Empirical Studies on the Relation between User Interface Design and Perceived Quality of Android Applications

by

Seyyed Ehsan Salamati Taba

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
August 2014

Abstract

The number of mobile applications has increased drastically in the past few years. According to the statistics, it is estimated that 167 billion mobile applications will be downloaded by 2015. However, some applications are superior to the others in terms of user-perceived quality. User-perceived quality can be defined as the user's opinion of a product. For mobile applications, it can be quantified by the number of downloads and ratings. Earlier studies suggested that user interface (UI) design is one of the major reasons that can affect the user-perceived quality of mobile applications. UI design is relatively a complex concept by its nature. In this thesis, we try to examine the affect of UI design on user-perceived quality by focusing on two different aspects of UI, namely UI complexity and UI reuse. We carry out our case studies on 1,292 Android applications from the Android market (*i.e.,* Google Play).

We find that our measurement of UI complexity quantified by the number of inputs and outputs confirms the findings of previous studies on UI complexity. UI complexity can affect the user-perceived quality, and we are able to provide guidelines for the proper amount of UI complexity that helps an application achieve high user-perceived quality. We observe that UI of mobile applications are widely reused among and across different mobile categories. Frequently used UI elements with certain characteristics can provide high user-perceived quality. Finally, we are able to extract practical

UI templates with high user-perceived quality for developers to design UIs with high quality. Developers and quality assurance personnel can use our guidelines to improve the quality of mobile applications.

Co-Authorship

Earlier versions of the work in the thesis were published as listed below:

- *An Exploratory Study on the Relation between User Interface Complexity and the Perceived Quality of Android Applications (Chapter. 4)*

  Seyyed Ehsan Salamati Taba, Iman Keivanloo, Ying Zou, Joanna Ng, and Tinny Ng, in Proceedings of the 14th International Conference on Web Engineering (ICWE 2014) , Late Breaking Results Track, 1-4th of July, 2014, Toulouse, France. Springer. Acceptance Rate = 33% [1].

  My contributions: drafting the research plan, gathering and analyzing the data, writing and presenting the paper.

Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisor Dr. Ying Zou for the continuous support of my M.Sc. study and research, for her patience, motivation, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis.

Besides my advisor, I would like to thank Dr. Iman Keinvanloo who provided insightful guidelines, and assisted patiently throughout my studies.

In my daily work I have been blessed with a friendly and cheerful group of fellow students in the Software Reengineering Research Lab. Shaohua Wang who provided advices for the problems I faced. Moreover, I should mention the help and support of Dr. Bipin Upadhyaya and Feng Zhang.

I appreciate the hard work from my committee members: Dr. Zhen Ming (Jack) Jiang, Dr. Ahmed E. Hassan, Dr. Hossam S. Hassanein, and Dr. Ying Zou.

I also thank Dr. Mei Nagappan who kindly shared their dataset of Android mobile applications used for the research presented in this thesis.

I thank to my wife Hana Fahim Hashemi without whom this effort would have been worth nothing. Your love, support and constant patience have taught me so much about sacrifice, discipline and compromise.

Last but not the least, I would like to thank my family: my parents who have

always supported, encouraged and believed in me through all the stages of my life.

Contents

Abstract **i**

Co-Authorship **iii**

Acknowledgments **iv**

Contents **vi**

List of Tables **ix**

List of Figures **xi**

List of Abbreviations **1**

Chapter 1: Introduction **2**
1.1 Background 2
1.1.1 Mobile Application Stores 3
1.1.2 Characteristics of Mobile Applications 4
1.1.3 Structure of Android Applications 5
1.2 Research Statement 8
1.3 Thesis Objectives 8
1.4 Organization of Thesis 9

Chapter 2: Related Work **11**
2.1 User Studies on the UI of Mobile Applications 11
2.2 Mining Mobile Applications 12
2.3 Studying the UI of Mobile Applications 13
2.4 Summary 15

Chapter 3: Processing Mobile Applications Extracted from Android Market **16**
3.1 Approach Overview 16

3.1.1 Calculating User-perceived Quality . . . 17
3.1.2 Extracting the Structure . . . 20
3.1.3 Identifying Functionalities . . . 20
3.2 Subject Systems . . . 21
3.3 Summary . . . 21

