UMETRIX - Proof of Concept

Neeraj Mathur

January 2018

1 Umetrix Framework

Fig 1 describes usability evaluation framework.

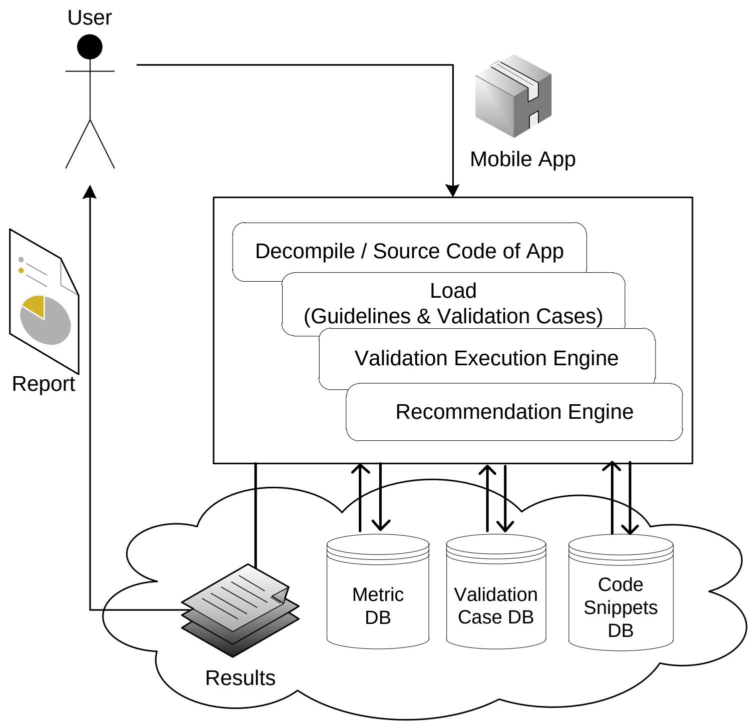


Figure 1: UMETRIX - Usability Evaluation Framework

2 Implementation - POC

In this section, we discuss the implementation of the tool for Android, based on our framework to evaluate the usability guideline using usability code-pattern analysis [?]. The implementation consists of two main components - Validation Case Generator and Guideline Evaluator.

Listing 1: Reveal or Hide Password

```
<android.support.design.widget.TextInputLayout
android:id="@+id/input_layout_password">
    <EditText android:id="@+id/txtPassword"
    android:inputType="textPassword"
    android:hint="Password"/>
</android.support.design.widget.TextInputLayout>
```

2.1 Validation Case Generator

The purpose of validation case generator is to provide a formal structure to validate the presence of Usability Code Pattern in a given mobile App's source code. For ease of usage, we provide a workflow designer which has the flexibility to design test case using drag and drop options instead of writing validation case manually. This helps non-programmers/testers to write validation case. If we consider an action as a Task, each task is supported with a toolbox that can be used to check a specific code structure. For example, flow/conditional steps (example FOR LOOP, WHILE LOOP, IF-ELSE, SWITCH), exception handling and nested condition checks for static code analysis.

We have built a prototype tool using Microsoft.NET to implement our framework. The details of extraction are provided in the next section. We have chosen Windows Workflow Foundation (WF) over other technologies as it provides the rich functionality of existing controls and ability to add your own custom logic specific to mobile app source code validation. Each workflow consists of activities that are small actions to detect specific code pattern. Each workflow corresponds to a validation case. A validation case plan is created consisting of validation cases. Fig **??** displays the validation case designer. It consists of three components as shown in Fig. 2

1. **Toolbox Usability Controls** - It lists the controls needed to create the Validation Case for a usability guideline. For example, *Custom Android Control* loads app related attributes and *Control flow* provides conditions and Error Handlers.
2. **Validation Case Work-flow** - This is a build area where the controls are added by using drag and drop options. The Work-flow of the validation case is created here.
3. **Arguments and Properties** - Input Variables and Constraints are defined here as arguments to evaluate the validation case against the source code of the mobile app.

