Technical Report No. NJU-SEG-2019-IC-003

2019-IC-003

Extracting Mapping Relations for Mobile User Interface Transformation

Ruihua Ji, Junyu Pei, Wenhua Yang, Juan Zhai, Minxue Pan, Tian Zhang

Internetware 2019

Extracting Mapping Relations for Mobile User Interface Transformation

Ruihua Ji[†], Junyu Pei[†], Wenhua Yang[‡], Juan Zhai[†], Minxue Pan[†*], and Tian Zhang[†*]
[†]State Key Laboratory for Novel Software Technology, Nanjing University, China
[‡]College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China
yiting.ji@gmail.com, imbarpjy@163.com, ywh@nuaa.edu.cn, {zhaijuan, mxp, ztluck}@nju.edu.cn

ABSTRACT

The development of mobile apps has become the current mantra for any business' success. The rise of many types of mobile devices and mobile OS has instantly created the need to develop multiple versions for the same app. In order to grasp as much market share as possible, it is desirable to have all the versions of an app demonstrate similar user interface (UI) appearances, to make users feel comfortable when switching from one platform to another and more likely to stick to the app. However, to ensure consistent UIs among cross-platform versions can be a challenging and costly endeavor, since different platforms have their own UI controls and programming languages. In this paper, we propose an automatic approach to transforming mobile app UIs across platforms, and illustrate our approach by transforming the UIs of iOS apps to Android ones. We leverage the enormous existing apps carefully designed by developers to achieve similar UI effects between iOS and Android versions, since these apps contain valuable knowledge of mapping relations between the iOS and Android UI controls. Starting from the reverse engineering of these apps, our approach separates each user interface into modules of adequate sizes. Then it maps the modules from both versions that contribute to the same visual and functional effect, and automatically mines the mapping relations. By applying the mined relations, our approach has successfully transformed the iOS app UIs into Android app UIs, as confirmed by a series of experiments.

CCS CONCEPTS

• Software and its engineering

KEYWORDS

Mobile applications, Cross-platform development, Graphical user interface, Code transformation

* Corresponding author

ACM Reference format:
Ruihua Ji, Junyu Pei, Wenhua Yang, Juan Zhai, Minxue Pan, and Tian Zhang. 2019. Extracting Mapping Relations for Mobile User Interface Transformation. In *Internetware '19: Proceedings of 11th Asia-Pacific Symposium on Internetware (Internetware '19), October 28-29, 2019, FuKuoka, Japan.* ACM, New York, NY, USA, 10 pages.
https://doi.org/10.1145/3361242.3361250

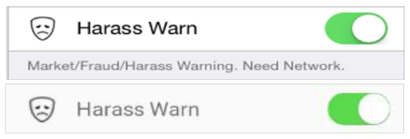
1 Introduction

The smartphone market has grown rapidly in recent years. This growth has fueled the demand for a wide range of new mobile apps by both consumers and businesses. Mobile apps have become a vital part of our lives, and contributed a large portion of profit to IT industry. Currently, there are several mobile platforms in the market, such as iOS, Android and BlackBerry OS. To attract more subscribers, companies are obliged to develop multiple versions for one app. The cross-platform development, particularly between Android and iOS, receives the most attention, since these two are currently the dominant platforms. Developers have to invest considerable effort in cross-platform development: different mobile platforms come with their own software development kits, use different programming languages, and provide custom APIs.

To address this challenge, developers and researchers have been working hard for effective cross-platform developing methods. For example, Xamarin [1] can compile simple C# programs to codes that can be executed on iOS or Android. J2objc [2] provides interfaces that can execute some Java codes in iOS system. Unfortunately, they are still preliminary and cannot handle app graphic user interfaces (GUIs) well. For mobile apps, GUIs are extremely important, since they directly interact with users. In order to attract more users, mobile apps must be GUI-friendly, or even artistic. In cross-platform development, it is desirable to have all the versions of an app demonstrate similar GUI appearances, so users will feel comfortable when switching from one platform to another and be more likely to stick to the same app. However, to ensure consistent UIs among cross-platform versions can be a challenging and costly endeavor, since different mobile platforms have their GUI controls. Only experienced developers who have good knowledge about different platforms can write cross-platform apps that have consistent GUIs.

One particular method is proposed to bypass this problem. Since all mobile platforms support Web pages, it is possible to rely on the system's Web browser to provide the GUIs. With this method, the app embeds a Web view that renders HTML codes, which then can display browser elements such as HTML buttons and input fields. In this way, developers can achieve a uniform look and feel across platforms. Representatives of this approach include Adobe PhoneGap [3], Cordova [4] and Sencha [5], etc. However, the display of Web elements cannot be loaded as fast as native GUI controls, not to mention that the Web-based method suffers more security issues since malicious scripts are more likely to be injected.

Another method employs the technique of GUI transformation. For example, the commercial tool Myappconverter [6] can convert each GUI control from iOS to Android, using fixed mapping relations between controls. Its official document [7] shows some of these mapping relations, e.g. *UILabel* in iOS is mapped to *TextView* in Android (*UILabel* and *TextView* mentioned here are GUI controls). This method can work for some apps. However, controls can be composed in flexible ways, and those fixed relations may not be able to handle complex GUIs. Here is an example. Fig. 2 shows the real code snippets of a page in WhoCall from iOS and Android versions. Fig. 2(a) contains one *imageView*, one *label* and one *switch*, as shown in Fig. 1(a). Besides, there is one text line in the bottom, which is the result of the attribute *footerTitle* of *tableViewSection* in line 1 that shows explanation text for the view section. Using the control-corresponding table given by Myappconverter, the code in Fig. 2(a) is transformed into its Android version in Fig. 2(b) (convert *tableViewSection*, *tableViewCell* and *tableViewCellContentView* to *FrameLayout*, *imageView* to *ImageView*, *label* to *TextView*, and *switch* to *ImageView*). Unfortunately, the generated Android code loses the bottom text, as shown in Fig. 1(b), since there is no rule to convert the *footerTitle* attribute. A new rule, which requires an extra GUI control *TextView*, is needed to make the transformed Android code have the similar appearance to the original iOS one. It is impractical to write all the rules manually, and an automatic


Harass Warn

Market/Fraud/Harass Warning. Need Network.

