58
B.H	Calculator App Test Script 8	59
B.I	Calculator App Test Script 9	60
B.J	Calculator App Test Script 10	61
C	Calculator Application	62
C.A	UnitConverter App Test Script 1	62
C.B	UnitConverter App Test Script 2	63
C.C	UnitConverter App Test Script 3	64
C.D	UnitConverter App Test Script 4	65
C.E	UnitConverter App Test Script 5	66
C.F	UnitConverter App Test Script 6	67
C.G	UnitConverter App Test Script 7	68
C.H	UnitConverter App Test Script 8	69
C.I	UnitConverter App Test Script 9	70
C.J	UnitConverter App Test Script 10	71

List of Figures

1	Block Diagram for Translation	8
2	Android App Structure	10
3	Layout to Visual Tool Architecture	20
4	Sikuli Input Text File	26
5	Sikuli Tool UI	27
6	Omni Notes Interaction Table	28
7	Calculator App Interaction Table	29
8	Unit Converter Interaction Table	29
9	Attempts for Translation Table	30
10	Discarded Image 1	31
11	Accepted Image 1	32
12	Discarded Image 2	32
13	Accepted Image 2	32
14	Discarded Screenshots By Apps	33
15	Similar Image Issue	34
16	Execution Attempts Passed	35
17	Omni Notes Application UI	37
18	Calculator Application UI	38
19	Unit Converter Application UI	39

ABSTRACT

Context: Mobile testing tools can be categorized in three classes, which are first generation (coordinate based), second generation (layout based) or third generation (visual based). All of them have their own advantages and disadvantages, but these shortcomings could be overcome by using a mixed approach where automated scripts can be translated from one generation to another one. Using a mixed approach for testing will improve the productivity of test scripts development, and mitigate the issues like fragility and maintainability. Since, some graphical aspects of the application could result in failure while performing visual GUI based testing but could be recovered with layout based ones. There is a possibility to create Visual scripts from existing available layout based test scripts

Goal: The aim of this work is to implement a tool that can automatically translate second generation layout based test scripts in the third generation visual GUI based test scripts which may be Sikuli or Eyeautomate

Method: A software project in Eclipse IDE is created using Java language with support of Selenium framework to achieve the above mentioned goal and then an experiment was conducted, to evaluate the success rate of the translated scripts. The experiment consisted in translating layout test scripts, developed for three open source mobile applications (Omni Notes, Calculator and Unit Converter Application), into Sikuli GUI based scripts.

Results: The translation was completed with very high success rate for all the test cases for all three applications (30 out of 30 test cases successfully translated), while the execution of those scripts showed less success rate comparatively

Conclusion: The experiment concluded that the translation from second generation (layout based) to third generation (visual based) test scripts is feasible in the mobile domain, and it reduces the efforts required for generating visual-based test cases. However more work is required to extend the proposed tool and experiment it in industrial environment, as well as to measure its capability of reducing fragility and maintenance effort for visual-based test suites.

1 INTRODUCTION

The experiment is the translation of layout based test scripts into GUI based test scripts. It was performed with help of different tools and using different concepts related to mobile testing. Before explaining about the actual experiment performed, we need to understand the concept of mobile testing in general and why there is a need to perform this experiment and how it is may be useful for the industrial use. This chapter introduces the project with its prerequisite knowledge and provides the motivation

1.1 Mobile Application Testing

Mobile application testing is a process by which application software developed for handheld mobile devices is tested for its functionality, usability and consistency. Mobile application testing can be an automated or manual type of testing. Mobile applications either come pre-installed or can be installed from mobile software distribution platforms.

1.1.1 Mobile Application Development Life cycle

There are different phases in development of a mobile application:

1. **Planning** - Planning is one of the most important aspects of every development it is the creation of mobile apps. one need to know answer of questions like, aim of the mobile application, kind of audience it will target, time required to develop the application and finally, platform on which the app will be developed
2. **Design** - It is advised to develop the designs and the mobile app development methodology (agile or waterfall, as per the requirement) in this phase only. Prototypes are the early sketches that help determine the flow of mobile app development in respect to app creation and designing.
3. **Coding** - The coding of an application is the most important stage as it involves all the technicalities to address.
4. **Testing** - It is important to identify all the bugs prior to the official app release. Apps with lots of bugs is never going to create the kind of impression that is required. Therefore, it is essential to test the application before its official release.
5. **App Launch** - It is the final stage in the entire mobile app development procedure. As your application is ready to be launched, it is now time for you to register it over different App Stores. Besides the two main App Stores that we know as Google Play and Apple App Store, mobile apps can also be stored on a range of other platforms too including the Amazon App Store.

1.1.2 Types of Mobile Application Testing

There are different types of mobile application testing:

- **Functional testing** - It ensures that the application is working as per the requirements. Most of the tests conducted for this is driven by the user interface and call flow.
- **Laboratory Testing** - It is usually carried out by network carriers, is done by simulating the complete wireless network. This test is performed to find out any glitches when a mobile application uses voice and/or data connection to perform some functions.
- **Performance Testing** - It is undertaken to check the performance and behavior of the application under certain conditions such as low battery, bad network coverage, low available memory, simultaneous access to the application's server by several users and other conditions. Performance of an application can be affected from two sides: the application's server side and client's side. Performance testing is carried out to check both.
- **Memory leakage testing** - Memory leakage happens when a computer program or application is unable to manage the memory it is allocated resulting in poor performance of the application and the overall slowdown of the system. As mobile devices have significant constraints of available memory, memory leakage testing is crucial for the proper functioning of an application
- **Usability Testing** - It is carried out to verify if the application is achieving its goals and getting a favorable response from users. This is important as the usability of an application is its key to commercial success (it is nothing but user friendliness). Another important part of usability testing is to make sure that the user experience is uniform across all devices.
- **Load Testing** - When many users all attempt to download, load, and use an app or game simultaneously, slow load times or crashes can occur causing many customers to abandon your app, game, or website. In-country human testing done manually is the most effective way to test load.
- **Crowd-sourced Testing** - A global community of testers provides easy access to different devices and platforms. A globally distributed team can also test it in multiple locations and under different network conditions.

1.1.3 Types of Testing Tools

1. **Coordinate Based Testing Tools** - These are First generation testing tools which identify elements in the layout through their coordinates. This technique is not much used due to its fragility

2. **Layout or Property Based Testing Tools** - They are also called Second generation testing tools. These allow them to identify a component based on its label or ID or a property as the text contained. This type of test is more robust of the previous one but does not check the actual aspect of the GUI and has a high maintenance cost
3. **Visual GUI Testing Tools** - These are third generation testing tools that identify elements through image recognition algorithms, which capture the actual appearance of the elements as displayed to the user.

1.2 Motivation

The pace of mobile application development is at the record high in today's time, every business wants to come on a mobile platform and wants to interact with the users. Since, mobile application testing should also develop with the same pace and most of the projects are developed with agile approach where development and testing occurs in parallel stages. Although the market is full of different mobile application testing tools. Layout based testing techniques and tools are the most in use. While visual GUI based testing tools were not so successful comparatively because of their lack of robustness and performance. Even if there is evidence of applicability, feasibility and usefulness of Visual GUI testing tools they are less commonly used than the Layout-based. But Layout-based testing tools cannot fully emulate human user as interactions through GUI properties do not verify the system's appearance as shown to the human user. Most researches aimed to compare the two techniques have concluded that a mixed approach is required where both kind of testing tools are used in parallel by the testers. The first advantage of a mixed approach is that it allows re-usability of already existing Visual tests suites. This would reduce the development effort for the test scripts and increase the productivity. Also, the layout based test scripts can be converted to visual test scripts automatically, and that too without manually recapturing the images. This is what we have tried to achieve in this experiment. By collecting screenshots of particular elements on the screen, which are required to execute the layout based test script, while same screenshots are passed on to GUI based tool for execution. This way, it would help to mitigate the fragility [1] issues also, since layout based testing is sensitive to code changes and visual testing is sensible to GUI changes. Also, if the layout based test cases does not work, then it is possible to recover them from visual test cases and vice versa, which reduces the maintenance cost also.

1.3 Layout Based to Visual based Translation of Test Scripts

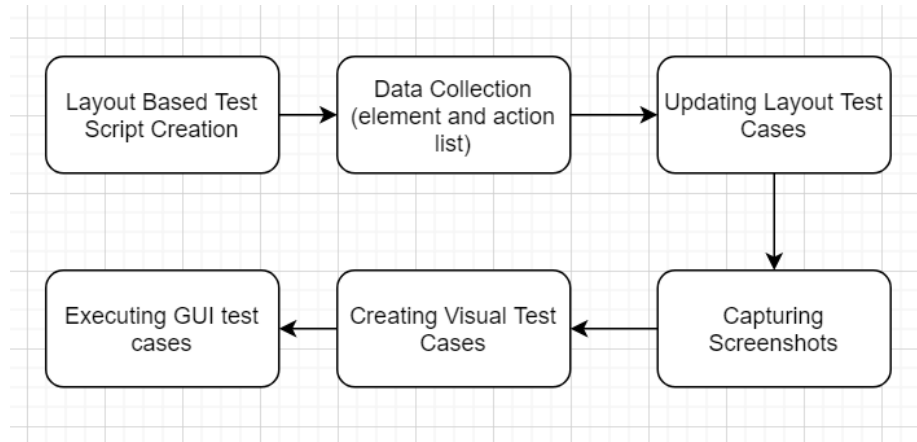


Figure 1: Block Diagram for Translation

1.3.1 Creating Layout Based Test Cases:

The initial stage is to create Layout based test cases. we have created our test case using java language in selenium framework using Eclipse IDE

1.3.2 Data Collection:

A parser java program taken an input file which is a layout based test script and parse through it, to collect all the necessary data from each line code which performs an interaction with the mobile application. The data collected are the locator, container, attribute, value, action performed

1.3.3 Translation Program Implementation:

A java program takes an input file which is a layout based test script, reads every line, add a functionality to take screenshot of the element under execution through a function, and writes the updated file as another java file.

1.3.4 Data Generation for Visual Test Cases:

The java file created in previous step is run for every layout based updated test script one by one. So, every time an element is interacted on the mobile screen, before that, the screenshot of that element is taken and saved at a location in the system. On other hand the same java program goes through the layout test case and collect all the action performed on all the elements in the test case, to create an updated text file for Visual text case.

1.3.5 Execution of Visual Test Cases:

The screenshots collected in previous step and text file generated, is used as input for the visual test cases. These GUI based test cases are then executed using visual testing tool - Sikuli

2 BACKGROUND

In this section, we are going to discuss about a generic android application structure. We are discussing the structure with the help of a mobile application development tool, which is Android Studio. We will also discuss about the Android Application testing techniques available and about the issues present and how they can be mitigated.

2.1 ANDROID APP STRUCTURE

Below are some important files/folders, and their significance is explained for the easy understanding of the Android studio work environment.

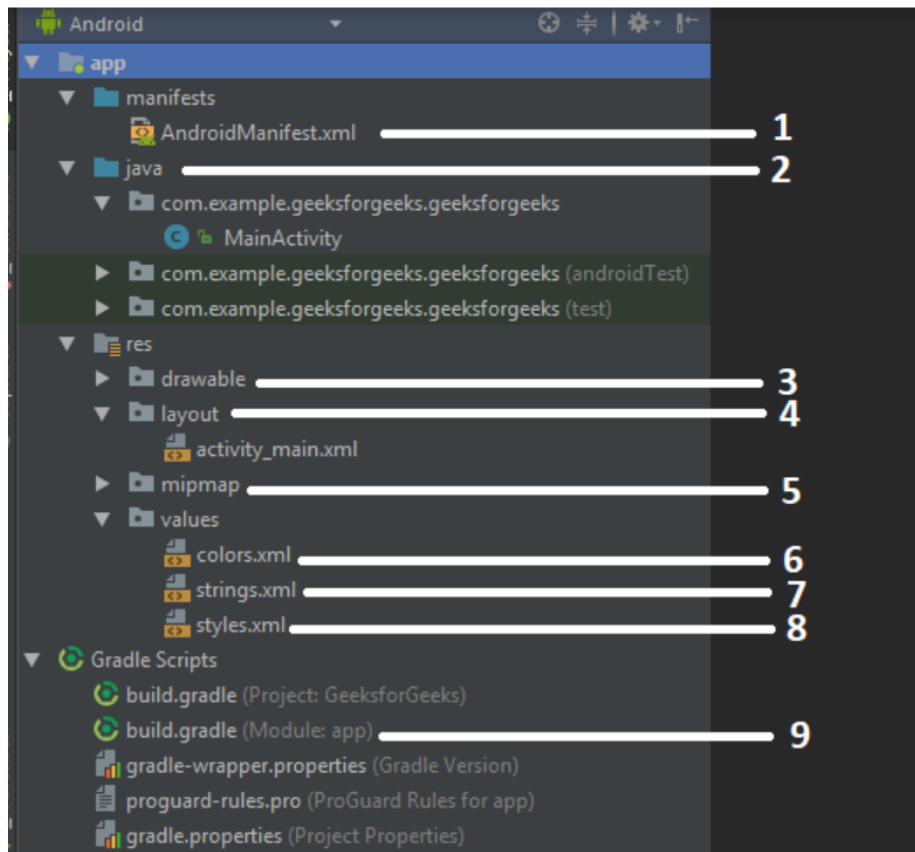


Figure 2: Android App Structure

1. **AndroidManifest.xml:** Every project in Android includes a manifest file, which is `AndroidManifest.xml`, stored in the root directory of its project hierarchy. The manifest file is an important part of our app because it defines the structure and metadata of our application, its components, and its requirements. This file includes nodes for each of the Activities, Services, Content Providers and Broadcast Receiver that make the application and using Intent Filters and Permissions, determines how they co-ordinate with each other and other applications. A typical `AndroidManifest.xml` file looks like:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.geeksforgeeks.geeksforgeeks">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>

```

2. **Java:** The Java folder contains the Java source code files. These files are used as a controller for controlled UI (Layout file). It gets the data from the Layout file and after processing that data output will be shown in the UI layout. It works on the back end of an Android application.
3. **drawable:** A Drawable folder contains resource type file (something that can be drawn). Drawables may take a variety of file like Bitmap (PNG, JPEG), Nine Patch, Vector (XML), Shape, Layers, States, Levels, and Scale.
4. **layout:** A layout defines the visual structure for a user interface, such as the UI for an Android application. This folder stores Layout files that are written in XML language. You can add additional layout objects or widgets as child elements to gradually build a View hierarchy that defines your layout file. Below is a sample layout file:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res-
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>

