INFERRING IN-SCREEN ANIMATIONS AND INTER-SCREEN TRANSITION FROM USER INTERFACE SCREENSHOTS

by

SIVA NATARAJAN BALASUBRAMANIA

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2017

Acknowledgements

I would like to thank my supervising professor, Dr. Christoph Csallner for his continuous support, endless patience and trust in my skills. He has been a great teacher and a mentor steering me in the right direction. I could not have imagined having a better advisor for my thesis.

Besides my advisor, I am thankful to Dr. Tuan Ngyuen for inspiring me and providing me the first glimpse of research and for supporting and motivating whenever needed. He has been a mentor, a colleague and a friend helping me in being successful!

I am grateful to Dr. Farhad Kamangar and Mr. David Levine for serving as members of thesis committee and for your insightful comments and questions.

Finally, I would like to thank my father, mother and sister for their support and trust in me throughout my life.

Apr 18, 2017

Abstract

REVERSE ENGINEERING MOBILE APPLICATION UI

USING REMAUI

SIVA NATARAJAN BALASUBRAMANIA, MS

The University of Texas at Arlington, 2017

Supervising Professor: Christoph Csallner

In practice, many companies have adopted the concept of creating interactive prototypes for explaining workflows and animations. Designing and developing a user interface is a time-consuming process, and the user experience of the application has a major impact on the success of the application itself. User interface designing marks the start of the app development, and it is very expensive regarding cost and time for making any modification after the coding phase kicks in. Currently, companies have adopted UI prototyping as part of the app development process. Third-party tools like Flinto or Invision use the high fidelity screen designs for making interactive prototypes, and other tools like Flash is used to prototype animations and other transition effects. With this approach, there are two major setbacks. Creating the screen designs (acts as the screen specification for color, dimensions, margin, etc.) and the navigations or animations takes a lot of time, but they are not reusable in the app development process. The prototypes could act as a reference for the developers, but none of the output artifacts is reusable in the developing the application With our technique of using REMAUI as a preprocessor to identify different UI elements like images, texts, containers on the input bitmap images. We have developed a user interface that allows users to interact with the preprocessed inputs and create links for inter-screen transitions on click, long click with effects like a slide, fade, and explode. We would be able to generate code for the intended navigation targeting a specific platform say Android. Additionally, we have developed a heuristic algorithm that analyses REMAUI processed input bitmaps and infers possible in-screen animations such as translation, scaling and fading using perceptual hashing.

In our experiment, we evaluated our prototype’s inter- screen transition on screenshots of 10 different third party applications and it generated android code for transition in less than 2s. We evaluated inferring in-screen animation on screenshots of top 30 third party Android application generated user interfaces similar to the original on comparing pixel-by- pixel (SSIM) and it takes 26s on an average for identifying possible animation.

Table of Contents

Acknowledgements iii

Abstract iv

List of Illustrations viii

List of Tables xi

Chapter 1 1

Introduction 1

Chapter 2 6

Motivating Example 6

2.1 Transition Extension: 6

2.2 Animation Extension 15

Chapter 3 19

Background 19

3.1 Native apps 19

3.2 Prototype 20

3.3 Hamming Distance 21

3.4 Android 21

3.5 REMAUI 26

3.6 Perceptual Hash 28

Chapter 4 32

Overview and Design 32

4.1 Inferring in-screen animation 32

4.2 Transition Extension 43

Chapter 5 50

Research Questions (RQ), Expectations (E), and Hypotheses (H) 50

Chapter 6 51

Evaluation 51

Prototype Information 51

Experiments 52

Chapter 7 67

Related Work 67

Chapter 8 69

Conclusion and Future work 69

Chapter 9 70

Appendix I 70

Test Applications for inferring in-screen animations 70

Appendix II 73

Test Application for inter-screen transitions 73

References 74

Biographical Information 77

List of Illustrations

Figure 1-1 Design-Development Process 1

Figure 2-1 Menu screen design of Amazon Underground app for Android 6

Figure 2-2 - Home screen design of Amazon Underground app for Android 6

Figure 2-3 - Working of Invision tool on Amazon app's menu screen 7

Figure 2-4 - Invision preview of menu screen design 8

Figure 2-5 - Invision preview of home screen design 8

Figure 2-6 - Working of Flinto on Amazon Underground app for Android 9

Figure 2-7 - Flinto preview of menu screen design 10

Figure 2-8 - Flinto preview of home screen design 10

Figure 2-9 - Working of REMAUI Transition extension on Amazon Underground App for Android. 12

Figure 2-10 - Working of REMAUI Transition Extension on Amazon Underground app for Android with Highlighted user click 13

Figure 2-11 - Working of REMAUI transition extension on Amazon Underground app for Android with line representing navigation 14

Figure 2-12 - Real world example Facebook's login screen animation 16

Figure 2-13 - Working of REMAUI - Animation extension on Facebook app for Android. 17

Figure 2-14 - REMAUI Animation extension generated code for Facebook Android animation 18

Figure 3-1 - Working of REMAUI on Google Hangouts screen design 28

Figure 3-2 - Calculated MSE / SSIM on Chrome application's landing screen 29

Figure 3-3-Overview of Perceptual Hashing 30

Figure 4-1-Example of In-screen animation 32

Figure 4-2 - Overview of in-screen animation inference 33

Figure 4-3 - Example of Step 1 processing on Facebook app for Android 34

Figure 4-4 - Chrome Landing screen animation Intial and Final states 38

Figure 4-5 - REMAUI processed Whatsapp initial state 40

Figure 4-6 - REMAUI processed Whatsapp final state 40

Figure 4-7 - Pruning Text View Hierarchy of Initial and Final states of WhatsApp for Android 41

Figure 4-8 - Snapshot of Generated Project Structure 42

Figure 4-9 - Code snippet for creating transition and scaling animations 43

Figure 4-10 - Search screen design of Google Play! Store 45

Figure 4-11 - App detail screen design of Google Play! Store 45

Figure 4-12 - Working of REMAUI Transition extension on Google Play! Store 46

Figure 4-13 - User interactions are highlighted and line represents a transition 46

Figure 4-14 - File structure of generated application with inter-screen transitions 48

Figure 4-15 - Generated Activity class with navigation 49

Figure 6-1 -Time taken for generating code vs Number of Transitions 54

Figure 6-2 - Identified View hierarchy is drawn over Music Free app screen design 55

Figure 6-3 - Identified View hierarchy is drawn over the Whatsapp app screen design ... 56

Figure 6-4 - Identified View hierarchy is drawn over the eBay app screen design 57

Figure - 6-5 - Runtime for inferring in-screen animation 58

Figure 6-6 - Identified View hierarchy is drawn over the Zedge app intial state screen design 59

Figure 6-7 - Identified View hierarchy is drawn over the Zedge app final state screen design 60

Figure6-8 - SSIM calculated for the test application's input screenshot vs. generated screenshot 62

Figure 6-9 - Screenshot of generated final state of Zedge app 63

Figure 6-10 - Identified View hierarchy is drawn over the Beacon Flashlight app initial state screen design 64

Figure 6-11 - Identified View hierarchy is drawn over the Beacon Flashlight app final state screen design 65

Figure 6-12 - Screenshot of the generated final state of Beacon Flashlight app for Android 66

List of Tables

Table 2-1 - Comparison of time taken for uploading and assigning navigation 15

Table 4-1 - Heuristics for Perceptual hashing 36

Table 4-2 - Comparison of Similar Image views 37

Table 4-3 - Comparison of Non-similar Image views 37

Table 4-4- Heuristic for Pruning Text View Hierarchy 41

Table 6-1 - Runtime of Transition extension on Amazon Underground app for Android .. 53

I

Introduction

Developing the user interface of the mobile application is expensive in terms of cost and time. In today's competitive market, time to market, intuitive user interface design and good user experience determine the quality of the mobile application [17]. For designing high-quality user interface, companies are building a separate team (design team) for creating high fidelity screen designs. Companies have formulated an app development process as below,

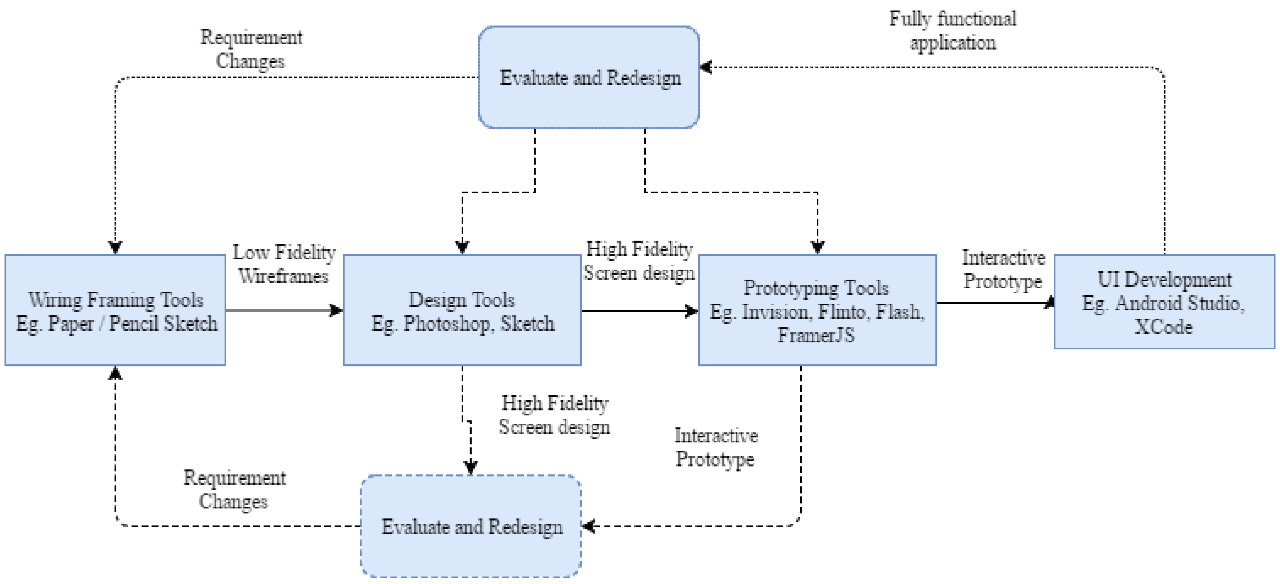

Requirement Changes
Evaluate and Redesign
Fully functional application
Low Fidelity Wireframes
High Fidelity Screen design
Interactive Prototype
Wiring Framing Tools Eg. Paper / Pencil Sketch
Design Tools Eg. Photoshop, Sketch
Prototyping Tools Eg. Invision, Flinto, Flash, FramerJS
UI Development Eg. Android Studio, XCode
High Fidelity Screen design
Interactive Prototype
Requirement Changes
Evaluate and Redesign

Figure 1-1 Design-Development Process

Design team interacts with different stakeholders, gather the app requirements, and sketch the low fidelity prototypes on paper. High-fidelity screen designs are designed using tools like Photoshop or Sketch. The high fidelity screen designs provide the exact look-and-feel of the final product. These designs are handed over to the programmers along with detailed explanation of the navigations/transitions, transition effects, and animations. Since these visual elements are hard to be documented, nowadays, designers are creating interactive prototypes using tools such as Invision[3], Flinto[4]. These tools allow a user to upload the visual designs and create navigational flows and effects.

Our key observation is that the tools mentioned above are not suitable for prototyping in-screen animations and they cannot generate code specific to a platform (Android or iOS). Still, designers use them for defining inter-screen in response to user interaction such as click, long click or swipe with the transition effects such as slide, fade, etc. These tools allow users to upload screen designs and create clickable rectangular regions on the input images called as Hotspots. Thereby enabling users to create more realistic or complex navigations between different screens/pages of the application. The output deliverable of this type of tools is an interactive prototype. It can be previewed on a web application or emulated within another application provided by these companies. They do not generate platform specific code and hence is not capable of generating an output deliverable capable of physically running a device itself. Also, these prototypes do not export a single artifact such as images, texts, dimensions of images, style information, etc. These tools are not capable of prototyping in-screen animation. These tools allow only a few specific set of actions (user interactions) and effects.

Consequently, designers either use conventional animation creator such as Adobe Flash, Adobe After Effects, etc. or modern framework like FramerJS[15]. The former is more of a design studio whereas the latter requires the knowledge of scripting technologies (Javascript and CoffeeScript). The output deliverable of the conventional ones says Flash is video files displaying each animation present and reworking these Flash animation videos for every design iteration is very expensive. Since the output deliverable is a video, developers cannot reuse the output deliverable in the development process. In the case of FramerJS, the animation is generated using JavaScript, FramerJS do create output deliverable targeting a platform (Android or iOS) and animation scripts developed for FramerJS is not consumable in the app development process for Android or iOS.

The designer spends at least 80 hours for creating the visual designs and prototypes [18]. That is a substantial time in the app development process. Even though designers spend a lot of time on sketching the app, the output deliverables such as screen designs of the application or the interactive prototypes and animation prototypes are not reusable in application development process. Since the deliverable of the prototyping tools is just an interactive prototype that viewed on a web application or emulated on a device that cannot be installed physically on a mobile device, they do not generate platform (Android or iOS) specific code for creating the screen navigation or animations. The programmers have to rework the entire work of the designers in the making of the inter-screen transitions and in-screen animations.

Additionally, iterating the design improves the quality of the user interface [17] that is, sketching the visual screen designs, creating a prototype, evaluating the prototype, modifying the visual design and repeating the entire process. Thus, the constant rework of maintaining the prototyping is another setback for designer’s productivity.

These challenges would become much more significant as many companies are adopting *Mobile First* strategy, and interestingly several companies are adopting *Mobile Only* strategy.

Specifically, our technique solves the following problems,

- Creating inter-screen prototypes (say Invision) and in-screen prototypes (say Flash) need separate platforms, adding more challenges while iterating the designs.
- Neither the interactive prototypes nor the in-screen prototypes are reusable in the actual application development process.

We focused on solving above mentioned challenges and came up with the following approach,

- Providing a unified platform for building prototypes both inter-screen transitions (navigations) and in-screen animations. It can be used for building prototypes that could be installed physically on a mobile device would save a lot of time and money in the development process.
- Using REMAUI for preprocessing the input images for generating the user interface code specific to a platform along with the following enhancements,
    - Transition extension: The computed view hierarchy is overlying on input images. We allow the users to create navigation links with transitions effects. Finally, on exporting, compilation ready source code containing code blocks for creating the specified transition is auto-generated.
    - Animation Extension: REMAUI Animation extension analyses the view hierarchy of the input bitmaps and infers the in-screen animation. Finally, on exporting, compilation ready source code for creating the inferred animation is auto-generated.

Summarizing the major contributions made,

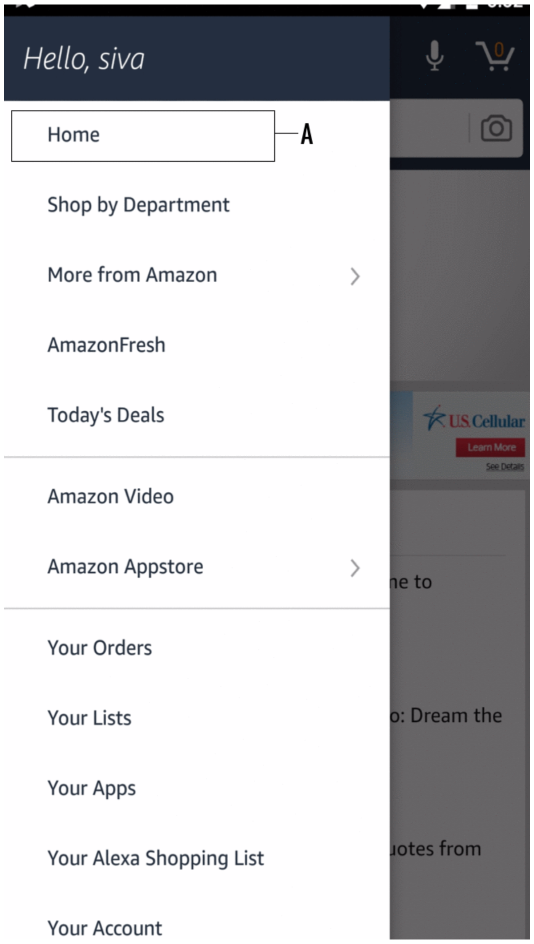
- Enhanced existing REMAUI with Animation and Transition extensions. Transition extensions allows the user to interact with computed view hierarchy and create navigation links. Whereas Animation extension analyses the computed view hierarchies and infers an in-screen animation. In both these cases, our prototype tool that generates code for Android UI, transition, and inferred animations.
- In an evaluation on 10 screenshots from Amazon Underground app for Android, the prototype generated code for transition in 1.7s
- In an evaluation on 10 out of Top 30 popular Android application, the prototype generated UIs similar to the original evaluated by pixel-by-pixel comparison. The average runtime for inferring is 26s.

I

Motivating Example

2.1 Transition Extension:

Let us consider an UI designer is creating an interactive prototype of the following real world example Amazon Underground App (version 10.3.0.200) for Android. Figure 2-1 and 2-2 represents the Application's menu screen and landing screen.