(a)

Harass Warn

(b)

Fig. 1: (a) Whocall iOS GUI (b) Android GUI with controls converted from iOS one by one

```
1 <tableViewSection
  footerTitle="Market/Fraud/Harass.Need Network.">
2  <tableViewCell id="Ork-z8-AXP">
3   <tableViewCellContentView id="P8K-xu-Nrw">         (a)
4    <imageView image="icon_spam.png" id="10h-lu-bEP">
5    <label text="Harass Warn" id="2e4-i3-bXw">
6    <switch id="QJC-Y7-LTP">
-----------------------------------------------------------------
1 <FrameLayout
2  <FrameLayout
3   <ImageView android:src="@drawable/icon_spam" />
4   <TextView android:text="Harass Warn" />
5   <ImageView android:src="@drawable/myswitch" />       (b)
6  </FrameLayout>
7 </FrameLayout>
8 <TextView
  android:text=" Market/Fraud/Harass.Need Network." />
```

Fig. 2: (a) iOS code snippet in Whocall (b) Android code snippet in Whocall

rule inference method can be useful.

In this paper, we propose an automatic data-driven approach to transforming mobile app GUIs, and illustrate our approach by transforming the GUIs from iOS to Android. We observed that enormous existing apps having both versions on iOS and Android platforms are carefully designed by developers to achieve similar UI effects. We collected 125 apps from Apple Store, consisted of the top 5 apps from 25 different categories, and downloaded these apps' Android versions from Google Play Store, to evaluate the GUI similarities between the two versions of an app. Among the 125 apps, 97 were coded with native GUI controls and thus could be served as experiment subjects, and the others were written in non-native GUI codes. Within the 97 apps, 71 apps (93.8%) have similar GUI pages, i.e., the same texts and images are placed in the same positions of the screens on both versions. To obtain the mapping relations of GUI controls, our approach reverse-engineers the apps, and get their GUI related codes. Then it separates the GUI controls in each screen into adequately sized modules. With a mapping strategy, it maps the modules from different versions that contribute to the same visual and functional effect, and automatically records the mapping relations of modules. To transform a given iOS app GUI, the approach divides its GUI into modules, and then for each module it leverages the mapping relations to generate the corresponding Android app GUI module. By composing the generated modules, our approach can transform the iOS app GUI to Android app GUI effectively, as confirmed by the case studies.

The contributions of this paper are summarized as follows:

1) We propose a novel approach to obtaining the mapping relations between native GUI controls on iOS and Android platforms from existing apps;
2) We propose an automatic approach which can transform the GUIs of iOS apps to Android apps while assure GUI consistency;
3) We implemented our approach as a prototype tool and generated over 1426 mapping rules. The case studies show that our approach can successfully transform iOS app GUI to Android ones.

The remainder of this paper is organized as follows: Section 2 presents the overview of our approach. In Section 3, we show how to map the GUI controls across platforms, and in Section 4, we give the details about how to transform GUIs of iOS apps to Android apps. Section 5 evaluates our work with real-world cases. Section 6 reviews related work and Section 7 concludes the paper and discusses future work.

2 Approach Overview

Our approach aims to discover the mapping relation between iOS and Android GUI controls, and use the knowledge of the mapping relations to convert the GUI of an iOS app (specifically, the Storyboard files and xib files created by Xcode producing the GUI of one iOS app [10]) into layout code of an Android app. The reason choosing iOS as the source platform and Storyboard or xib files as the source code is that: (1) these files are in XML format, where controls and attributes all are arrayed in forms

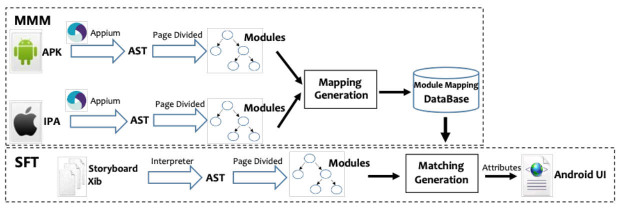


Fig. 3: The overall architecture of our approach

and are easy to be analyzed; and (2) we find that most iOS attributes affiliated to GUI controls have counterparts in Android, and thus iOS pages are more likely to be fully transformed.

The overview of our approach is illustrated in Fig. 3. The approach consists of two parts: one is for mining mapping modules (MMM), and the other is to transform iOS StoryBoard files to Android GUI code (SFT). The top half of the Fig. 3 shows the MMM process, which is to extract mapping relations between iOS and Android apps, in terms of modules that contain a set of GUI controls. As discussed earlier, there are abundant apps sharing similar GUI appearance on iOS and Android platforms. Generally, different controls need to be combined together to make these cross-platform apps look similar. The types of controls to use and the way to compose them are the key knowledge for developers in cross-platform development. To extract the knowledge and organize them into mapping relations, we need to analyze the usage of controls for apps in iOS and Android. We first use Appium [8] to decompile the Android and iOS versions of the apps and then encode the generated GUI controls and their relations into the form of trees. A tree represents a GUI page consisting of a set of controls. The relations between controls and the essential displaying attributes (e.g., attribute *footerTitle* showing a line of texts as a footer note) are also included in the tree. Then we divide iOS trees into modules of adequate size, with which we look for an Android module that contains the counterpart controls, i.e., controls in the same position showing similar visual contents. As the two versions of apps have similar appearance, it is safe to assume that the modules containing similar visual elements of the two versions are likely to be found. The mapping relations between the modules in apps from different platforms are recorded as rules, which are stored in a database, prepared for GUI transformation.

The bottom half of Fig. 3 shows the SPT part, which is responsible for the transformation of the Storyboard and xib files in an iOS app to Android code, using the module mapping database. First, the source storyboard and xib files are analyzed and encoded into a set of trees, which are divided into modules in the same way in the MMM process. Then, we search the database for each module and use the mapped Android module in the mapping relation as the template, from which we create the native code of the Android GUI. Finally, for the generated Android module, the necessary control attributes, such as the positions and colors, are generated using the information obtained from analyzing Storyboard and xib files.

To make our approach effective, we need to carefully weigh the impacts of two key points. One is the number of controls one module should contain when dividing pages. Having too few controls in one module may lead to the loss of information. For example, in Fig. 2(a), dividing the *tableViewSection* (line 1) and the *Label* (line 5) into different modules lose the relationships between the two controls and make the generation of the footnote text impossible. Meanwhile, if one module contains too many controls, it can be difficult to use the mapping relations, since different apps have a smaller chance of having larger similar modules. The other is the ways to exploit the extracted mapping relations, and the ways to deal with different attributes affiliated to the controls. The iOS and Android platforms are different in managing controls' attributes, which could lead to different layout forms. For example, for control positioning, iOS employs a unified positioning approach using a tuple (x, y, height, width) to specify the left-corner position and size of a control, while Android controls' positioning can be represented by attribute *left* (absolute position) or attribute *layout_marginLeft* (relative position). The former works with *FrameLayout*, and the latter works with *RelatedLayout*.