```

4. **mipmap:** Mipmap folder contains the Image Asset file that can be used in Android Studio application. You can generate the following icon types like Launcher icons, Action bar and tab icons, and Notification icons.
5. **colors.xml:** colors.xml file contains color resources of the Android application. Different color values are identified by a unique name that can be used in the Android application program. Below is a sample colors.xml file:

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>

```

6. **strings.xml:** The strings.xml file contains string resources of the Android application. The different string value is identified by a unique name that can be used in the Android application program. This file also stores string array by using XML language. Below is a sample strings.xml file:

```

<resources>
    <string name="app_name">GeeksforGeeks</string>
</resources>

```

7. **styles.xml:** The styles.xml file contains resources of the theme style in the Android application. This file is written in XML language. Below is a sample styles.xml file:

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkAc
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

8. **build.gradle(Module: app):** This defines the module-specific build configurations. Here you can add dependencies what you need in your Android application.

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 26
    defaultConfig {
        applicationId "com.example.geeksforgeeks.geeksforgeeks"
        minSdkVersion 16
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-andr
        }
    }
}
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:26.1.0'
    implementation 'com.android.support.constraint:constraint-1
    testImplementation 'junit:junit:4.12'
```

2.2 ANDROID APPLICATION TESTING

Android applications can be tested through layout or property based application

tools or visual GUI based tools. Both kind of tools has been explained below:

2.2.1 Layout Based Testing:

These are the testing tools which do the automation mobile testing based on written scripts or code using tools. The most famous tool used is Appium. Appium is an open-source tool for automating native, mobile web, and hybrid applications on iOS mobile, Android mobile, and Windows desktop platforms. Native apps are those written using the iOS, Android, or Windows SDKs. Mobile web apps are web apps accessed using a mobile browser (Appium supports Safari on iOS and Chrome or the built-in 'Browser' app on Android). Hybrid apps have a wrapper around a "webview" – a native control that enables interaction with web content. Projects like Apache Cordova or Phonegap make it easy to build apps using web technologies that are then bundled into a native wrapper, creating a hybrid app. Importantly, Appium is "cross-platform": it allows you to write tests against multiple platforms (iOS, Android, Windows), using the same API. This enables code reuse between iOS, Android, and Windows test suites. Few of the concepts which are necessary for these testing tools are:

- (a) Client/Server Architecture: Appium is at its heart a web server that exposes a REST API. It receives connections from a client, listens for commands, executes those commands on a mobile device, and responds with an HTTP response representing the result of the command execution. The fact that we have a client/server architecture opens up a lot of possibilities: we can write our test code in any language that has a http client API, but it is easier to use one of the Appium client libraries. We can put the server on a different machine than our tests are running on. We can write test code and rely on a cloud service like Sauce Labs to receive and interpret the commands.
- (b) Session: Automation is always performed in the context of a session. Clients initiate a session with a server in ways specific to each library, but they all end up sending a POST /session request to the server, with a JSON object called the 'desired capabilities' object. At this point the server will start up the automation session and respond with a session ID which is used for sending further commands.
- (c) Desired Capabilities: Desired capabilities are a set of keys and values (i.e., a map or hash) sent to the Appium server to tell the server what kind of automation session we're interested in starting up. There are also various capabilities which can modify the behavior of the server during automation. For example, we might set the platformName capability to iOS to tell Appium that we want an iOS session, rather than an Android or Windows one. Or we might set the safariAllowPopups capability to true in order to ensure that, during a Safari automation session, we're

allowed to use JavaScript to open up new windows. See the capabilities doc for the complete list of capabilities available for Appium.

- (d) Appium Server: Appium is a server written in Node.js. It can be built and installed from source or installed directly from NPM: The beta of Appium is available via NPM with `npm install -g appium@beta`. It is the development version so it might have breaking changes. Please `uninstall appium@beta` (`npm uninstall -g appium@beta`) before installing new versions in order to have a clean set of dependencies.
- (e) Appium Clients: There are client libraries (in Java, Ruby, Python, PHP, JavaScript, and C) which support Appium's extensions to the WebDriver protocol. When using Appium, you want to use these client libraries instead of your regular WebDriver client. You can view the full list of libraries [here](#).

2.2.2 Visual GUI Based Testing

There are two types of interfaces for a computer application. Command Line Interface is where you type text and computer responds to that command. GUI stands for Graphical User Interface where you interact with the computer using images rather than text. GUI testing is defined as the process of testing the system's Graphical User Interface of the Application Under Test. GUI testing involves checking the screens with the controls like menus, buttons, icons, and all types of bars - toolbar, menu bar, dialog boxes, and windows, etc. To generate a set of test cases, test designers attempt to cover all the functionality of the system and fully exercise the GUI itself. The difficulty in accomplishing this task is twofold: to deal with domain size and with sequences. In addition, the tester faces more difficulty when they have to do regression testing. The second problem is the sequencing problem. Some functionality of the system may only be accomplished with a sequence of GUI events. For example, to open a file a user may have to first click on the File Menu, then select the Open operation, use a dialog box to specify the file name, and focus the application on the newly opened window. Increasing the number of possible operations increases the sequencing problem exponentially. This can become a serious issue when the tester is creating test cases manually. Regression testing is often a challenge with GUIs as well. A GUI may change significantly, even though the underlying application does not. A test designed to follow a certain path through the GUI may then fail since a button, menu item, or dialog may have changed location or appearance. Another method of generating GUI test cases simulates a novice user. An expert user of a system tends to follow a direct and predictable path through a GUI, whereas a novice user would follow a more random path. A novice user is then likely to explore more possible states of the GUI than an expert.

There are two main activities which are need to be done during GUI based testing:

- (a) Mouse position capture: A popular method used in the CLI environment is capture/playback. Capture playback is a system where the system screen is “captured” as a bit mapped graphic at various times during system testing. This capturing allowed the tester to “play back” the testing process and compare the screens at the output phase of the test with expected screens. This validation could be automated since the screens would be identical if the case passed and different if the case failed. Using capture/playback worked quite well in the CLI world but there are significant problems when one tries to implement it on a GUI-based system. The most obvious problem one finds is that the screen in a GUI system may look different while the state of the underlying system is the same, making automated validation extremely difficult. This is because a GUI allows graphical objects to vary in appearance and placement on the screen. Fonts may be different, window colors or sizes may vary but the system output is basically the same. This would be obvious to a user, but not obvious to an automated validation system.
- (b) Event Capture: To combat this and other problems, testers have gone ‘under the hood’ and collected GUI interaction data from the underlying windowing system. By capturing the window ‘events’ into logs the interactions with the system are now in a format that is decoupled from the appearance of the GUI. Now, only the event streams are captured. There is some filtering of the event streams necessary since the streams of events are usually very detailed and most events aren’t directly relevant to the problem. This approach can be made easier by using an MVC architecture for example and making the view (i. e. the GUI here) as simple as possible while the model and the controller hold all the logic. Another approach is to use the software’s built-in technology, to use an HTML interface or a three-tier architecture that makes it also possible to better separate the user interface from the rest of the application. Another way to run tests on a GUI is to build a driver into the GUI so that commands or events can be sent to the software from another program. This method of directly sending events to and receiving events from a system is highly desirable when testing, since the input and output testing can be fully automated and user error is eliminated.

2.3 ANDROID TESTING ISSUES

There are many key challenges for mobile application testing. Some of the common issues are discussed below. While two specific issue fragility and maintainability are also discussed after them, which are important for our project in particular, because we are doing translation from one type of text scripts to another. The most common issues during mobile application testing are:

- Diversity in Mobile Platforms: There are different mobile operating systems in the market. The major ones are Android, iOS, and Windows Phone. Each operating system has its own limitations.

- Device availability: Access to the right set of devices when there is an ever-growing list of devices and operating system versions is a constant mobile application testing challenge. Access to devices can become even more challenging if testers are spread across different locations.
- Scripting: The variety of devices makes executing a test script (scripting) a key challenge. As devices differ in keystrokes, input methods, menu structure and display properties single script does not function on every device.
- Compatibility: It is necessary to test the compatibility; suppose an application can work on the high resolution and it doesn't work on the lower resolution.
- Variety of mobile devices: Mobile devices differ in screen input methods (QWERTY, touch, normal) with different hardware capabilities.

The two main issues for android application testing which are important for our project also are:

- (a) Fragility: Fragility [1] is defined as the quality of being easily broken or damaged, if we go by the dictionary meaning of the word. In terms of automated test cases it refers to non functionality of test cases, means being broken, when there is some change in the GUI of the application. Fragility of GUI tests is considered as a big problem for developers to not select GUI testing because even a small changes in the user interface may break entire test suites. Identification of test fragility in software projects manually has been proved a time consuming process and required a careful inspection of different version of the test code together with the application production code. For this reason, an automatic classification approach is used - which says, anytime a pre existent method of GUI test class is modified we assume the change is due to test fragility Fragility Metrics: With an automatic inspection of test code, information about modified methods and classes can be obtained. Based on this collected data, fragility of test suites can be approximated. The number of modified classes with modified methods can be different from the total number of modified classes in three different cases.

I When the modifications performed to the classes involve non-significant portions of code like comments, imports, declarations.

II when the changes made to the classes include only additions of test methods.

III when the modifications performed to the classes involve only removal of test methods.

Addition and removal of test methods are thought to be the result of a new functionality or a new use case of applications, since they are not considered as an evidence of fragility of test classes. on the other hand,

modifications of test methods may be strictly linked with fragility. The fragility of the tests can be estimated with two metrics based on the raw count of classes and methods modified. There is a need to mitigate the issues caused by fragility

- (b) Maintainability: Maintainability [1] is the process of testing the system's ability to update, modify the application if required. This is very important part as the system is subjected to changes all through the software life cycle. Checking of how easily it is to maintain a system is the main aim of this testing type. The system or application support involves analysis, modification, and testing the product. Categories of product maintainability are:

- I Corrective maintenance: It is about fixing the problems. The system maintaining may be specified in the context of time, spent for diagnostics and errors removal, detected in the system.
- II Perfective maintenance: This is related to the system's modifications. The maintaining may be measured by the number of efforts which are necessary for performing the required system improvements.
- III Adaptive maintenance: This includes adaptiveness to the changes in the special environment. The maintaining may be defined by the number of forces which are required for the system adaptiveness.
- IV Preventive maintenance: It discusses about the actions for reducing expenditures, needed for system support.

Maintaining the cost of testing artifacts is also an issue. Difficulties in Evolving and Maintaining GUI Scripts/Models is a concern, since, Of the available approaches for automated testing of mobile apps, test scripts recorded manually or written with automation APIs/Frameworks are the most vulnerable to app evolution and fragmentation. Generating test scripts is time consuming. As an application evolves, test scripts need to be updated when changes modify the GUI (or GUI behavior) as expected in the scripts (e.g., the id of a component is modified or a component is removed from a window). Automation APIs like Espresso allow for declaring GUI events partially decoupled from device characteristics, but, the scripts are coupled to change-prone component ids. As of today there is no current approach for automatically evolving scripts written or recorded using Automation APIs. There is a need to mitigate this issue also.

