Attribute Extraction Method for
GUI-Usability Evaluation of Android Applications

Masanori OHGITA†, Masanao KOEDA†

† Osaka Electro-Communication University, Shijonawate, Osaka, JAPAN
mt15a001@oecu.jp, koeda@osakac.ac.jp

Abstract. Today, a huge number of applications with graphical user interfaces (GUIs) are available on various smartphones. However, it is difficult to design a user-friendly GUI for everyone. We aim to provide appropriate improvement tips for the developers of GUI applications on smartphones. The GUI usability varies according to the characteristics of each application. This study describes a method to extract the attributes from Android GUI applications. We then use machine learning to look for hidden design models in excellent GUI applications.

Keywords. Android, GUI, User Interface, Usability

1 Introduction

Universal Design is required for all applications with a graphical user interface (GUI). In addition, application development and publishing have become easy for everyone. However, inexperienced developers who have little knowledge of UI design may find it difficult to make good applications with useful user interfaces. The typical GUI evaluation method is a questionnaire survey [1] or self-testing by the developer. However, these approaches are difficult for beginning developers to apply.

In this study, we present a GUI evaluation system that automatically evaluates an application and offers suggestions to improve its usability. By using a machine-learning method, we expect to extract hidden and common key concepts from excellent GUI applications. These extracted concepts may not be mentioned in well-known GUI design guidelines. We present a system that proposes a much more useful design, suited to the characteristics of the application, based on the extracted results.

Previously, we proposed a usability-evaluation method that did not depend on the existing guidelines [2]. Additionally, we developed an original GUI analysis method, called “Static Analysis,” which could process at high speed, and we analyzed several applications using this method. However, it required manual pre-processing, and was not suited to analyzing a huge number of applications. In this paper, we present a new system with improved analysis methods.
2 Towards the System for Usability Improvement

In our study, we apply general concepts for user-interface design. The first one is "The Four Principles of Design," which is an important concept in general design. The principles consist of proximity, repetition, contrast, and alignment. They are used in many GUI applications. Fig. 1 shows an example of the interface design of a GUI application. The image under the mouse pointer is colored to identify it as the selected one. However, this is not always the best methods. There are many other ways to improve a GUI; e.g., expanding the GUI component size, etc. A more flexible usability-improvement method is needed to consider the design and features of each application.

Another concept is "GUI Design Guidelines." Well-known platforms generally provide guidance to design the GUI for an application; e.g., "Android Design" [3] for Android OS as the smart-device platform, "iOS Human Interface Guidelines" [4] for iOS, etc. By taking rules from these guidelines, we may automatically evaluate the usability of a GUI. However, evaluation results based on the guidelines could become uniform and monotonous.

To automatically evaluate the GUI usability and suggest improvements, we execute the following steps. At this time, Step 1(a) and 2 are realized. The entire process runs automatically using a Perl script that we developed.

1. Extract the GUI attributes and the improvement methods
   (a) Attribute analysis - Attribute extraction that later mentioned in chapter 3.
   (b) Behavior analysis (improvement-method extraction)
2. Attribute classification (GUI feature extraction using machine learning)
3. Cluster the features and improvement methods

Fig. 1. Example of Item Selection in a GUI

3 GUI Attribute Analysis

In the attribute analysis of this study, we converted a GUI structure to the tree structure described below. We adopted the Android application environment because the specifications have been published clearly.
3.1 Application Analysis and GUI Tree Generation

Firstly, we obtained the package file (APK) of the target application, and installed it on an Android Virtual Device (AVD). Then, we developed a customized version of the "Hierarchy Analyzer," which analyzes the GUI-application structure. Using this tool, we obtained the GUI tree shown in Code 1 that contain the parameters of that GUI part (View). Then, captured images are shown in Fig. 2 from each view.

The original version of the Hierarchy Viewer is included in the "Android Software Development Kit (SDK)" and embedded in the Integrated Development Environment (IDE) as a plug-in. Therefore, it must be used in the IDE and operated manually. Our customized version has the following advantages over the original version.