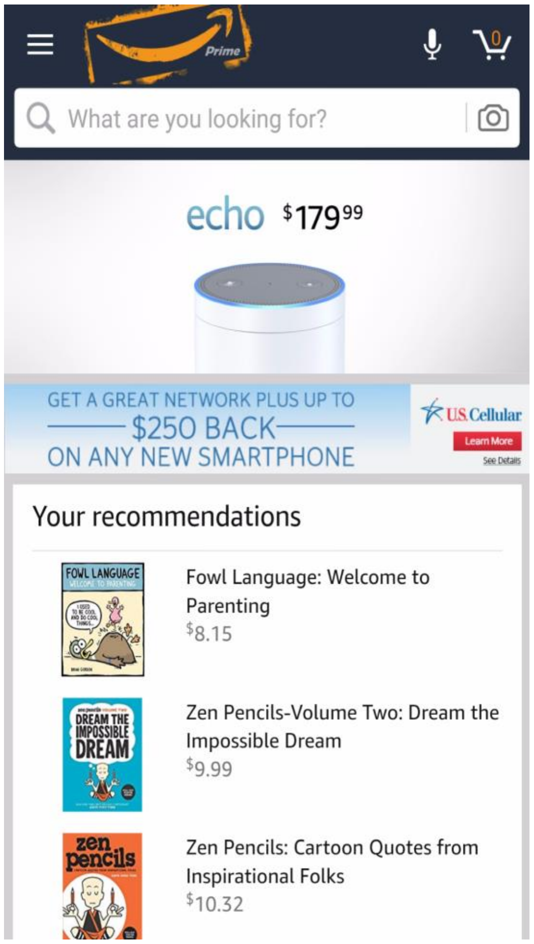

Prime
What are you looking for?
echo $179.99
GET A GREAT NETWORK PLUS UP TO
$250 BACK
ON ANY NEW SMARTPHONE
US Cellular
Learn More
See Details
Your recommendations
Fowl Language: Welcome to Parenting
$8.15
Zen Pencils-Volume Two: Dream the Impossible Dream
$9.99
Zen Pencils: Cartoon Quotes from Inspirational Folks
$10.32

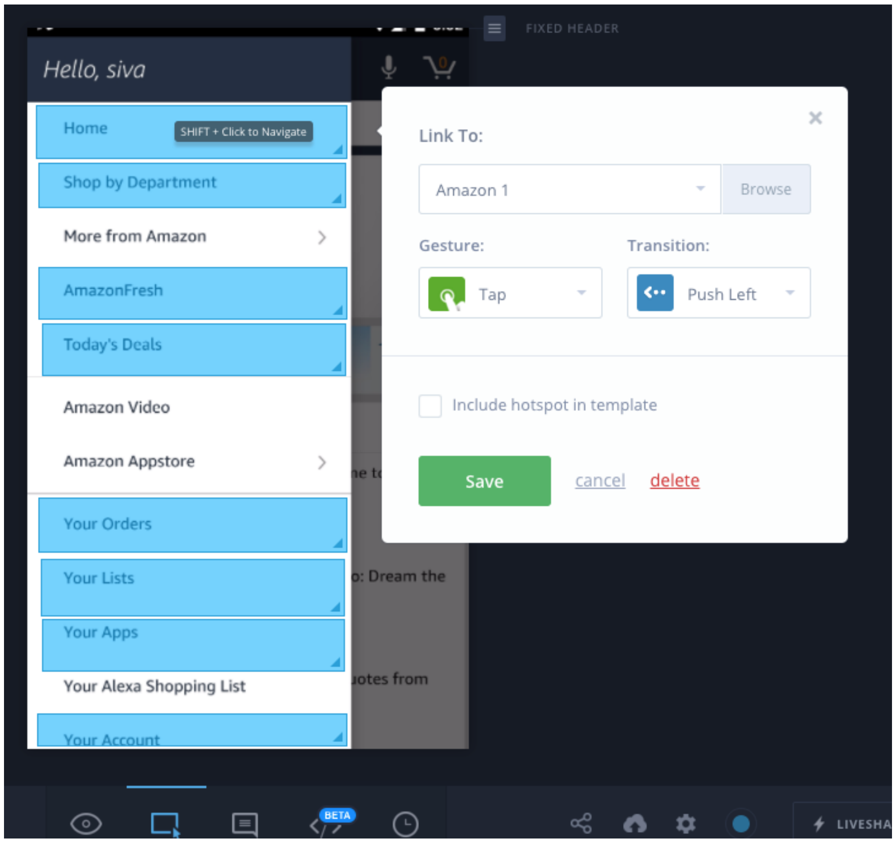
Figure 2-1 Menu screen design of Amazon Underground app for Android

Figure 2-2 - Home screen design of Amazon Underground app for Android

The designer has to create a prototype that allows the user to tap/click on the area represented by '*A*' in Figure 2-1. On tapping the specified area, the landing screen slides in forming a transition or navigation.

2.1.1 Case Study: Invision

Invision, a web-application allows the user to create a clickable rectangular area called as *Hotspot* on the input bitmap. "Build" mode is selected as shown in Figure 2.3. In this mode, Invision allows the user to draw rectangular regions over the image. A rectangular area surrounding the menu *Home* is drawn and shown in Figure 2.3.



Link To: Amazon 1

Gesture: Tap

Transition: Push Left

Figure 2-3 - Working of Invision tool on Amazon app's menu screen

On clicking the specified location, the panel pops out allowing the user to select the triggering action (*tap*) with transition effect (*Push-left)* and destination screen design.

After saving the navigation, the designer has to switch to the "Preview" mode shown in Figure 2.4. This mode would emulate the navigation on the web browser as shown below. Clicking on the "Home" menu triggers the Home screen design shown in Fig 2-5 slides in

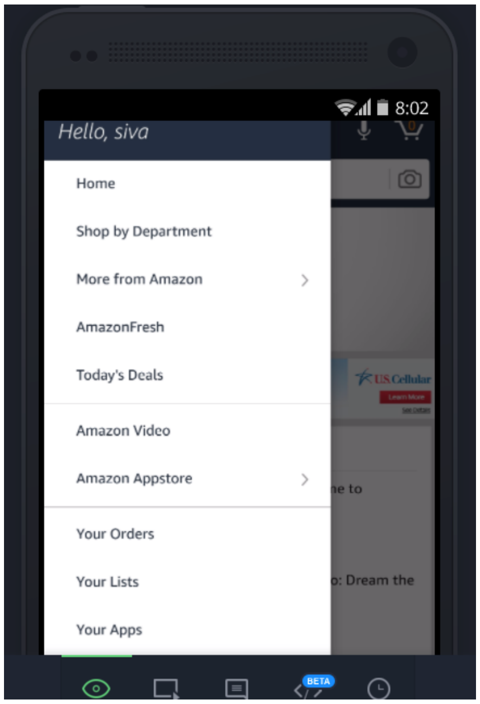


Figure 2-4 - Invision preview of menu screen design

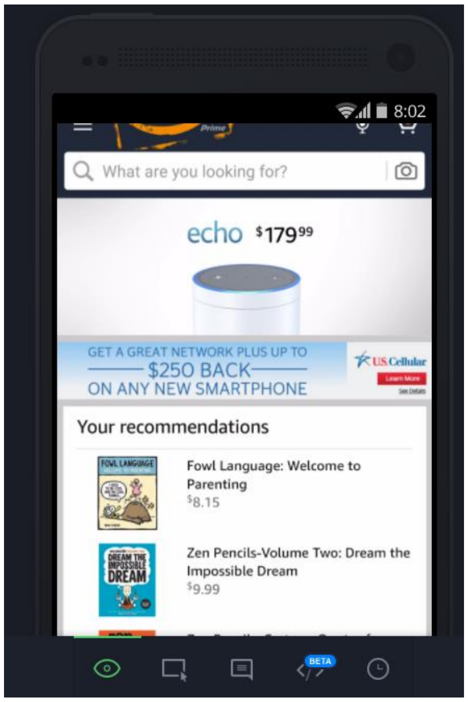
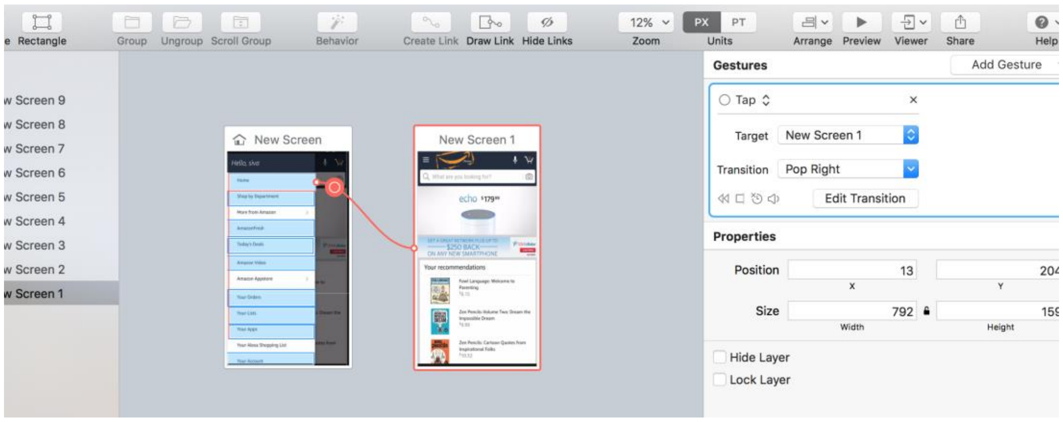

8:02
What are you looking for?
echo $179.99
GET A GREAT NETWORK PLUS UP TO
$250 BACK
ON ANY NEW SMARTPHONE
US Cellular
Your recommendations
Fowl Language: Welcome to Parenting
$8.15
Zen Pencils-Volume Two: Dream the Impossible Dream
$9.99
BETA

Figure 2-5 - Invision preview of home screen design

2.1.2 Case Study: Flinto

Flinto provides a stand-alone application for Mac. Similar to Invision, Flinto allows drawing hotspots over the image and creating links representing the navigation.



Zoom 12%

Tap

Target New Screen 1

Transition Pop Right

Position 13 (X) 204 (Y)

Size 792 (Width) 159 (Height)

Figure 2-6 - Working of Flinto on Amazon Underground app for Android

Figure 2-6 shows the working of Flinto prototyping tool. One significant advantage over a web-based tool is that the addition of input bitmaps is instantaneous. The user does not have to wait for uploading the screen designs. The *Rectangle* toolbar button allows the user create a hotspot over the image. *Create Link* or *Draw Link* toolbar button is used to specify the linkage between different screens. Also, additional information such as trigger event (*Tap*), transition effect (*Pop Right*) and target are defined. *Preview* toolbar pops out the simulation of the navigation as shown below,

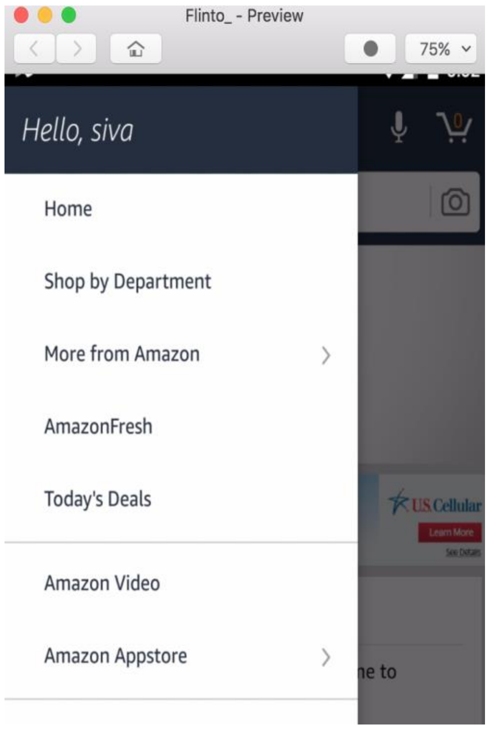


Figure 2-7 - Flinto preview of menu screen design

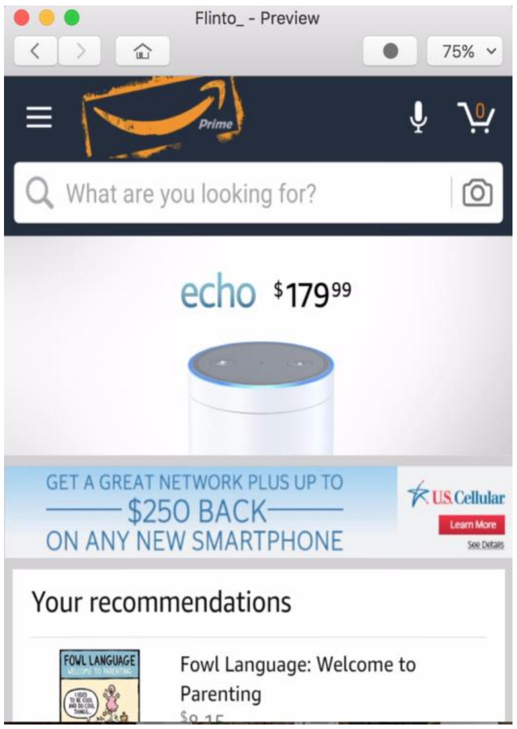


Figure 2-8 - Flinto preview of home screen design

Clicking on the hotspot area defined over the *Home* menu would navigate to the home screen as shown in Figure 2.8 with a Pop effect.

2.1.3 REMAUI with Transition extension

As we know designing is an iterative process, in all these tools, any modification made to the input source image would invalidate the hotspot information, or the links created. There is a constant rework of creating hotspots and links. As mentioned before, these prototypes are throw-away prototypes. After spending quite some time on creating the prototype, it is not useful in the development process.

Our approach is developing a tool for users to interact with REMAUI preprocessed input bitmaps. REMAUI would apply computer vision and OCR on the source images, identify different visual elements and arrange in a hierarchical order. The input screen designs are displaying these visual elements as an overly. Users interact with these hotspots and define various transition and effects. This way of automatically populating the hotspots has a significant advantage in the development process eliminating the constant recreation of these hotspots for every iteration. REMAUI with the transition extension exports source code of Android UI, specified transition, images, and text. The generated source code is compilation ready. Our tool packages the necessary artifacts such as source code, layouts, styles and strings as an android application. If needed, we can install the generated application installed physically on a device.

Figure 2-9 represents the working of REMAUI transition extension on Amazon Underground App for Android. We provide the Menu screen design (Figure 2-1) and Home screen design (Figure 2-2) as the input.

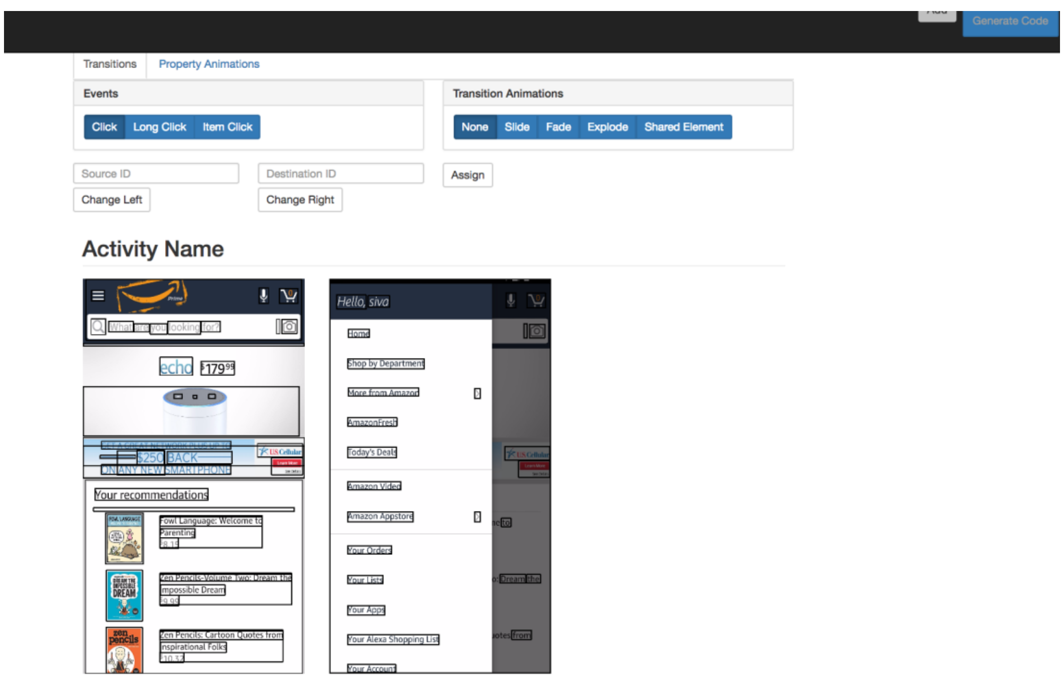
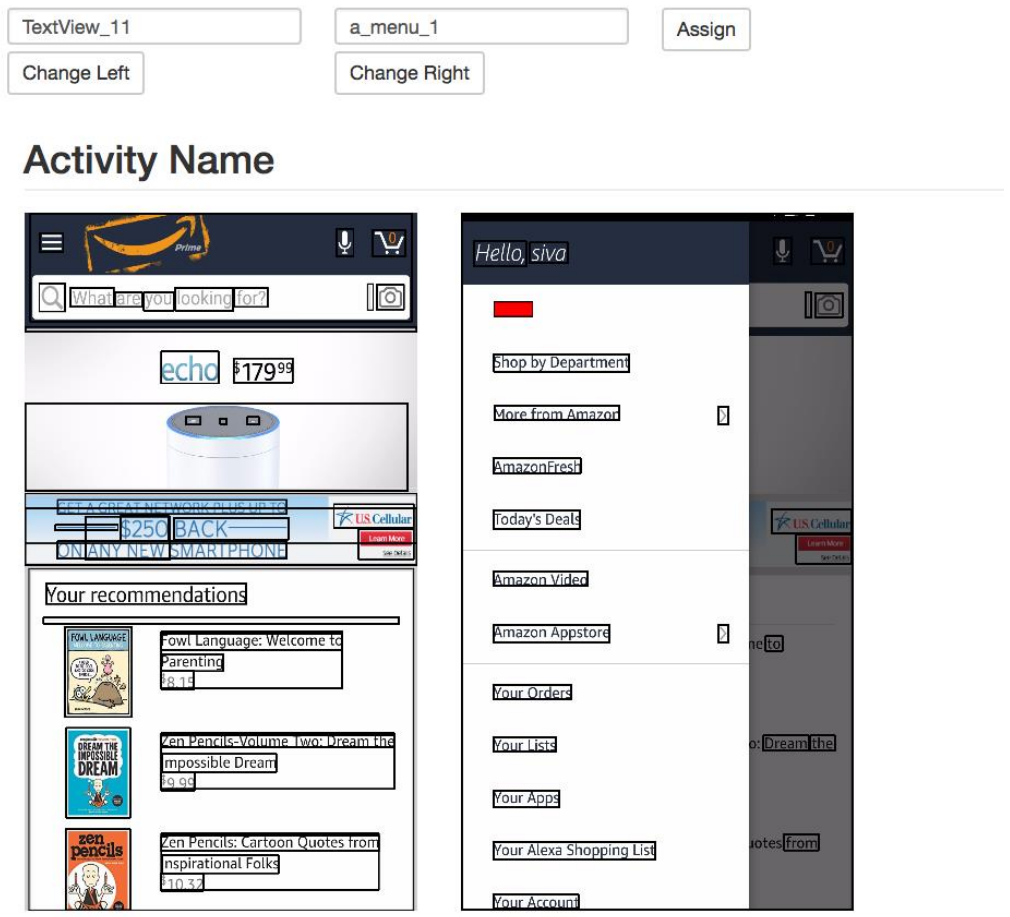


Figure 2-9 - Working of REMAUI Transition extension on Amazon Underground App for Android.