3 TOOL ARCHITECTURE

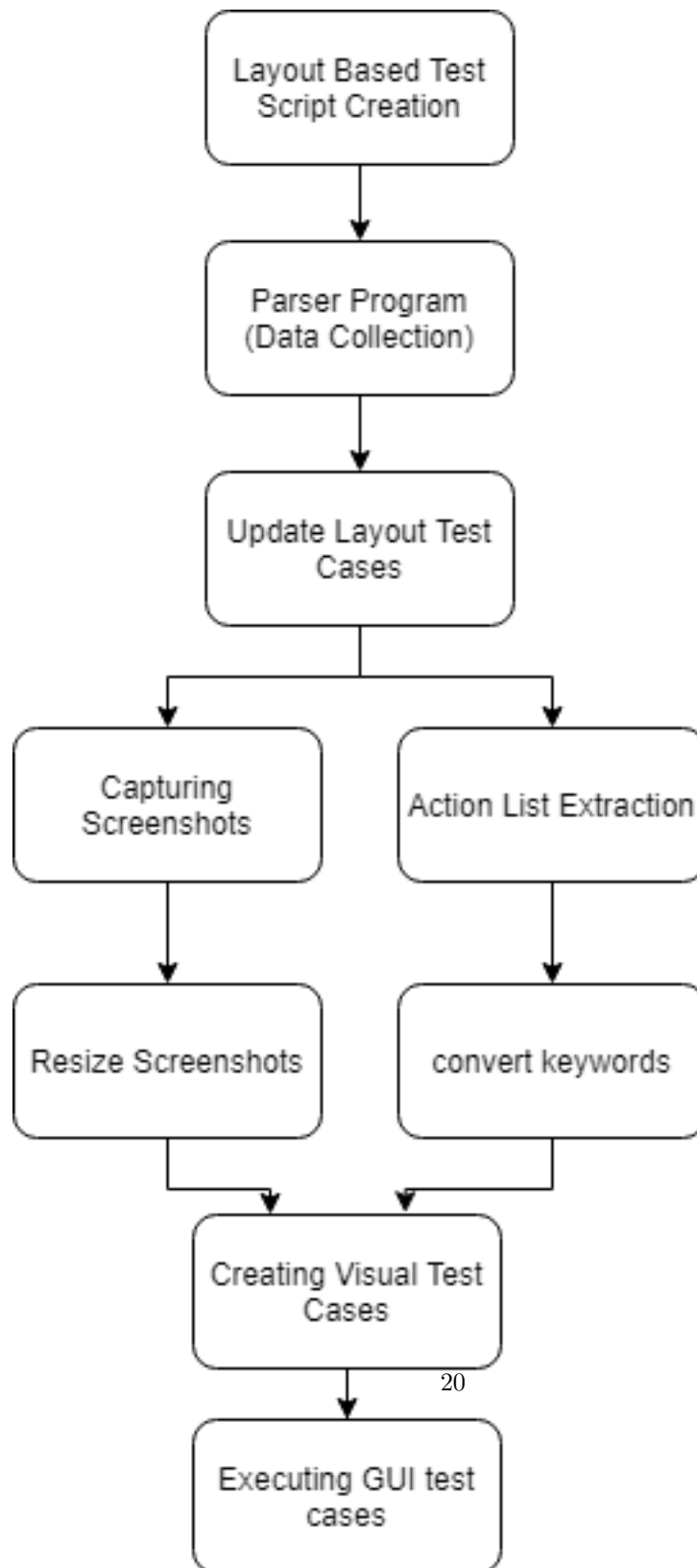


Figure 3: Layout to Visual Tool Architecture

3.0.1 Layout Based Test Script Creation:

The initial stage is to create Layout based test cases. we have created our test case using java language in selenium framework using Eclipse IDE. Every test case has more than five interactions within the mobile screen. Most of the test cases include an assert statement to check the presence of particular text. Since, we have done the experiment on three different mobile applications, in total we created 30 test cases, 10 for each application.

The corresponding desired capabilities are defined.

```
dc.setCapability(MobileCapabilityType.AUTOMATION_NAME, "Appium");
dc.setCapability(MobileCapabilityType.PLATFORM_NAME, "Android");
dc.setCapability(MobileCapabilityType.PLATFORM_VERSION, 9.0);
dc.setCapability(MobileCapabilityType.DEVICE_NAME, "Android Emulator");
dc.setCapability(MobileCapabilityType.AUTOMATION_NAME, AutomationName.ANDROID_UIAUTOMATOR2);
dc.setCapability(MobileCapabilityType.APP,
    "E:\\Omni-Notes\\omniNotes\\build\\outputs\\apk\\alpha\\debug\\OmniNotes.apk");
```

3.0.2 Parser Program:

The second task is to write a parser program, which takes any already written Layout based java test code for mobile testing, and generate an excel sheet showing all the attributes from a code line. For that we need to know the action functionality list, means a list of all the operations that can be done by on an element. for example, operations like, click or input text, or whatever which we can translate to third generation. we have to look at a single line code and try to extract all useful information from it. In general we can divide the information given in a single line code into these section:

1. location attribute by which you try to locate the element on the page: eg. `findElementByXPath`, `findElementById`.
2. container of the element: eg. `span`, `div`, `button`
3. attribute of the element by which you are trying to uniquely identify the element: eg. `text`, `id`
4. Value of the attribute you have chose to uniquely identify the element: `CERCA`
5. action which has to be performed on that element: eg. `click`, `input text`.

We can see an example for all the above mentioned points:

```
driver.findElementByXPath("//span[contains(text(),'CERCA')]").click();
```

From the above line we can extract following information:

1. Locator : `XPath`
2. Container : `span`

3. Attribute : text
4. Value : CERCA
5. Action : click

For creating a parser program, we have used java as a programming language with selenium library and we have written the code in Eclipse IDE. we have created different methods for different functionalities. These methods are named as locatorType, containerType, attributeType, valueType and actionType respectively for all the values which we have to extract from a every single line appium code. All the methods make use of StringUtils class and methods like substringBetween() to extract the useful information from the line code. These methods are called inside the main method respectively. While to read the appium code file line by line, we used Buffer-reader class, which runs till the end of the file using a while loop. Figure ??.

Below is the example of a method created:

```
// this method extract the locator type used by the driver.
// few examples are: driver.findElementByXPath, driver.findElementById
public static String locatorType(String locator) {
    int count = 0;
    if(locator.contains("findElementBy")) {
        String loc_1 = StringUtils.substringBetween(locator, "driver.findElementBy", "(");
        if(loc_1.length() != 0 ) {
            count = count + 1;
            return loc_1;
        }
    }else if(locator.contains("findElement(")){
        String loc_2 = StringUtils.substringBetween(locator, "driver.findElement(By.", "(");
        if(loc_2.length() != 0) {
            return loc_2;
        }
    }
    return null;
}
```

The program runs in such a way that, every time a line is read from the file, it is passed as a parameter to particular function which extract the useful information from the line, and returns back the result to a variable inside the main function. While the value inside this variable is appended to a linked-list created for every type or we can say for every method. When all the lines in the file are read and stored in the respective linked list, these linked list are converted into array.

At last, the values inside these linked list converted into array, is printed on an excel sheet, by running a for loop. While the path of the excel sheet is given in the program only, which can be changed on choice, since the file is created on run time. Apache POI library was very useful in printing the values into

the excel sheet. also, we need to add the apache POI jar files into external jars in eclipse build path after downloading from google. i faced few errors while using this library, like null pointer exception, since null value was thrown while creating a new row. while i was able to solve this by creating a private static variable of row class and running a for loop separately to create all the row headers first.

3.0.3 Update Layout Test Cases:

A java program created in Eclipse takes a layout test case as an input, and adds a line of code to take screenshot before every line of code which do some interaction on the mobile application screen.

```
Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/fab_note").click();

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/detail_title").sendKeys("First Note");

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/detail_content").click();
```

For example, the lines of code in the above image are updated with including two lines of code before every line, as shown in the image below.

```
Thread.sleep(200);
MobileElement element1 = driver.findElementById("it.feio.android.omninotes.alpha:id/fab_note");
AddScreenshot.elementScreenshot(driver, element1 , "element1");
driver.findElementById("it.feio.android.omninotes.alpha:id/fab_note").click();

Thread.sleep(200);
MobileElement element2 = driver.findElementById("it.feio.android.omninotes.alpha:id/detail_title");
AddScreenshot.elementScreenshot(driver, element2 , "element2");
driver.findElementById("it.feio.android.omninotes.alpha:id/detail_title").sendKeys("First Note");

Thread.sleep(200);
MobileElement element3 = driver.findElementById("it.feio.android.omninotes.alpha:id/detail_content");
AddScreenshot.elementScreenshot(driver, element3 , "element3");
driver.findElementById("it.feio.android.omninotes.alpha:id/detail_content").click();
```

3.0.4 Capturing Screenshots:

When the updated layout based test case is executed through eclipse IDE, every time elementScreenshot() method is executed, a screenshot of the element to be interacted on mobile screen is taken and saved at a location in the system. For completion of this task, Add Screenshot program contains a method elementScreenshot() method. Below is the code lines used:


```

File scrFile = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);

BufferedImage fullImg = ImageIO.read(scrFile);
// Get the location of element on the page
Point point = ele.getLocation();
System.out.println(point);
int eleWidth = ele.getSize().getWidth();
int eleHeight = ele.getSize().getHeight();

System.out.println(eleWidth + "and" + eleHeight);
// Crop the entire page screenshot to get only element screenshot
BufferedImage eleScreenshot = fullImg.getSubimage(point.getX(), point.getY(),
    eleWidth, eleHeight);

```

3.0.5 Resize Screenshots:

This is part of the translation which was performed on the extracted data. The screenshots which were captured as mentioned previously, were not suitable to use for the GUI based test cases. They need to be modified according to the aspect ratio of the android emulator. so, inside the same method the resizing of screenshots was done. Below is the code used for this purpose:

```

// resizing the image
Image reImage = eleScreenshot.getScaledInstance(eleWidth / 3, eleHeight / 3,
    Image.SCALE_DEFAULT);
eleScreenshot = new BufferedImage(eleWidth / 3, eleHeight / 3,
    BufferedImage.TYPE_INT_ARGB);
Graphics2D g2d = eleScreenshot.createGraphics();
g2d.drawImage(reImage, 0, 0, null);
g2d.dispose();

```

3.0.6 Action List Extraction:

In the same program Add Screenshot, one method is written to extract all the actions which are performed on any elements on the mobile screen in that particular test case. Below is the code used

```

public static String actionType(String lineCode) {
    String act = StringUtils.substringBetween(lineCode, ".", "(");
    if (lineCode.contains("driver.find")) {
        if (act.length() != 0) {
            if (act.equals("sendKeys")) {
                String text = "";
                text = StringUtils.substringBetween(lineCode, "sendKeys(", ")")
                    .replaceAll("^\\\"+|\\\"+$", "");
                return act + " " + text;
            } else if (act.equals("isDisplayed")) {
                String assertText = "";
                assertText = StringUtils.substringBetween(lineCode, "text,", "'")
                    .replaceAll("^\\\"+|\\\"+$", "");
                return act + " " + assertText;
            } else if (act.equals("getText")) {
                String assertEquals = "";
                assertEquals = StringUtils.substringBetween(lineCode, "getText(),\\\"", "\\\"")
                    .replaceAll("^\\\"+|\\\"+$", "");
                return act + " " + assertEquals;
            } else {
                return act;
            }
        }
    }
    return act;
}

```

3.0.7 Convert to GUI based Keywords:

The action list which is extracted in the previous step contains the actions performed on respective elements from the layout based test case. These actions were click, send keys, assert True. etc. While in visual testing tool, in our case particularly for sikuli, the keywords used are different. So, a modification was required. Below is the code to do the modification:

```

for (int i = 0; i < actionList.size(); i++) {
    guilist.add(actionList.get(i));
    String action = actionList.get(i).toString();
    if (action.contains("click")) {
        guilist.set(i, "click(\"element\" + i + ".png\")\nwait(2)");
    } else if (action.contains("sendKeys")) {
        String update = action.replace("sendKeys ", "");
        guilist.set(i, "type(\"" + update + "\")\nwait(2)");
    } else if (action.contains("isDisplayed")) {
        String update = action.replace("isDisplayed ", "");
        guilist.set(i, "exists(\"element\" + i + ".png\")\nwait(2)");
    } else if (action.contains("getText")) {
        String update = action.replace("getText ", "");
        guilist.set(i, "exists(\"" + update + "\")\nwait(2)");
    }
}

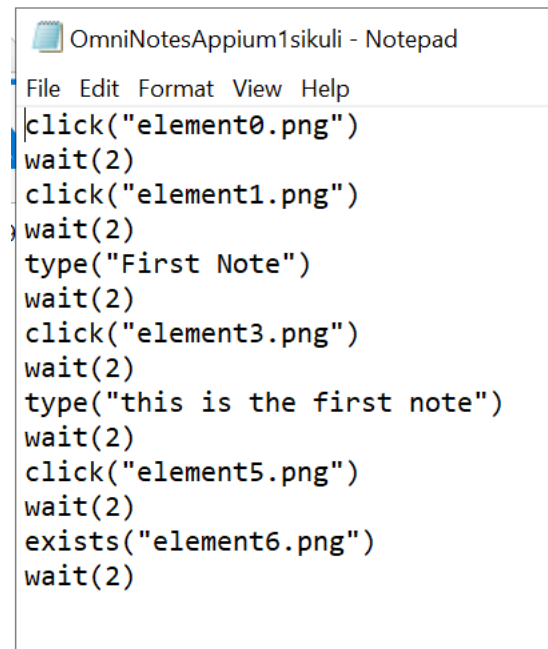
```

we can see in the code above the actions performed in the layout based test case are written into a text file, after converting them according the corre-

sponding keywords used in visual testing tool sikuli. The following changes are done:

- (a) click event the text file is modified eg. click(Element.png)
- (b) sendKeys action is replaced with type keyword in text file eg. type("this is first note")
- (c) assertTrue statements in test script, exists keyword is used in text file. eg. exists ("one is smaller")

So, finally an updated text file which can be used as an input for the sikuli tool will look like:



```
File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
type("First Note")
wait(2)
click("element3.png")
wait(2)
type("this is the first note")
wait(2)
click("element5.png")
wait(2)
exists("element6.png")
wait(2)
```

Figure 4: Sikuli Input Text File

3.0.8 Creating Visual Test Cases:

After resizing of screenshots and modification of text files, all the outputs from the program are stored at sikuli folder location only, so that it is easy to access them. Separate folder were made using the program code only, for different test cases. It means three main folders for three different applications and ten sub

folders for each test case were created. All the corresponding screenshots and text files are put in these folder.

