Automated Cross-Platform GUI Code Generation for Mobile Apps

Sen Chen*, Lingling Fan*, Ting Su*, Lei Ma†, Yang Liu*, Lihua Xu‡
*Nanyang Technological University, Singapore †Harbin Institute of Technology, China
‡New York University Shanghai, China
ecnuchensen@gmail.com

***Abstract*—Android and iOS are the two dominant platforms for building mobile apps. To provide uniform and smooth user experience, app companies typically employ two teams of programmers to develop UIs (and underlying functionalities) for these two platforms, respectively. However, this development practice is costly for both development and maintenance. To reduce the cost, we take the first step in this direction by proposing an automated cross-platform GUI code generation framework. It can transfer the GUI code implementation between the two mobile platforms.**

Specifically, our framework takes as input the UI pages and outputs the GUI code for the target platform (e.g., Android or iOS). It contains three phases, i.e., component identification, component type mapping, and GUI code generation. It leverages image processing and deep learning classification techniques. Apart from the UI pages of an app, this framework does not require any other inputs, which makes it possible for large-scale, platform-independent code generation.

***Index Terms*—Cross-platform, Code Generation, Mobile App**

I. Introduction

Nowadays, over 3.8 million Android apps and 2 million iOS apps are striving to gain users on Google Play and Apple App Store [3]. As we know, the most famous mobile apps are event-driven programs with rich Graphical User Interfaces (GUIs), and a pleasant user experience is crucial for an app's success in the highly competitive markets [6], [13].

Currently, a great amount of research focuses on improving the efficiency of GUI code development and generation [4], [5]. However, they only focuses on one type of mobile platforms (e.g., Android system or iOS). Actually, most of mobile apps have two similar versions on the two different platforms by using different developing languages (i.e., Java and Object-C). Many leading industrial companies (e.g., Google) attempt to leverage the native development scheme for cross-platform code development (e.g., Flutter[1] and ReactNative[2]). Their solutions is to develop the code only once and make it run on the two different mobile systems. Nevertheless, two main problems are as follows: (1) the technology is immature and still at the preliminary stage; (2) It is definitely difficult to put such scheme into practice due to the competitive relationship between different industrial companies, such as Google and Apple, as well as their corresponding products (e.g., Android system and iOS).

[1]https://flutter.io/

[2]https://facebook.github.io/react-native/

Fig. 1: `Facebook` in Android

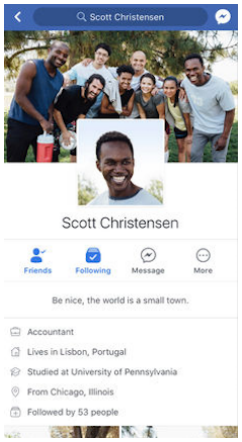

Fig. 2: `Facebook` in iOS

Typically, the industrial companies maintain two developing teams in practice for Android and iOS app development. The functionalities of the two versions are usually very similar in the real world. As shown in Fig. 1 and Fig. 2, the two versions of `Facebook` have similar login pages, especially the GUI component types, such as `Button`, `TextView`, etc. From the point of GUI design, the commonality of the two versions is still very high. However, the two teams rarely have interaction or discussion on GUI page design and implementation. In industry, the companies have to cost more to employ different areas of experts and developers for the two developing teams. As for the GUI designers and developers working for different platforms, there's nearly nothing they can share when designing app GUI or implementing GUI code.

Thus, it is much-need to propose an automated cross-platform GUI code generation framework for transferring the GUI code implementation between the two mobile platforms. As for the framework, given the GUI pages of an Android/iOS app, and no other inputs are required, the implementation code of the corresponding iOS/Android GUI pages should be generated automatically. However, it faces the following challenges: (1) It is difficult to extract the components in the GUI pages accurately. (2) The types of the extracted components cannot be identified only relying on the image processing techniques. (3) There's no such a technique that maps and transfers the GUI implementation code of the two different platforms (i.e., Android and iOS).