- Simple build process. It extracted the minimum required projects from the Android SDK and Development Tools (ADT).
- Automatic process using command-line options
- GUI tree is exported in the JavaScript Object Notation (JSON) file format
- Views of each GUI are captured and the images saved in the Portable Network Graphics (PNG) file format

```
{
  "zero":{
    "image": "/tmp/gui-refine-cache/latele.nekocalc/zero.png",
    "name": "android.widget.Button",
    "property_drawing:getAlpha()": "1.0",
    "property_drawing:getRotation()": "0.0",
    "property_drawing:getSolidColor()": "0",
    "property_drawing:getX()": "0.0",
    "property_drawing:getY()": "99.0",
    "property_layout:getBaseline()": "69",
    "property_layout:getHeight()": "99",
    "property_layout:getWidth()": "118",
    "property_layout:mBottom": "198",
    "property_layout:mLeft": "0",
    "property_layout:mRight": "118",
    "property_layout:mTop": "99",
    "property_scrolling:mScrollX": "524229",
    "property_scrolling:mScrollY": "0",
    "property_text:getTextSize()": "52.0",
    ...
  },
  ...
}
```

Code 2. GUI tree generated by our Hierarchy Analyzer
3.2 Dataset Generation from the GUI Tree

To extract the attributes from the GUI tree generated in the previous section, the attribute value of each view is calculated. A dataset is generated from each view obtained from the GUI tree, using the following Perl module called the "Dataset Generator." By repeating these steps, each module makes a dataset of each window. We implemented the following modules.

- General-purpose dataset generator:
  GUIRefine::DataSetGenerator::Android::General
- Dataset generator for each view type:
  GUIRefine::DataSetGenerator::Android::Button (example)

The former generator will generate attributes for all views. There are contains the following attributes. In addition, some attributes about color will be generated from capture-image of Fig. 2.

- numColors: Number of colors on the screen
- mostUsedColor: Most used color(s) on the screen
- ...

The latter generator will generate attributes for each view type. Following is an example for the Button view.

- num: Number of buttons on the screen
- minWidth: Width of the smallest button on the screen
- minHeight: Height of the smallest button on the screen
- minDistTop: Distance from the top of the screen to the button
- minDistLeft: Distance from the left edge of the screen

4 Experiment

We conducted an experiment to confirm the feature extraction from GUI applications. Specifically, we analyzed the attributes from multiple applications listed in Table 1. We compared the attribute-value trends that were output by the "Random Forests" algorithm using the R language. The target application in this experiment was a simple calculator; specialized calculators were excluded (e.g., hex number converters).

Each application are compared using the attributes in Table 2. The “Name”, “Author”, “Version”, and other column of each application in that table were acquired from the Google Play Store in Japanese or English in Dec. 2015. The “Assign” col-
umn is a classification variable in this experiment. A value of “A” means the application is easy to use; a value of “B” means it is hard to use. For this classification, we referred to the evaluation scores and review comments of each application in the Google Play Store. In this experiment, we used only the startup screen of each target application.

The results of this experiment are shown in Fig. 3. That figure shows the graph of the frequency distribution of the eight selected attributes of all 27 attributes from the GUI tree. That graph shows the trend of each attribute in each application using the box-and-whisker plot, and the X-axis in each inner graph shows the attribute value that described in Table 2.

In our previous study [2], the target application was not focused and various applications for different purposes were analyzed together. In addition, the previous method could only analyze simple applications, and the target applications used for analysis were quite few. By using the new tool, we can analyze many more applications and focus on a specific application type. We could not extract the trends and guidelines for this target application. The reason is under investigation.

Table 1. Experimental Applications (18 items)