Chapter 4: An Exploratory Study on the Relation between User Interface Complexity and the Perceived Quality of Android Applications 23
4.1 Motivation . . . 24
4.2 Research Questions . . . 25
4.3 Study Design . . . 26
4.3.1 Data Processing . . . 27
4.3.2 Analysis Methods . . . 29
4.4 Study Results . . . 30
4.4.1 **RQ4.1: Can our measurement approach quantify UI complexity?** . . . 31
4.4.2 **RQ4.2: Does UI complexity have an impact on the user-perceived quality within each category of mobile applications?** . . . 34
4.4.3 **Findings.** . . . 36
4.4.4 **RQ4.3: Does UI complexity have an impact on the user-perceived quality of the functionalities in mobile applications?** . . . 38
4.5 Summary . . . 43

Chapter 5: An Exploratory Study on Reused User Interface Elements of Andriod Applications 45
5.1 Motivation . . . 46
5.2 Research Questions . . . 47
5.3 Study Design . . . 48
5.3.1 Data Processing . . . 49
5.3.2 Analysis Methods . . . 54
5.4 Study Results . . . 54
5.4.1 **RQ5.1: To what extent UI elements set are reused?** . . . 55
5.4.2 **RQ5.2: Does reused UI elements set impact user-perceived quality within and across categories?** . . . 59
5.4.3 **RQ5.3: Does reused UI elements set have an impact on the user-perceived quality of the functionalities in mobile applications?** . . . 65

5.4.4 **RQ5.4: Can we extract UI templates from reused UI elements sets of high user-perceived quality?** 71
5.5 Summary 76

Chapter 6: Conclusion **77**
6.1 Contributions 77
6.2 Threats to Validity 78
6.2.1 Construct Validity 78
6.2.2 Internal Validity 79
6.2.3 External Validity 80
6.3 Future Work 81

Bibliography **82**

Appendix A: Sample Screen-shots for Retrieved UI Templates **90**

List of Tables

3.1 Summary of the characteristics of different categories 22

4.1 Proposed Application and Activity Level Metrics 29
4.2 Input and Output Tags 29
4.3 Summary of the number of inputs, outputs, and UI elements of different categories 30
4.4 Kruskal-Wallis test results for application level UI metrics in different categories. 33
4.5 Wilcoxon rank sum test results for the usage of application level UI metrics in categories with high and low user-perceived quality. 33
4.6 Difference of usage of used UI metrics between the applications with low and high user-perceived quality in each category. 36
4.7 Difference of usage of activity level UI metrics between the activities with low and high user-perceived quality for each functionality in each category. 42

5.1 Summary of the characteristics of different categories 51
5.2 Jaccard similarity measure for RES reused across different categories. 59
5.3 Difference of the user-perceived quality for the RES that are reused within each two pair of categories. 62

5.4 Difference in the usage of different characteristics of RES within various categories. 64
5.5 Difference of usage of activity level UI metrics between the activities with low and high user-perceived quality for each functionality in each category. 69
5.6 List of related words for Login/Sign in functionality in the Shopping category 73
5.7 UI templates for different sub-functionalities 74

List of Figures

3.1 Overview of our data collection process. 17

3.2 Information that are available for a mobile application in Android market. 18

3.3 Distribution of user-perceived quality among different categories. . . . 19

3.4 Part of a Sample XML File 21

4.1 Overview of our data collection process. 27

4.2 Part of a Sample Smali File 28

4.3 Sorted categories based on the distribution of their user-perceived quality. 32

5.1 Overview of our data collection process. 48

5.2 A UI XML file with its corresponding UI elements. 49

5.3 Percentage of Total Elements Reused in each Category 57

5.4 Percentage of Files Associated with RES in each Category 58

A.1 Screen-shot of the User Agreement page of the Shopzilla application. 91

A.2 Screen-shot of the Privacy Policy page of the OfficeMax application. . 92

A.3 Screen-shot of the Sign in page of the Groupon application. 93

A.4 Screen-shot of the Date Time page of the Walgreen application. . . . 94

A.5 Screen-shot of the Photo Preview page of the Walgreen application. . 95

A.6 Screen-shot of the Address Entry page of the eBay application. 96

A.7 Screen-shot of the Shopping Cart page of the Newegg application. . . 97
A.8 Screen-shot of the Search page of the eBay application. 98
A.9 Screen-shot of the Rating page of the AppRedeem application. 99
A.10 Screen-shot of the Setting page of the eBay application. 100

List of Abbreviations

The following table describes the significance of various abbreviations and acronyms used throughout the thesis. The page on which each one is defined or first used is also given.