3.0.9 Executing GUI Test Cases:

When we have received all the necessary data, the screenshots and text files. The all test cases were executed through sikuli tool one by one and results are stored in screenshots. Below is the example:

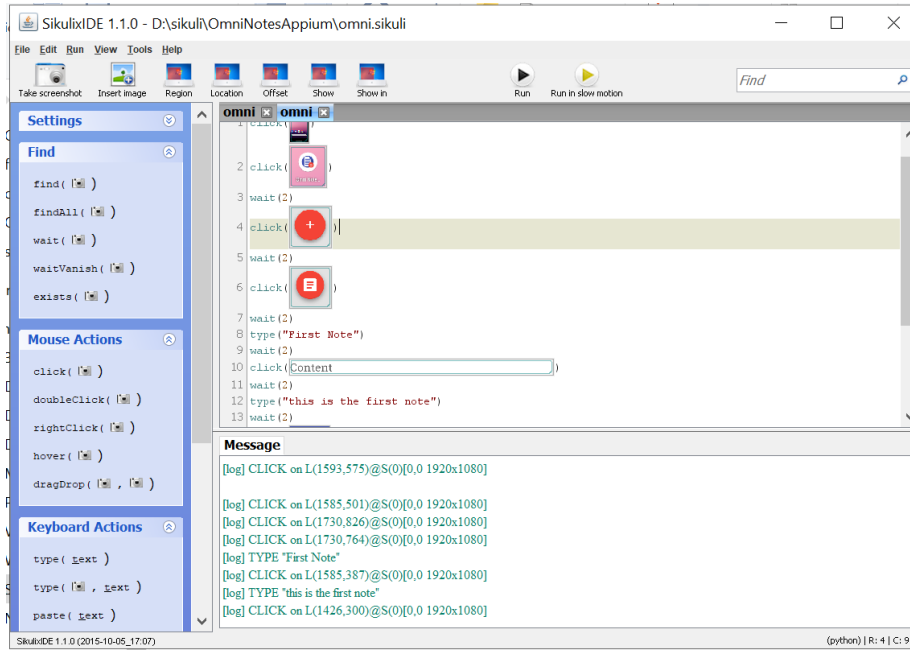


Figure 5: Sikuli Tool UI

4 EVALUATION

4.1 Definition of the Experiment:

The outcomes of the experiment were very exciting, as we got to know about different challenges and opportunities seen to improve the process of converting second generation test scripts to third generation test scripts. In total there were 30 layout based test cases, 10 for each mobile application which we did the experiment on and every test case had more than five interactions. Out of 30 layout test scripts, we were able to run all of them after converting them into

GUI based test cases. Overall, There was 100 percent success rate to convert all the test scripts into GUI based scripts.

4.2 Interaction Log Data:

During the experiment, for each test script there were five or more than five interactions inside the mobile application. so. for every application for all the test cases written, the data has been collected about the number of interactions and the type of interactions performed. Below are the data collected for all three applications:

1. Omni Notes Application:

C	D	E	F	G
Test Case Name	Total Number of Interactions	Number of Interactions per Type		
		click	sendKeys/ type	assert/exists
Omni Notes Test Script 1	7	4	2	1
Omni Notes Test Script 2	5	5	0	0
Omni Notes Test Script 3	6	5	1	0
Omni Notes Test Script 4	8	6	2	1
Omni Notes Test Script 5	7	6	1	0
Omni Notes Test Script 6	7	6	1	0
Omni Notes Test Script 7	7	5	1	1
Omni Notes Test Script 8	6	6	0	0
Omni Notes Test Script 9	9	6	2	1
Omni Notes Test Script 10	5	3	1	1

Figure 6: Omni Notes Interaction Table

2. Calculator Application:

C	D	E	F	G
Test Case Name	Total Number of Interactions	Number of Interactions per Type		
		click	sendKeys/ type	assert/exists
Calculator App Test Script 1	7	6	0	1
Calculator App Test Script 2	7	6	0	1
Calculator App Test Script 3	7	6	0	1
Calculator App Test Script 4	7	6	0	1
Calculator App Test Script 5	6	5	0	1
Calculator App Test Script 6	6	5	0	1
Calculator App Test Script 7	7	6	0	1
Calculator App Test Script 8	7	6	0	1
Calculator App Test Script 9	7	6	0	1
Calculator App Test Script 10	7	6	0	1

Figure 7: Calculator App Interaction Table

3. Unit Converter Application

C	D	E	F	G
Test Case Name	Total Number of Interactions	Number of Interactions per Type		
		click	sendKeys/ type	assert/exists
Unit Converter App Test Script 1	9	7	1	1
Unit Converter App Test Script 2	9	7	1	1
Unit Converter App Test Script 3	9	7	1	1
Unit Converter App Test Script 4	9	7	1	1
Unit Converter App Test Script 5	9	7	1	1
Unit Converter App Test Script 6	9	7	1	1
Unit Converter App Test Script 7	9	7	1	1
Unit Converter App Test Script 8	9	7	1	1
Unit Converter App Test Script 9	9	7	1	1
Unit Converter App Test Script 10	9	7	1	1

Figure 8: Unit Converter Interaction Table

4.3 Attempts for Translation

When layout based test scripts were converted to visual based scripts, some times the translation did not took place in one attempt. Most of the time the updated layout test script failed to complete the execution and generate all the screenshots required as input for the visual based tool sikuli. so, the data collected for attempts made by different test scripts for completely convert into visual based scripts is shown in the table below:

C	D	E	F
Test Case Number	Omni Notes Applicatoin Attempts	Calculator Applicaton Attempts	UnitConverter Application Attempts
Test Case 1	2	1	2
Test Case 2	1	2	1
Test Case 3	3	2	1
Test Case 4	1	1	1
Test Case 5	2	1	2
Test Case 6	2	1	2
Test Case 7	1	2	1
Test Case 8	3	1	1
Test Case 9	1	1	1
Test Case 10	1	1	1

Figure 9: Attempts for Translation Table

Most of the time the issue was with the sleep time, as the element was not found in the DOM or not present on the page. Few times the issue was that, script was unable to create a new session with the appium server. But at last all the test cases were translated successfully.

From the above table we can see that out of 30 test cases 19 were able to be translated into visual script in one attempt only. so, we can conclude that:

$$\text{Percentage}(\text{Single Attempt Translation}) = (19/30)*100 = 63.3 \text{ percent}$$

4.4 Issues faced in Translation:

Firstly, it took more than one attempt to run updated Layout based test scripts to generate screenshots. Most of the time the problem was with the waiting time means the thread.sleep statement, means element was not visible on the page or it was not present in the DOM. This issue was resolved by increasing the thread sleep time. In few of the interactions the sleep time was less than 500ms, but in some cases the sleep time was increased till 1000ms which made the code to run successfully. The line of code which required to be changed was:

```
Thread.sleep(1000);
```

Similar scenario was seen during execution of sikuli based scripts, which was mostly because of slowness of the android emulator or the whole system. This issue was resolved again here also by increasing the waiting time. The line of code which was added in the text file and changed in case script was failing was:

```
wait(2)
```

There were some very specific and exciting observations made during the experimental which are explained below:

- (a) Few screenshots were not recognized by the sikuli framework, so those test cases failed in between. these images were renamed as discarded images and new images were generated using the sikuli framework only. After adding new screenshots, the test scripts were again run from the beginning and resulted in passing of the script. Also sometimes a blank image is generated by the screenshot program, which was useless because it guaranteed that, when you run this script in sikuli it will fail.

Examples of discarded images and respective accepted images are given below:

The image shows a close-up of a button on a mobile screen. The button has a light gray background and the text 'Creation date' in a bold, black, sans-serif font. The button is rectangular and appears to be part of a larger application interface.

Figure 10: Discarded Image 1

The above element was discarded by the sikuli tool, after which an screenshot was taken using sikuli tool functionality and below image was gener-

ated.

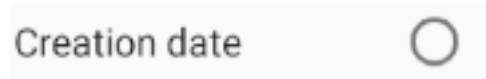


Figure 11: Accepted Image 1

Sometimes it was strange to know, why the screenshot was failing, as you see in the below example.

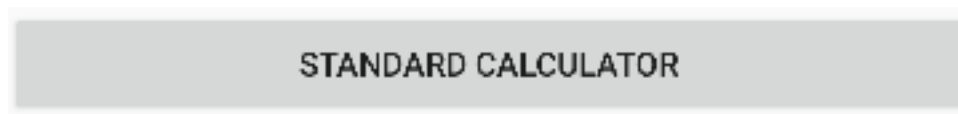


Figure 12: Discarded Image 2

In this case, the above element received from element screenshot method was rejected, while it look similar to the screenshot image generated by the sikuli tool.

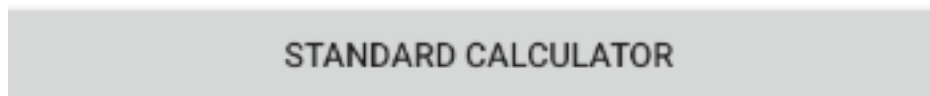


Figure 13: Accepted Image 2

After collecting data from all the three different applications about the discarded images, the log is put into the table shown below:

Test Case Number	Omni Notes Applicatoin	Calculator Applicaton	UnitConverter Application
Test Case 1	1	1	1
Test Case 2	0	0	0
Test Case 3	1	0	1
Test Case 4	2	0	2
Test Case 5	1	0	1
Test Case 6	0	0	0
Test Case 7	1	0	1
Test Case 8	0	0	0
Test Case 9	2	0	2
Test Case 10	2	0	2

Figure 14: Discarded Screenshots By Apps

- (b) False Positive cases were found during the execution of sikuli test scripts.
For example,

I In the mobile application unit converter, the screenshot was for "Grams" value in a drop-down list, while another option "Grains" was selected by the tool.

II During execution of a test script for calculator mobile application, where a division button was clicked by the tool, while the addition button was expected to be clicked.

- (c) When two similar images are found on the same page of the mobile, then the script clicked on the first image, however it was expected to click on the second image. But when the screenshots are generated by the sikuli tool, the mouse clicked on the desired image even though there were two similar images present on the mobile screen.

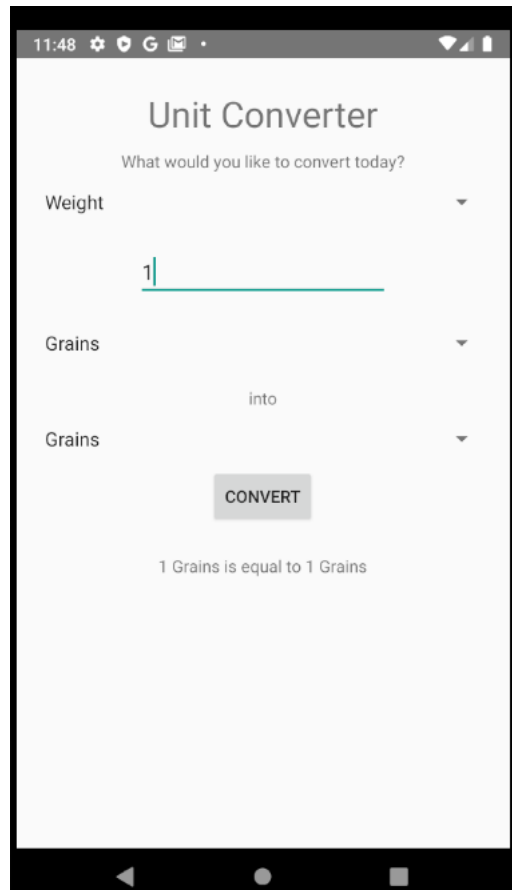


Figure 15: Similar Image Issue

In total there were 6 test cases which were affected by discarded screenshots problem, 2 test cases affected by False Positive problem and 2 test case affected by similar images issue.

4.5 Time Taken For Translation:

Different test cases took different time span for translation from layout to visual script. Time factor was affected from all the issues mentioned in above section during translation of the script. Translation can be divided into small parts, and the time taken was different on average for different phases of translation. For Example(for every test case on average with time in approx.):

Average Time to Updated Test Script: 1min

Average Time to create Screenshots: 4min

Average Time to Update Text File: 1min

Total average time for complete translation = 5min

4.6 Execution Attempts:

After solving all the issues seen in the previous step, the test cases are executed a fixed number of times i.e. 10. After executing each test case from all the three applications respectively, the results were collected from first 10 attempts, as shown below:

Test Case Number	Omni Notes Applicatoin Attempts	Calculator Applicaton Attempts	UnitConverter Application Attempts
Test Script 1	10	10	10
Test Script 2	8	10	10
Test Script 3	10	10	9
Test Script 4	10	10	10
Test Script 5	1	10	10
Test Script 6	9	10	10
Test Script 7	10	9	10
Test Script 8	10	10	10
Test Script 9	9	10	10
Test Script 10	10	10	10

Figure 16: Execution Attempts Passed

The execution results were very satisfactorily, as most of the times the test cases were executed successfully.

4.7 Test Suites and Apps Used:

There are different tools used for performing this experiment. Below are the name of the tools and apps which are used and for what purpose.