3 Mapping Module Mining

In this section, we will define the concept of module in our approach and introduce how to mine the GUI module mappings between iOS and Android. Briefly, a module is in the form of a tree consisting of several controls, attributes, and their relationships. Our module mapping process takes apps of both iOS and Android versions as input. For each app, we construct layout trees for each screen of both versions. Then we separate the layout trees into several modules. As the two versions of one app have similar GUI appearances, we can identify module mappings between iOS and Android, which can be used as relations to transform an iOS screen into an Android one.

In our approach, we divide all the controls listed in the developer webs [9, 10] into two categories: *function control* and *container control*. A function control is a control that cannot have any child node in the layout, and is visible on screen. It is used to interact with users, give messages about words/pictures, and/or trigger events. Common function controls include *label*, *textView* and *imageView*, etc. The controls not belonging to function controls are in the container control category, which are used to place function controls or other container controls. They must have child nodes, and are usually invisible by default. However, we also notice that sometimes iOS container controls can also have some visible functions, such as the *tableView-Section* in Fig. 2(a) that provides the functionality of showing footnotes. These kinds of features further show that mapping controls one to one from the two platform is infeasible, and a more complex mapping is needed.

Since the apps in different platforms are expected to work in similar appearance, our intuition is that the layout trees of the same screens should be mappable. To have an accurate mapping, each layout tree is divided into several subtrees that do not share any controls. These subtrees are defined as modules in our approach and are used as basic units for the mapping.

Definition 3.1. Module A module is a tree-form structure made up of nodes and edges. A module consists of a root node and potentially many levels of additional nodes that form a hierarchy. A module is donated by a 4-tuple <R, N, E, D> where,

1) R is the root node of this module which is the control containing all the other nodes in the module. At the same time, R can be a child of another node outside the module.
2) N is the set of all the nodes in this module. Each n ∈ N represents a control or a module in the tree. If one node is a control, it is donated by a 5-tuple: node id, control name, type, attributes, children pointers.
3) E is the set of directed edges. Each edge connects one node n_1 to another node n_2, if n_2 is directly contained in n_1.
4) D is the depth of the module tree, the maximum distance from the root node to the leaf nodes.

Considering the left tree shown in Fig. 4, it is the layout tree of one screen of an app named KuGou in its iOS version. The left subtree of node 2 can be seen as a 2-depth module. The nodes 2, 4, 5, 7, 8 consist set N, and node 2 is R. Among them, nodes 2, 4, 5 are container controls, while nodes 7, 8 are function controls (marked by underlines). The following four subsections will describe the mapping process using this tree as an example.

3.1 Page Layout Tree Generation

The first step of the mapping process is to generate the layout trees for each screen of the apps. These layout trees are the basis for the module mapping, which are extracted from the installation package of an app. Installation packages of the iOS version are downloaded from Apple App store and those of the Android version are downloaded from GooglePlay Store. Each package is composed of compiled binary codes and resources, e.g., pictures. Note that there exist some apps whose iOS versions and Android versions do not look the same, so they are filtered out since the generated modules cannot be mapped in the subsequent steps.

For each app, we leverage Appium [8] to analyze the GUI layouts of pages in the iOS and the Android version apps. Appium is an open source automated test tool for apps. Appium can run the app under test in a device based on a given script. It also provides APIs for saving GUI layouts of pages of the app under test as XML format files. Each line in these XML files contains the complete information of a control, including the control's name, the package's name and the control's attributes. To obtain similar pages on the iOS and Android version app, the prepared scripts for the two versions of an app are designed to execute functions in the same order.

Then we encode these GUI layouts into tree structures named as

Algorithm 1: iOS Modules Generation

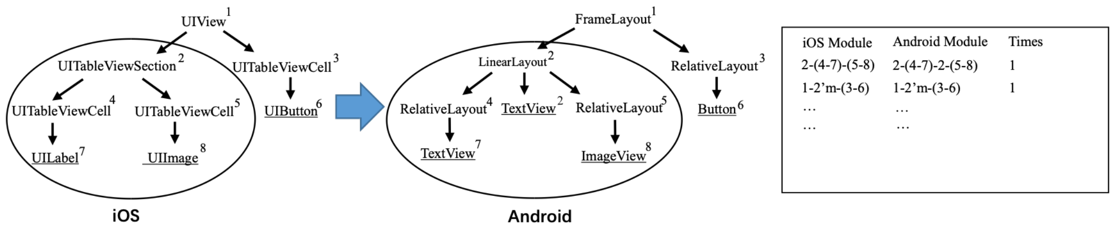
```
Input: iOS layout tree (T), depth
Output: a set of iOS modules (S)
1  NodeSet ← LRD(T);
2  foreach node in NodeSet do
3    if depth == Get_Tree_Depth(Root(node)) then
4      module ← Mark_as_Module(node);
5      Add_Into_Set(module, S);
6      Mark_Node_As_Leaf(node);
7      Remove_Node_In_Tree_From_Set(Sub_Tree(node), NodeSet);
8    end
9  end
10 return S;
```

Page Layout Trees (PLTs). Each node in a PLT is a 5-tuple *(id, name, type, attributes, children pointers)*, in which *id* is used to uniquely identify a control, *name* is specified what the control is (e.g., a *label* or a *imageView*), *type* specifies whether the control is a container or a function one, *attributes* stores the values of necessary attributes of the control (such as positions and label texts), and *children pointers* point to the direct child nodes if it is a container node. Every page is encoded into one tree, as shown in Fig. 4.

3.2 Module Construction

In this step, we divide the Android and iOS layout trees into modules. We notice that if we divide the two trees of both versions at the same time, they might be divided into different structures and be difficult to execute the following mapping process. Since the goal of our approach is to transform iOS pages into Android ones, we only need to make sure that all iOS controls can be mapped to Android controls in all the modules, but not vice versa. Dividing iOS layouts first and then matching them to the Android controls is a reasonable approach to maintain this unidirectional mapping relationship.