Abbreviation	Meaning	Page
PC	Personal Computer	2
UI	User Interface	2
iOS	iPhone Operating System	3
RIM	Research In Motion	4
GPS	Global Positioning System	4
SDK	Software Development Kit	5
APK	Android PaKage	5
XML	Extensible Markup Language	7
UPQ	User-Perceived Quality	13
NI	Number of Inputs	29
NO	Number of Outputs	29
NE	Number of Elements	29
ANI	Average Number of Inputs	29
ANO	Average Number of Outputs	29
ANE	Average Number of Elements	29
NA	Number of Activities	29
LDA	Latent Dirichlet Allocation	30
RES	Reused ui Elements Set	45
RDM	Reused elements set Distribution Metric	51
DRM	Developer-customized Reused element set Metric	52
RLM	Reused elements set Length Metric	52
PER	Percentage of UI Elements Reused	53
FAR	Files Associated with RES	54

Chapter 1

Introduction

1.1 Background

A mobile application is a computer program designed to run on smartphones, tablet computers and other mobile devices. Mobile applications are pervasive in our society and play a vital role in our daily lives. Users can perform similar tasks both on smartphones and PCs [2] such as: checking e-mails or browsing the web. Mobile applications are usually available through mobile application stores (*e.g.,* Apple's application store). Developing applications for mobile devices requires considering the constraints and features of these devices (*i.e.,* small screen size, poor network connection, and computational power) to achieve high user-perceived quality. In the following sub-sections, first, we introduce mobile application stores. Second, we talk about the characteristics of mobile applications, their user interface (UI) design and user-perceived quality. Finally, we briefly describe the architecture of Android mobile applications as we have conducted our studies on these types of applications.

1.1.1 Mobile Application Stores

Mobile applications are usually available through application stores, which began appearing in 2008 and are typically operated by the owner of the mobile operating systems, such as the Apple application store [3], Android market (Google Play) [4], Windows phone store [5], BlackBerry application world [6], Amazon application store [7], Nokia store [8], and Samsung application store [9]. Apple's application store for iOS was one of the first application distribution services, opened on July 10th in 2008. It started with around 800 applications available, and as of July 2014 it has more than 1 million applications, which had been downloaded 50 billion times. According to statistics the revenue of Apple's application store has been over $26.5 billions by the end of 2013 [10]. Some mobile applications are free, while others must be bought. For applications with a price, generally a percentage, 20-30%, goes to the distribution provider (such as Apple), and the rest goes to the producer of the mobile application [11]. It is also important to mention that free mobile applications account for nearly 90 percent of total mobile application store downloads by the end of 2013 [12].

Android market (Google Play) was opened in October 25th in 2008. It started with less than 100 applications which looked weak compared to Apple's application store at the time. However, Android has more than 1.3 million applications available on its store (*i.e.,* Google Play). The number of applications downloaded from this platform is around 60 billion applications by the end of 2013. The revenue of Android market is reported $10 billions in 2014 which is less than Apple store with $26.5 billions [13].

Besides these two leading mobile platforms (*i.e.,* iOS and Android). The following statistics demonstrate the tremendous success of mobile applications among all the aforementioned stores:

1. **Number of downloads:** The number of downloads for mobile applications in various platforms (*e.g.,* iOS, Android, BlackBerry, and Windows Phone) by the end of 2014 is predicted to be 127 billions, increased from 7 billion in 2009 [14].

2. **Revenue:** The global revenue of mobile application stores of Apple, Google, Nokia and RIM in 2013 has been $26 U.S. billions, and it is estimated to generate revenues of $74 U.S. billions by 2017 [15].

In application stores users can rate and write reviews for mobile applications which reflects the quality of mobile applications perceived by the users. User-perceived quality can be defined as follow:

User-perceived Quality

User-perceived quality can be defined as the user's opinion of a mobile application. User-perceived quality can be influenced by UI design, stability and performance of applications [16]. User-perceived quality of previous users of a mobile application can give users a judgmental power to choose between mobile applications. However, users do not always have complete information about different aspects of a mobile application; indirect measures such as: number of downloads or ratings available by the mobile stores may be their only basis for comparing different mobile applications.

1.1.2 Characteristics of Mobile Applications

Mobile devices differ significantly from desktop computers and laptops. Mobile devices run on battery and have less computational power than personal computers and also have more features such as location detection (*i.e.,* GPS) and camera [17].