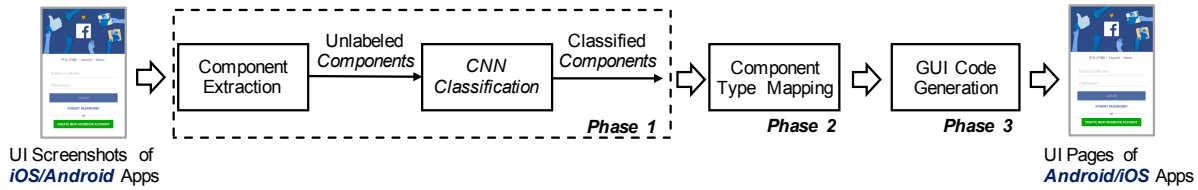

Fig. 3: Overview of our framework

To address the above problem and challenges, in this paper, we propose a framework to automatically generate the cross-platform GUI code. Our framework contains three phases: (1) *Component Identification*, which extracts the components with their corresponding attributes from the GUI pages by leveraging image process techniques, and identifies the corresponding types by utilizing deep learning classification algorithms; (2) *Component Type Mapping*, which maps the component types from one platform to another (e.g., `TextView` in Android → `textView` in iOS); (3) *GUI Code Generation*, based on the above two phases, we further generate the GUI code for the target platform. In summary, we make the following contributions in this paper.

- We are the first to generate cross-platform GUI code for different mobile platforms (e.g., Android and iOS platforms), instead of using native app development for independent platforms.
- Our approach only requires the UI pages of an app, which makes large-scale GUI code generation possible and without platform limitations. Moreover, we propose and maintain an extensible mapping relations between different platforms for GUI component type transferring.

We introduce a new research direction that automated and data-driven mobile app code generation could be achieved by leveraging Computer Vision (CV) and Artificial Intelligence (AI). CV corresponds to the rich Graphical User Interface of mobile apps, and AI corresponds to the millions of mobile apps, as well as a huge of metadata. That means, CV and AI can shed a light on software engineering communities on the direction of mobile code generation, both GUI code and logic code generation, instead of using traditional program analysis [7], [8] (e.g., data-flow analysis and symbolic execution).

II. Preliminaries

In this section, we briefly introduce the mobile GUI, image processing, and deep neural networks that are used in our framework.

Mobile GUI. Android graphical user interface (GUI) framework is famous for the multi-interactive activities. GUI is implemented as a layout hierarchy, which supports a variety of pre-built components such as structured layout objects (e.g., `LinearLayout` and `RelativeLayout`) and components (e.g., `Button`, `EditText`, and `TextView`), and allows developers to build the GUI for the app. Each layout object may contain several components together with their attributions (e.g., `android:id`, `android:text`).

IOS GUI also contains multiply rich components. Although the implementation code is completely different, most of component types can be matched to the Android component types, making automatic conversion between codes possible. Such as, the correspondences between `TextView` and `textVew`, `ImageView` and `imageView`, and `Button` and `button`.

Image Processing. We briefly introduce two image processing techniques: Canny edge detection [1] and Edge dilation [2]. Canny edge detection is an edge detection technique that uses multi-stage algorithm to detect a wide ranges of edges in images. It aims to accurately catch as many edges shown in the image as possible and localize the center of the edge. It can also be used to detect the edges of components (e.g., `Button`, `EditText`, and `TextView`) in a given GUI page. Edge dilation is usually used after canny edge detection to further merge the adjacent elements. It gradually enlarges the boundaries of regions so that the holes within the regions become smaller or disappeared.

Deep Neural Networks. Convolutional Neural Networks [9] (CNN) is a type of deep neural networks which is commonly applied to analyzing visual images. It leverages multi-layers with convolution filters to automatically locate features for different tasks such as image classification.