Definition 3.1 shows that the modules we intend to map should be the subtrees from the PLTs, and Section 2 has illustrated that too many or too few controls in the subtrees may not benefit the matching process. Thus, we need to choose subtrees with an appropriate size. To divide the trees, we have defined different segmentation strategies based on the depth of the subtree, which is the distance between the root node and the farthest leaf node. The evaluation of the experiment in Section 5 demonstrates that different depths can lead to different transformations, and we found that the best depth is 2, based on trial and error experiments. Algorithm 1 shows the procedure of dividing the PLT into a set of modules. In line 1, we get all the nodes with procedure LRD, which performs a post-order traversal on the PLT. For each



iOS Module	Android Module	Times
2-(4-7)-(5-8)	2-(4-7)-2-(5-8)	1
1-2'm-(3-6)	1-2'm-(3-6)	1
…	…	
…	…	

Fig. 4: Page layout trees of the app KuGou in iOS and Android platforms

Algorithm 2: Module Pairs Generation

```
Input: iOS module (M), Android layout tree (T)
Output: Module map (Map)
1  F ← Get_All_Function_Controls(M);
2  foreach element in F as N_iOS do
3    do
4      N_And ← Get_Mapped_Android(N_iOS);
5      N_iOS ← Get_Parent_Node(N_iOS);
6      do
7        N_And ← Get_Parent_Node(N_And);
8        COUNT ← 0;
9        foreach child in Get_Child_Node(N_iOS) do
10         if Belong_To(child, Sub-tree(N_And)) then
11           COUNT ++;
12         end
13       end
14       if IsEqual(COUNT, Get_Child_Size(N_iOS)) then
15         Set_Mapped_Android(N_iOS, N_And);
16       end
17     while Be_Root(N_And, T);
18   while Be_Root(N_iOS, M);
19 end
20 if Get_Mapped_Android(Get_Root(M)) != null then
21   module←Mark_As_Module(Get_Mapped_Android(Get_Root(M)));
22   return Map(M, module);
23 end
```

node in the set (line 2 to 3), we calculate the depth of the subtree by considering this node as the root of the subtree. If the depth equals to the given depth, we perform the following four steps (line 4 to 7): (1) make this subtree a module; (2) add the module into Set S; (3) mark the root of the subtree as a leaf node, instead of an intermediate node; and (4) remove all the nodes in the subtree, except for the root of the subtree, from the NodeSet to avoid redundant traversal. In the iOS tree of Fig. 4, we mark node 1 and 2 as modules: {{1, 2-"module", 3, 6}, {2, 4, 5, 7, 8}}, and they are added into the module mapping database.

3.3 Mapping Controls of iOS to Android

Given the iOS modules acquired in the last step and the layout tree of the Android version, it is still hard to divide the Android modules and map the controls because of the huge mapping space and possibility. We propose to map the function controls before container controls, for the following reasons: (1) as mentioned above, functional controls are usually visible and have more influences on the pages' appearance; (2) mapping relations of container controls are hard to confirm, since location information becomes useless. For example, one *Framelayout* is designed to contain a *TextView*, but its size does not necessarily represent anything, since it can be much larger than the *TextView* by containing some blank space; and (3) finally, the mapping of container controls has lots of alternatives since different combinations of container controls can achieve the same effect, and thus the choice of different container controls depends on the habit of developers.

If two versions of one page look the same, the most intuitive identification for mapping one iOS function control to one Android function control is that they are displayed in the same position. For example, two buttons from iOS and Android pages look similar, but may be implemented entirely different in coding ways. The Android button may be one *CheckBox* plus one *TextView*, while the iOS button may be one clickable *imageView*. These two GUI elements have no other common features but positions on screen. A control's position is a two-tuple key-values *(x, y)*, in which x represents the value of the distance away from the screen's left border (x is a relative value compared to the screen width, whose value is 1), and y represents the value of the distance away from the screen's top border (y is a relative value compared to the screen height, whose value is 1).For example, a control's position could be represented as (0.2, 0.3), which means the control is 0.2 away from left border and 0.3 away from top border when the screen width is 1 and the screen height is 1.

We cannot treat these key-value tuples as the location directly, because all the pages have been re-sized in various kinds of devices. Fortunately, we can still acquire the locations by making the two pages into the same coordinate system with the page size. Our approach considers the two GUI elements, which have the same words and are the closest to the bottom right of the screen, as reference object, and records their key-values as W_iOS, H_iOS and W_And, H_And. If no elements have the same words, we record the screen sizes directly. In this coordinate system, the positions of controls are represented by the distances to the reference object, and we use Euclidean metric to represent the distance between two objects (Formula (1)). For each iOS function control, we rank all the distances with Android function controls and take the one with the minimum distance mapped. Our approach takes more into consideration since not all the apps are consistently designed in both two platforms. If the minimum distance is larger than one threshold, this control and the module it belongs to are abandoned and the mapping is aborted. By trial and error, we find out that the suitable threshold should be 60/(H_iOS + H_And).

$$\begin{cases} distance\_X = X\_iOS - X\_And \times W\_iOS / W\_And \\ distance\_Y = Y\_iOS - Y\_And \times H\_iOS / H\_And \\ distance = \sqrt{distance\_X^2 + distance\_Y^2} \end{cases} \quad (1)$$

3.4 Mapping iOS and Android Modules

In this step, we will take a different strategy to map the container controls, where no positions, sizes or other attributes but only the parent-children relations are exploited. The parent nodes are container controls and the children nodes can be container controls, function controls or modules. We require that container controls to be mapped between Android and iOS platforms should have one-to-one mapped children nodes.

Algorithm 2 shows the process of mapping container controls. It takes the iOS modules and an Android PLT which has been processed in Section 3.3 as inputs and stores the module mapping relations in the database. First, we collect all the function controls F in the iOS module in line 1. From the leaf nodes, we traversal to the root, and mark the ones that can be mapped to Android controls (line 2 to 19). One iOS container control can be mapped with one Android container control, only when all its children belong to the subtree are mapped to Android controls (line 9 to 16). In line 20 to 22, if we map controls for the whole iOS subtree, we mark such a subtree as a module and store it in the database.

For example (in Fig. 4), the iOS module in the circle has function controls Node 7 and 8. Following Algorithm 2, we first take out Node 7 *UILabel* and its parent Node 4 *UITableViewCell*. We have known from Section 3.3 that the mapped Android node for Node 7 is the *TextView* and its parent node is *RelativeLayout*. Obviously, all the children belonging to the two container controls are completely mapped (*UILabel-TextView* and *UIImage-ImageView*). Node 4 and its parent Node 2 *UITableViewSection* are then mapped. In the same way, we can get the target Android node *LinearLayout*. Note that there is an unmapped node, Node 2 *TextView*. We map this node to *UITableViewSection*, since the generated map relations can tolerant such difference between the two version modules we think. Thus, we mark Android subtree rooted with Node 2 *LinearLayout* as a module. Node 1 *UIView* and *FrameLayout* can also be mapped similarly.

When module mapping database have thousands of modules, one iOS module could be mapped to multiple different Android modules. We create a flag field for each pair of modules to record its occurrence times. This flag records how often the pair is adopted by developers in their projects. The most frequently used modules are considered in the transformation process and the rest of the possible module pairs will only be used if the transformation cannot meet the requirement.