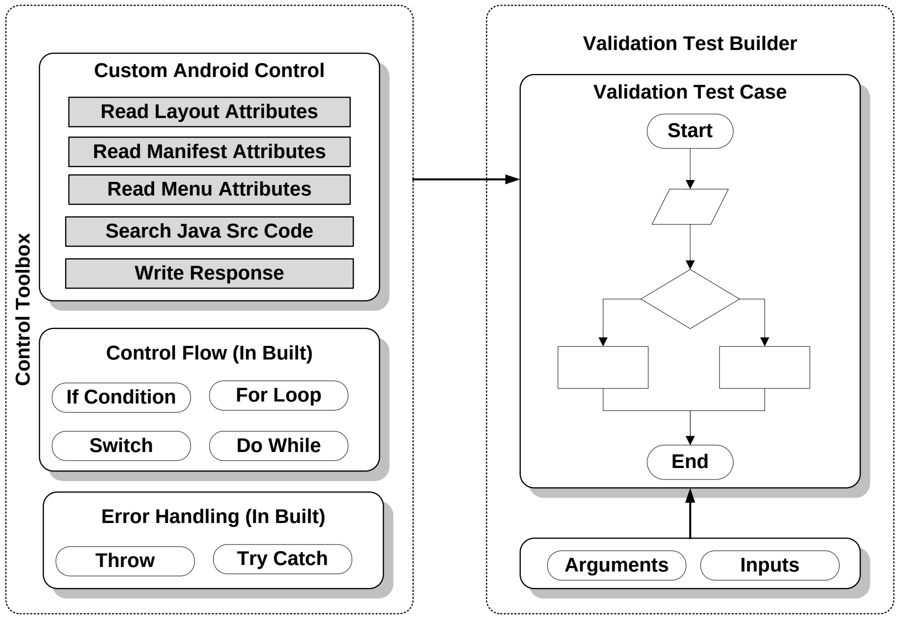
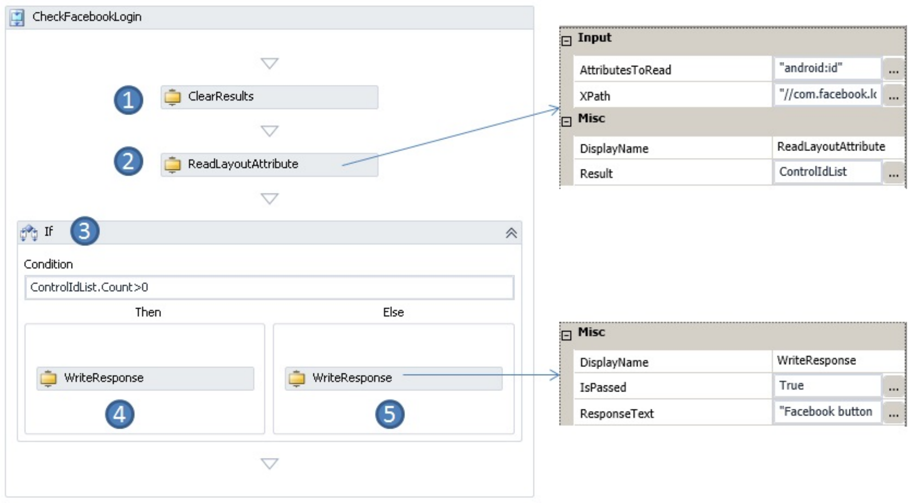


Figure 2: Validation Case Generator - Architecture

Fig 3 shows an example of a validation case to check the presence of social media login button (Facebook, Twitter, and Google to be specific) in the APK file. This guideline is loaded from the RuleDB to evaluate the presence of this feature in mobile app source code. Following are the details of steps specified in Fig 3.

1. **ClearResults** - Clears any available previous validation case results
2. **ReadLayoutAttributes** - To read all available layout (.XML) files and find the elements which matches the *xpath* defined in the properties section as input. Here for facebook login button, we will look for the Xpath which matches the string *"//com.facebook.login.widget.LoginButton"* and returns the value to "*android:id*" attribute in a variable called "FacebookButtonIdLists" so as to capture the response from validator step
3. **Condition** - It is a IF/ELSE checker used as part of condition check. We use the logic *"FacebookButtonIdLists.Count¿0"* to count if the value is greater than zero. This is an inspection service, which runs against the source code to generate the response.
4. **WriteResponse** - Success or Failure responses are posted based on the Response Text. If the *IsPassed* value is True, this step is executed
5. **WriteResponse** - If the *IsPassed* value is False, this steps is executed

The above validation case can be saved in a *.xml* file format so as to be re-used on future apps without re-building the entire validation case using the validation case designer. These *.xml* validation case files can be stored in the



Input	
AttributesToRead	"android:id"
XPath	"//com.facebook.lc
Misc	
DisplayName	ReadLayoutAttribute
Result	ControlIdList

Misc	
DisplayName	WriteResponse
IsPassed	True
ResponseText	"Facebook button

Figure 3: Example Validation Case (Check Facebook Button)