<table>
<thead>
<tr>
<th>Name</th>
<th>Author</th>
<th>Ver.</th>
<th>Score</th>
<th>Assign</th>
<th>Mention to Usability in User Review</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculator</td>
<td>andanapps</td>
<td>2.0.5</td>
<td>4.1</td>
<td>A</td>
<td>- / -</td>
</tr>
<tr>
<td>Calculator Pro free</td>
<td>Apalon Apps</td>
<td>2.0</td>
<td>4.1</td>
<td>A</td>
<td>2 / 5</td>
</tr>
<tr>
<td>CALCU Free</td>
<td>Designer Calculators</td>
<td>2.1.0</td>
<td>4.5</td>
<td>A</td>
<td>- / -</td>
</tr>
<tr>
<td>Calculator Plus Free</td>
<td>Digitalchemy, LLC</td>
<td>4.9.2</td>
<td>4.4</td>
<td>A</td>
<td>23 / 2</td>
</tr>
<tr>
<td>Simple Calculator</td>
<td>Kugelschreiber</td>
<td>3.0</td>
<td>4.1</td>
<td>A</td>
<td>- / -</td>
</tr>
<tr>
<td>LateCalc</td>
<td>Latele Inc.</td>
<td>1.5.0</td>
<td>3.6</td>
<td>B</td>
<td>5 / 3</td>
</tr>
<tr>
<td>Nekocalc</td>
<td>Latele Inc.</td>
<td>1.3.0</td>
<td>4.3</td>
<td>A</td>
<td>1 / -</td>
</tr>
<tr>
<td>Takano tsume Calculator</td>
<td>Loop-Sessions LLC</td>
<td>1.31.0</td>
<td>3.8</td>
<td>B</td>
<td>- / -</td>
</tr>
<tr>
<td>Colorful calculator</td>
<td>R225.zero</td>
<td>1.6</td>
<td>3.9</td>
<td>A</td>
<td>23 / -</td>
</tr>
<tr>
<td>Panecal</td>
<td>Appsys</td>
<td>6.0.2.2</td>
<td>4.1</td>
<td>A</td>
<td>3 / 1</td>
</tr>
<tr>
<td>TigerCalc</td>
<td>tripot</td>
<td>1.1</td>
<td>3.4</td>
<td>B</td>
<td>- / -</td>
</tr>
<tr>
<td>RealCalc</td>
<td>Quartic Software</td>
<td>2.3.1</td>
<td>4.5</td>
<td>A</td>
<td>- / -</td>
</tr>
<tr>
<td>Cho Simple Calculator 2</td>
<td>Nobuo CREATE</td>
<td>1.3</td>
<td>3.9</td>
<td>B</td>
<td>- / -</td>
</tr>
<tr>
<td>Pretty Calculator</td>
<td>Pretty Pet Co</td>
<td>1.1.3v</td>
<td>4.0</td>
<td>A</td>
<td>- / -</td>
</tr>
<tr>
<td>Oto-san Calculator</td>
<td>SoftBank Corp.</td>
<td>3.0.0</td>
<td>3.6</td>
<td>B</td>
<td>1 / 3</td>
</tr>
<tr>
<td>Calculator</td>
<td>7th Gear</td>
<td>1.0</td>
<td>3.9</td>
<td>B</td>
<td>- / -</td>
</tr>
<tr>
<td>Calculator</td>
<td>TricolorCat</td>
<td>1.10.7</td>
<td>4.1</td>
<td>A</td>
<td>8 / 2</td>
</tr>
<tr>
<td>Calculator</td>
<td>Woodsmall inc.</td>
<td>1.6.5</td>
<td>4.0</td>
<td>A</td>
<td>13 / 3</td>
</tr>
</tbody>
</table>
Table 2. Comparison Attributes in Dataset (Excerpt)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button.distLeft</td>
<td>Distance from the screen left edge to Button</td>
<td>Pixels (px)</td>
</tr>
<tr>
<td>Button.distTop</td>
<td>Distance from the screen top to the Button</td>
<td>Pixels (px)</td>
</tr>
<tr>
<td>Button.minHeight</td>
<td>Height of the smallest Button</td>
<td>Pixels (px)</td>
</tr>
<tr>
<td>Button.minWidth</td>
<td>Width of the smallest Button</td>
<td>Pixels (px)</td>
</tr>
<tr>
<td>Button.num</td>
<td>Number of Buttons on the screen</td>
<td>Pieces</td>
</tr>
<tr>
<td>EditText.distLeft</td>
<td>Distance from the screen left edge to EditText</td>
<td>Pixels (px)</td>
</tr>
<tr>
<td>EditText.distTop</td>
<td>Distance from the screen top to the EditText</td>
<td>Pixels (px)</td>
</tr>
<tr>
<td>EditText.minHeight</td>
<td>Height of the smallest EditText</td>
<td>Pixels (px)</td>
</tr>
</tbody>
</table>

Fig. 3. Experimental Results: Frequency Distribution (Excerpt)
5 Conclusions

We proposed a system that suggests methods to improve a GUI by evaluating many high-rated GUI applications. We used machine learning, which is not bound by the GUI Design Guidelines. We developed the "Hierarchy Analyzer," analyzed the attributes and extracted the features from GUI applications. Specifically, the system analyzed a running application on a virtual machine. To make a dataset for evaluation, the system obtained a GUI tree from the running application using the "Hierarchy Analyzer," and calculated the attributes from the GUI tree.

We can easily analyze many applications using the new method. We can also consider the application licenses, because our method does not need to de-compile the application. We hope to reduce the burden of all developers by continuing this study.

References