4 Storyboard File Transformation

Storyboard Files Transformation aims to leverage the mapping relations of modules in the database and transform iOS page codes (in the form of Storyboard or xib files) to Android GUI codes. Controls and attributes are the key points during transformation. In our context, we classify the attributes into the static and dynamic ones. The dynamic attributes in iOS are hard to find mapped objects in Android, and we ignored them here. The important static attributes we have found are listed in Table 1, as well as the mapping relations between iOS and Android. Based on whether it affects the positions of controls, we classify the static attributes into layout-dependent and layout-independent ones. The layout-dependent attributes which have been marked in the third column of Table 1 should be considered during transformation. This transformation process consists of the following three steps.

First, we collect all the Storyboard/xib files from the source program and encode them into PLTs (Section 3.1). Then we divide these PLTs following Algorithm 1 (Section 3.2) into modules with the depth used by the module mapping database, so that these derived modules could be used to match the mapping relations in the module mapping database in the next step.

Second, we search the database for the iOS module of the stored mapping relations which are similar to the newly derived modules in the first step. The two modules are similar when the following two conditions meet: (1) the two modules should have the same structure, and the controls contained by the nodes in the same position of the two modules should be the same; (2) the same controls contained by the nodes in the same position of the two modules should have the same layout-dependent attributes. To each derived module from the first step, we check stored mapping relations containing any of the found similar iOS modules, and get the Android modules mapped to. If there is no matched mapping relation found for one newly derived module, it indicates that the PLTs containing such module could not be converted to Android code using our approach. To get a complete Android layout, the derived Android modules will be combined. Starting from the Android module which is mapped by the iOS module containing the root of the input PLT, we recursively replace the "module" marks (Section 3.2) with the other Android modules. Only if no "module" marks are left, the complete Android layout will be finished. Specifically, we choose the Android module to replace one "module" mark (MM0) following these steps: (1) find the iOS module (M1) mapped to the Android module (M0) containing MM0, (2) find the "module" mark (MM1) with the same place of MM0 in M1, (3) find the iOS module (M2) representing the subtree whose root is the leaf node with MM1 in M1, and (4) output the Android module mapped by M2 to replace MM0.

Finally, we construct the Android layout codes by combining the generated Android modules from the second step with the information obtained from analyzing Storyboard and xib files. The

Table 1. Collected iOS attributes from developer site and the corresponding Android attributes

Common		
layout_marginTop	Rect.y	
layout_marginLeft	Rect.x + ∑PreControl.Rect.width	
layout_width	Rect.width	
layout_height	Rect.height	
gravity	contentMode	√
background	Color IF Key == backgroundColor	
visibility	hidden	
onClick	userInteractionEnabled,multipleTouchEnabled	√
scrollbars	scrollEnabled	√
state_selected,Color	selectionStyle	√
--	flexibleMaxX	
--	flexibleMaxY	
--	horizontalHuggingPriority	
--	verticalHuggingPriority	
Text		
textColor	Color IF Key == textColor	
textSize	FontDescription.pointSize	
textStyle	type	
hintText	placeholder	
layout_gravity	baselineAdjustment	
centerHrizontal	contentHorizontalAlignment	√
centerVertical	contentVerticalAlignment	√
textColorHighlight	Color IF Key == highlightedColor	
singleLine	lineBreakMode	√
--	adjustsFontSizeToFit	
--	translatesAutoresizingMaskIntoConstraints	
Image		
src	State.image	
scaleType	contentMode	√
Bottom		
radius	buttonType	
singleLine	lineBreakMode	√

layout-independent attributes can be used to construct Android layout codes directly by following the simple mapping rules in Table 1, except two groups. One group marked by "--" in Table 1, such as “flexibleMaxX”, cannot be replaced with any Android attributes. However, these attributes make few differences on the appearance and can be ignored. The other group has different applying modes on the two platforms and thus need further discussion. such as Color and Position, which are the most important and tough ones. **Color**: iOS develop site suggests two kinds of color expression systems. One is *calibratedWhite* (*<white, alpha>*). The *white* is the grayscale value and the *alpha* is the opacity value. This kind cannot display color, and always be used in black-white background. The other is *calibratedRGB* (*<red, green, blue, alpha>*), and the *alpha* is also the opacity value. The values of all six attributes are between 0 to 1. The Android color system uses four hexadecimal values for the three primary colors and opacity, and each value is between 00 and ff. When transformation, we can get Android color by multiplying 255 and every attributes' values in *calibratedRGB* tuples, or multiply 255 and the two attributes' values in *calibratedWhite* tuples and repeating the *white* value the three-primary colors. **Position**: Controls in Storyboard/Xib must take *Rect* shown as a 4-tuple *<x, y, width, height>*, where *<x, y>* is relative positions in the father control. Oppositely, Android pages have three kinds of layout ways called *FrameLayout*, *LineLayout* and *RelativeLayout*. *FrameLayout* uses absolute positions, *LineLayout* uses relative positions to the prior element in the same container, and *RelativeLayout* uses relative positions to father container element. Thus, we have to turn them into one special tuple *<layout_marginTop, layout_marginLeft>* differently, which is suitable in iOS.

Fig. 5 shows a transformation example from an iOS control *UILabel* to an Android control *TextView*. The pointSize in line 3 is turned into *textSize* in line 8, while other attributes are transformed as follows.

5 Evaluation

To evaluate our approach, we have implemented the mapping and transformation approach in a tool called UITrainDroid. We used UITrainDroid transformed source projects in iOS 10 into target Android 7.0 (API level 24) projects and used Android Studio (AS) to help us evaluate the similarity between the pairs of projects. Notice that the evaluation of the UI appearance is quite subjective, and we conducted a questionnaire survey to evaluate the transformation results. Besides, the modules obtained in our

```
<label contentMode="center" text="H" id="DyA">
  <Rect x="0.0" y="402" width="320" height="40">          (a)
  <FontDesCription pointSize="14">
  <Color white="1" colorSpace="CalibratedWhite">
-------------------------------------------------------
<TextView
  android:id="@+id/DyA"
  android:layout_width="320sp"
  android:layout_height="44sp"
  android:layout_marginTop="402sp"
  android:layout_marginLeft="0sp"                        (b)
  android:textColor="#000000"
  android:textSize="14sp"
  android:gravity="center"
  android:text="H" />
```

Fig. 5: (a) Code of iOS Control - Label (b) Code of Android Control - Textview

work has dominant impacts on the effectiveness of code transformation between different platforms, while the depth of one module is important in our context, and we have attempted to discuss the depth of a module in Section 3. So, we focused on the following three research questions in this paper.