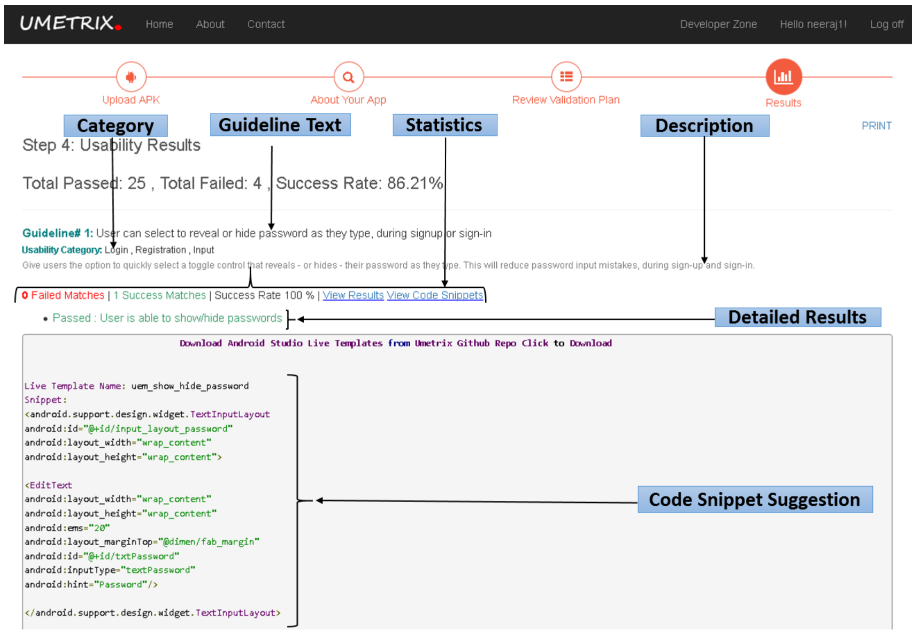
RuleDB along with other usability guidelines to load/unload the rules on a new app for usability evaluation.

2.2 Guideline Evaluator

The purpose of Guideline Evaluator is to facilitate a sequential process through which users can upload their mobile app and perform guideline based usability evaluation. It is a web-based tool as shown in Fig.4 for developers, testers and even for non-programmers. It has two modes - *Developer* and *Normal*. *Developer* - helps users to upload the .xml validation cases built using the Validation Case Generator. *Normal* - helps users to upload mobile app file, launch desired guidelines to evaluate, view results and code-pattern recommendation. The fig.5 shows the architecture of Guideline Evaluator.

Decompiler: We use a third-party decompiler to extract the source code artifacts like Layout XML, JAVA code, Manifest files etc. from the mobile app. We decompile the APK file using open source tools like *apktool, dex2jar & JD-core*. *APK tool* will provide an output of activity layouts XML which contains the details of mobile app layouts in XML format and the rest of the code in Java classes file called .dex files. These classes.dex files are converted to JAR file with the help of a third party *dex2jar tool*. The *jd-core* tool is used to extract java classes from the generated JAR file. The XML layout file and the Java Classes files are used as input for evaluation.

Databases: We maintain 3 databases for Web evaluation. *Validation Test Cases database* to store and maintain validation cases built using validation case generator. *Validation Plan Database* to store the testing plan with usability guideline and its associated test cases. *Metric database* to store evaluation



Step 4: Usability Results

Total Passed: 25 , Total Failed: 4 , Success Rate: 86.21%

Guideline# 1: User can select to reveal or hide password as they type, during signup or sign-in

Usability Category: Login , Registration , Input

Give users the option to quickly select a toggle control that reveals - or hides - their password as they type. This will reduce password input mistakes, during sign-up and sign-in.

0 Failed Matches | 1 Success Matches | Success Rate 100 % | View Results View Code Snippets

- Passed : User is able to show/hide passwords

```
Download Android Studio Live Templates from Umetrix Github Repo Click to Download


Live Template Name: uem_show_hide_password
Snippet:
<android.support.design.widget.TextInputLayout
android:id="@+id/input_layout_password"
android:layout_width="wrap_content"
android:layout_height="wrap_content">

<EditText
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:ems="20"
android:layout_marginTop="@dimen/fab_margin"
android:id="@+id/txtPassword"
android:inputType="textPassword"
android:hint="Password"/>

</android.support.design.widget.TextInputLayout>
```

Figure 4: Guideline Evaluator - UI

results and reports for future analysis.

Validation Execution Engine: It executes validation plan and generates validation results based on presence or absence of a usability guideline as shown in Fig.6. The result contains the Guideline Number, Guideline details including Guideline text and it's description along with the usability category. If the guideline fails to identify the code-pattern, it returns *1 Failed Matches* as a response. If the validation case is successful, it returns *1 Success Matches*. Similarly if there are multiple success/failure matches for validation case instance(s), it will return the respective results. It also provides *Success Rate* in percentage. This Success rate percentage is an aggregate value of all the validation cases executed in one instance. We could see that there are 3 validation cases listed in the fig 6 which are passed together with detailed results.