Desktop applications tend to be used in long sessions. However, mobile applications tend to be used frequently, but for short session durations.

Quick startup time, responsiveness, and focused purpose are important characteristics of good mobile applications. Applying all these features in a mobile application can provide a highly productive user experience. Moreover, as a part of the development process for mobile applications, User Interface (UI) design is essential in the creation of mobile applications.

User Interface Design

User interface (UI) design is defined as the design of computers, mobile devices, software applications, and websites with the focus on the user's experience and interaction [18]. Unlike traditional design where the goal is to make the object or application physically attractive, the goal of user interface design is to make the user's interaction experience as simple as possible. To this end, different mobile platforms provided basic design and development principles for the developers such as: Android UI design guidelines [19], iOS UI guidelines [20] and Windows Mobile UI guidelines [21]. The general goal of these guidelines is to help developers design mobile applications with good user-perceived quality.

1.1.3 Structure of Android Applications

Android applications are written in Java programming language using Android Software Development Kit (SDK). The Android SDK compiles the code into an Android PaKage (APK) file which is an archive file with a ".apk" extension. One APK file contains all the contents of an Android application, and is the file that Android devices

use to install applications.

Application components are the essential building blocks of an Android application. There are four different types of application components, including activities, services, content providers and broadcast receivers. We describe them as follows:

Activities

An activity represents a single screen with a user interface (UI). For example, a shopping application like eBay might have one activity that shows a list of products, another activity to search for product, and another activity to add a product to the shopping cart. Although the activities work together to form a cohesive user experience in the eBay application, each one is independent of the others.

Services

A service is a component which runs in the background without direct interaction with the user. As the service has no user interface, it is not bound to the lifecycle of an activity. Services are used for repetitive and potentially long running operations, i.e., Internet downloads, checking for new data, data processing, or updating contents.

Content providers

A content provider component supplies data from one application to others on request. For example, the Android system provides a content provider that manages the user's contact information. As such, any application with the proper permissions can query part of the content provider to read and write information about a particular person.

Broadcast receivers

Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. For example, a broadcast announced by the system that the screen has turned off, the battery is low, or a picture was captured can be received by applications to perform appropriate behaviors.

Among the aforementioned application components, users only interact with activities. An Android application consists of several activities. The guidelines [19] for Android developers recommends that an activity is a single, focused task that the user can do. Each activity represents a single-screen user interface (UI). As a result, only one activity can be in the foreground for the users to interact with.

There are two ways to declare a UI layout for an activity: i) Declaring UI layout elements in an XML file (a.k.a., UI XML file), or ii) Instantiating UI layout elements programmatically. Our premise in this work is towards using UI XML file approach since it is the recommended way by Android design guidelines [19]. An XML file defines a human-readable visual structure for a user interface. Applications that Instantiate UI layout elements programmatically are excluded from our study since our analysis and data gathering approach cannot handle them.

Every Android application has an AndroidManifest.xml (manifest) file in its root directory. It contains meta-data information of an application(*e.g.,* the path to the source code of activities, permissions). In addition to the activities and compiled code, manifest file plays an important role as a source of information.

1.2 Research Statement

As mentioned above, the large number of downloads, and revenues for mobile applications imply that a new software industry is emerging. In this competitive market, developers should be more careful in designing applications with respect to mobile characteristics (*e.g.,* small screen size, network problems and computational power [17]) to reach high user-perceived quality [22]. Developers' negligence in the importance of UI design is one of the major reasons for users to abandon a task on mobile applications and switch to PC [22].

We believe that in-depth empirical studies on currently existing mobile applications are needed in order to pave the ground for developers to design user interfaces that provide users better perceived quality. To the best of our knowledge no one has focused on recommending detailed characteristics of UI design among existing mobile applications, and how it can affect the user-perceived quality of mobile applications.

1.3 Thesis Objectives

In this thesis, we investigate the effect of different aspects of UI design (*i.e.,* UI complexity, UI reuse) on the user-perceived quality of mobile applications in two empirical studies. The studies are all conducted on 1,292 Android applications from the Android market (*i.e.,* Google Play) since it is one of the most active application markets today [14]. In this thesis, our objectives are as follows:

1. **To identify the relation between user interface complexity and user-perceived quality (Chapter 4)**.