REMAUI preprocess the input images are preprocesses and identify the view hierarchy of the input images. Our prototype represents the computed view hierarchy as rectangular box over the image. The web application allows the user to click on the rectangular area as in Figure 2.10. When the user clicks on a rectangular box, the prototype highlights the box with *Red* color and the Text boxes above the screen designs displays the property ID of the element. In this case, Figure 2-10 displays *TextView_11* as the property ID. The code generator use these property IDs.


TextView_11

a_menu_1

Activity Name

Figure 2-10 - Working of REMAUI Transition Extension on Amazon Underground app for Android with Highlighted user click

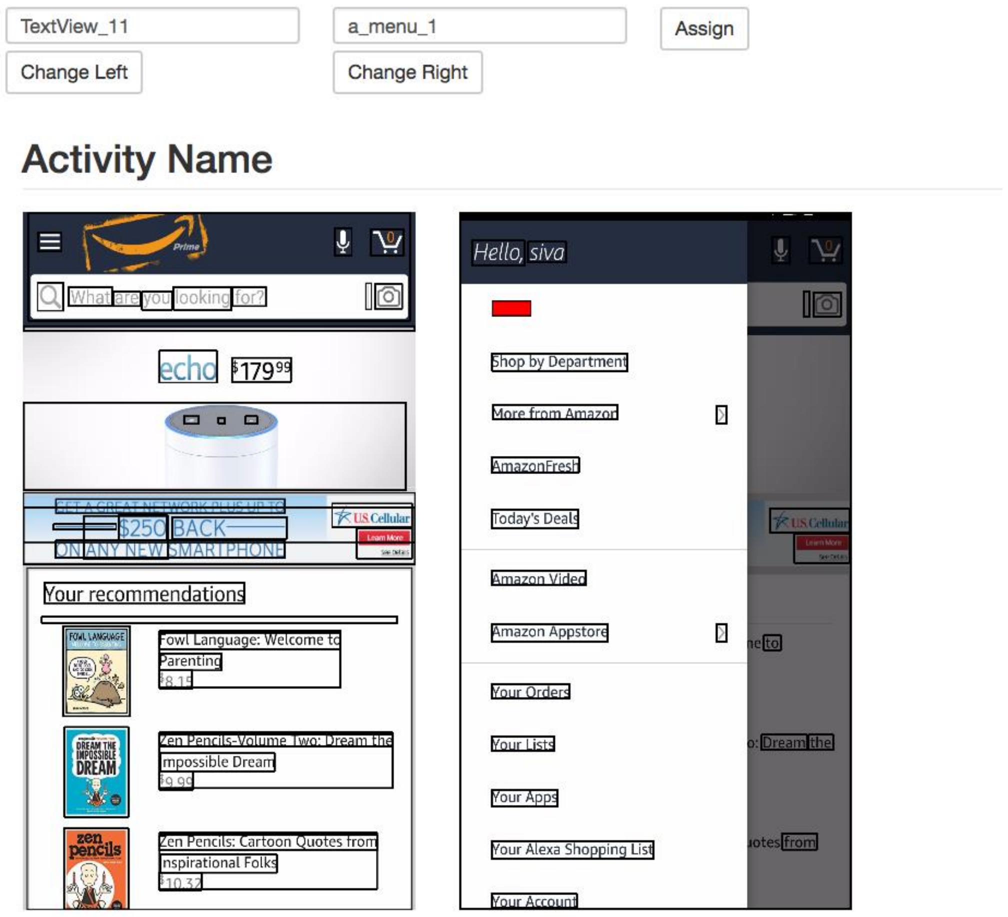


Figure 2-11 - Working of REMAUI transition extension on Amazon Underground app for Android with line representing navigation

The designer selects the *Event* and *Transition Animation* from the web interface and clicks on a rectangular region. The user then clicks and drags towards the destination activity (in this case Amazon Home screen design). When we release the mouse button, a line is drawn representing a navigation flow. Figure 2-11 shows the creation of navigation. Clicking on *Assign* button saves the navigation. Clicking on the *Generate Code* button generates the Android code for UI and transitions.

Time taken for building a basic interactive prototype for Amazon Underground app for Android (version 10.3.0.200) for 10 screen designs on Invision, Flinto and REMAUI with transition extension are as follows,

Product	Time taken in seconds	
Invision	Upload Inputs	37.8
	Assigning navigation	122
Flinto	Upload inputs	12
	Assign navigation	192
REMAUI with Transition extension	REMAUI processing	102.3
	Assigning navigation	135
	Code generation	1.7

Table 2-1 - Comparison of time taken for uploading and assigning navigation

On an average, it takes 4 – 5hrs for developing an UI for Android or iOS from screen shots.

2.2 Animation Extension

Consider the following real-world example Facebook for Android (version 105.0.0.23.137) Figure 2-12 (a) is representing the initial state of the animation, Figure 2-12 (b) is displaying the intermediate state and Figure 2-12 (c) displaying the final state of the animation. Visually, the Facebook logo image undergoes two transformations slides towards the top, forming the *Translation* animation and grows before reaching its final position, forming the *Scaling* animation. The logo changes its position and increases in its size as it reaches the intermediate state. Then the system renders the elements not part of the animation. From an Android developer's standpoint, the initial, intermediate and final screens are all part of same *Activity.* It is an example of *View Property* animation.

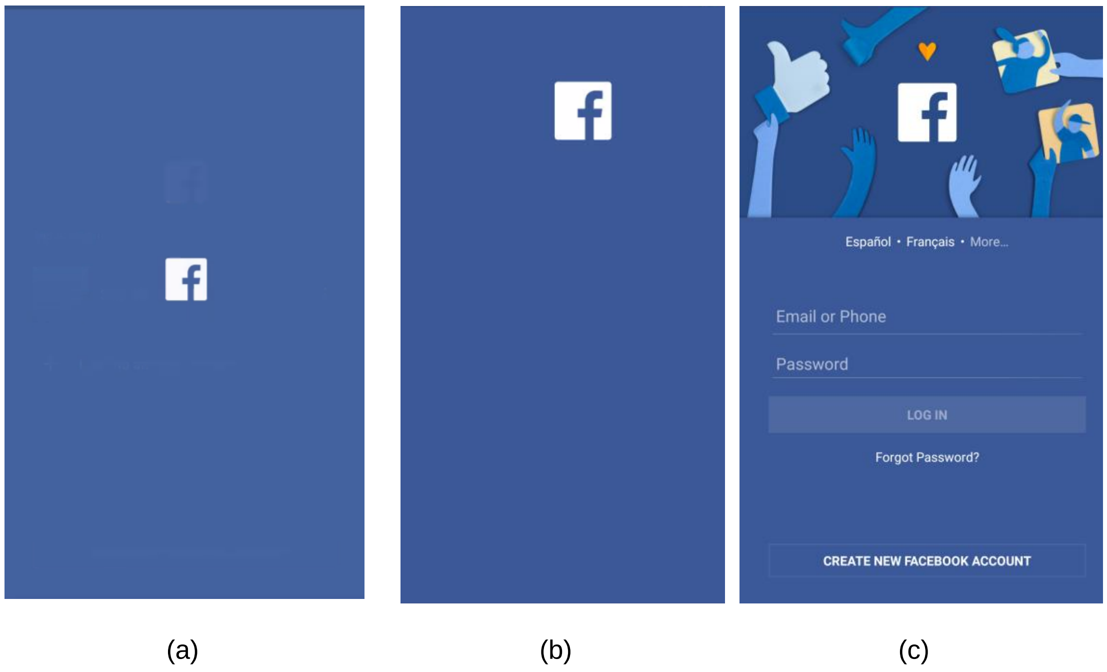


Figure 2-12 - Real world example Facebook’s login screen animation

Nowadays, the design team would be using conventional approach of using Flash to prototype this animation or a modern approach to using FramerJS or similar tool to prototype the animation. With both these approaches, creating a prototype needs a prototyping tool different from tools used for prototyping app’s navigation/workflow. This adds a significant overhead in the design phase. Similar, to the challenges in prototyping workflows, any change in screen design would require a complete rework of the animation prototype.

These tools allow the user to export the preview in specified format accessing through their application. Embedding a prototyping within another application would cause some delay in the orchestration of the animations or some lagging in playing it on devices because of the hardware constraint. The user experience does not match with real application’s user experience. Our approach is to keep a unified platform for prototyping application workflow/navigation and to reduce time by predicting possible animations. Since our approach is using REMAUI as the preprocessing technique, we would be able to generate code for creating the animation and the UI itself. Hence, the designers/developers would be able to obtain the real experience of the final app as prototype would contain expected UI, navigations, and animations. Predicting animations would be an extension of REMAUI itself. The designers/developers would provide the necessary inputs such as screen designs of the initial state and the final state of the animation. Using the output of REMAUI, we could identify the different views and their positions, and by using image-processing techniques, we could compute the view undergoing the animation.

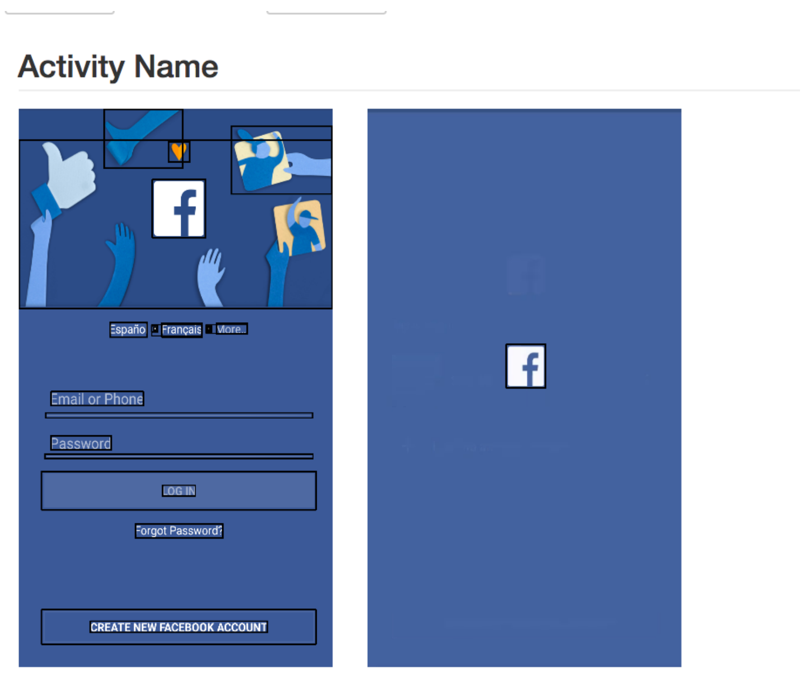
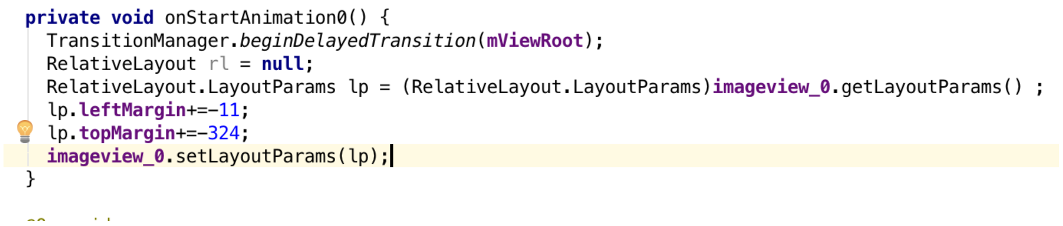


Figure 2-13 - Working of REMAUI - Animation extension on Facebook app for Android

Figure 2-13 represents the working of REMAUI with Animation extension on Facebook app for Android. The initial state in Figure 2-12 (a) and the final state in Figure 2-12 (b) of Facebook Login screen animation are the inputs. Applying REMAUI on the input screen design generates the view hierarchy. Our prototype analyses view hierarchies of the input screen design and infer possible in-screen animation. Clicking on the *Generate Code* generates the Android code for UI, animation. Figure 2.14 shows the snippet of the generated code.



```
private void onStartAnimation0() {
    TransitionManager.beginDelayedTransition(mViewRoot);
    RelativeLayout rl = null;
    RelativeLayout.LayoutParams lp = (RelativeLayout.LayoutParams)imageview_0.getLayoutParams() ;
    lp.leftMargin+=-11;
    lp.topMargin+=-324;
    imageview_0.setLayoutParams(lp);
}
```

Figure 2-14 - REMAUI Animation extension generated code for Facebook Android animation

REMAUI Animation extension also generates the necessary layout XML file, and other JAVA attributes and methods for defining the view elements say *imageview_0* and *mViewRoot.* Code blocks for setting up the visibility of the elements as part of the animation is also auto-generated.

The time taken to infer a possible in-screen animation from the Facebook login screen designs is 27s.

I

Background

3.1 Native apps

Native apps are applications that run physically on a device and are coded specifically to run on an operating system such as Android or iOS. Each of these platforms provides comprehensive set User Interface (UI) widgets. Also, these platforms provide a detailed specification for designing UI such as Material design [7] and iOS Human Interface Guidelines.

Mobile friendly website (Responsive Web Designs) is an alternative for developing UI that fits all sizes and platforms. It is possible to create an UI very similar to that of a native UI widget, but the user experience is not the same. Native apps provide a lot of control over the hardware/device such as GPS or camera. It is even possible to spawn background threads and pre-process and cache information before loading. Native mobile apps could reduce the amount of data transferred between the server and app because the UI is rendered locally on the device. On the contrary, a mobile website is just a client, the wait time for the server to respond back is significantly more, and JS, CSS and images are loaded externally from the server. Apart from following the design guidelines of the platforms, UI designers must consider the following parameters in android,

1. Screen sizes and orientations – Appropriate layout with constraints should be used to support various screen sizes and orientations.
2. Screen density – resources dimensions such as width, height, margin, etc. should be defined in resolution independent pixels (dpi) and bitmaps

(images) should be scaled appropriately for low, medium, high and extra-high densities.

Designing an app UI is very different from designing desktop (PC) applications. Mobile apps are designed to support these variations whereas PC applications work in fixed width mode. The second major difference is processing speed. Desktop CPUs are much faster when compared to current generation multicore mobile CPUs [1]. Responsiveness of the mobile application is very crucial for the smooth function of the application. In the case of Android, the application being unresponsive for 5s triggers “Application Not Responding” (ANR) dialog and provides the user an option to quit. This happens a lot of time when compared to PCs because of the reduced memory and processing power. The last major difference is the battery. Mobile applications should be designed considering lesser battery capacity. For instance, an application performing too many background processing tends to consume more battery. Hence, there should be a proper trade-off between battery consumption and performance (responsiveness).

3.2 Prototype

The prototype is an early model of a product built to test a concept. Nowadays, many software companies have adopted rapid prototyping methodologies. Ideally, designers would make prototypes for communicating their ideas effectively. These prototypes are of two type’s namely low fidelity, and high fidelity prototypes based on the tools used [16]. Low fidelity prototypes consist of a series of static screen designs sketched on paper in general. Tools like Invision and Flinto helps in building interactive prototypes. The digital prototypes represent the complete functionality of the application and the final look and feel of the product. In the modern developmental process, most companies validate the concept using digital prototypes. In addition, the final prototype acts as the reference for developers as well further in the development process. *Prototype Builders* are tools created for building interactive prototypes. For example, *Invision* is allowed the users to upload the bitmap image of the screen designs and create navigation workflows across the screens. The prototype could be previewed on a web browser and a mobile device as well.

3.3 Hamming Distance

Hamming distance between two strings of equal length is the number positions at which the corresponding symbols are different. For example Hamming distance between 1011101 1001001 is two.

3.4 Android

3.4.1 GUI Framework

Android GUI framework consists of UI elements such as layouts (containers) and widgets (leaf nodes). Layouts are special kind of widgets, which could hold other widgets and layouts could even nest other layouts. Different types of layouts are Linear Layout, Relative Layout, Frame Layout and Grid Layout.

Commonly used widgets are as follows; *View* is generic view and parent Class of all widgets. Text View is read-only text. Edit Text is editable text element; Image View is bitmap; Button is text button.

Activity is the visual representation of an android application. The activity consists of widgets and users interact with the widgets. An application can have several activities. Users could navigate from one Activity to another with various effects.

Android is an event-based framework and events are bound to actions. User interaction is the input mechanism to invoke an action. Actions are the response to an interaction. They are user-defined methods registered as a callback to an event.

With the introduction of material designs, animations became an integral part of the application. Android framework is pretty robust that allows animating any view element. *View animation* is the transformations such as position, size, and rotation applied on the view elements.

3.4.2 Input Events

Android supports several ways for the user to interact with the application. To support user interactions Android framework provides two major components, events, and events listeners. In programming terms, an event is an object that is created when something changes within the UI, for example, clicking on a button, typing an edit field, etc. An event listener is an interface that contains a callback method. View objects such as Text View, ImageView, Button, etc. registers itself with the event listeners. Event listeners respond to the events of a particular view object. Android framework is responsible for triggering the event listeners based on the user interaction. Commonly used events are Click / tap, long click, swipe left / swipe right and the corresponding callback methods are onClick, onLongClick, onTouch respectively

3.4.3 Transition

The Introduction of Material Design [7] remarkably enhanced transition and animation frameworks. Activity transitions in material design app provide a visual connection between different activities through motion and transformation. It allows specifying animation effects enter and exit transitions while navigating from Activity to Activity (Screen-to-Screen).

- An Enter transition specifies how views in an activity enter the scene. Say for example, in the *slide* exit transition, the views enter the scene with a sliding out towards left or right.
- An Exit transition specifies how views in an activity enter the scene. Say for example, in the *fade-out* transition, the views exit the scene with a fading out animation.

Material Design introduced in Android 5.0 (API Level 21) supports transition framework [8] with these following enter and exit transitions,

- Explode – Moves views in or out from the center of the scene
- Slide – Moves views in or out from one of the edges of the scene
- Fade – Adds or removes a view from the scene by changing its opacity.

Consider the following example of a simple transition. Below is the snippet of two different activities such as *MainActivity* and *DetailsActivity*.

MainActivity.java

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_transition);
    setupWindowAnimations();
}

private void setupWindowAnimations() {
    Slide slide = TransitionInflater.from(this).inflateTransition(R.transition.activity_slide);
    getWindow().setExitTransition(slide);
}
```

DetailsActivity.java

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_transition);
    setupWindowAnimations();
}

private void setupWindowAnimations() {
    Fade fade = TransitionInflater.from(this).inflateTransition(R.transition.activity_fade);
    getWindow().setEnterTransition(fade);
}
```

On navigating from MainActivity to DetailsActivity, views in the MainActivity moves out from one of its edges while the views in the DetailsActivity show up with a fade-in effect.