- (a) Eclipse IDE: The programs like parser and Add Screenshots are written in Eclipse IDE and in java language. The choice of the tool was very obvious, as it is an open source tool and provides an integrated development environment. Also, all the test scripts for all the three applications are written in java language in Eclipse.
- (b) Appium: Appium provides an open source test automation framework. Appium server is used to run the code written in eclipse on Android emulator. The necessary desired capabilities are mentioned in the java program to connect with the appium server. Also, the appium element inspector is used to identify the xpath locators for different element in the mobile applications.
- (c) Android Studio: This tool was used to build the project after downloading the code for mobile applications from Github. It was also used to create and run the android emulator.
- (d) Sikuli: Sikuli is an image-based open source tool to automate the GUI and can be used on any platform like Windows/Linux/Mac/Mobile. This GUI based testing tool is used for running the updated text based test scripts generated through the experiment.
- (e) Omni Notes Mobile Application: This was our first mobile application on which we did the experiment. Its a note making and saving application. The home page of the application looks like this:

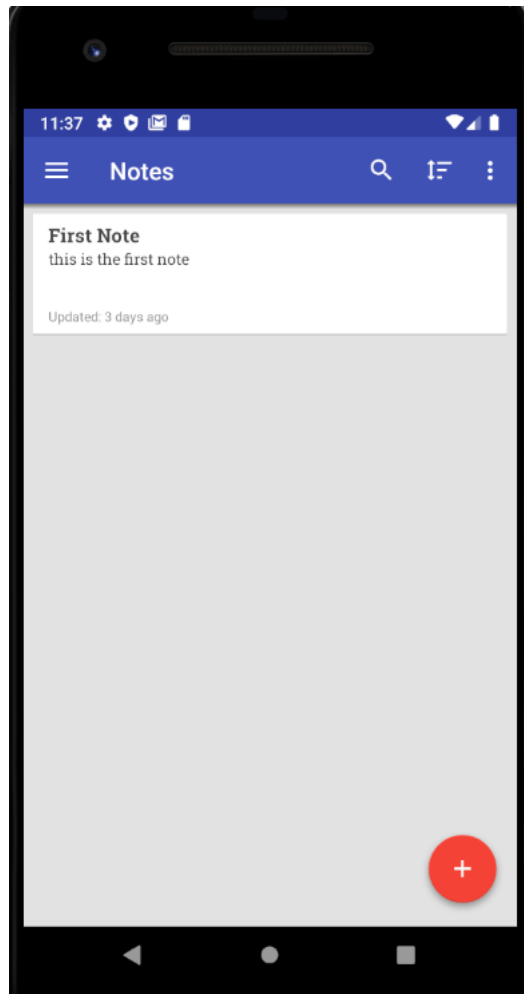


Figure 17: Omni Notes Application UI

- (f) Calculator Mobile Application: This was our second mobile application on which we did the experiment. It resembles like any other calculator mobile application. The home page of the application looks like this:

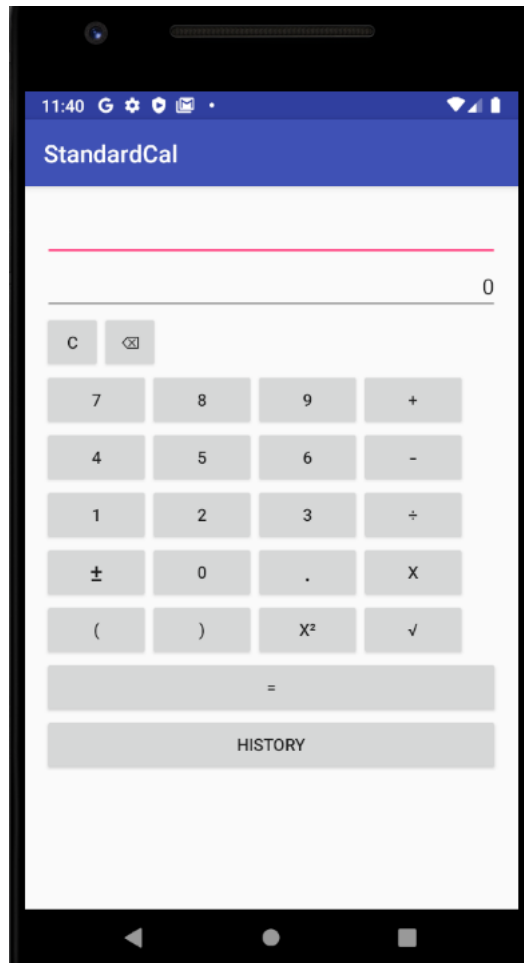


Figure 18: Calculator Application UI

- (g) Unit Conversion Mobile Application: This was our third mobile application on which we did the experiment. The home page of the application looks like this:

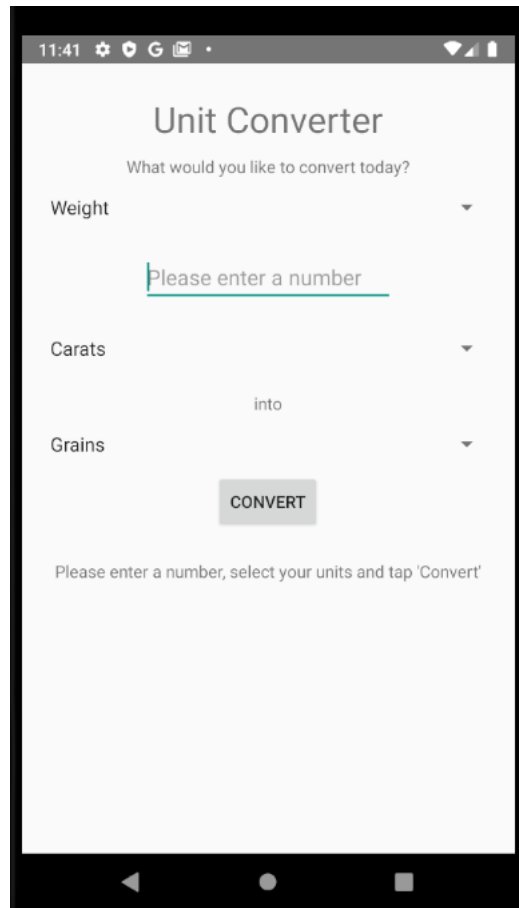


Figure 19: Unit Converter Application UI

4.8 Results:

The results obtained from the experiment were very exciting as we were able to translate all the layout based test cases into visual based test cases. Below one example of one test case from each of the three applications is shown, while results for all the test cases are shown in the appendices chapter. The results are shown in a way, starting from the java code for layout test script, which after going through parser and modification, generates a text file with the same number of interactions as they were happening in the java code, followed which it was executed in sikuli tool to generate successful results:

4.8.1 Omni Notes Test Script Example

```
driver.findElementById("it.feio.android.omninotes.alpha:id/fab_expand_menu_button").click();

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/fab_note").click();

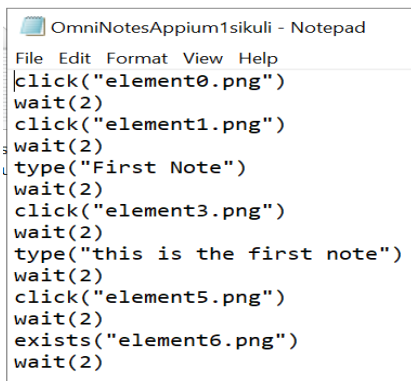
Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/detail_title").sendKeys("First Note");

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/detail_content").click();

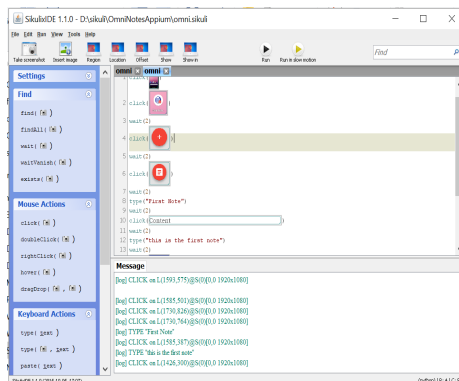
driver.findElementById("it.feio.android.omninotes.alpha:id/detail_content").sendKeys("this is the first note");

driver.findElementByXPath("//android.widget.ImageButton[@content-desc='drawer open']").click();

Thread.sleep(500);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'this is the first note')]").isDisplayed());
```



(a) Text file



(b) Sikuli Results

4.8.2 Calculator App Test Script Example

```
driver.findElementById("anubhav.calculatorapp:id/button1").click();

Thread.sleep(1000);
driver.findElementById("anubhav.calculatorapp:id/clear").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num5").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/plus").click();

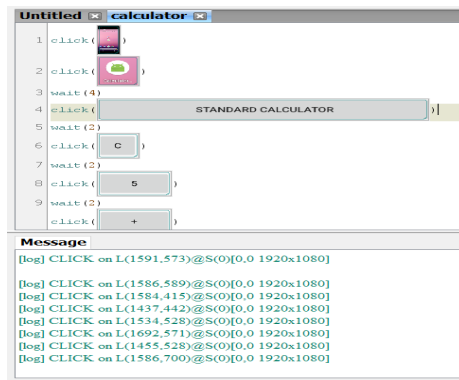
Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num4").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/equal").click();

Thread.sleep(200);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'9.0')]").isDisplayed());
```

```
File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
click("element2.png")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
exists("element6.png")
wait(2)
```

(a) Text file



(b) Sikuli Results

4.8.3 Unit Converter Test Script Example

```
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerCategory").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Weight')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/input").sendKeys("1");

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsBase").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Pounds')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsResult").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Ounces')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/btnConvert").click();

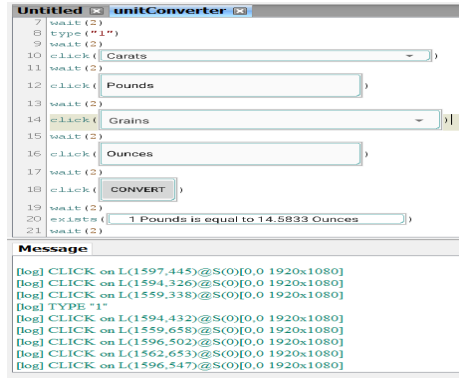
Thread.sleep(500);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'1 Pounds is equal to '
+ "14.5833 Ounces')]").isDisplayed());
```

```

File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
type("1")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
click("element6.png")
wait(2)
click("element7.png")
wait(2)
exists("element8.png")
wait(2)

```

(a) Text file



(b) Sikuli Results

5 Threats to Validity:

In this chapter, we are going to discuss about the inherent limitations of the experiment which we have performed. This can be divided into three main categories as mentioned below:

5.1 Threats to External Validity:

The experiment done here is at a small scale compared to an industry environment. The data-set collected and used is also very specific. so, the results generated from this experiment can not be generalized and applied to all other projects. Firstly, since the interactions and keywords used are specific to sikuli tool, so implementing them to other automation GUI testing tools will generate issues. For example, if you run the same test suite for Eye Automate will result in failure. But after some modifications, we would be able to run the same test suites in Eye Automate also. By far the number of interactions were less and platform was same for all the applications, so few issues may come if platform is changed or an application with completed different appearance and logic. But these issues are temporary technical limitations, and which will be addressed in the future versions.

5.2 Threats to Internal Validity:

The result of the experiment shows that the translation success rate was 100 percent. But there were few issues while executing the translated scripts which

required some manual changes. since all the assumptions made during the experiment were correct and the calculations are done in a standard method to calculate the effectiveness of the tool.

5.3 Threats to Construction Validity:

During the experiment standard procedures are being followed, like a test case is considered successful if it is passed, and tool effectiveness is calculated by the total number of test cases passed. While during executing there were some False Positive cases found, which was solved by taking the screenshots using the sikuli tool itself. But there was no False Negative. Blank images were discarded, which resulted in failure of particular test cases

6 CONCLUSION

6.1 Conclusion of Experiment

GUI based automated testing is the solution for today's fast growing mobile application market, where time and accuracy plays an important roles. So far, we have seen in our experiment that the translation of layout based test scripts to visual based test scripts reduces the human efforts required for testing. So, a mixed approach where we can use both layout and visual based test cases is very helpful to increase the productivity of the project.

GUI based automated testing is becoming a common approach among the testing community. But GUI based approach alone is not feasible to do effective testing, which motivates to use hybrid approach which is a mix of second generation layout based and third generation GUI based testing. But, this has proved to be a challenge as both the techniques have common benefits and shortcomings with respect to speed, robustness and capability to emulate the human user, which make the two approaches complementary to each other. In this experiment, we proposed a solution for translation of test scripts from one generation to another. This approach will enlighten the beneficiaries to harness the positive aspects of both the generations while mitigating the shortcomings of particular approaches.

Although the failure in execution after the translation in some test cases were due to fragility issues, but success is not guaranteed when the test cases itself

is faulty. a faulty situation can generate when an element changes at run time or based on the current time, for example an interaction with an element works fine before the translation and fails after that. Eventually, this experiment have the potential to become a useful tool for the testing and developer community.

6.2 Expected Future Work

Although tool was able to translate the interactions which were provided in the layout based test scripts but it does not guarantee that all the interactions for all the tools may be translated easily. The factors like mobile application and device platform can play important role in the success or failure of these kind of translation. so, an extensive study is required which matches the parameters of an industrial approach. Also, the current tool does not cover all the interactions possible on a mobile application screen, so future work focused on testing all the interactions will be very helpful for the testing community.