Recommendation Engine: It uses Lucene .NET Search engine to index code snippets from a central repository and provides matching code pattern suggestions to guidelines which are absent in the given mobile app as shown in Fig.4 as code snippet suggestion.

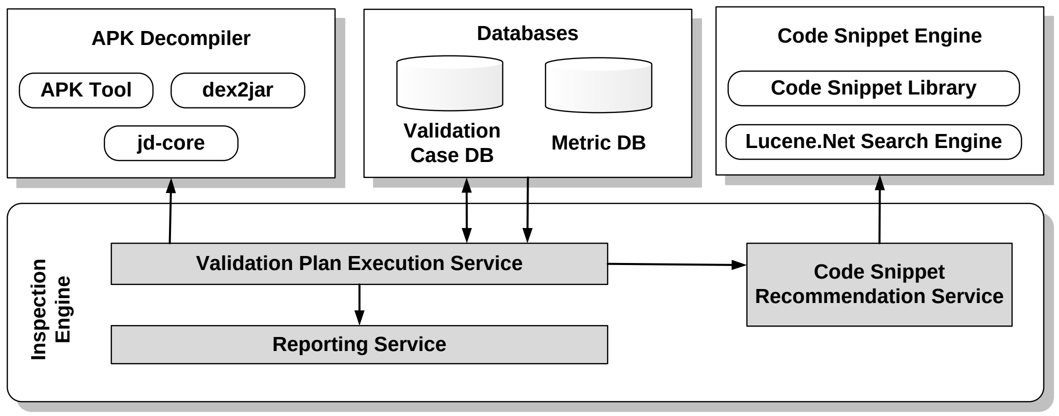
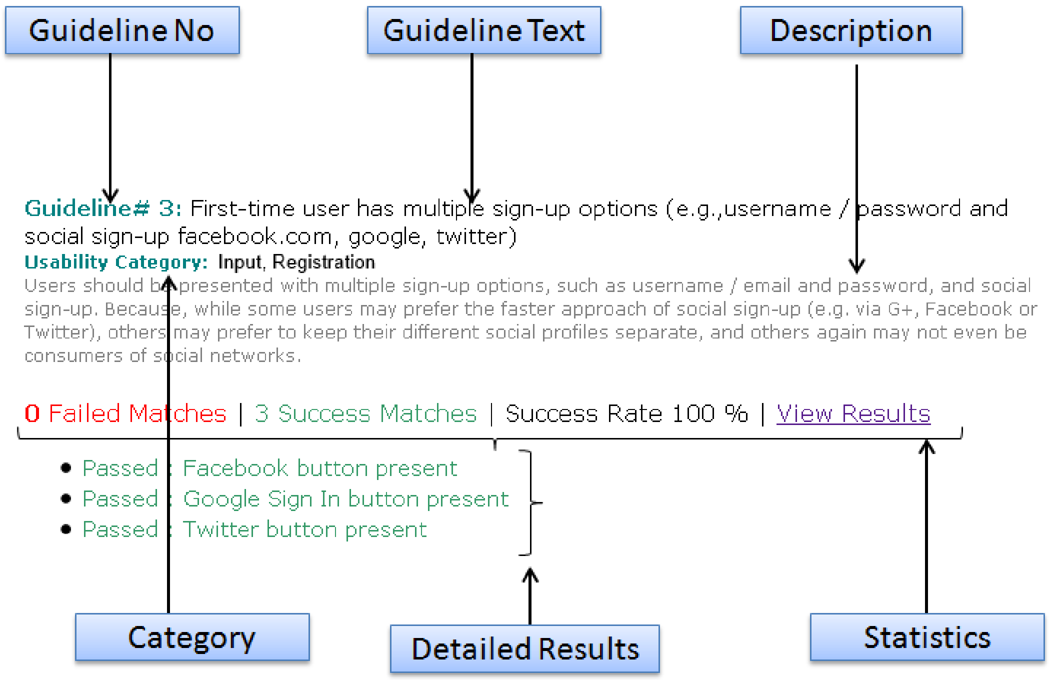


Figure 5: Guideline Evaluator - Architecture



Guideline# 3: First-time user has multiple sign-up options (e.g.,username / password and social sign-up facebook.com, google, twitter)

Usability Category: Input, Registration

Users should be presented with multiple sign-up options, such as username / email and password, and social sign-up. Because, while some users may prefer the faster approach of social sign-up (e.g. via G+, Facebook or Twitter), others may prefer to keep their different social profiles separate, and others again may not even be consumers of social networks.

0 Failed Matches | 3 Success Matches | Success Rate 100 % | View Results

- Passed : Facebook button present
- Passed : Google Sign In button present
- Passed : Twitter button present

Figure 6: Usability Validation Test results