RQ1. Is our approach able to generate correct Android GUI code which produces Android pages having similar appearances with the iOS pages?

RQ2. How well is the generated Android GUI code used to develop the Android applications in the Android developers' eyes?

RQ3. What is the suitable depth in module generation that can have the best effect on the overall approach?

5.1 Metrics for Evaluation

We use “Element Displayed Correctly” and “Page Displayed Correctly” to measure the similarity of the appearances of the Android and iOS pages. One GUI element is transformed and displayed correctly when meeting all the three conditions: (1) every control of the iOS element is transformed to a control in the Android code, (2) the controls of two visible elements in the two version pages have the same color, position and content, and (3) if one of the two visible elements can be clicked, the other should be clickable, and they have the same click events. Note that the elements here should be visible. One transforms a page correctly, when the following two conditions meet: (1) each iOS module can map to an Android one based on the mapping relations in the prepared Module Mapping Database, and (2) each iOS control can be map to one or more Android controls, but does not require strict consistency of content and click events.

We conducted a questionnaire survey to evaluate the UI appearance produced by the transformation results. The questionnaire focused on two aspects: appearance and function. Appearance refers to the intuitive feeling when viewing the page information. Function refers to the intuitive feeling when using the application. And the two evaluation indicators were divided into five levels: terrible, bad, normal, good, perfect. The participants could choose “perfect” if there is no difference between the two version apps, “good” if some slight differences, “normal” if some obvious but acceptable differences, and “terrible” or “bad” if serious differences in their feelings.

5.2 Benchmark Applications and Module Mapping Database

Our evaluation has been conducted on 8 iOS applications selected based on the following criteria: (1) the source code of the iOS applications must be available, so that we can analyze the GUI pages with Xcode; (2) we only consider applications whose GUI codes are made up of Storyboard or xib files; (3) the applications not use native controls are excluded, including games and web applications; (4) the applications should represent different application categories, such as productivity, entertainment, and tools, and from different repositories. Table 2 presents the eight projects. Coding [11] provides a developer-oriented cloud development platform with all kinds of functional pages like sign up, setting, alarms, and work description. Whocall [12] is a call re-

Table 2: Examples of iOS projects transformed into Android

App Name	Page	Line	Function Control		Container Control		Element Displayed Correctly	Page Displayed Correctly
			iOS	Android	iOS	Android		
Coding	79	12184	597	615	680	513	171(85.1%)	76(96.2%)
Wocall	4	386	17	20	23	23	12(100%)	4(100%)
PlainReader	2	325	18	18	23	22	17(94.4%)	2(100%)
DouBan	10	1200	54	54	34	32	18(94.7%)	10(100%)
Zulip	14	1402	34	34	37	34	20(81.7%)	13(92.9%)
Doppio	4	293	19	19	10	7	10(100%)	4(100%)
News-YC	10	1044	42	42	27	23	14(64.8%)	7(70%)
WNXHuntForCity	18	1440	75	75	31	30	75(79.5%)	15(83.3%)

minder app, PlainReader [13] is used to read electronic documents and DouBan [14] shows the picture region of DouBan platform. These four apps' pages are formed with Storyboard files, and the rest is in the form of xib files. Zulip [15] is an office chat software with fast and powerful search experience. Doppio [16] works for Starbucks and News-YC [17] works for Hacker News. WNXHuntForCity [18] imitates ChengMi guiding for city food. These projects have been well developed and maintained for a long time. Among them, Coding has more than 10 thousand lines of GUI codes and totally 79 pages, with 597 function controls and 680 container controls.

We also prepared a large module mapping database to offer the module-to-module relations from iOS to Android. So far, all applications were collected manually, about 18 eligible apps follow the recommendation list of Google Play. We prepared both versions from Google Play and Apple Store. After MMM process, 8703 valid mapped modules are generated. Among them, 1426 different modules were added into the database. Some control constructs are frequently used, and the most frequently used one module appears up to 244 times. This indicates that most applications in the real world have similar GUI design styles, which strengthens the applicability of our tools. The tree depth in Mapping Modules Mining was set to 2, and we will discuss the effect of different depths in evaluation RQ3.

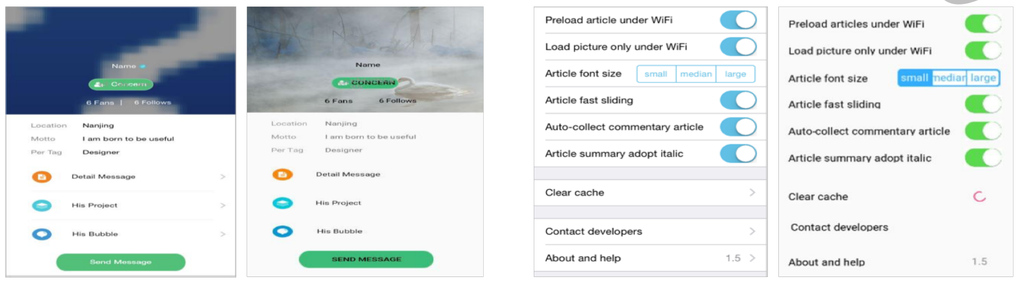
5.3 Results

To answer RQ1, we applied SFT (Section 4) with the prepared Module Mapping Database (Section 5.2) on the eight benchmark applications (Section 5.2). The Android GUI codes output by SFT consists of xml layout files, java code files, and other resources. To compare the Android pages produced by the generated Android GUI codes with the iOS pages, we put the SFT outputs of each benchmark application in a separate Android project in Android Studio (AS), using which one can preview the generated Android pages. The comparison results are collected in Table 2. Note that the module depth used here is 2, and the threshold is 60/(H_iOS + H_And).

The columns "Element Displayed Correctly" and "Page Displayed Correctly" in Table 2 show that UITrainDroid performs well on these cases. Especially for *Coding*, only 3 in 79 pages cannot be transformed and more than 85% (171 in 201) elements are transformed correctly. We can observe that *Coding*, *Wocall*, *PlainReader*, and *DouBan* are transformed better than others. The reason may be that one page of Xib is always cut into multiple pieces and placed in different files by the developers, which will bring a lot of difficulties to Appium to deal with the Xib file. In general, our tool can successfully transform most of iOS page code correctly.

Fig. 6 shows the two pages selected in the experiment cases, which contain as many layouts and elements as possible, such as images and progress bars. We can see that the pages before and after the transformation are almost identical. However, some incorrect details still can be found. For example, "Send Message" will be automatically converted to uppercase. Most of these problems are caused by different design patterns between the two platforms. So, if one attribute cannot be mapped to some combinations with similar functionality, it will be ignored, during control-to-control transformation. So, we designed the metrics in Section 5.1 based on this consideration, and that is the reason why we have some "100%" in Table 2.