References

- [1] Riccardo Coppola, Maurizio Morisio and Marco Torchiano.
Scripted GUI Testing of Android Apps: A Study on Diusion, Evolution and Fragility.
- [2] Pavneet Singh Kochhar, Ferdian Thung, Nachiappan Nagappan, Thomas Zimmermann, and David Lo1
Understanding the Test Automation Culture of App Developers
- [3] Mario Linares-Vasquez1, Carlos Bernal-Cardenas2, Kevin Moran2, and Denys Poshyvanyk
How do Developers Test Android Applications?
- [4] LucaArdito,RiccardoCoppola, MarcoTorchiano
Towards Automated Translation between Generations of GUI-based Tests for Mobile Devices
- [5] Mario Linares-Vásquez1, Kevin Moran2, and Denys Poshyvanyk2
Continuous, Evolutionary and Large-Scale: A New Perspective for Automated Mobile App Testing

Appendices

A Omni Notes Application

A.A Omni Notes Text Script 1

```
driver.findElementById("it.feio.android.omninotes.alpha:id/fab_expand_menu_button").click();

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/fab_note").click();

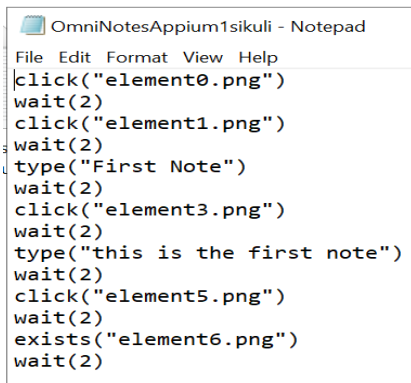
Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/detail_title").sendKeys("First Note");

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/detail_content").click();

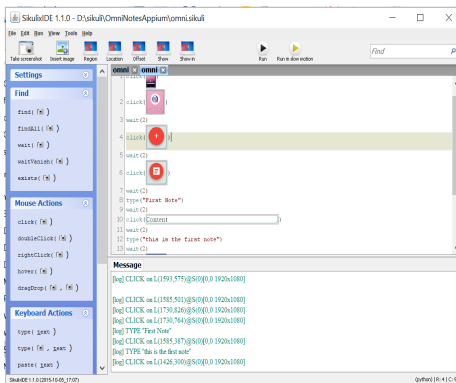
driver.findElementById("it.feio.android.omninotes.alpha:id/detail_content").sendKeys("this is the first note");

driver.findElementByXPath("//android.widget.ImageButton[@content-desc='drawer open']").click();

Thread.sleep(500);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'this is the first note')]").isDisplayed());
```



(a) Text file



(b) Sikuli Results

A.B Omni Notes Text Script 2

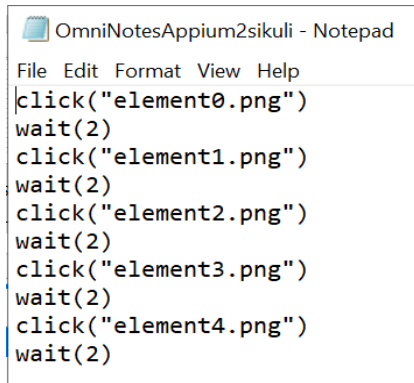
```
driver.findElementByXPath("//android.widget.ImageButton[@content-desc=\"drawer open\"]").click();

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/settings").click();

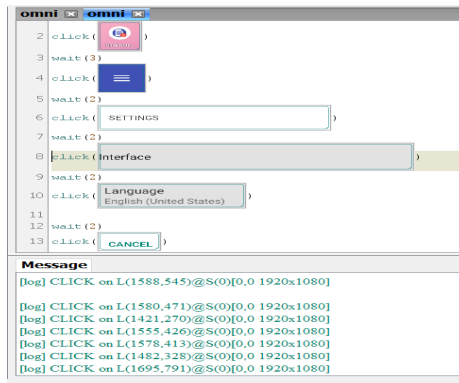
Thread.sleep(500);
// click on Interface
driver.findElementByXPath("(//*[class=\"android.widget.RelativeLayout\"][4])").click();

Thread.sleep(200);
// click on Language
driver.findElementByXPath("(//*[class=\"android.widget.RelativeLayout\"][2])").click();

Thread.sleep(200);
// click on cancel button
driver.findElementByXPath("(//*[class=\"android.widget.Button\"][1])").click();
```



(a) Text file



(b) Sikuli Results

A.C Omni Notes Text Script 3

```
driver.findElementById("it.feio.android.omninotes.alpha:id/fab_expand_menu_button").click();

Thread.sleep(300);
driver.findElementById("it.feio.android.omninotes.alpha:id/fab_checklist").click();

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/detail_title").click();

driver.findElementById("it.feio.android.omninotes.alpha:id/detail_title").sendKeys("First Checklist");

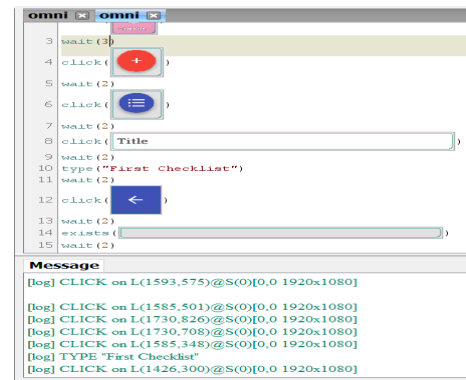
driver.findElementByXPath("//android.widget.ImageButton[@content-desc=\"drawer open\"]").click();

Thread.sleep(200);
assertTrue(driver.findElementByXPath("(//*[contains(@text, 'First Checklist')]).isDisplayed());
```

OmniNotesAppium3sikuli - Notepad

```
File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
click("element2.png")
wait(2)
type("First Checklist")
wait(2)
click("element4.png")
wait(2)
exists("element5.png")
wait(2)
```

(a) Text file



(b) Sikuli Results

A.D Omni Notes Text Script 4

```
driver.findElementById("it.feio.android.omninotes.alpha:id/fab_expand_menu_button").click();

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/fab_note").click();

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/detail_title").sendKeys("Work Note");

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/menu_category").click();

Thread.sleep(300);
driver.findElementByXPath("//*[@contains(@text,'ADD CATEGORY')]").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Title')]").sendKeys("Work");

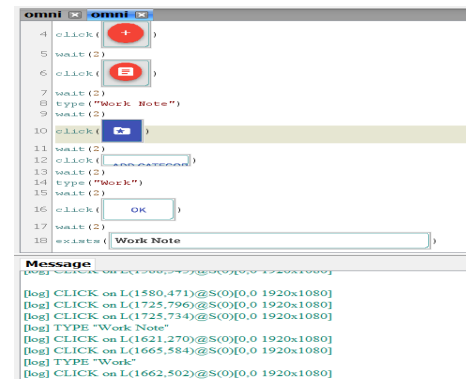
driver.findElementById("it.feio.android.omninotes.alpha:id/save").click();

Thread.sleep(500);
assertTrue(driver.findElementById("it.feio.android.omninotes.alpha:id/detail_title").isDisplayed());
```

OmniNotesAppium4sikuli - Notepad

```
File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
type("Work Note")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
type("Work")
wait(2)
click("element6.png")
wait(2)
exists("element7.png")
wait(2)
```

(a) Text file



(b) Sikuli Results

A.E Omni Notes Text Script 5

```

driver.findElementById("it.feio.android.omninotes.alpha:id/fab_expand_menu_button").click();

Thread.sleep(300);
driver.findElementById("it.feio.android.omninotes.alpha:id/fab_checklist").click();

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/detail_title").click();

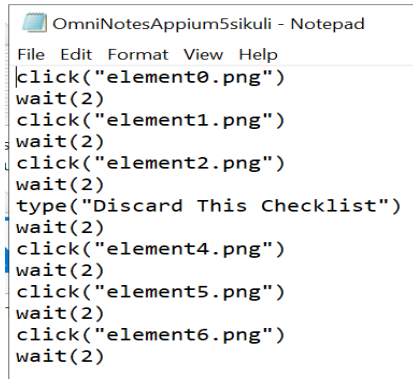
driver.findElementById("it.feio.android.omninotes.alpha:id/detail_title").sendKeys("Discard This Checklist");

Thread.sleep(300);
driver.findElementByXPath("//android.widget.ImageView[@content-desc=\"More options\"]").click();

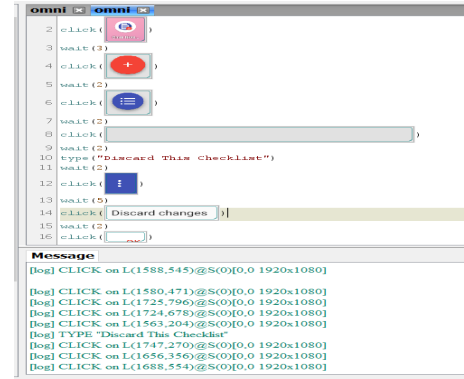
Thread.sleep(300);
driver.findElementByXPath("//*[@class=\"android.widget.TextView\"])[3]").click();

Thread.sleep(100);
driver.findElementByXPath("//*[contains(@text, 'OK')]").click();

```



(a) Text file



(b) Sikuli Results

A.F Omni Notes Text Script 6

```

driver.findElementById("it.feio.android.omninotes.alpha:id/fab_expand_menu_button").click();

Thread.sleep(300);
driver.findElementById("it.feio.android.omninotes.alpha:id/fab_checklist").click();

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/detail_title").click();

driver.findElementById("it.feio.android.omninotes.alpha:id/detail_title").sendKeys("Checklist with Timestamp");

driver.findElementByXPath("//*[contains(@resource-id, 'menu_attachment')]").click();

Thread.sleep(200);
driver.findElementByXPath("//*[contains(@text, 'Timestamp')]").click();

Thread.sleep(200);
driver.findElementByXPath("//android.widget.ImageButton[@content-desc=\"drawer open\"]").click();

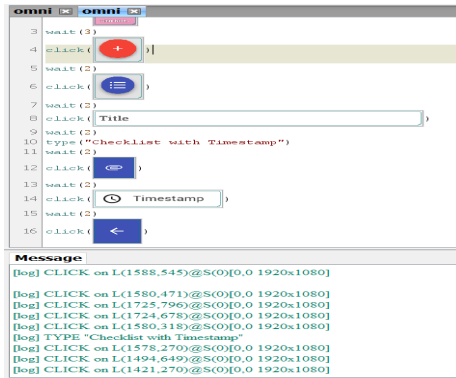
```

```

OmniNotesAppium6sikuli - Notepad
File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
click("element2.png")
wait(2)
type("Checklist with Timestamp")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
click("element6.png")
wait(2)

```

(a) Text file



(b) Sikuli Results

A.G Omni Notes Text Script 7

```

driver.findElementByXPath("//android.widget.ImageButton[@content-desc=\"drawer open\"]").click();

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/settings").click();

Thread.sleep(500);
// click on Notifications
driver.findElementByXPath("//*[@class=\"android.widget.RelativeLayout\"][7]").click();

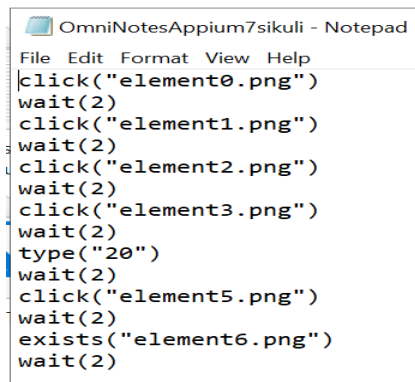
Thread.sleep(200);
// click on Seconds reminder delay
driver.findElementByXPath("//*[@class=\"android.widget.RelativeLayout\"][4]").click();

Thread.sleep(200);
// enter text in Seconds
driver.findElementByXPath("//*[@class=\"android.widget.EditText\"]").sendKeys("20");

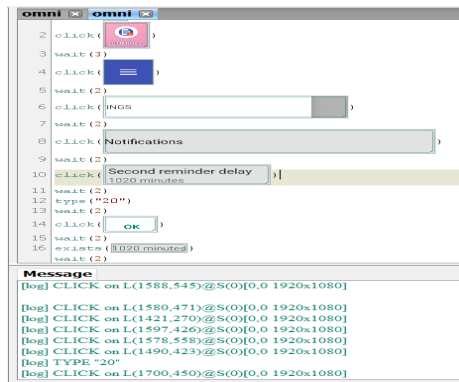
driver.findElementByXPath("//*[@contains(@text,'OK')]").click();

Thread.sleep(200);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'1020 minutes')]").isDisplayed());

```



(a) Text file



(b) Sikuli Results

A.H Omni Notes Text Script 8

```

driver.findElementByXPath("//android.widget.ImageView[@content-desc=\"More options\"]").click();

// click on - Reduced View
Thread.sleep(200);
driver.findElementByXPath("//*[@contains(@resource-id,'title')]").click();

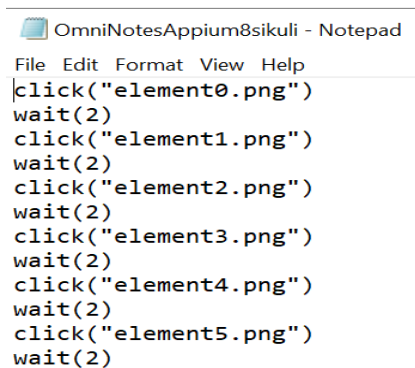
Thread.sleep(200);
driver.findElementByXPath("//android.widget.ImageView[@content-desc=\"More options\"]").click();

// click on - Expanded View
Thread.sleep(300);
driver.findElementByXPath("//*[@contains(@resource-id,'title')]").click();

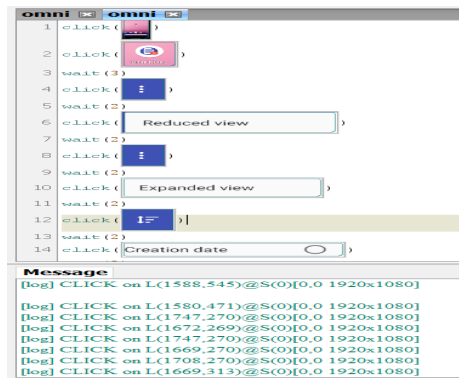
Thread.sleep(200);
driver.findElementByXPath("//android.widget.TextView[@content-desc=\"Sort\"]").click();

Thread.sleep(200);
driver.findElementByXPath("//*[@contains(@text,'Creation date')]").click();