III. Framework Design

As shown in Fig. 3, our framework takes as input the UI pages from Android or iOS apps and outputs the UI code for the target platform. Specifically, our framework contains three phases: (1) *GUI component identification*, which first extracts the components in the UI pages by utilizing image processing techniques, and then identifies the types of components (e.g., `Button`, `TextView`) by leveraging the deep learning algorithm (i.e., CNN classification); (2) *Component type mapping*, which maps the identified types of components to the corresponding components of target platform; (3) *UI code generation*, which generates the GUI implementation code based on the component types together with their attributes from the above two phases.

A. Component Identification

The process of component identification involves two steps: *component detection* and *component classification*.

Component Detection. To extract the components of the given UI pages, we first detect the edges of all components in each page through canny edge detection algorithm [1]. Since the detected edges are too coarse to use directly, edge dilation [2] is always used to cooperate with canny edge detection to optimize the component detection. We thus merge the adjacent elements by leveraging edge dilation which gradually enlarges

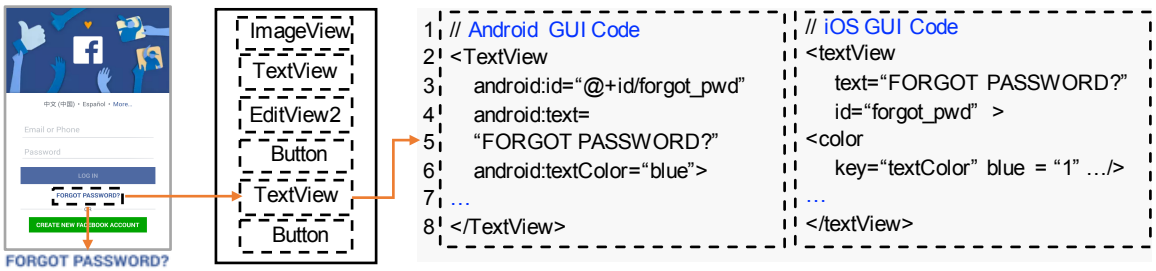

Fig. 4: Component type mapping between Android and iOS GUI code

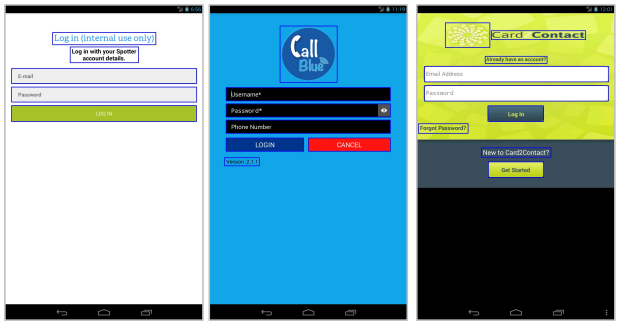

Fig. 5: Examples of component detection

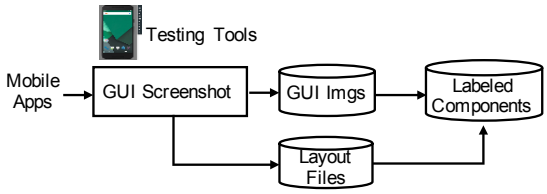

Fig. 6: Labeled component data collection

the boundaries of regions so that the holes within the regions become smaller or disappeared. As shown in Fig. 5, the components can be detected by the image processing algorithms.

Component Classification. Although we identified the components of the given UI pages in the first phase, the types of these components still remain unknown, which is an essential characteristic for further GUI code generation. To achieve this, we utilize Convolutional Neural Network [9] (CNN) for this classification task. Specifically, we use CNN to train two classification models based on two types (i.e., Android and iOS) of large-scale labeled component images, respectively.

Each model takes as input volume and outputs an N dimensional vector where N is the number of classes that the program has to choose from. We take `EditText`, `Button`, and `TextView`, etc., for consideration. Thus, N would be different here for different platforms and our model is to classify a cropped component as one of the N types. Note that the output of the fully connected layer (also the whole CNN) will be the probability of these N classes (the sum of them is 1).