3.4.4 Animation

Animation package [9] is robust and used for animating any view element in an activity. Animations provide a visual cue of the intended action. Android supports API to perform a tween animation on Views. The programmer provides with the initial and final state of the view and tweening technique generate the intermediate states/frames. The supported tween transformations are,

- Translate
- Scale
- Rotate
- Alpha

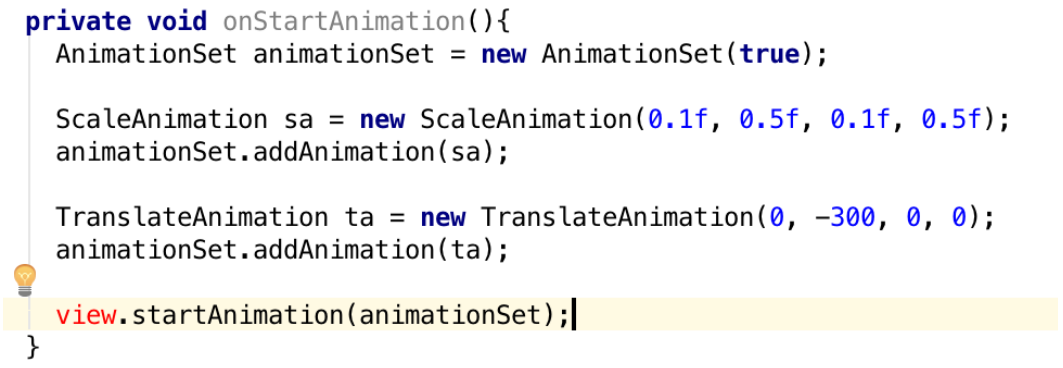
For example, applying a translation transformation to a Text View, programmers define the initial and final positions in code, and the Android system generates a smooth animation changing the view’s position to its final state. We can define these animations by either using declarative XML style or using APIs [9]. When using the declarative XML style for defining the animation, programmers create the correct XML files and places them in /res/anim/ directory of the Android project. In the case of APIs, *Animation* is the base class for all transformations. It contains the necessary attributes for the animation such as *duration*, *repeatMode,* and *repeatCount.*

Playing a sequence of such animations grouped together forming an AnimationSet [9]. AnimationSet acts a single transformation, and it contains other simple transformation defined earlier. Properties defined for the atomic animations stands valid when grouped together.

The difference between using XML-based declarative style and APIs for generating the intended animation is the former is statically bound, and latter creates the animation at runtime. REMAUI code generation module uses the APIs for creating an animation. Developing animation with APIs offers more control over the animation for the developers.

Let us consider the animation in Android's Facebook Application login screen as shown in Fig 2.9 (a) and (c). The Facebook Icon is undergoing a translation and scaling transformations.

The following code snippet shows the Android code developed to create an animation,



```
private void onStartAnimation(){
    AnimationSet animationSet = new AnimationSet(true);

    ScaleAnimation sa = new ScaleAnimation(0.1f, 0.5f, 0.1f, 0.5f);
    animationSet.addAnimation(sa);

    TranslateAnimation ta = new TranslateAnimation(0, -300, 0, 0);
    animationSet.addAnimation(ta);

    view.startAnimation(animationSet);
}
```

The two different animation Scale and Translate animations are grouped together to form an animation set. *startAnimation()* method is invoked to play the entire set of animations. At the end of the animation, the target view reaches its final state.

3.5 REMAUI

REMAUI is a software tool to reverse engineer the screen design and generate working UI code specific to a platform (Android / iOS). Android application has a declarative style for defining the UI for various elements such as text views, image views, buttons, and declaring layouts as an XML file called as layout file. This layout file would contain the specification for its elements such as its position, width, height and other style attributes. Nesting Layout tags are possible for placing the view elements at specific positions. In other words, layouts act as a container for different elements and even other nested layouts. Layout XML file would contain two types of nodes, container nodes (layouts) and leaf nodes (text, image, buttons, etc.) forming a view hierarchy.

The working of REMAUI is very similar to that of humans. It identifies the different visual elements on the screen and classifies them as layout or text or image. As widely recommended by the Android platform, REMAUI follows declarative GUI programming [12]. REMAUI identify visual elements and arrange them in a hierarchical fashion with parent –child relationship with layouts acting as a parent and other elements acting like children. This kind of view (images or texts) within another view (layout) forms the view hierarchy. REMAUI converts the View hierarchy to layout XML file as defined by the Android platform. Since REMAUI generate prototypes specific to a platform; it generates the JAVA code targeting Android platform. Generated UI code contains the necessary constructs for binding the layout XML with the *activity*. REMAUI generated UI Code, and layout definitions are compilation ready. REMAUI compiles the source code against Android SDK and packages assets generating a native prototype.

The working of REMAUI works in three main steps,

1. Visual elements (text, images, layouts) are identified using Computer Vision
2. Text recognition by OCR
3. REMAUI internally uses Tesseract as the OCR Tool. Tesseract performs on par with existing commercial OCR tools. Tesseract's precision and recall are less than one. Therefore, even a powerful OCR tool could classify non-text as text. REMAUI removes the false positives by merging the results of computer vision and OCR. REMAUI further refines the output by merging nearby visual elements forming a single element thus forming a minimal *View Hierarchy.*

Step1 comprises of the following activities, translating the View hierarchy to layout XML file and identifying Assets such images and icons. In Step 2 consists of identifying Text styles like size, color, bold, italics.

Consider the following Fig 3.1 (Image Source: http://cseweb.uta.edu/~tuan/REMAUI/), shows the working of REMAUI on Google Hangouts conceptual drawing of iOS platform. REMAUI initially identifies different visual elements using computer vision operations. OCR identifies text from non-textual elements. REMAUI prunes the UI view hierarchy and merges the closer view elements. At this stage, REMAUI identifies List View using heuristic approaches, such as *Identify by Item Size* and *Identify by drawables*. REMAUI analyses the view hierarchy, and if there is a pattern of repeating visual elements then, it may contain a ListView. When the repeating visual elements have a constant size, then a

List view is recognized. Similarly, when the repeating visual elements have the same visual element, then a List view is recognized.

REMAUI extracts icons used in the application and saves as separated icons in the required folder (/res/*drawables*). REMAUI performs OCR and stores text content and styles such as font size, font type, and font color as *strings.xml* and *styles.xm*l respectively. REMAUI generates necessary JAVA source code for binding the UI with content, and the entire app is compilation ready with Android SDK, and generating the output deliverable (.apk file) targeting a physical device.

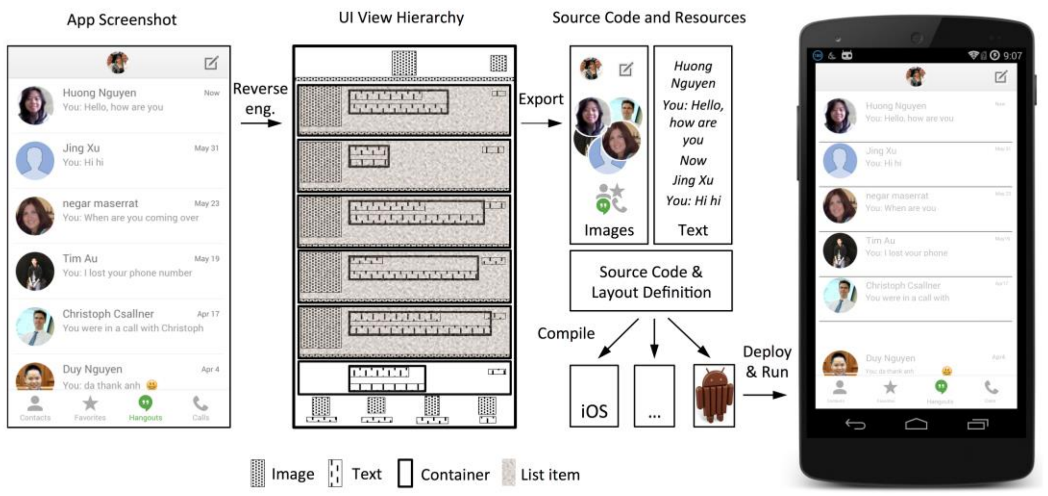


Figure 3-1 - Working of REMAUI on Google Hangouts screen design

3.6 Perceptual Hash

Perceptual hashing [10] is an algorithm that produces a fingerprint of images. The Hamming distance computed between the perceptual hashes of the source images are smaller if the same features are present in the source images. Thus, perceptual hashes could be used to identify similar images. Other commonly used way of finding similar images is performing a pixel-by-pixel (Mean-Squared-Error) match. Unlike pHash, MSEs do not perform well for all cases.

Consider the following example shown in Fig 3.2

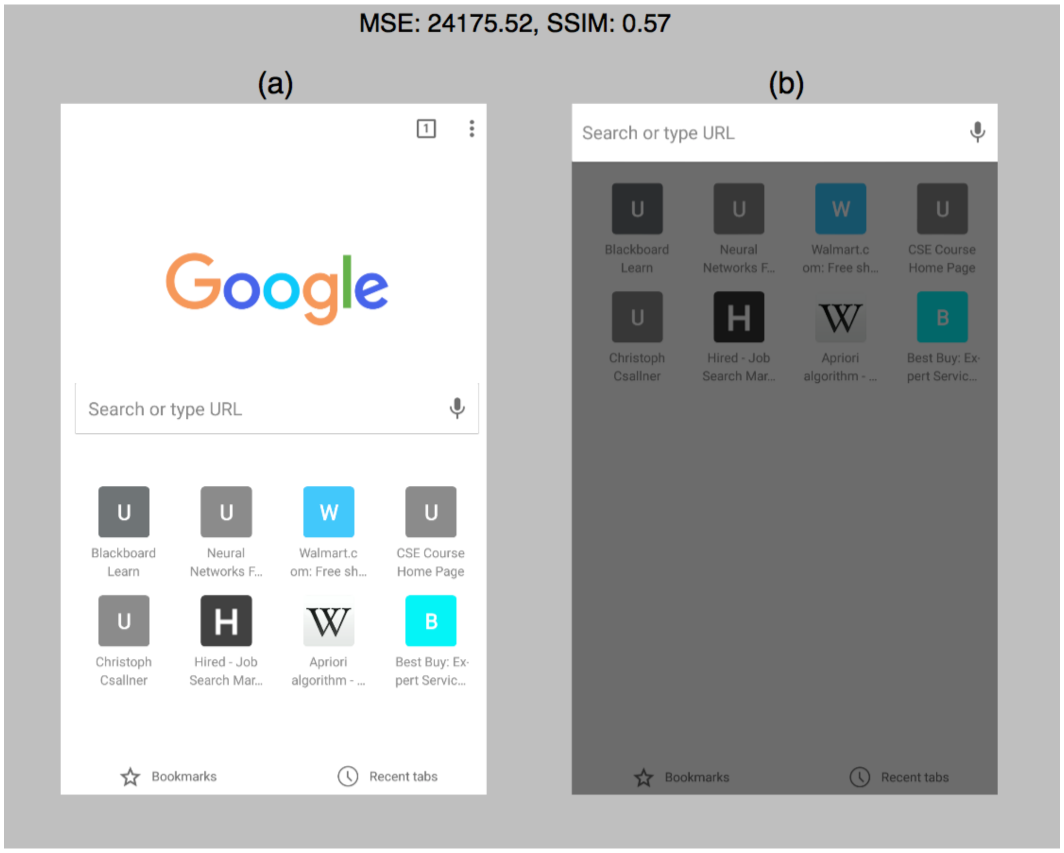


Figure 3-2 - Calculated MSE / SSIM on Chrome application's landing screen

Fig 3.2 (a) and (b) represent the Android version of Chrome application's landing screen initial and final state respectively. The input screens differ in a few elements' position and brightness. Using SSIM [19] algorithm, the similarity was calculated as 0.57. SSIM has a range of 0 to 1. The SSIM index increases with increase in similarity. Identical images have SSIM index as 1.0.

Unlike MSE, pHash works better even when the inputs images are different in scale, color, brightness. Since pHash looks for the dominant feature in the image and does not perform a pixel-to-pixel match. For example, as of ImageMagick version 6.9.4, compares the source images were compared based on the metric pHash. The syntax is as follows,

Compare –metric phash <image1> <image2>

Using the same source images mentioned above the comparison by pHash metric calculated it as **309.709.** This number represents the similarity between the source images with zero for identical ones.

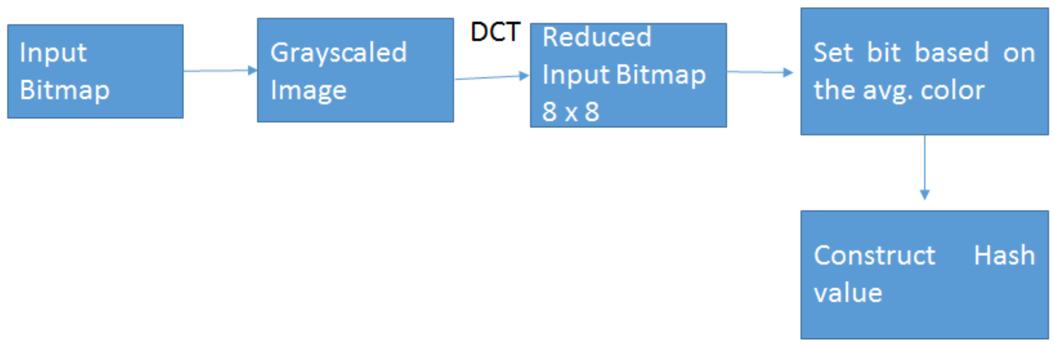


Figure 3-3-Overview of Perceptual Hashing

The working of Perceptual hashing (Figure 3-3) will elaborate more on the robustness of this algorithm. The first step is resizing the source input image to common resolution say (32x32), and we apply the Discrete Cosine Transformation (DCT) on the resized image. Because of using DCT, the resultant image contains higher significant components on the top left and lower significant components on the bottom right. We then select the higher significant components from top-left (8x8) and discarding the rest of the image. Finally, we generate a 64-bit hash string using the reduced DCT components (8x8). The similarity between any two generated hash values is calculated using Hamming distance.

When two hash values are identical, the Hamming distance between them is zero. The similarity increases with a decrease in Hamming distance between them.

With our approach to recognize an in-screen animation automatically, the first step is to compute if the source images are similar. We compute Perceptual Hash or pHash for all image views. We can infer the similarity between input images by computing Hamming distance between the respective pHash values. When the Hamming distance is lesser than a threshold, then the image views are similar. Likewise, when the Hamming distance is greater than a threshold, the image views are similar. After identifying the similar from the screen designs, we are analyzing if they constitute an animation.

I

Overview and Design

4.1 Inferring in-screen animation

The following image (Figure 4-1) displays the login screen animation in Facebook app for Android. Figure 4-1 highlights the image view elements that translates and scales. In the case of Android, it provides APIs that accepts the initial and final state (position, size) of view elements and automatically generates the intermediate frames.

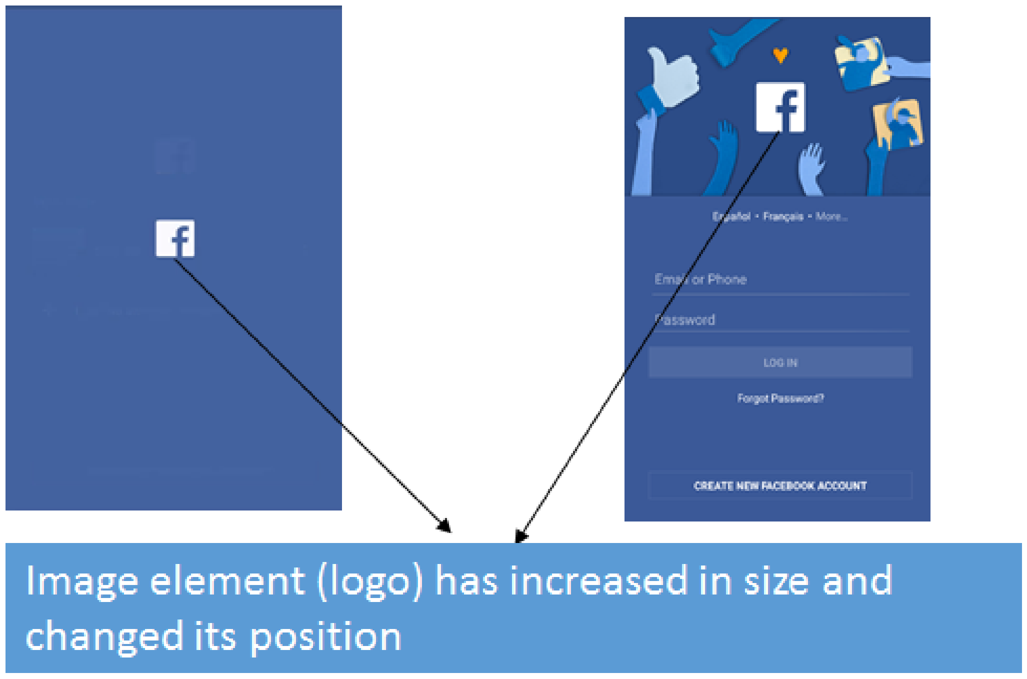

Español · Français · More...
Email or Phone
Password
LOG IN
Forgot Password?
CREATE NEW FACEBOOK ACCOUNT
Image element (logo) has increased in size and changed its position

Figure 4-1-Example of In-screen animation

The working of in-screen inference is very similar to humans. The steps involved are as follows,

1. Identifying view elements present in the inputs
2. Identifying matching view elements present on both the inputs

3. Inferring possible animation

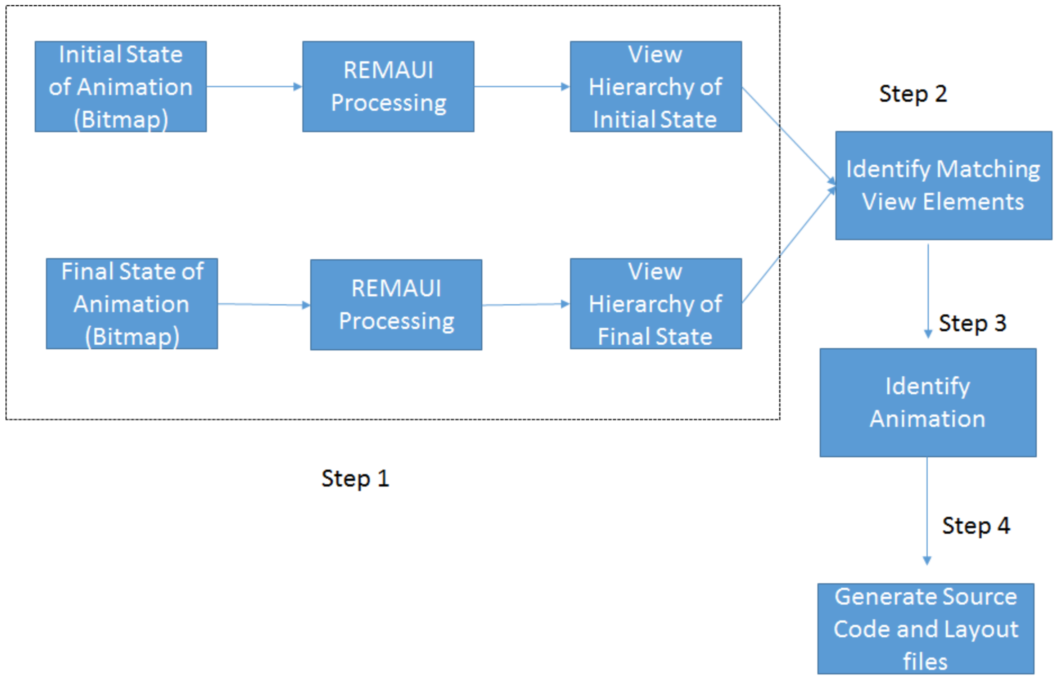

Initial State of Animation (Bitmap)
REMAUI Processing
View Hierarchy of Initial State
Final State of Animation (Bitmap)
REMAUI Processing
View Hierarchy of Final State
Step 1
Step 2
Identify Matching View Elements
Step 3
Identify Animation
Step 4
Generate Source Code and Layout files

Figure 4-2 - Overview of in-screen animation inference

Figure 4-2 displays the various steps involved in in-screen animation inference.