```



(a) Text file



(b) Sikuli Results

A.I Omni Notes Text Script 9

```

driver.findElementById("it.feio.android.omninotes.alpha:id/fab_expand_menu_button").click();

Thread.sleep(300);
driver.findElementById("it.feio.android.omninotes.alpha:id/fab_checklist").click();

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/detail_title").sendKeys("First Checklist");

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/menu_category").click();

Thread.sleep(300);
driver.findElementByXPath("//*[@contains(@text,'ADD CATEGORY')]").click();

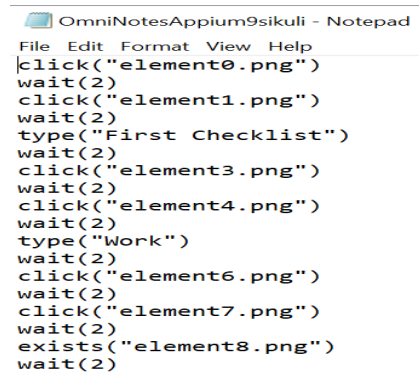
Thread.sleep(1500);
driver.findElementByXPath("//*[@contains(@resource-id,'category_title')]").sendKeys("Work");

driver.findElementById("it.feio.android.omninotes.alpha:id/save").click();

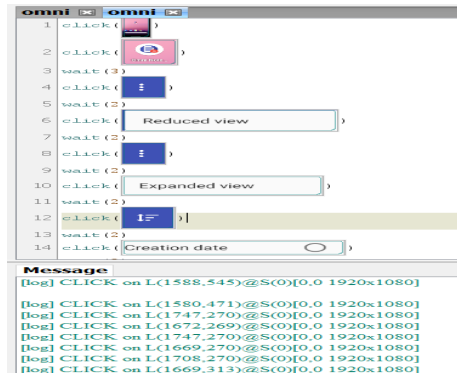
Thread.sleep(200);
driver.findElementByXPath("//android.widget.ImageButton[@content-desc=\"drawer open\"]").click();

Thread.sleep(500);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'First Checklist')]").isDisplayed());

```



(a) Text file



(b) Sikuli Results

A.J Omni Notes Text Script 10

```

driver.findElementById("it.feio.android.omninotes.alpha:id/fab_expand_menu_button").click();

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/fab_note").click();

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/detail_title").sendKeys("First Note");

driver.findElementByXPath("//android.widget.ImageButton[@content-desc=\"drawer open\"]").click();

Thread.sleep(200);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'First Note')]").isDisplayed());

```

```

OmniNotesAppium10sikuli - Notepad
File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
type("First Note")
wait(2)
click("element3.png")
wait(2)
exists("element4.png")
wait(2)

```

(a) Text file

```

omni omni
1 click(
2 click(
3 wait(3)
4 click(
5 wait(2)
6 click(
7 wait(2)
8 type("First Note")
9 wait(2)
10 click(
11 wait(2)
12 exists("First Note")

```

Message

```

[log] CLICK on L(1588,545)@S(0)[0,0 1920x1080]
[log] CLICK on L(1580,471)@S(0)[0,0 1920x1080]
[log] CLICK on L(1725,796)@S(0)[0,0 1920x1080]
[log] CLICK on L(1725,734)@S(0)[0,0 1920x1080]
[log] TYPE "First Note"
[log] CLICK on L(1421,270)@S(0)[0,0 1920x1080]

```

(b) Sikuli Results

B Calculator Application

B.A Calculator App Test Script 1

```

driver.findElementById("anubhav.calculatorapp:id/button1").click();

Thread.sleep(1000);
driver.findElementById("anubhav.calculatorapp:id/clear").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num5").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/plus").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num4").click();

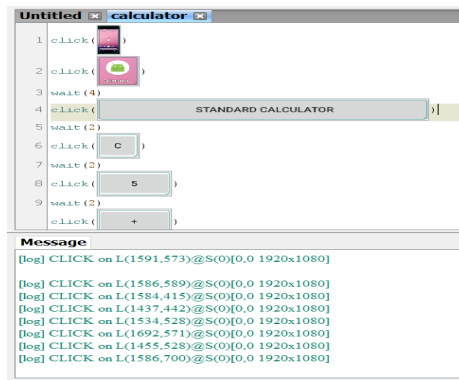
Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/equal").click();

Thread.sleep(200);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'9.0')]").isDisplayed());

```

```
File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
click("element2.png")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
exists("element6.png")
wait(2)
```

(a) Text file



(b) Sikuli Results

B.B Calculator App Test Script 2

```
driver.findElementById("anubhav.calculatorapp:id/button1").click();

Thread.sleep(1000);
driver.findElementById("anubhav.calculatorapp:id/clear").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num5").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/minus").click();

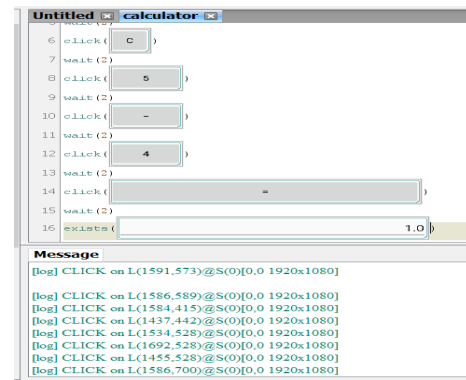
Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num4").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/equal").click();

Thread.sleep(200);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'1|.0')]").isDisplayed());
```

```
File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
click("element2.png")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
exists("element6.png")
wait(2)
```

(a) Text file



(b) Sikuli Results

B.C Calculator App Test Script 3

```
driver.findElementById("anubhav.calculatorapp:id/button1").click();

Thread.sleep(1000);
driver.findElementById("anubhav.calculatorapp:id/clear").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num5").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/multiply").click();

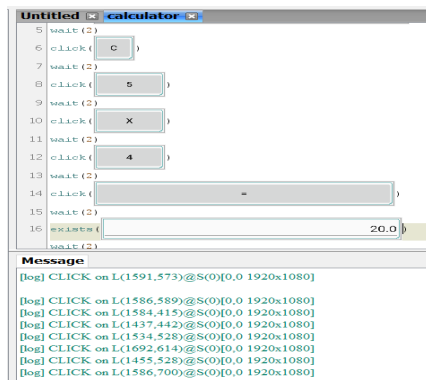
Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num4").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/equal").click();

Thread.sleep(200);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'20|0')]").isDisplayed());
```

```
File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
click("element2.png")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
exists("element6.png")
wait(2)
```

(a) Text file



(b) Sikuli Results

B.D Calculator App Test Script 4

```
driver.findElementById("anubhav.calculatorapp:id/button1").click();

Thread.sleep(1000);
driver.findElementById("anubhav.calculatorapp:id/clear").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num6").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/divide").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num3").click();

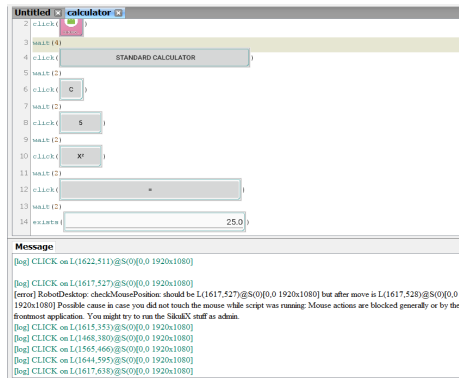
Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/equal").click();

Thread.sleep(200);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'2|.0')]").isDisplayed());
```



```
File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
click("element2.png")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
exists("element6.png")
wait(2)
```

(a) Text file



(b) Sikuli Results

B.E Calculator App Test Script 5

```
driver.findElementById("anubhav.calculatorapp:id/button1").click();

Thread.sleep(1000);
driver.findElementById("anubhav.calculatorapp:id/clear").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num5").click();

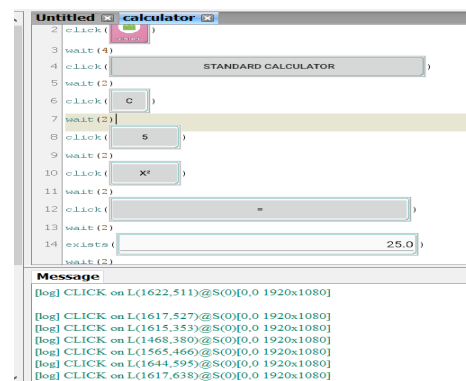
Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/square").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/equal").click();

Thread.sleep(200);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'25.0')]").isDisplayed());
```

```
File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
click("element2.png")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
exists("element5.png")
wait(2)
```

(a) Text file



(b) Sikuli Results

B.F Calculator App Test Script 6

```
driver.findElementById("anubhav.calculatorapp:id/button1").click();

Thread.sleep(1000);
driver.findElementById("anubhav.calculatorapp:id/clear").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num4").click();

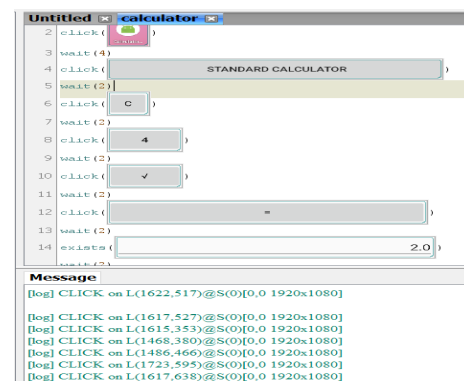
Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/sqrt").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/equal").click();

Thread.sleep(200);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'2.0')]").isDisplayed());
```

```
File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
click("element2.png")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
exists("element5.png")
wait(2)
```

(a) Text file



(b) Sikuli Results

B.G Calculator App Test Script 7

```

driver.findElementById("anubhav.calculatorapp:id/button").click();

Thread.sleep(1000);
driver.findElementById("anubhav.calculatorapp:id/clear").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num2").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/xpowy").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num3").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/equal").click();

Thread.sleep(200);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'8.0')]").isDisplayed());

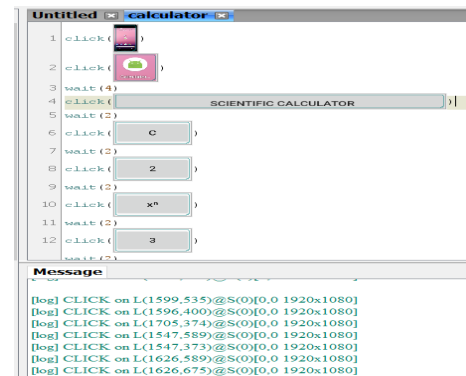
```

```

File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
click("element2.png")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
exists("element6.png")
wait(2)

```

(a) Text file



(b) Sikuli Results

B.H Calculator App Test Script 8

```
driver.findElementById("anubhav.calculatorapp:id/button").click();

Thread.sleep(1000);
driver.findElementById("anubhav.calculatorapp:id/clear").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num1").click();

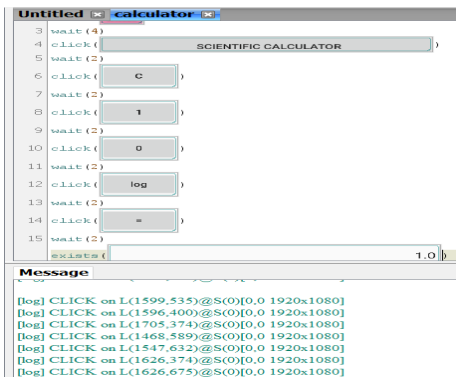
Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num0").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/log").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/equal").click();

Thread.sleep(200);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'1.0')]").isDisplayed());
```

```
File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
click("element2.png")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
exists("element6.png")
wait(2)
```



(a) Text file

(b) Sikuli Results

B.I Calculator App Test Script 9

```
driver.findElementById("anubhav.calculatorapp:id/button").click();

Thread.sleep(1000);
driver.findElementById("anubhav.calculatorapp:id/clear").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num9").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num0").click();

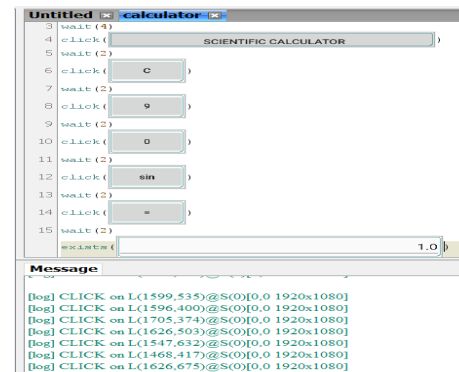
Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/sin").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/equal").click();

Thread.sleep(200);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'1.0')]").isDisplayed());
```

```
File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
click("element2.png")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
exists("element6.png")
wait(2)
```

(a) Text file



(b) Sikuli Results

B.J Calculator App Test Script 10

```
driver.findElementById("anubhav.calculatorapp:id/button1").click();

Thread.sleep(1000);
driver.findElementById("anubhav.calculatorapp:id/clear").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num5").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/plus").click();

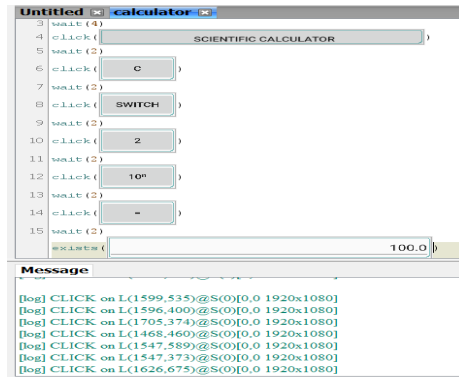
Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/num4").click();

Thread.sleep(200);
driver.findElementById("anubhav.calculatorapp:id/equal").click();

Thread.sleep(200);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'9.0')]").isDisplayed());
```

```
File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
click("element2.png")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
exists("element6.png")
wait(2)
```