To answer RQ2, we generated the Android-version applications of the eight benchmark applications (Section 5.2) using the generated Android GUI codes from the previous experiments. Then, we conducted a questionnaire survey among 60 participants



Location Nanjing
Motto I am born to be useful
Per Tag Designer

Location Nanjing
Motto I am born to be useful
Per Tag Designer

Preload article under WiFi
Load picture only under WiFi
Article font size small median large
Article fast sliding
Auto-collect commentary article
Article summary adopt italic
Clear cache
Contact developers
About and help 1.5

Preload articles under WiFi
Load picture only under WiFi
Article font size [✓ small] median large
Article fast sliding
Auto-collect commentary article
Article summary adopt italic
Clear cache
Contact developers
About and help 1.5

Fig. 6: Two examples of transformed experiment subjects with iOS in the left and Android in the right

along with the installation packages of the generated Android-version applications to see whether the generated Android GUI code worked well. Half of our participants were Android developers from industry working on Android mobile application development for many years, and the other half were undergraduates.

Finally, 58 participants replied to us, and their selections are shown in Fig. 7. Most of them (about 10% of them thought it was perfect, about 20% thought it was good, and about 60% thought it was normal) felt that our transformation results were enough in appearance and function. According to the survey, the generated Android GUI code by applying our approach proved to perform well when cross-platform developing.

To answer RQ3, we applied STF on application *Coding* by changing the depth input in Algorithm 1, and collected the comparing results, "Element Displayed Correctly" and "Page Displayed Correctly", between the Android pages and the iOS pages.

The column "Module Number" in Table 3 shows the number of generated mapping modules after MMM with specific depths. The most effective mapping modules and the most correct pages or elements can be seen when the depth is 2. Also, we found that the results became bad when the depth exceeds 2, and only a few empty pages or table pages can be successfully converted.

6 Related Work

Many cross-platform mobile development solutions [19, 20, 21] have been proposed to make applications suitable for different platforms. Two kinds of solutions are adopted mostly to create native web apps. The web apps are usually based on web technologies such as HTML5 and JavaScript [3, 4], which can be applied to different platforms. A native app made up with native codes only works in a certain platform and we have to translate versions between different languages if we want to apply them in different platforms. These apps are more efficient in loading time and more secure than web apps, and there have been several tools [22, 23] created for translating cross-platform native codes. Xamarin [1, 24] tries to create shareable C# code, which is Ahead-of-Time (AOT) compiled to generate iOS project and Just-in-Time (JIT) compiled to generate Android one, and thus these two generated native codes can approach native efficiency, but

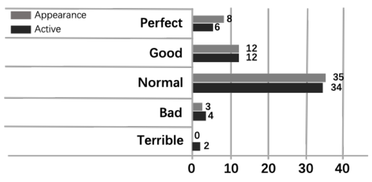


Fig. 7: Research about the evaluation with the transformation

Table 3: The transformation for *Coding* with different depths

Depth	Module Number	Correct Element	Correct Page
1	1147	162(80.6%)	76(90.3%)
2	1426	171(85.1%)	76(90.3%)
3	973	39(19.4%)	23(29.1%)
4	697	17(8.5%)	11(13.9%)

cannot deal with GUI transformation because GUI compilation is more interface-based. MyAppConverter [6] using semantically driven code transformation to create native code can only work from Storyboard and xib projects to Android projects, and it tries to convert controls one by one and claims a relationship stable for each GUI element, but it always fails when transforming complex iOS project because of controls' usage diversity on multiple platforms. RAPPT [25] uses rules to build models of native GUI codes, and the rules are stored in the system beforehand so the rules database is not extendable. It will fail when some models that have not been inside the system occur. Besides, its transformed scenes cannot reach the actual needs at all. Our work takes similar rules with RAPPT but the rules are not provided by developers but mined from the same projects with both versions. Because of the automation process of mapping, we can extend the rules database continually.

The model-driven approach seems to be an important way in creating, analyzing and testing apps and has been promised to be able to increase developer productivity and reduce costs [26]. Nguyen et al. [27] develop a framework for generating the master/detail Android design pattern. The focus of the research is on the small patterns of projects rather than creating a full application. Ribeiro et al. [28] use a UML based approach to describe the multistage mobile app development containing boilerplate code suitable for multiple platforms. Henning et al. [29] claim that their work can consider any view and control as models and connect models from different platforms. These works try to convert Android applications into models, including native logical code and GUI code, but they do not focus on the usage of the generated models. Also, Mona et al. [30] define models to detect inconsistencies in multi-platform mobile apps, which is similar to our work but focuses on the Android test. One new proposed solution called ICPMD [31] focuses to produce native apps by combining the trans-compilation approach and the model-driven development approach, but the GUI part of the native is still created by developers.

The mining-based approaches are becoming popular in many areas, including mobile development. Mining approaches have been used in automatically creating cross-platform API mappings [32, 33, 34, 35, 36, 37] and mining for source code and code generation [38, 39, 40, 41, 42]. These works use different learning algorithms to find patterns with source codes, relate the API or generate native code, but controls relations are more complex than native codes and have never been studied.

7 Conclusion and Future Work

We have presented an approach for automated GUI transformation between different platforms, in particular from iOS into Android. UITrainDroid is tailored to create Android projects with Storyboard/Xib files from iOS projects. We have evaluated the effectiveness of UITrainDroid on real-world applications, and it has successfully produced runnable Android versions.

UITrainDroid relies on Appium, which outperforms better than other similar reverse engineering tools. However, we cannot get the completely precise layouts from the page currently by using

Appium. This could cause that the modules in the database be different from the real cases in iOS and Android development and further lead to mistakes in transformation. Nowadays, GUI reverse engineering depends more on the screenshots but not the device's running memory. We believe that a more effective GUI reverse engineering method or software open source process will improve the performance of UITrainDroid in the future. UITrainDroid is tailored to transform Storyboard/Xib to Android code. It does not mean that our approach cannot be applied to other scenarios, such as transforming Swift files in iOS projects or even Android code. The difficulty in these scenarios is that Swift or Android GUI files are difficult to be encoded, which requires the same process with what a compiler does. Therefore, a more advanced encoding scheme can extend UITrainDroid into other kinds of cross-platform developments in future work.

ACKNOWLEDGMENTS

This research is supported by the National Key R & D Program (No. 2017YFB1001801), the National Natural Science Foundation (Nos. 61972193, 61802179, 61802166), and the Jiangsu key R & D Plan (No. BE2017004-4) of China.