Step 1 – Identifying view elements

The first step in inferring in-screen animation is identifying the view elements such as text views, image views etc. present in input and final states. We use the identified view elements for further processing.

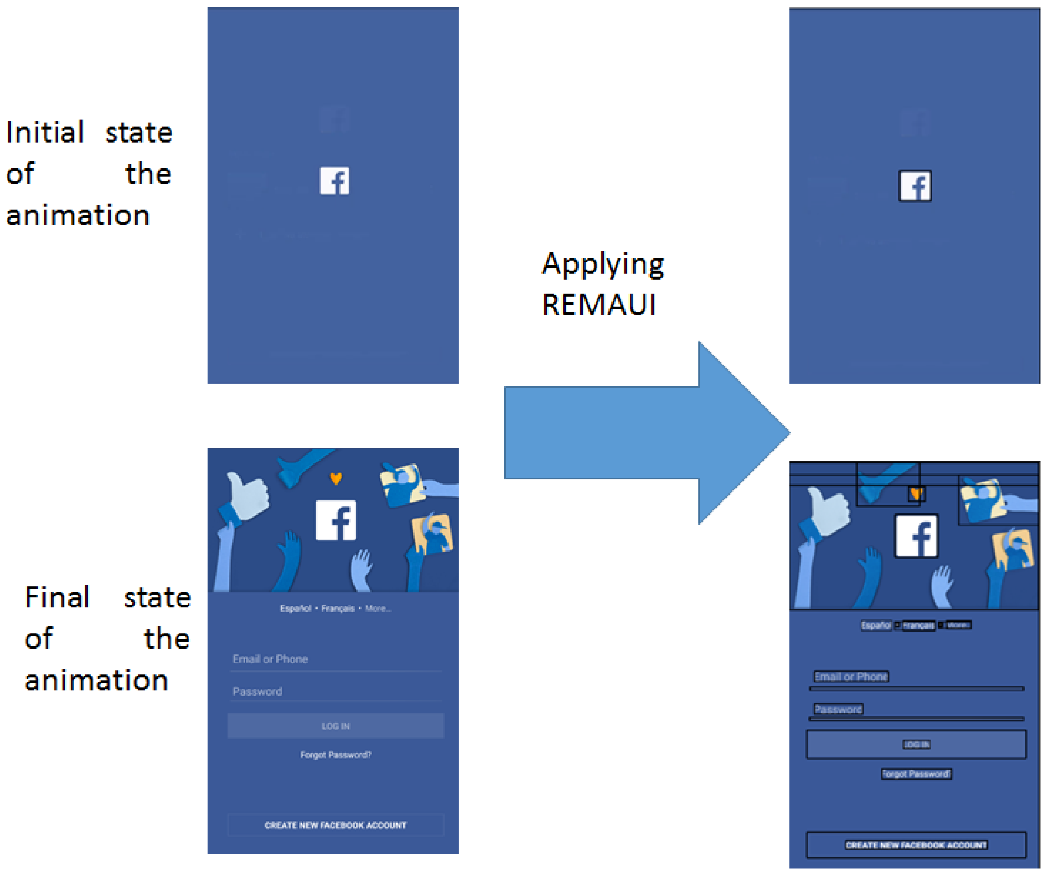

Initial state of the animation
Final state of the animation
Applying REMAUI
Español • Français • More...
Email or Phone
Password
LOG IN
Forgot Password?
CREATE NEW FACEBOOK ACCOUNT

Figure 4-3 - Example of Step 1 processing on Facebook app for Android

Figure 4-3 shows the Step 1 processing on Facebook app for Android. We provide the initial and final state of the animation as input and applying REMAUI generates the view hierarchy of the inputs. The black rectangular boxes represents the view hierarchy.

Step 2 – Identifying Matching View Elements

The next step is identifying the common elements between the screen designs. Identifying matching text view elements is different from identifying matching image view elements.

Step 2 (a) – Identifying matching image view elements

Few common techniques are

1. Mean Squared Error
2. Histogram Comparison [13]
3. Perceptual Hashing

Mean Squared Error (MSE) performs a pixel-to-pixel comparison for identifying the common element. This technique can be applied only when the comparable elements are of the same size (width x height). Using this we cannot identify the resizes elements (image views) shown in Figure 4-1. Hence, we are not adopting this technique.

Another technique is to compute the Histogram of the comparable elements and identifying if they are similar. In this technique, we compute a histogram, one for each channel red, green and blue. We can identify similar images by comparing these based on the standard metrics [7]. This technique is heavily dependent on the color intensity. Histogram matching technique does not work when one of the input images is gray scaled. In other words, it is not color-invariant. Thus, Mean Squared Error (MSE) and Histogram are not ideal choices.

Perceptual Hashing is one of the best contenders for our requirement since it is more accurate in identifying the similarity between images even with scaling or change in color intensities. The runtime complexity of all the techniques is O (n x m) as we are processing a 2D image and identifying similar view is by comparing each image view with a runtime complexity of O (k x l) with k and l as the number of view elements in the source images. The total runtime time complexity is O (kl (nm)). However, we can reduce runtime complexity by caching the perceptual hashes. We cache the perceptual hash for each non-textual element and compute the Hamming distance on the fly. Thus reducing total complexity is O ((k+l) (nm)). With this method of using pHash is much faster and memory efficient than other techniques.

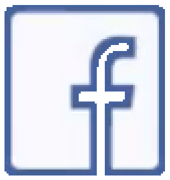
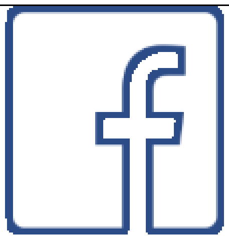
For example, consider the Facebook for Android's login animation (Figure 4-1). This example is exploring in detail the working of pHash technique when comparable images change by scale or size. Our prototype generates the visual hierarchy of the screen designs and represents the visual hierarchy as a rectangular box. Our prototype computes the pHash of all the images and caches in memory. The Facebook icon changes its size and position as part of the animation.

Based on our experiments, the threshold for Hamming distance is 5 that is two image views are similar if and only if the Hamming distance between the perceptual hashes is less than 5.

Name	Heuristics
Hamming distance < 5	Similar elements

Table 4-1 - Heuristics for Perceptual hashing

Consider the following examples,

Image		

Perceptual Hash Value	304FFEC61BFF9	104EFEC61BDD9
Hamming Distance	3	
Similarity	Yes	

Table 4-2 - Comparison of Similar Image views

Since the images are similar to one another, the Hamming distance between the perceptual hashes is 3 and is less than the threshold

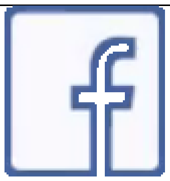
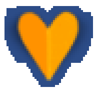
Image		
Perceptual Hash Value	304FFEC61BFF9	1FDFFF3EFE47F
Hamming Distance	23	
Similarity	No	

Table 4-3 - Comparison of Non-similar Image views

Since the images are very different to one another, the Hamming distance between the perceptual hashes is 23 and is greater than the threshold.

Consider another example, Android Chrome (Version 53.0.2785.135) App's landing screen animation in Figure 4.4. This case study explains the robustness of using Perceptual Hashing technique even when one of the inputs is gray scaled as part of an animation.

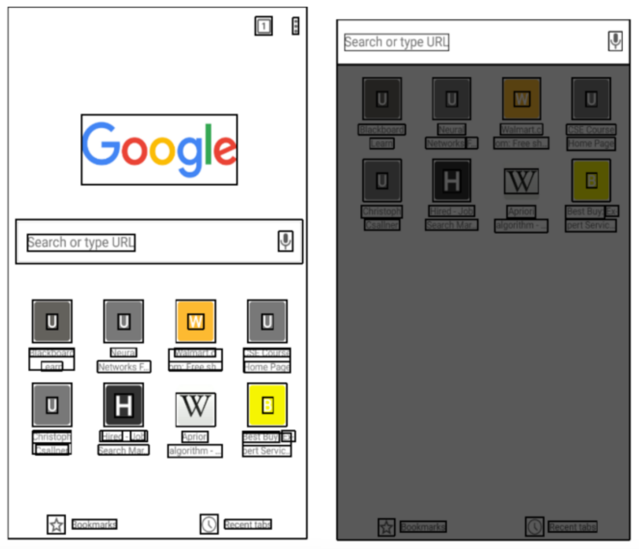


Figure 4-4 - Chrome Landing screen animation Intial and Final states

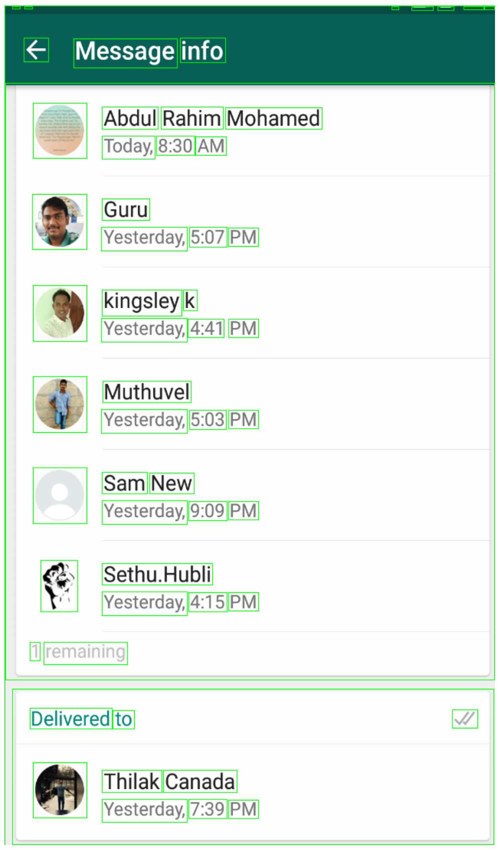
When the search “Search or type URL” edit text gets the focus, the edit text and thumbnails below it are pulled to the top forming a *Translation* animation. Since the edit text is focused, the entire screen is gray scaled.

REMAUI identifies the view hierarchy and it computes the pHash values of all images and caches it. For example, when comparing, and , REMAUI looks up the pHash value of these images and computes the hamming distance as 4. This value is still less than the threshold value (5).

Step 2 (b) – Identifying matching text view elements

Identifying matching text view elements is much simpler. A simple content based matching (text comparison) is sufficient for identifying matching text view elements. A simple text comparison could lead to incorrect inference. Hence, pruning text view hierarchy is essential.

Let us consider a more complex inference, applying our REMAUI technique on the Android version of Whatsapp application (2.17.107) as shown below in
Figure 4-5 and 4-6. Similar to the previous one, REMAUI identifies view hierarchy and the system computes the perceptual hash value of all image views and caches them in memory.
Then REMAUI tries to identify similar image views by computing the respective Hamming distance. We can identify similar text views by comparing their content.


Message info

Abdul Rahim Mohamed
Today, 8:30 AM

Guru
Yesterday, 5:07 PM

kingsley k
Yesterday, 4:41 PM

Muthuvel
Yesterday, 5:03 PM

Sam New
Yesterday, 9:09 PM

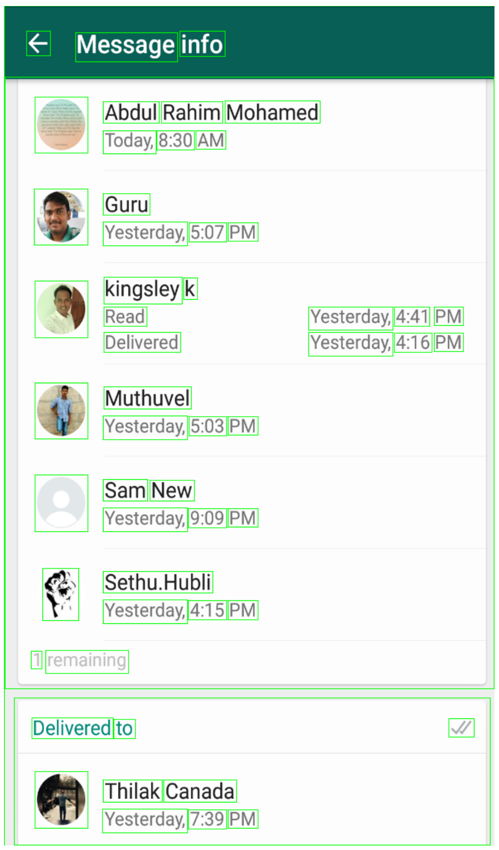
Sethu.Hubli
Yesterday, 4:15 PM

1 remaining

Delivered to

Thilak Canada
Yesterday, 7:39 PM

Figure 4-5 - REMAUI processed Whatsapp initial state


Message info

Abdul Rahim Mohamed
Today, 8:30 AM

Guru
Yesterday, 5:07 PM

kingsley k
Read Yesterday, 4:41 PM
Delivered Yesterday, 4:16 PM

Muthuvel
Yesterday, 5:03 PM

Sam New
Yesterday, 9:09 PM

Sethu.Hubli
Yesterday, 4:15 PM

1 remaining

Delivered to

Thilak Canada
Yesterday, 7:39 PM

Figure 4-6 - REMAUI processed Whatsapp final state

In this scenario, the text “Yesterday” is repeating quite often. When we compare text elements by their content, we might not get the correct matching element. To rectify this, we would be making a complete pass over the view hierarchy (as shown in Fig 4.7 and 4.8) and remove matching elements and thus reduce the candidates eligible for possible animation.

Name	Heuristics
Difference in A and B elements' position < 10 ^ Difference in A and B element's dimension < 10	Eligible for pruning

Table 4-4- Heuristic for Pruning Text View Hierarchy

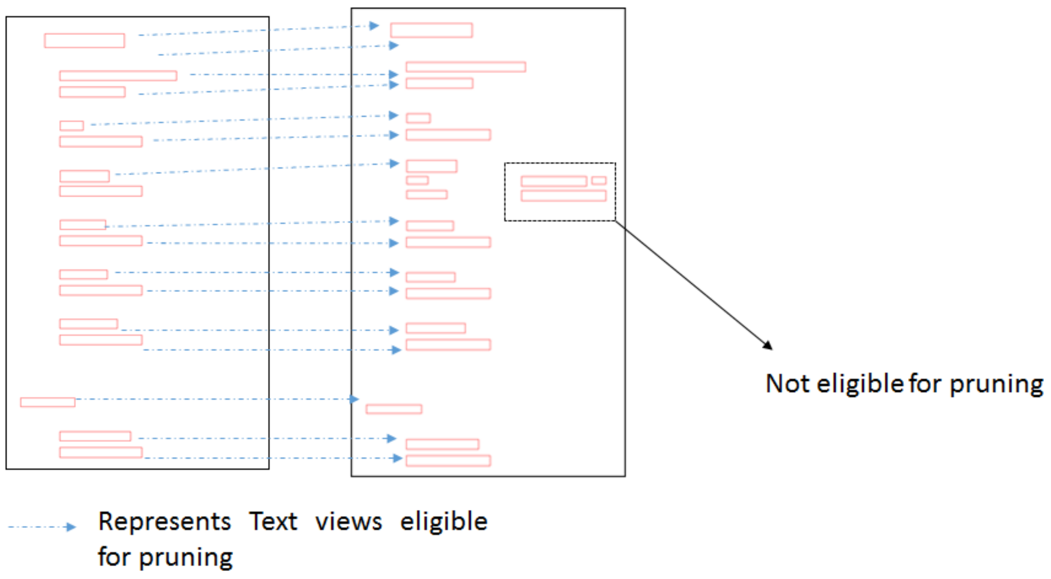


Figure 4-7 - Pruning Text View Hierarchy of Initial and Final states of WhatsApp for Android

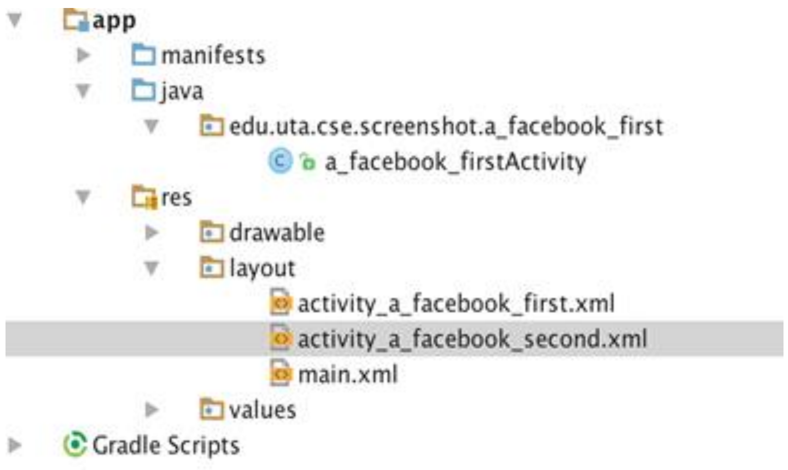
Figure 4-7 shows the text view hierarchy of initial and final states of WhatsApp for Android with elements eligible for pruning. We reduce the search space for a possible animation by eliminating elements based on their positions (Table 4-1). We iterate through the view elements in a view hierarchy, and we compare against all elements in the other view hierarchy. We eliminate elements that are of the same dimension and positioned at the exact same place on both the view hierarchies. With this approach, we would be able to eliminate false positives caused by text comparisons.