(a) Text file



(b) Sikuli Results

C Calculator Application

C.A UnitConverter App Test Script 1

```
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerCategory").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Weight')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/input").sendKeys("1");

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsBase").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Pounds')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsResult").click();

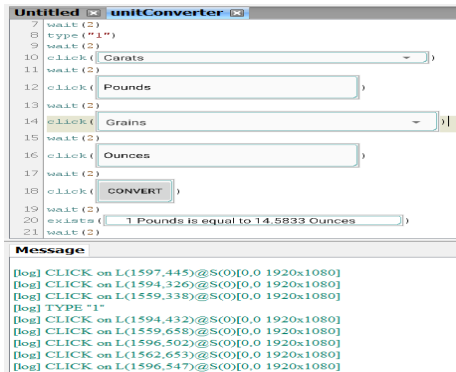
Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Ounces')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/btnConvert").click();

Thread.sleep(500);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'1 Pounds is equal to '
+ "14.5833 Ounces')]").isDisplayed());
```

```
File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
type("1")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
click("element6.png")
wait(2)
click("element7.png")
wait(2)
exists("element8.png")
wait(2)
```

(a) Text file



(b) Sikuli Results

C.B UnitConverter App Test Script 2

```

driver.findElementById("com.rcarvalho.unitconverter:id/spinnerCategory").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Weight')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/input").sendKeys("1");

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsBase").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Grains')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsResult").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Kilograms')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/btnConvert").click();

Thread.sleep(1500);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'1 Grains is equal to "
+ "0.0001 Kilograms')]").isDisplayed());

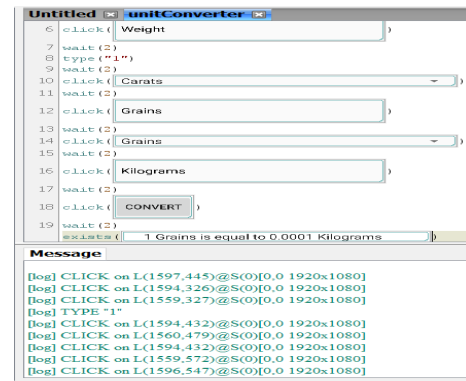
```

```

File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
type("1")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
click("element6.png")
wait(2)
click("element7.png")
wait(2)
exists("element8.png")
wait(2)

```

(a) Text file



(b) Sikuli Results

C.C UnitConverter App Test Script 3

```

driver.findElementById("com.rcarvalho.unitconverter:id/spinnerCategory").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Distance')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/input").sendKeys("1");

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsBase").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Yard')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsResult").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Feet')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/btnConvert").click();

Thread.sleep(1000);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'1 Yard is equal '
+ "to 3 Feet')]").isDisplayed());

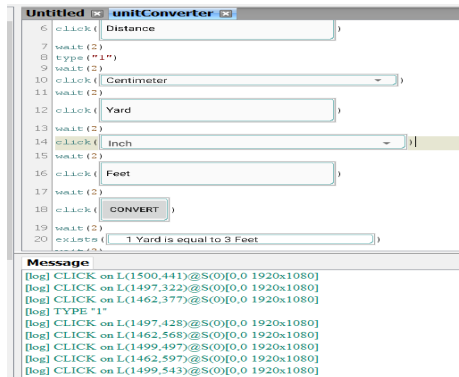
```

```

File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
type("1")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
click("element6.png")
wait(2)
click("element7.png")
wait(2)
exists("element8.png")
wait(2)

```

(a) Text file



(b) Sikuli Results

C.D UnitConverter App Test Script 4

```

driver.findElementById("com.rcarvalho.unitconverter:id/spinnerCategory").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Distance')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/input").sendKeys("1");

Thread.sleep(500);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsBase").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Mile')]").click();

Thread.sleep(500);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsResult").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Kilometer')]").click();

Thread.sleep(500);
driver.findElementById("com.rcarvalho.unitconverter:id/btnConvert").click();

Thread.sleep(1000);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'1 Mile is equal to "
+ "0.0485 Kilometer')]").isDisplayed());

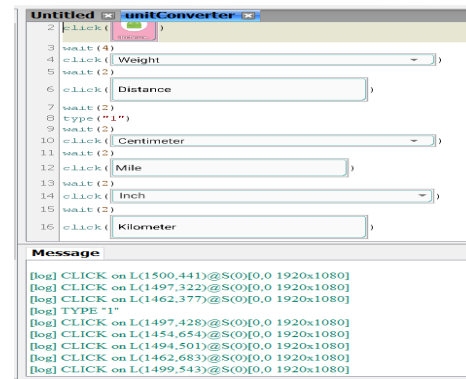
```

```

File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
type("1")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
click("element6.png")
wait(2)
click("element7.png")
wait(2)
exists("element8.png")
wait(2)

```

(a) Text file



(b) Sikuli Results

C.E UnitConverter App Test Script 5

```

driver.findElementById("com.rcarvalho.unitconverter:id/spinnerCategory").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[contains(@text,'Volume')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/input").sendKeys("1");

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsBase").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[contains(@text,'Liter')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsResult").click();

Thread.sleep(500);
driver.findElementByXPath("//*[contains(@text,'Milliliter')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/btnConvert").click();

Thread.sleep(1000);
assertTrue(driver.findElementByXPath("//*[contains(@text,'1 Liter is equal "
+ "to 1000 Milliliter')]").isDisplayed());

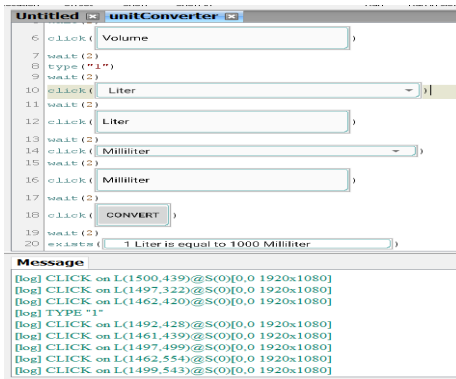
```

```

File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
type("1")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
click("element6.png")
wait(2)
click("element7.png")
wait(2)
exists("element8.png")
wait(2)

```

(a) Text file



(b) Sikuli Results

C.F UnitConverter App Test Script 6

```

driver.findElementById("com.rcarvalho.unitconverter:id/spinnerCategory").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[contains(@text,'Volume')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/input").sendKeys("1");

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsBase").click();

Thread.sleep(500);
driver.findElementByXPath("//*[contains(@text,'US Gal')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsResult").click();

Thread.sleep(500);
driver.findElementByXPath("//*[contains(@text,'Liter')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/btnConvert").click();

Thread.sleep(500);
assertTrue(driver.findElementByXPath("//*[contains(@text,'1 US Gal is equal "
+ "to 4.0582 Liter')]").isDisplayed());

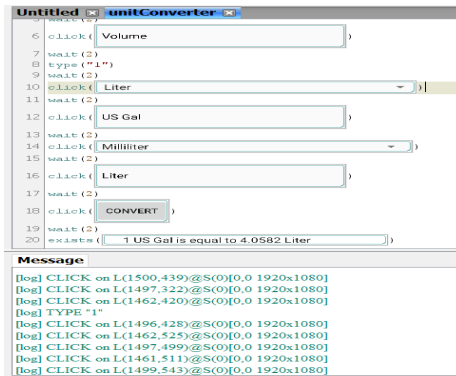
```

```

File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
type("1")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
click("element6.png")
wait(2)
click("element7.png")
wait(2)
exists("element8.png")
wait(2)

```

(a) Text file



(b) Sikuli Results

C.G UnitConverter App Test Script 7

```

driver.findElementById("com.rcarvalho.unitconverter:id/spinnerCategory").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[contains(@text,'Speed')]").click();

Thread.sleep(500);
driver.findElementById("com.rcarvalho.unitconverter:id/input").sendKeys("1");

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsBase").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[contains(@text,'km/h')]").click();

Thread.sleep(500);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsResult").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[contains(@text,'mile/h')]").click();

Thread.sleep(500);
driver.findElementById("com.rcarvalho.unitconverter:id/btnConvert").click();

Thread.sleep(1000);
assertTrue(driver.findElementByXPath("//*[contains(@text,'1 km/h is equal "
+ "to 0.6214 mile/h')]").isDisplayed());

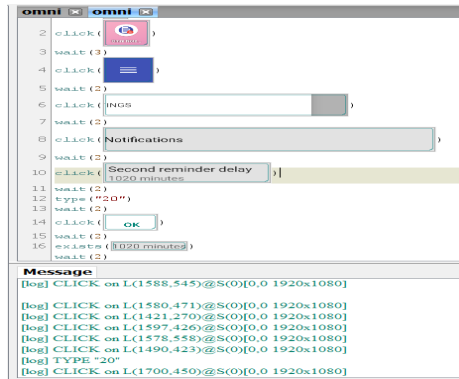
```

```

File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
type("1")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
click("element6.png")
wait(2)
click("element7.png")
wait(2)
exists("element8.png")
wait(2)

```

(a) Text file



(b) Sikuli Results

C.H UnitConverter App Test Script 8

```

driver.findElementById("com.rcarvalho.unitconverter:id/spinnerCategory").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[contains(@text,'Speed')]").click();

Thread.sleep(500);
driver.findElementById("com.rcarvalho.unitconverter:id/input").sendKeys("1");

Thread.sleep(500);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsBase").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[contains(@text,'knot')]").click();

Thread.sleep(200);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsResult").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[contains(@text,'km/h')]").click();

Thread.sleep(500);
driver.findElementById("com.rcarvalho.unitconverter:id/btnConvert").click();

Thread.sleep(1000);
assertTrue(driver.findElementByXPath("//*[contains(@text,'1 knot is equal "
+ "to 1.8520 km/h')]").isDisplayed());

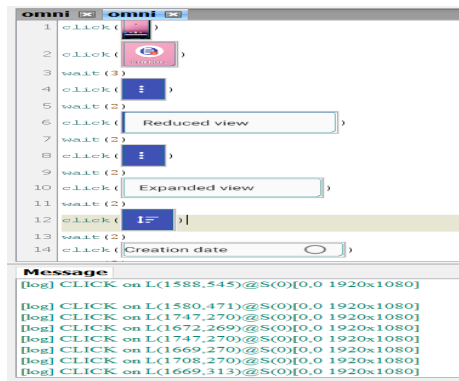
```

```

File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
type("1")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
click("element6.png")
wait(2)
click("element7.png")
wait(2)
exists("element8.png")
wait(2)

```

(a) Text file



(b) Sikuli Results

C.I UnitConverter App Test Script 9

```

driver.findElementById("it.feio.android.omninotes.alpha:id/fab_expand_menu_button").click();

Thread.sleep(300);
driver.findElementById("it.feio.android.omninotes.alpha:id/fab_checklist").click();

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/detail_title").sendKeys("First Checklist");

Thread.sleep(200);
driver.findElementById("it.feio.android.omninotes.alpha:id/menu_category").click();

Thread.sleep(300);
driver.findElementByXPath("//*[@contains(@text,'ADD CATEGORY')]").click();

Thread.sleep(1500);
driver.findElementByXPath("//*[@contains(@resource-id,'category_title')]").sendKeys("Work");

driver.findElementById("it.feio.android.omninotes.alpha:id/save").click();

Thread.sleep(200);
driver.findElementByXPath("//android.widget.ImageButton[@content-desc='drawer open']").click();

Thread.sleep(500);
assertTrue(driver.findElementById("//*[@contains(@text,'First Checklist')]").isDisplayed());

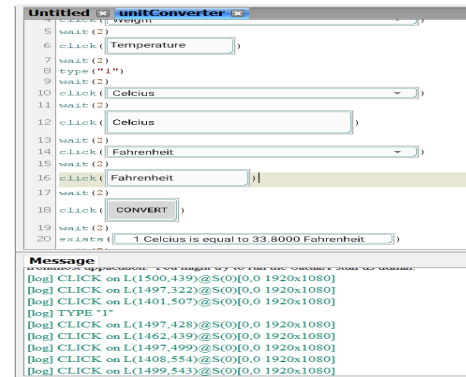
```

```

File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
type("1")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
click("element6.png")
wait(2)
click("element7.png")
wait(2)
exists("element8.png")
wait(2)

```

(a) Text file



(b) Sikuli Results

C.J UnitConverter App Test Script 10

```

driver.findElementById("com.rcarvalho.unitconverter:id/spinnerCategory").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Temperature')]").click();

Thread.sleep(500);
driver.findElementById("com.rcarvalho.unitconverter:id/input").sendKeys("1");

Thread.sleep(500);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsBase").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Fahrenheit')]").click();

Thread.sleep(500);
driver.findElementById("com.rcarvalho.unitconverter:id/spinnerUnitsResult").click();

Thread.sleep(1000);
driver.findElementByXPath("//*[@contains(@text,'Celcius')]").click();

Thread.sleep(500);
driver.findElementById("com.rcarvalho.unitconverter:id/btnConvert").click();

Thread.sleep(1000);
assertTrue(driver.findElementByXPath("//*[@contains(@text,'1 Fahrenheit is equal to "
+ "17.2222 Celcius')]").isDisplayed());

```

```

File Edit Format View Help
click("element0.png")
wait(2)
click("element1.png")
wait(2)
type("1")
wait(2)
click("element3.png")
wait(2)
click("element4.png")
wait(2)
click("element5.png")
wait(2)
click("element6.png")
wait(2)
click("element7.png")
wait(2)
exists("element8.png")
wait(2)

```

(a) Text file



(b) Sikuli Results