REFERENCES

[1] Xamarin, "Xamarin," Aug.2017. [Online].Available: https://developers.xamarin.com/
[2] Google, "J2objc," Aug. 2017. [Online]. Available: https://github.com/google/j2objc
[3] Adobe, "Adobe phonegap," Aug. 2017. [Online]. Available: https://phonegap.com/
[4] Apache, "Apache cordova," Aug. 2017. [Online]. Available: http://cordova.apache.org/
[5] Sencha, "Sencha," Aug. 2017. [Online]. Available: https://www.sencha. com/
[6] Myappconverter, "Myappconverter," Aug. 2017. [Online]. Available: https://apps.myappconverter.com/
[7] MyappconverterWeb, "Myappconverterdocument," Aug.2017. [Online]. Available: https://docs.myappconverter.com/
[8] Appium, "Appium," Aug. 2017. [Online]. Available: http://appium.io/slate/en/master/
[9] Android, "Android developer," Aug. 2017. [Online]. Available: https:// developer.android.com/reference/classes.html
[10] iOS, "ios developer," Aug. 2017. [Online]. Available: https://developer.apple.com/ios/human-interface-guidelines/ui-controls/buttons/
[11] Coding, "Coding," Aug. 2017. [Online]. Available: https://github.com/ Coding/Coding-iOS
[12] Wocall, "Wocall," Aug. 2017. [Online]. Available: https://gitbug.com/ Quotation/Whocall
[13] PlainReader, "Plainreader," Aug. 2017. [Online]. Available: https://github.com/guojiubo/PlainReader
[14] DouBan, "Douban," Aug. 2017. [Online]. Available: https://gutbuh.com/TonnyTao/DoubanAlbum
[15] zulip, "zulip," Aug. 2017. [Online]. Available: https://github.com/zulip/zulip-ios-legacy
[16] Doppio, "Doppio," Aug. 2017. [Online]. Available: https://github.com/ chroman/Doppio
[17] News-YC, "News-yc," Aug. 2017. [Online]. Available: https://github.com/grp/newsyc
[18] WNXHuntForCity, "Wnxhuntforcity," Aug. 2017. [Online]. Available: https://github.com/ZhongTaoTian/WNXHuntForCity
[19] M. Palmieri, I. Singh, and A. Cicchetti, "Comparison of cross-platform mobile development tools," 2012 16th International Conference on Intelligence in Next Generation Networks, pp. 179–186, 2012.
[20] A. Holzinger, P. Treitler, and W. Slany, "Making apps useable on multiple different mobile platforms: On interoperability for business application development on smartphones," International Conference on Availability, Reliability, and Security, pp. 176–189, 2012.
[21] S. Xanthopoulos and S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications," Proceedings of the 6th Balkan Conference in Informatics, pp. 213–220, 2013.
[22] V. Tunali and S. Zafer, "Comparison of popular cross-platform mobile application development tools," Celal Bayar University: Faculty of Technology Department of Software Engineering, Maltepe University: Faculty of Engineering and Natural Sciences Department of Software Engineering, 2015.
[23] J. Friberg, "Evaluation of cross-platform development for mobile devices," 2014.
[24] N. Boushehrinejadmoradi, V. Ganapathy, S. Nagarakatte, and L. Iftode, "Testing cross-platform mobile app development frameworks (t)," 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 441–451, 2015.
[25] S. Barnett, R. Vasa, and J. Grundy, "Bootstrapping mobile app development," Proceedings of the 37th International Conference on Software Engineering-Volume 2, pp. 657–660, 2015.
[26] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," Synthesis Lectures on Software Engineering, vol. 3, no. 1, pp. 1–207, 2017.
[27] T.-D. Nguyen and J. Vanderdonckt, "User interface master detail pattern on android," Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems, pp. 299–304, 2012.
[28] A. Ribeiro and A. R. da Silva, "Evaluation of xis-mobile, a domain specific language for mobile application development," Journal of Software Engineering and Applications, vol. 7, no. 11, p. 906, 2014.
[29] H. Heitkö̈tter, T. A. Majchrzak, and H. Kuchen, "Cross-platform model- driven development of mobile applications with md 2," Proceedings of the 28th Annual ACM Symposium on Applied Computing, pp. 526–533, 2013.
[30] M. E. Joorabchi, M. Ali, and A. Mesbah, "Detecting inconsistencies in multi-platform mobile apps," 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), pp. 450–460, 2015.
[31] W. S. El-Kassas, B. A. Abdullah, A. H. Yousef, and A. M. Wahba, "Enhanced code conversion approach for the integrated cross-platform mobile development (icpmd)," IEEE Transactions on Software Engineering, vol. 42, no. 11, pp. 1036–1053, 2016.
[32] A. Gokhale, V. Ganapathy, and Y. Padmanaban, "Inferring likely mappings between apis," Proceedings of the 2013 International Conference on Software Engineering, pp. 82–91, 2013.
[33] M.P.Robillard,E.Bodden,D.Kawrykow,M.Mezini,andT.Ratchford, "Automated api property inference techniques," IEEE Transactions on Software Engineering, vol. 39, no. 5, pp. 613–637, 2013.
[34] H. Zhong, S. Thummalapenta, T. Xie, L. Zhang, and Q. Wang, "Mining api mapping for language migration," Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1, pp. 195–204, 2010.
[35] R. Nix and J. Zhang, "Classification of android apps and malware using deep neural networks," 2017 International Joint Conference on Neural Networks (IJCNN), pp. 1871–1878, 2017.
[36] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deep api learning," Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 631–642, 2016.
[37] V. Srivastava, M. D. Bond, K. S. McKinley, and V. Shmatikov, "A security policy oracle: Detecting security holes using multiple api implementations," ACM SIGPLAN Notices, vol. 46, no. 6, pp. 343–354, 2011.
[38] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pp. 297–308, 2016.
[39] E. Wong, T. Liu, and L. Tan, "Clocom: Mining existing source code for automatic comment generation," 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp. 380–389, 2015.
[40] R. Holmes, R. Cottrell, R. J. Walker, and J. Denzinger, "The end-to-end use of source code examples: An exploratory study," 2009 IEEE International Conference on Software Maintenance, pp. 555–558, 2009.
[41] F. Lv, H. Zhang, J.-g. Lou, S. Wang, D. Zhang, and J. Zhao, "Codehow: Effective code search based on api understanding and extended boolean model (e)," 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 260–270, 2015.
[42] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: finding relevant functions and their usage," Proceedings of the 33rd International Conference on Software Engineering, pp. 111– 120, 2011.