Step 3 – Inferring animation

The next step is inferring in-screen animations. After identifying the matching view elements, we can infer possible animations. When we identify a difference in the elements position, then it infers a translation animation. Similarly, when we identify a difference in elements dimensions, then it infers a scaling animation.

Step 4 – Export

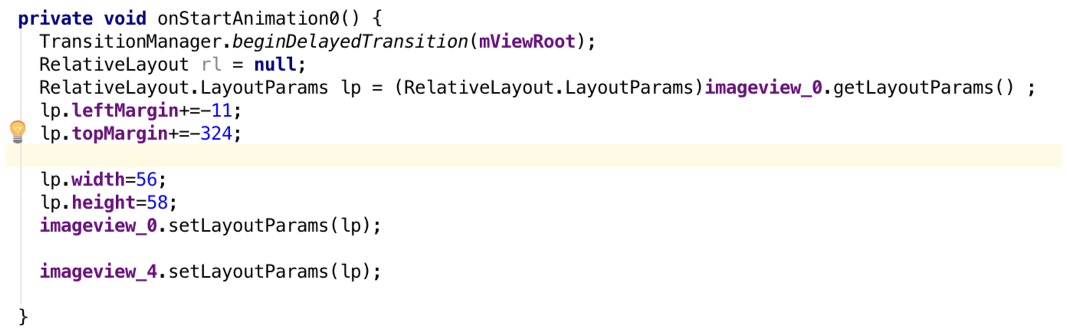
In the case of in-screen animation the initial and final state are part of the same activity thus the generated application has only one Activity (Figure 4-8).



```
▼ app
    ▶ manifests
    ▼ java
        ▼ edu.uta.cse.screenshot.a_facebook_first
              a_facebook_firstActivity
    ▼ res
        ▶ drawable
        ▼ layout
              activity_a_facebook_first.xml
              activity_a_facebook_second.xml
              main.xml
        ▶ values
▶ Gradle Scripts
```

Figure 4-8 - Snapshot of Generated Project Structure

Figure 4-9 displays the snippet of code for creating a translation and scaling animation.


```
private void onStartAnimation0() {
  TransitionManager.beginDelayedTransition(mViewRoot);
  RelativeLayout rl = null;
  RelativeLayout.LayoutParams lp = (RelativeLayout.LayoutParams)imageview_0.getLayoutParams() ;
  lp.leftMargin+=-11;
  lp.topMargin+=-324;

  lp.width=56;
  lp.height=58;
  imageview_0.setLayoutParams(lp);

  imageview_4.setLayoutParams(lp);

}
```

Figure 4-9 - Code snippet for creating transition and scaling animations

This method in Figure 4-9 updates the left and top margin thereby changing the position of the element and changes the width and height thereby changing the size of the element.

4.2 Transition Extension

It is normal for an application to have more than one activity and users navigate from activity to another with transition effects. REMAUI with transition extension enables the users to define navigation from one screen to another in response to events such as Click, Long Click and with predefined transitions effect.

The users upload screen designs to the web interface. REMAUI identify visual hierarchy and displays on the web interface. The system allows the users to interact and create navigations and effects. Currently, we support the following events such as *Click, Long Click and Item Click* and transition effects such as *Slide*, *Fade, Explode and Shared Element.* Our tool does not support the following events related to Edit Text such KeyPress, Focus. The existing version of REMAUI does not identify Edit Text widgets, because of that our tools is not capable of handling events related to Edit Text view object. We allow the user to upload multiple screen designs and create links for navigating from one screen to another. The user has to click on the “Generate code” button to generate Android code for UI, and transitions.

The code generator module is robust enough to handle,

1. Multiple events listeners set for the same view element. In other words, our tool allows user to create the two different links navigating to different screens for two different events but the same view element. For example, our tool allow a user to create a navigation defined as Clicking on an Image navigates to screen 1 and Long clicking on the same image navigates to screen 2.
2. Same event listeners set for different view elements. In other words, our tool allows a user to create a different links navigating to different screen for the same event but different view elements. For example, our tools allow user to create multiple navigations defined the same event such as Clicking but for different elements, says Clicking on Image view 1 navigates to Screen1, clicking on Image View 2 navigates to Screen2, etc.

Let us consider the following example that illustrates the working of REMAUI transition extension with the input images shown in Fig 4-10 and 4-11. REMAUI identifies the view hierarchy and the supported web application displays the view hierarchy as rectangular boxes. Figure 4 – 12 illustrates the working REMAUI with Transition extension on Google Play Store app (version 7.6.08)

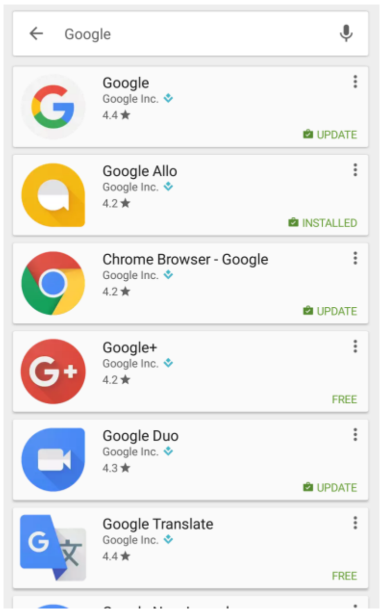


Figure 4-10 - Search screen design of Google Play! Store

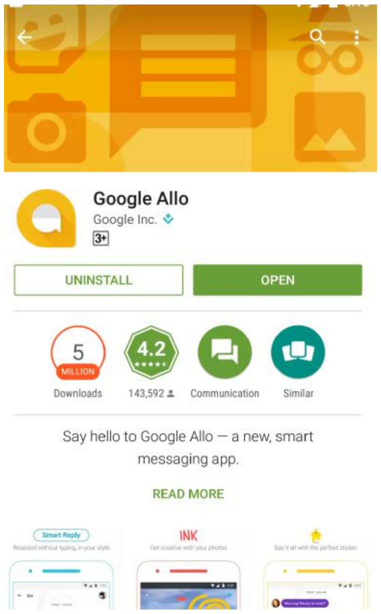


Figure 4-11 - App detail screen design of Google Play! Store

Our tool allows a user to create links for navigation along with transition effects. In this example, tapping on *Google Allo* icon, the app navigates to the second screen

Our tool allows the user to interact with the rectangular boxes, click, and drag to the destination screen design. Figure 4-13 shows the working of REMAUI transition extension and line representing navigational links.

.

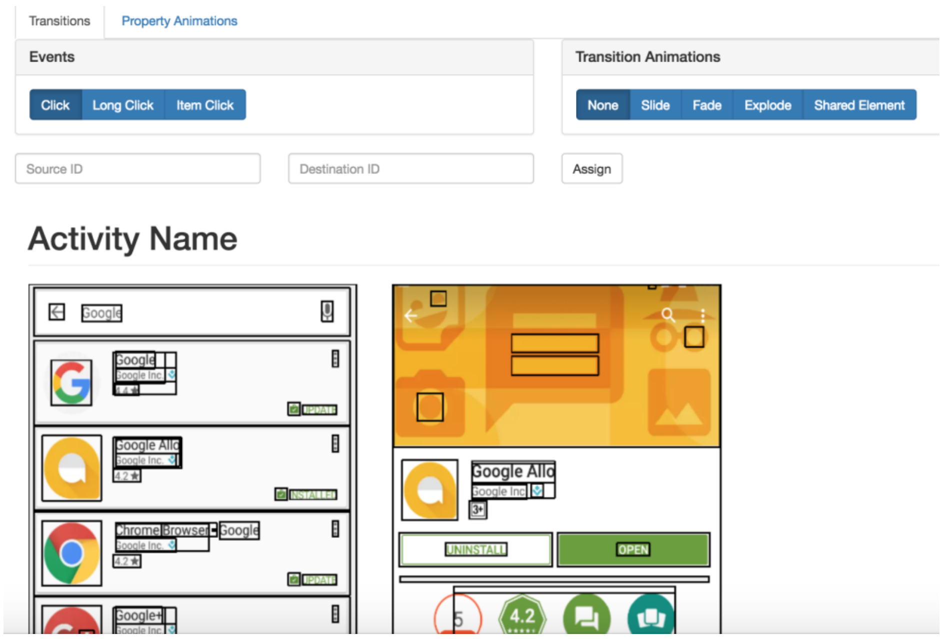


Figure 4-12 - Working of REMAUI Transition extension on Google Play! Store

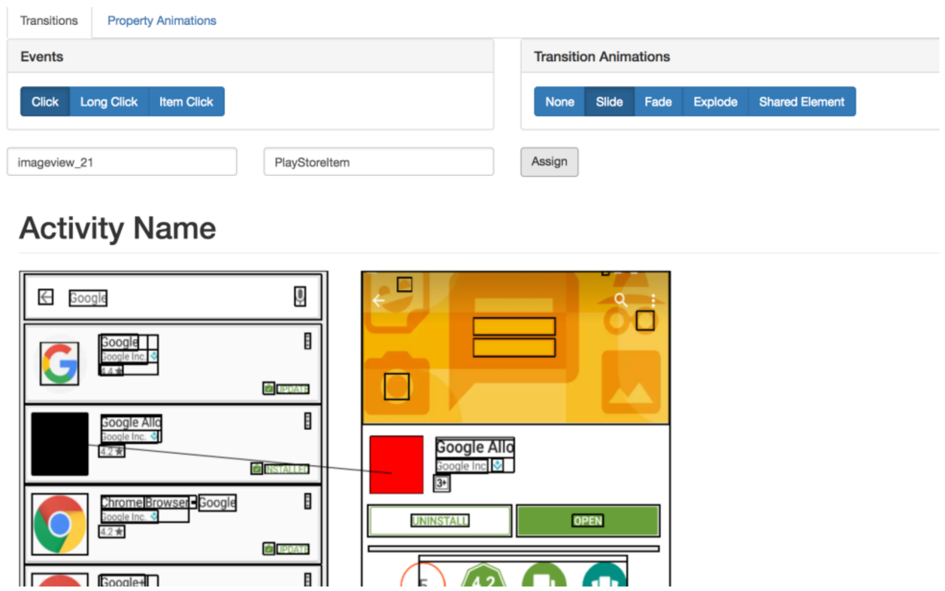


Figure 4-13 - User interactions are highlighted and line represents a transition

When the user clicks on the *“Assign Event”*, REMAUI handles the request that is sent to the server. It contains the information about the click element, destination activity, type of event and transition effect. The web application handles the requests and stores the information about the navigation in a predefined XML format.

When the user clicks on the *“Generate Code”*, REMAUI handles the request and parses the corresponding XML that contained information about the navigations. The REMAUI generates the android UI code, navigations and animation effects. The REMAUI uses APIs for generating animations instead of declarative XML styles.

Apart from generating compilation ready java source code, REMAUI packages the generated source code and other assets such as layout.xml, strings.xml, images into an Android executable (.apk) file.

Using REMAUI with Transition extension generates artifacts that are wholly consumable in the app development process.

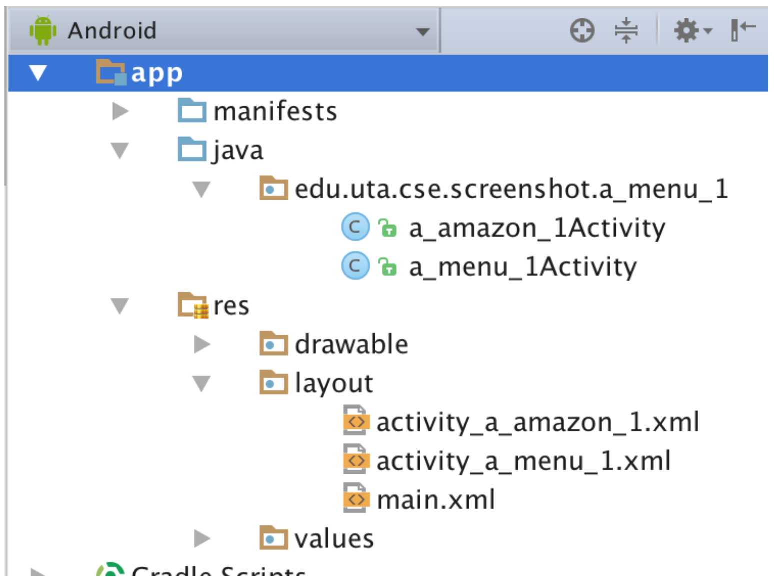
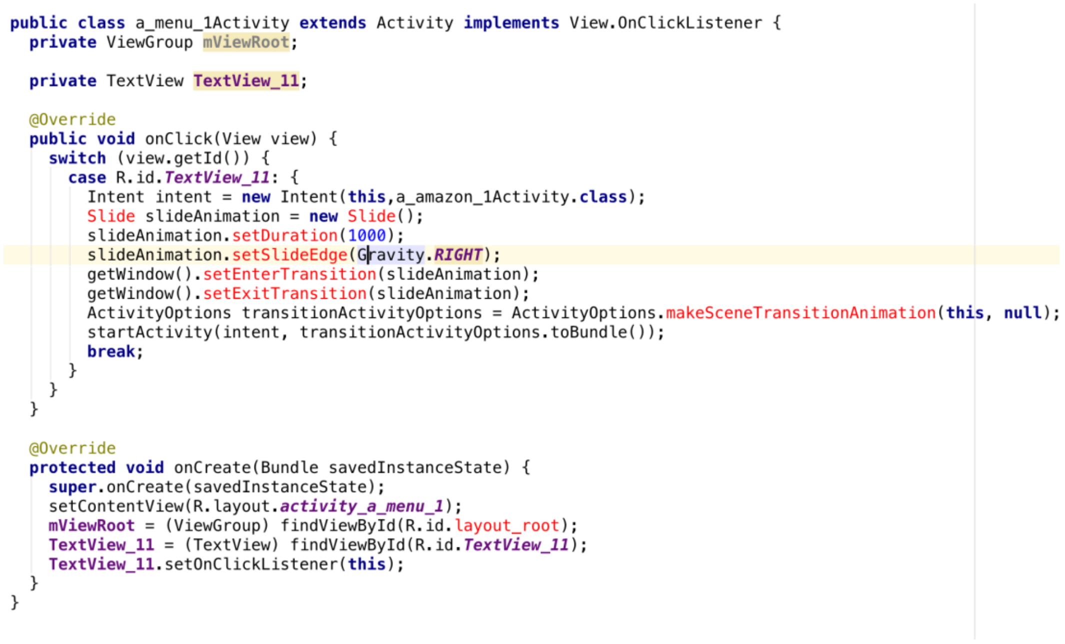


Figure 4-14 - File structure of generated application with inter-screen transitions

The generated application contains one layout file for each input screen design and one Activity class for each input screen design. Figure 4-14 displays the file structure with two Activity classes and two layout files.



```java
public class a_menu_1Activity extends Activity implements View.OnClickListener {
  private ViewGroup mViewRoot;

  private TextView TextView_11;

  @Override
  public void onClick(View view) {
    switch (view.getId()) {
      case R.id.TextView_11: {
        Intent intent = new Intent(this,a_amazon_1Activity.class);
        Slide slideAnimation = new Slide();
        slideAnimation.setDuration(1000);
        slideAnimation.setSlideEdge(Gravity.RIGHT);
        getWindow().setEnterTransition(slideAnimation);
        getWindow().setExitTransition(slideAnimation);
        ActivityOptions transitionActivityOptions = ActivityOptions.makeSceneTransitionAnimation(this, null);
        startActivity(intent, transitionActivityOptions.toBundle());
        break;
      }
    }
  }

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_a_menu_1);
    mViewRoot = (ViewGroup) findViewById(R.id.layout_root);
    TextView_11 = (TextView) findViewById(R.id.TextView_11);
    TextView_11.setOnClickListener(this);
  }
}
```

Figure 4-15 - Generated Activity class with navigation

The Activity class implements the *OnClickListener* and overrides the *onClick* method. *Intent* is the message passing mechanism in Android. Transition effect (Slide) is instantiated and *startActivity* method instructs the Android system to navigate to a new new screen.

I

Research Questions (RQ), Expectations (E), and Hypotheses (H)

To evaluate this extension of REMAUI, we ask (a) runtime for generating code for Transitions (b) runtime for recognizing type of animation (c) Qualitative analysis on generated UI vs. the inputs.

- RQ1: What is the runtime for generating code for Transitions?
  - E1: We expect REMAUI with Transition extension helps users to create working native prototype faster than conventional ways.
  - H1: REMAUI with Transition extension could generate code for navigations in less than 30s

- RQ2: What is the runtime for predicting and generating animations?
  - E1: Given the variety of animations, we do not expect REMAUI animation extension to predict all animations.
  - H1: REMAUI predicts the animations (translation, scaling and fade in / out) from the conceptual UI drawings.

- RQ3: Is a REMAUI assisted with animation or transitions recognition generates UI visually similar to one created by the user?
  - E1: Given the ability to generated complicated animations (sprite animations and 3D), we do not expect to do well for those applications
  - H1: REMAUI assisted animation generator produces visually similar animations.

I

Evaluation

Prototype Information

We developed a web application that acts as the wrapper for the core REMAUI actions. We provide the input screen designs to the web application and then the web application invokes the REMAUI core modules for identifying the view hierarchy, images, and texts and generates source code for animations and transitions. The workflow of the experiments are as follows,

- Transition Extension

  The web application works on the input screen designs provided by the user. The web application invokes the REMAUI processing of the input images. The web application allows the user to create navigation links and effects.

- Animation Extension

  We capture the initial and final screen designs of the animation and we provide the input bitmaps to the web application. The web application infers the possible in-screen animation and generates the Android source code. The source code is compiled, and the web application generates the executable. The generated prototype is installed on the device. We play the animation on the device and capture the screenshot of the final state of the auto-generated animation.