B. Component Type Mapping

Android and iOS apps are developed through different development languages (i.e., Java and Object-C). Therefore, the two types of implementation code cannot be transferred directly. We propose and maintain an extensible mapping relations between the two types (corresponding to the two platforms) of components. For example, as shown in Fig. 4, the corresponding GUI code of the component `TextView` extracted from the login page of `Facebook`, which is implemented in both Android version and iOS version. The login page from `Facebook` contains a link for resetting the password "FORGOT PASSWORD?". Android app uses the component of `TextView`, along with their corresponding attributes, such as "`android:text`" and "`android:textColor`." While the UI code implementation of the corresponding iOS app also uses key-value pairs to implement the component `textView` with component attributes (e.g., `<color>`).

C. GUI Code Generation

Code generation is the last phase, we generate GUI code for each kind of components according to its attributes, and add the code into the overall layout of the GUI code file. We maintain two types of GUI code templates for Android and iOS platform apps to generate GUI code. For example, as for `EditText`, we obtain its GUI code by also considering its text hint and background color. We then use "`android:hint`" and "`android:background`" to implement the GUI code.

IV. Training Dataset Collection

Fig. 6 shows the data collection process. We crawled 37,251 and 8,951 unique Android and iOS apps from Google Play Store and iTunes App Store, respectively. These apps belong to multiply categories including social (e.g., Facebook and Twitter), finance (e.g., HSBC), business (e.g., Gmail), news (e.g., CNN News), etc. With the help of dynamic testing tools, such as, UIAutomator[3] and Stoat [11], [12] for Android apps, ideviceinstaller[4] for iOS apps, we run each app on emulators (e.g., Android 4.3) configured with the default settings for 30 minutes and take screenshots of the explored screens during runtime. Finally we obtain billions of original UI screenshots. Meanwhile, we use these testing tools to extract component information (i.e., component types and coordinate positions) for the explored app screens. Actually, not all the apps can be successfully launched on the emulator due to version update warnings, Google service update warnings, and third-party library support. Our goal of screenshot

[3]https://developer.android.com/training/testing/ui-automator/
[4]https://github.com/libimobiledevice/ideviceinstaller/

Fig. 7: Original

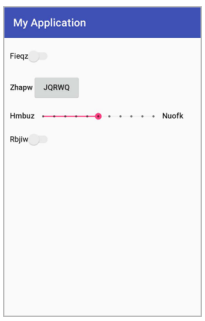

Fig. 8: PIX2CODE

Fig. 9: UI2CODE

collection which enables large-scale component analysis is to ensure multiple sets of screenshots and components, rather than completely explore each app or get all components in each screenshot. Although the `Layout Files` information from the testing tools do not contain all components and may contain minor errors, it would not affect the collection of our training dataset. Finally, the result data set contains 1,842,580 unique Android screenshots, which is the biggest raw data set of UI screenshots as far as we know, and the database of iOS screenshots is being constructing.

V. Preliminary Observation

We observe preliminary results from the following two aspects: *The accuracy of CNN classification and the UI similarity of generated pages*.

Accuracy of Classification. To demonstrate the effectiveness of the CNN classifier, we choose several state-of-the-art machine learning classifiers (e.g., `Logistic Regression`, `SVM`, and `K-nearest Neighbors`) as the baselines. We then use the typical metrics (e.g., Accuracy, Precision, and Recall) to compare the classification results. Based on our preliminary experiments and observation, we unveil that the CNN classification outperforms all baselines, achieving more than 85% accuracy, while the baselines achieve 20%-70% accuracy.