For evaluation, we performed the experiments on a 2.7Ghz Intel Core i5 Mac Book Pro with 16GB of RAM.

Experiments

a. Subjects

We explored the Third party Android applications published on Google play store for the dataset. Since RQ1 is for evaluating the Transition extension, we needed an application with more number of (different) screen designs that we can navigate within the application. This is contrary to the dataset needed for the evaluating Animation extension (RQ2, RQ3). For evaluating Animation extension, we need a couple of screen designs representing the initial and final state of the transformation.

We selected Top 100 Free Apps in Apr 2017; we preselected non-games applications. Not all application contains transition effects; we selected application with more than 5 transition effects discarding the rest and it left us with 10 applications comprising of 107 screenshots. We evaluated the RQ1 using this dataset.

We selected Top 100 Free Apps in Apr 2017 and went through all screen in a depth first search manner in search of an in-screen animation. Only thirty applications contained an in-screen animation. In case of applications having multiple in-screen animations, we arbitrarily chose one. Thus forming a dataset with 30 in-screen animations, one per application. This group of data is used for evaluating RQ2 and RQ3.

b. RQ 1 – Runtime for inter-screen transitions extension

Table 6-1 shows the runtime of REMAUI Transition extension. We provided 10 Screen designs of Amazon Underground App for Android as input, and the three major steps are,

1. REMAUI processing
2. Creating links or navigations with effects
3. Generating Android code for UI and transitions.

Step 1 has a longer runtime than the other steps (2) and (3). The time taken for Step 2 depends on the input image and the user's familiarity with the tool. Step 3 takes the least runtime, and this depends on the number links or navigation defined.

Step	Time taken in seconds
REMAUI Processing	102.3
Creating Links / Navigation	135
Code generation	1.7

Table 6-1 - Runtime of Transition extension on Amazon Underground app for Android

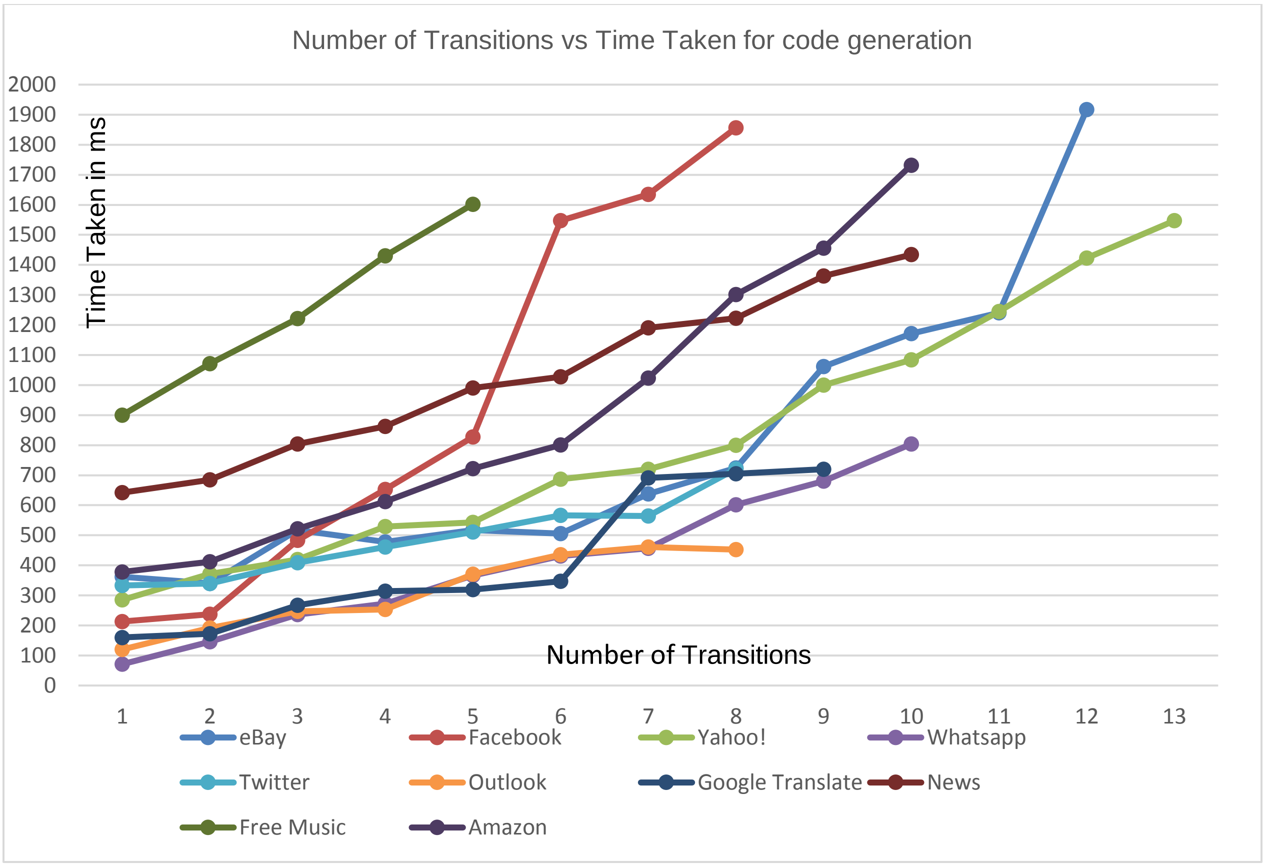


Figure 6-1 -Time taken for generating code vs Number of Transitions

Figure 6-1 shows the runtime of all 10 applications considered for evaluation. Number of transitions is plotted against the time taken for generating the code. We could notice that applications are requiring different time to generate code for the same number of transitions. The prototype reads the generated layout XML into memory as part of the code generation module. Even though we do not modify the generated layout XML for inter-screen transition, the code generation method is common for generating inter-screen transition and in-screen animation and it is having an impact on the time taken for inter-screen transitions.

For example, let us consider the runtime of using Music Free for Android (version 1.69) and Whatsapp for Android (version 2.16.396) inputs.

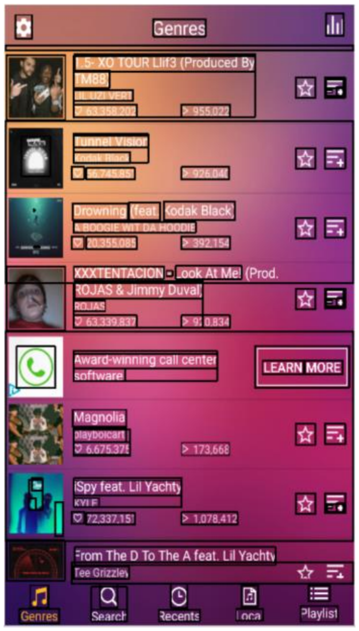


Figure 6-2 - Identified View hierarchy is drawn over Music Free app screen design

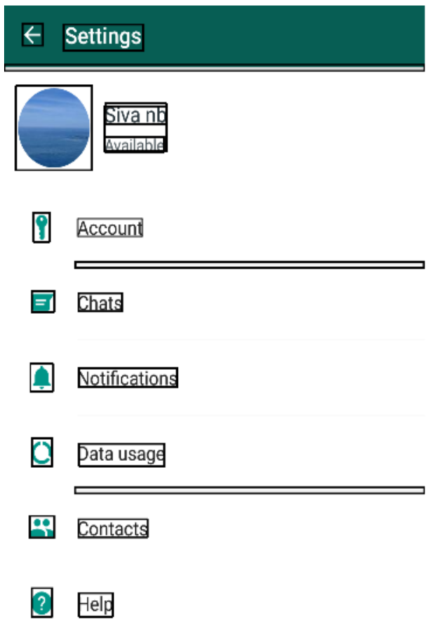


Figure 6-3 - Identified View hierarchy is drawn over the Whatsapp app screen design

We could see that input of Music Free (Figure 6-2) application has more number of view elements than the input of WhatsApp (Figure 6-3) application and hence the generated XML from Music Free input would contain more number of XML nodes than that of WhatsApp. Consequently, the time taken for loading Music Free app's XML takes more time than WhatsApp's XML and causing a variation in the total runtime.

Another variation we could identify is sudden spike in runtime. For example, in the case of eBay for Android (version 4.10.5), the runtime increases drastically in runtime from transition 11 to 12.

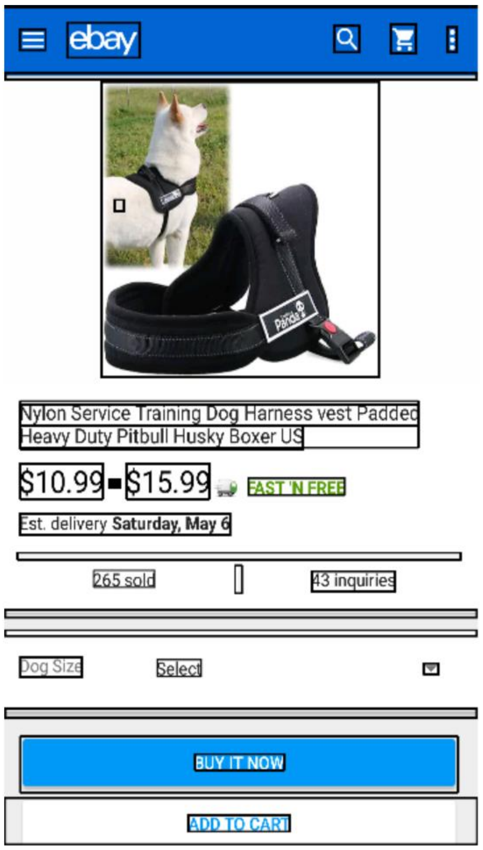

ebay
Nylon Service Training Dog Harness vest Padded Heavy Duty Pitbull Husky Boxer US
$10.99 - $15.99
FAST 'N FREE
Est. delivery Saturday, May 6
265 sold
43 inquiries
Dog Size
Select
BUY IT NOW
ADD TO CART

Figure 6-4 - Identified View hierarchy is drawn over the eBay app screen design

Figure 6-4 shows the input used. We could see that there are many view elements and hence the XML generated is larger. Similar to the previous scenario, the amount of time required for loading them in memory is greater, thereby increasing the total runtime sharply.

c. RQ 2 – Runtime for Inferring in-screen animation

The major steps in inferring in-screen animations are,

1. Identifying view elements
2. Identifying matching view elements
3. Pruning Text view hierarchy

4. Export layouts and code

On an average, the time taken for inferring in-screen animation using our prototype is 26s and the maximum is 69s..

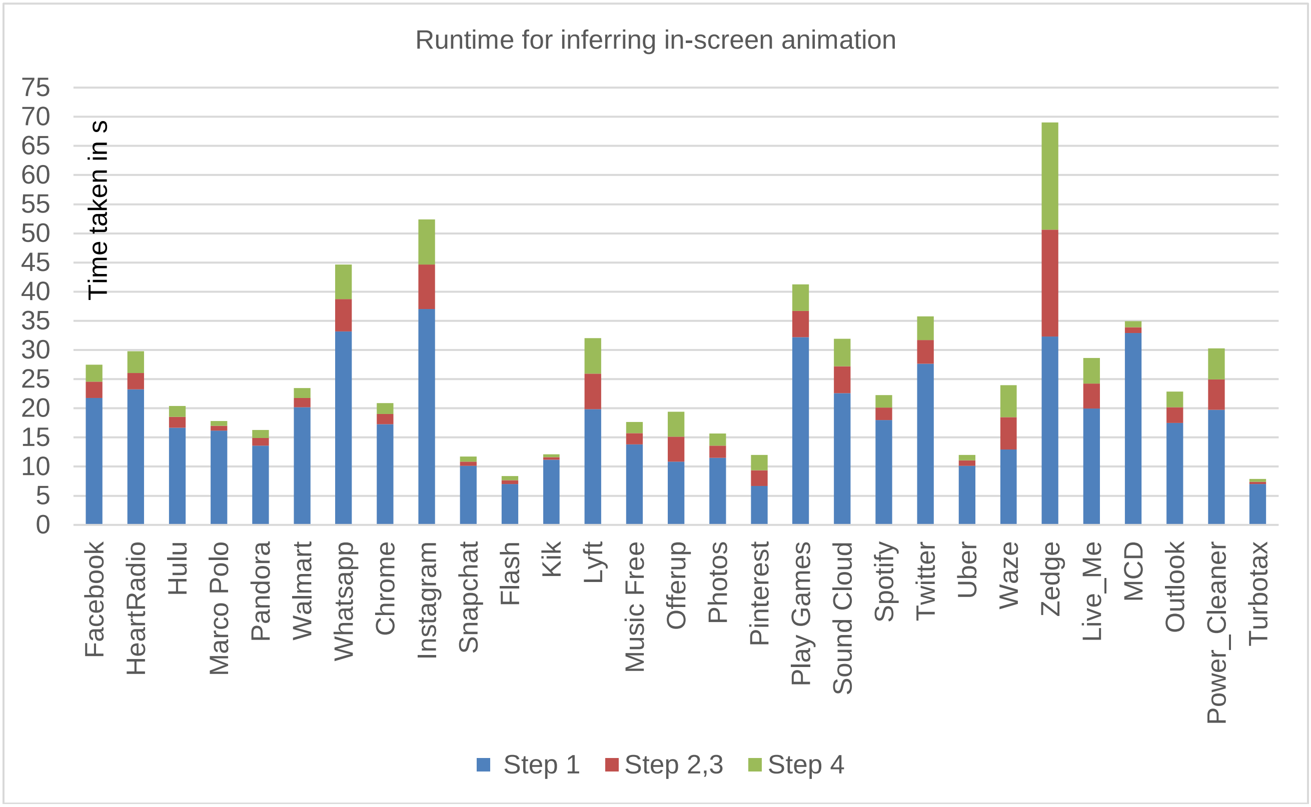


Figure - 6-5 - Runtime for inferring in-screen animation

Figure 6-5 shows the time taken for each application for inferring animation and generating android code. It is very clear that step 1 consumes the more time than other steps. The time taken for generating code (Step 4) is very similar to that of Step 2, 3. Step 1 is applying REMAUI on both the inputs. Step 2, 3 is identifying matching view images views and text views and inferring animations. The time taken for generating code is similar to Step 2, 3 because for each inferred animation the number lines of code (generated) increases.

The highest runtime is for inferring in-screen animation in Zedge app for Android (version 5.16.5). Total time required for inferring in-screen animation is 69s. Step 1 takes 32s, Step 2, 3 takes 18s and Step 4 takes 18s. Step 1 takes most of the time required for inferring in-screen animation. Step 2, 3 determines the amount of time needed for Step 4 and hence they have similar runtimes.

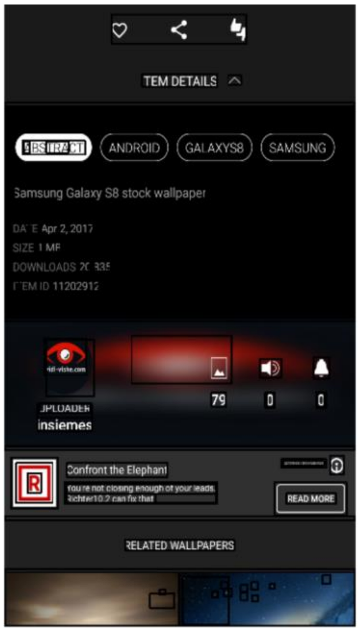


Figure 6-6 - Identified View hierarchy is drawn over the Zedge app intial state screen design

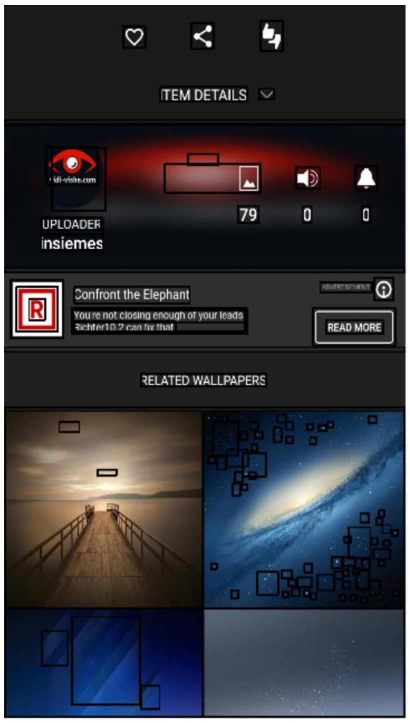


Figure 6-7 - Identified View hierarchy is drawn over the Zedge app final state screen design

Figure 6-6 and 6-7 shows the identified view hierarchies over the input. Since, there are many view elements, the Step 1 (Identifying view elements) took much longer time. Step 2, 3 and Step 4 work on the identified view elements and thus they have higher runtime too.

d. RQ 3 – SSIM Analysis:

Structure Similarity (SSIM) Index Method[19] is a method of measuring image quality. SSIM values ranges from [0 to 1]. We calculate SSIM using the formula,

$$\text{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

- μ_x the average of x;
- μ_y the average of y;
- σ_x^2 the variance of x;
- σ_y^2 the variance of y;
- σ_{xy} the covariance of x and y;

Variance is the expectation of squared deviation of a variable from its mean. We calculated variance using the formula,

$$\operatorname{Var}(X) = \operatorname{E}\left[(X - \mu)^2\right].$$

$\mu = \operatorname{E}[X]$, where E[X] is the expected value of X

We calculated covariance using the formula,

$\operatorname{cov}(X, Y) = \operatorname{E}\left[(X - \operatorname{E}[X])(Y - \operatorname{E}[Y])\right].$, where E[X] and E[Y] are the expected values of X and Y.

The value of SSIM index increases with increase in similarity. Identical images would have SSIM index as 1.0. For this experiment, we used the SSIM part of Scikit-Image package.

We compared the final state of the original screenshot against the screenshot of the final state reached through the generated code for all 30 applications. Figure 6-8 shows the calculated SSIM index against the application.

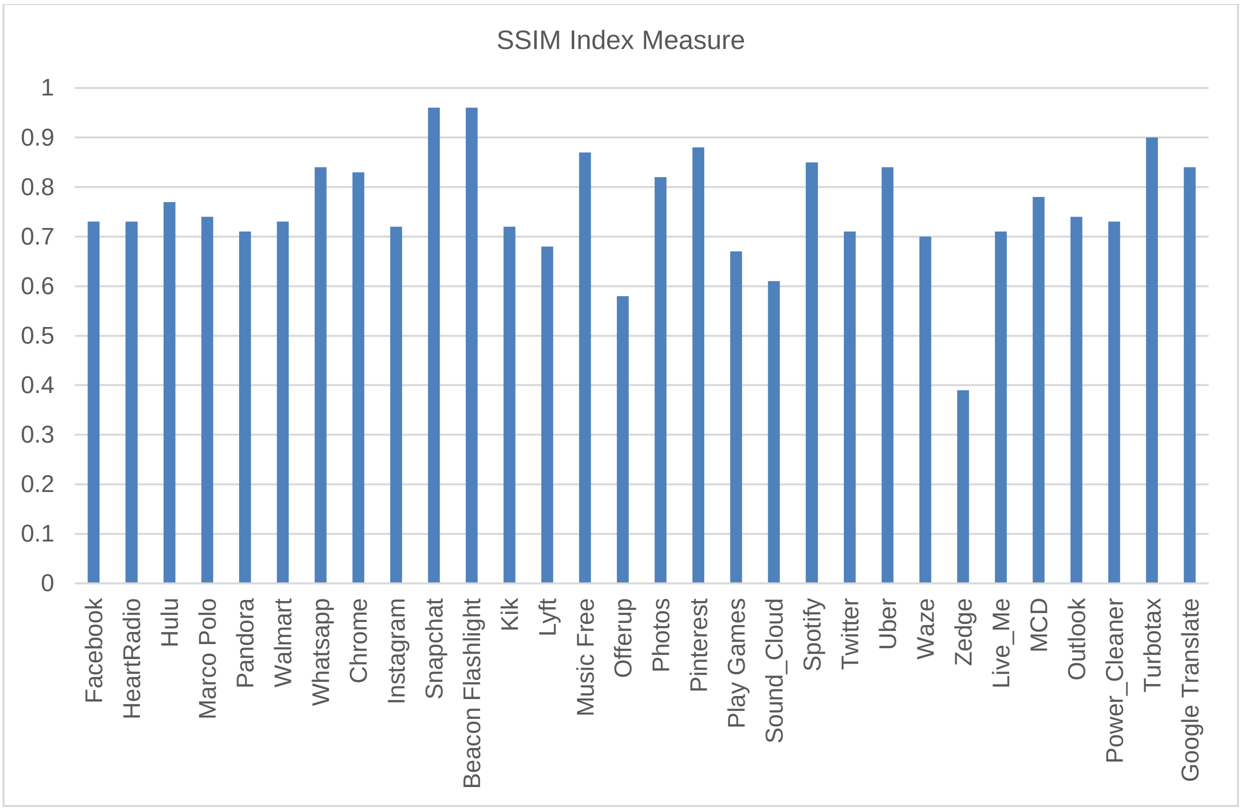


Figure6-8 - SSIM calculated for the test application's input screenshot vs. generated screenshot

In our experiment, the worst-case result is for the Zedge for Android (version 5.16.5). Figure 6-6 and 6-7 displays the identified view hierarchies of the initial and final states.

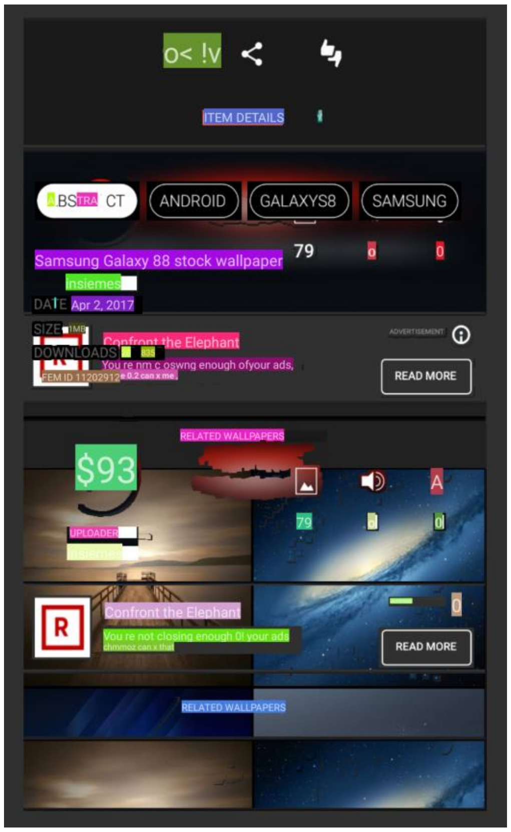


Figure 6-9 - Screenshot of generated final state of Zedge app

Figure 6-9 displays the screenshot of the final state achieved through the generated animation code. Since, the application is dark themed, the identification of view elements were not so accurate there by resulting in poor results.

The best-case scenario is for Beacon Flashlight application for Android (version 1.34) with an similarity of 0.96

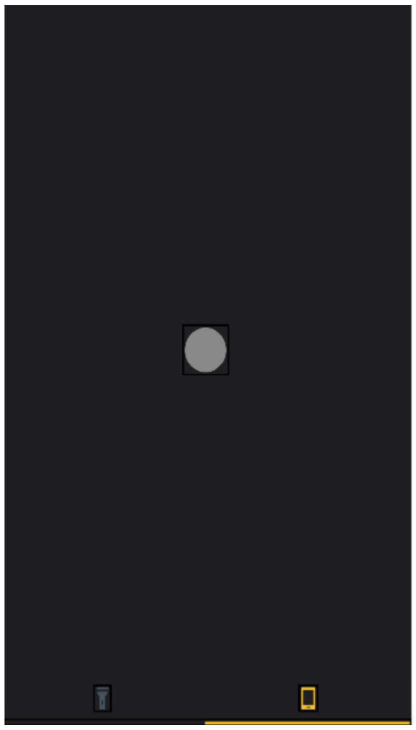


Figure 6-10 - Identified View hierarchy is drawn over the Beacon Flashlight app initial state screen design

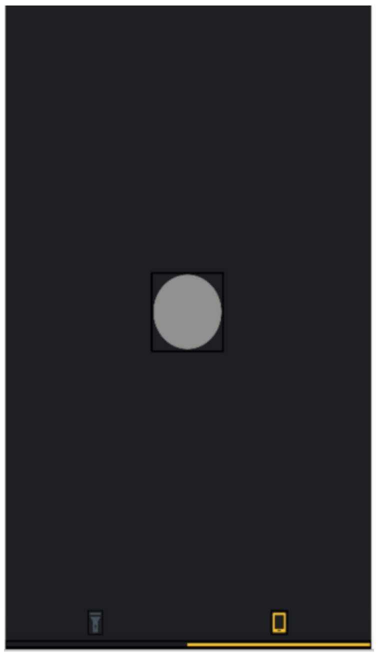


Figure 6-11 - Identified View hierarchy is drawn over the Beacon Flashlight app final state screen design

Figure 6-10 and 6-11 shows the input initial state with the identified view hierarchy. The identification of view elements is good and the view hierarchy is simple. This eliminates the possibility of false inference.

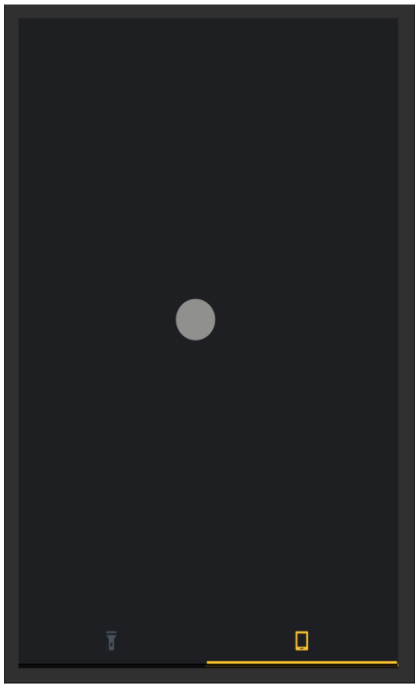


Figure 6-12 - Screenshot of the generated final state of Beacon Flashlight app for Android

Figure 6-12 displays the screenshot of the final state reached through the generated animation code, which is very similar to the input final state.

I

Related Work

The mobile application industry has widely accepted the rapid prototyping methodology. Prototypes are the systematic way of evaluating the concept of the application [16] [20]. Prototypes are of two types namely low-fidelity prototypes and high-fidelity prototypes based on the amount of information present on the prototype.

In software development process the challenges faced by the designer and developers is not a recent one. Always, there have been a constant need and areas of research and opportunities. With the advent of mobile technology, there is a constant need for designing intuitive UI design and better user experience. Even with the adaptation of modern tools, the communication gap exists between the designers and developer communities [2].

GUI builders like AppInventor[21] assists developers in the front-end app development. Developers are allowed to select various UI widget and drag-and-drop them in designing the UI. From a designer's viewpoint, designers are always looking for custom/complex designs[6]. Adding transitions and animations to the prototype needs basic programming skills. In general, designers are non-programmers and do not prefer GUI builders provided by Android Studio or XCode and even third-party GUI builders.

UI reverse engineering technique used in the tool, *Androider* [22] solves a problem of porting Graphical User Interface (GUI) from one platform to another. Androider helps in porting UI from Java Swing to Android SDK or even from Android SDK to Objective C.

For instance when porting Java Swing to Android; Android ports Swing code to declarative XML representation. Androider does not reverse engineer the input bitmaps to GUI code but extracts information from the applications in-memory representation using Java Reflection. Unlike Androider, REMAUI with Animation extension infers in-screen animation from static screen designs.

Working of MobiDev[23] is very similar to that of REMAUI both works on static screen designs. MobiDev identifies view elements by recognizing matching view elements against a model. The UI elements such as Text Field, Check Box, Label etc. needs be sketched in particular style. MobiDev recognizes the specific shape and generates the UI code targeting a platform such as Android. The existing version of REMAUI works on any input bitmap. It recognizes elements based on Computer vision and OCR techniques and there is no need to represent UI elements in any specific shape or style. Our prototype infers animations based on the position and size of the view elements REMAUI with Animation and Transition extensions do not need any modification made to their visual screen designs.

Commercial tools like Zeplin increases the reusability of artifacts from design phase to development phase. Zeplin allows importing designs / sketches from Photoshop or Sketch and extracts the text styles designed by the designers and generates the necessary snippet for *layout.xml*, *styles.xml and colors.xml*. REMAUI extracts the layout information and styles information from plain bitmap image and generates Android code for binding the UI with data. Our prototype generates necessary JAVA and XML code transitions and animations, which Zeplin do not support.

I

Conclusion and Future work

Developing mobile application UI is an expensive process, and interactive prototypes are bridging the communication gap between designers and developers. Still, these prototypes are reusable in the development process and most cases the creating and maintaining the prototypes are demanding a substantial amount of time. REMAUI with transition and animation extension unifies transition prototyping tools and animation prototyping tools, and these prototypes are wholly consumable in the development process. In addition, Animation extension is capable of inferring in-screen animations with generated UIs similar to the originals in terms of pixel-by-pixel comparison.

We plan to (1) generalize the export step to iOS platform. (2) Currently, REMAUI extracts icons and logo from the original image, we plan to point REMAUI to the assets folder, and REMAUI automatically identifies the correct image or logo and packages the right resolution image to the specified folders.

I

Appendix I

Test Applications for inferring in-screen animations

Following are the list of applications used for experiments,

S.No	Name	Package Name	Version	In-Screen Animation
1.	Facebook	com.facebook.katana	105.0.0.23.137	Login screen
2.	Heart Radio	com.thisisglobal.player.heart	4.10.5	Login screen
3.	Pandora	com.pandora.android	7.9	Login screen
4.	Whatsapp	com.whatsapp	2.16.396	Message Info screen
5.	Chrome	com.android.chrome	53.0.2785.135	New Tab screen
6.	Hulu	com.hulu.plus	2.27.5	Login screen
7.	Instagram	com.instagram.android	10.15.0	New message screen
8.	Marco Polo	co.happybits.marcopolo	0.99.0	Login screen
9.	Snapchat	com.snapchat.android	10.5.6.0	Splash screen
10.	Walmart	com.walmart.android	4.3.1	Login screen
11.	Beacon Flashlight	com.jiubang.fastestflashlight	1.34	Flashlight screen

12.	Kik	Kik.android	11.15.0.15115	Splash screen
13.	Lyft	Me.lyft.android	4.26.3	Invite friends screen
14.	Music Free	com.zentertain.freemusic	1.69	Search screen
15.	OfferUp	com.offerup	2.5.3	Item detail screen
16.	Google Photos	com.google.android.apps.photos	2.12	Search screen
17.	Pinterest	com.pinterest	6.13.0	Pin details screen
18.	Google Play Games	com.google.android.play.games	3.9.08	App details screen
19.	SoundCloud	com.soundcloud.android	2017.04.07	Home screen
20.	Spotify	com.spotify.music	8.3.0.681	Search screen
21.	Twitter	com.twitter.android	6.27.1	Add people screen
22.	Uber	com.ubercab	3.131.4	Promotions screen
23.	Waze	com.waze	4.22.1	Menu screen
24.	Zedge	net.zedge.android	5.16.5	Home screen
25.	Live.Me	com.cmcm.live	3.5.60	Login screen
26.	McDonald’s	com.mcdonalds.app	5.3.0	Login screen

27.	Outlook	com.microsoft.office.outlook	2.1.138	Settings screen
28.	Power Clean	com.lionmobi.powerclean	1.2.12	Adding Ignore list screen
29.	TurboTax	com.intuit.turbotax.mobile	3.4.0	Splash screen
30.	Google Translate	com.google.android.apps.translate	5.8.0	Home screen

Appendix II

Test Application for inter-screen transitions

S.No	Name	Package Name	Version
1.	Facebook	com.facebook.katana	105.0.0.23.137
2.	eBay	com.ebay.mobile	4.10.5
3.	Music Free	com.zentertain.freemusic	1.69
4.	Twitter	com.twitter.android	6.27.1
5.	Outlook	com.microsoft.office.outlook	2.1.138
6.	Google Translate	com.google.android.apps.translate	5.8.0
7.	Whatsapp	com.whatsapp	2.16.396
8.	Amazon Underground	com.amazon.mShop.android.shopping	10.8.0.200
9.	Yahoo! Mail	com.yahoo.mobile.client.android.mail	5.14.5
10.	News Break	com.particlenews.newsbreak	3.1.2

References

[1] – M. Hapern, Y. Zhu, V.Reddi “Mobile CPU’s Rise to Power: Quantifying the Impact of Generational Mobile CPU Design Trends on Performance, Energy, and User Satisfaction”, *High Performance Computer Architecture (HPCA) 2016*. [online]. Available: http://matthewhalpern.com/publications/mobile-cpus-hpca-2016.pdf

[2] – Kony, Inc., “Bridging the Gap: Mobile App Design & Development”, http://forms.kony.com/rs/konysolutions/images/Bridging_Gap_Brochure_dec_10_14.pdf, Dec 2014, accessed Dec 2016.

[3] – Invision, Inc., https://www.invisionapp.com/, accessed Feb 2017

[4] - Flinto, Inc., https://www.flinto.com/lite, accessed Feb 2017

[5] - Marvel App Inc., https://marvelapp.com/, accessed Feb 2017

[6] – T. A. Nguyen, C. Csallner “Reverse Engineering Mobile Application User Interfaces with REMAUI”, *30th IEEE/ACM Internation Conference on Automated Software Engineering (ASE)*, pp. 248 – 259, 2015.

[7] – Google, Inc., “Material Design for Android”, https://developer.android.com/design/material/index.html, accessed Dec 2016.

[8] - Google, Inc., “Package android.transition”, https://developer.android.com/reference/android/transition/package-summary.html, accessed Dec 2016.

[9] - Google, Inc., “Package AnimationSet”, https://developer.android.com/reference/android/view/animation/AnimationSet.html, accessed Dec 2016.

[10] – C. Zauner, “Implementation and Benchmarking of Perceptual Image Hash Function”, Master’s thesis, Upper Austria University of Applied Sciences, 2010.

[Online]. Available: http://phash.org/docs/pubs/thesis_zauner.pdf

[11] – Google Inc., “Top Free in Android Apps, ”https://play.google.com/store/apps/collection/topselling_free?hl=en, accessed Dec 2016.

[12] – Google, Inc., “Layouts”, https://developer.android.com/guide/topics/ui/declaring-layout.html, accessed Jan 2017.

[13] – OpenCV, “Histogram Comparison”, http://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/histogram_comparison/histogram_comparison.html, accessed Jan 2017.

[14] – Apple, Inc., “iOS Human Interface Guidelines”, https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles/, accessed Apr 2017

[15] – Motif Tools BV, “https://framer.com/”, accessed Apr 2017.

[16] – J. Rudd, K. Stern and S. Isensee “Low vs. High-fidelity prototyping debate”, Interactions pp 76-85, 1996

[17] - J. A. Landay and B. A. Myers, “Interactive sketching for the early stages of user interface design,” in *Proc. ACM SIGCHI Conference on Human Factors in Computing Systems (CHI).* ACM, May 1995, pp. 43–50.

[18] – Clutch Inc, “Cost to Build a Mobile App: A Survey”, https://clutch.co/app-developers/resources/cost-build-mobile-app-survey, accessed Apr 2017

[19] – Z. Wang, A.C. Bovik, ”Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures,” Signal Processing Magazine, IEEE, vol. 26, no. 1, pp. 98-117, Jan. 2009

[20] – M. Aleksy, “An Apporach to Rapid Prototyping of Mobile Applications”, Advanced Information Networking and Applications(AIANA), IEEE, pp. 1072-1077, Mar. 2013

[21] Google Inc., About – App Inventor for Android.
http://appinventor.mit.edu/explore/about-us.html.

[22] – E. Shah, E. Tilevich, "Reverse-engineering user interfaces to facilate porting to cross mobile devices and platforms", SPLASH '11 Workshops, ACM, pp. 255 – 260, Oct 2011.

[23] - J. Seifert, B. Pfleging, E. del Carmen Valderrama Bahamondez, M. Hermes, E. Rukzio, and A. Schmidt, "Mobidev: A tool for creating apps on mobile phones", in Proc. 13th Conference on Human-Computer Interaction with Mobile Devices and Services (Mobile HCI). ACM, Aug. 2011, pp. 109–112.

[24] - Zeplin, Inc., https://www.Zeplin.io/, accessed Feb 2017

Biographical Information

Siva Natarajan Balasubramania was born in Tirunelveli, India in 1990. He received his B.E Computer Science degree from Anna University, Chennai in 2007. He worked for three years in with Cognizant Technology Solutions and HeyMath! He started his Master's degree spring 2015.

He is very interested in mobile application specifically Android which inspired him to pursue his research on mobile software engineering. He wishes to continue pursuing his research aspirations.