UI Similarity of the Generated Pages. We have implemented several basic pages, such as login pages of social and financial apps. We compare the generated UI pages with the existing UI code generation techniques, such as PIX2CODE [4] and UI2CODE [5]. We use UI2CODE and PIX2CODE to generate the targeted UI pages. Based on the successfully generated UI pages, we measure the similarity using two widely used image similarity metrics [10]: mean absolute error (MAE) and mean squared error (MSE). Fig. 8 and Fig. 9 show the generated UI pages by PIX2CODE and UI2CODE respectively. The two techniques aim to reduce the burden on the GUI code development, but they are not competent to generate an almost the same UI page due to lack of realistic GUI-hierarchies of components and containers of UI pages. Moreover, they cannot extract component attributes, such as coordinate position, color and type. The similarity between the generated UI pages by UI2CODE and PIX2CODE and the original pages is mainly between 60%-70%. The similarity may be higher than what it looks like due to the white background.

VI. Conclusion and Discussion

In this paper, we propose a new idea, as well as a new research direction for automated cross-platform GUI code generation. It can improve the efficiency of mobile development, and further shorten the developing lifecycle. CV and AI techniques can be used to enable the automated UI code generation for mobile apps. With the growing usage of mobile apps, millions of apps can be collected for data-driven analysis. The large-scale dataset can be regarded as the training set for the AI techniques (e.g., machine learning, deep learning algorithms, and natural language processing (NLP)). The CV techniques can help extract the components of UI pages, as well as their attributes. Additionally, CV techniques may significantly improve the efficiency in many software research areas such as app testing. It can help extract the UI texts to generate meaningful text inputs for app pages.

Acknowledgments

We appreciate the reviewers' constructive feedback. This work is partially supported by NSFC Grant 61502170, NTU Research Grant NGF-2017-03-033 and NRF Grant CRDCG2017-S04.

References

[1] (2018) Canny Edge Detection. [Online]. Available: https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html

[2] (2018) Dilatation Edge. [Online]. Available: https://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

[3] (2018) Number of apps available in leading app stores as of 1st quarter. [Online]. Available: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[4] T. Beltramelli, "pix2code: Generating code from a graphical user interface screenshot," *arXiv preprint arXiv:1705.07962*, 2017.

[5] C. Chen, T. Su, G. Meng, Z. Xing, and Y. Liu, "From UI design image to GUI skeleton: a neural machine translator to bootstrap mobile GUI implementation," in *Proceedings of the 40th International Conference on Software Engineering*. ACM, 2018, pp. 665–676.

[6] S. Chen, L. Fan, C. Chen, T. Su, W. Li, Y. Liu, and L. Xu, "StoryDroid: Automated generation of storyboard for Android apps," in *Proceedings of the 41th ACM/IEEE International Conference on Software Engineering, ICSE 2019*, 2019.

[7] L. Fan, T. Su, S. Chen, G. Meng, Y. Liu, L. Xu, and G. Pu, "Efficiently manifesting asynchronous programming errors in android apps," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)*. ACM, 2018, pp. 486–497.

[8] L. Fan, T. Su, S. Chen, G. Meng, Y. Liu, L. Xu, G. Pu, and Z. Su, "Large-scale analysis of framework-specific exceptions in Android apps," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, 2018, pp. 408–419.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[10] T. A. Nguyen and C. Csallner, "Reverse engineering mobile application user interfaces with REMAUI," in *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015.

[11] T. Su, "Fsmdroid: Guided gui testing of android apps," in *Proceedings of the 38th International Conference on Software Engineering Companion*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 689–691.

[12] T. Su, G. Meng, Y. Chen, K. Wu, W. Yang, Y. Yao, G. Pu, Y. Liu, and Z. Su, "Guided, stochastic model-based GUI testing of Android apps," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 2017.

[13] S. E. S. Taba, I. Keivanloo, Y. Zou, J. Ng, and T. Ng, "An exploratory study on the relation between user interface complexity and the perceived quality," in *International Conference on Web Engineering*. Springer, 2014, pp. 370–379.