Earlier studies suggested that user interface (UI) barriers (i.e., input or output challenges) can affect the user-perceived quality of mobile applications [23]. User-perceived quality can be defined as the user's opinion of a product. For mobile applications, it can be quantified by the number of downloads and ratings [24]. In this study, we explore the relation between UI complexity (*i.e.,* quantified by the number of inputs and outputs) and user-perceived quality in Android applications. Furthermore, we strive to provide guidelines for the proper amount of UI complexity that helps an application achieve high user-perceived quality.

2. **To measure the impact of reuse in user interface design on user-perceived quality (Chapter 5)**.

   A recent study by Ruize et al. [25] has shown that code in the Android Market is being reused significantly, which implies that many applications in the Android Market use the same logic. However, reuse in mobile applications is not limited to the source code (i.e., logic). Another important part that can be reused and plays a vital role in constructing the user-perceived quality of smartphone applications is their user interface (UI) design. We study the extent of reuse in the UI of mobile applications. Moreover, we observe whether exploiting reused UI elements can improve user-perceived quality or not.

1.4 Organization of Thesis

The remainder of this thesis is organized as follows:

- chapter 2 presents current research work related to mobile applications.

- Chapter 3 presents our overall approach for processing mobile applications extracted from Android Market.

- Chapter 4 presents an empirical study that examines the relation between UI complexity and user-perceived quality of Android mobile applications.

- Chapter 5 studies the extent of reuse for the UI of mobile applications, and investigates whether using frequently used UI elements can provide high user-perceived quality or not.

- Finally, chapter 6 concludes the thesis with highlighting our results, the limitations, and threats to validity, and potential directions for future work.

Chapter 2

Related Work

In this chapter, we discuss previous user studies on the UI of mobile applications, and studies that conduct experiments on existing mobile applications.

2.1 User Studies on the UI of Mobile Applications

Researchers have tried to get more insight into mobile applications and their characteristics from different perspectives. One of the best ways of getting to know about users and their needs on smartphones is to carry out user studies. User studies yield conclusions and hypothesizes that are of great value for future research studies.

A user study on 14 smartphone users by Kane et al. [23] shows that participants use some applications more on their smartphones than PCs. In another study, Karlson et al. investigate the types of barriers (*e.g.,* input and output challenges) people face when performing tasks on their smartphones. They report how frustrating the experience of task disruption is for smartphone users by using a survey on 24 users. In this thesis, we build on top of the findings of these two user studies. We investigate whether via static analysis and our metrics we can observe similar findings that UI complexity, as one of the barriers of doing tasks on smartphone [22], can have

an impact on user-perceived quality of mobile applications. However, little or no empirical research on currently existing mobile applications has been conducted to understand the characteristics of UI of mobile applications and how it can impact the user-perceived quality of mobile applications. In the following subsections we present different studies on mobile applications that have been trying to uncover insightful information by investigating currently existing mobile applications.

2.2 Mining Mobile Applications

Harman et al. introduce App Store Mining [26], and discuss factors of success for mobile applications. Their results show that there is a strong correlation between customers rating and the number of downloads. In this thesis, we use this study as a motivation for defining factors influence user-perceived quality of mobile applications by means of measures (*e.g.,* number of downloads) extracted from Android Market. Shabtai et al. study has been one of the first studies to conduct a formal study on Android PaKage (APK) files [27]. They apply machine learning methods to build a classifier for Android games and tools to detect malwares. They achieved 89% of accuracy in classifying applications into these two categories. Minelli and Lanza develop SAMOA [28], a new tool designed to help developers better understand the development and evolution of their mobile applications by gathering and visualizing basic source code metrics (e.g., size and complexity). Following the same line of work as these studies, we also try to get more insight about the characteristics of mobile applications by analyzing the existing mobile applications on the Android Market. The aforementioned studies analyze the source code of mobile applications to assess the characteristics and quality of them. However, we focus on the UI of

mobile applications to assess the quality (*i.e.,* user-perceived quality). Shirazi et al. [29] present a process to gain insights into mobile user interfaces by analyzing the 400 most popular free Android applications. However, they did not provide any evidence that there exists a relation between different characteristics of UI (*e.g.,* UI complexity) and user-perceived quality of mobile applications. In this thesis, we go beyond the analysis of source code, and we take into account the constructed UI elements of mobile applications to see their relation with user-perceived quality of mobile applications.

Various projects have focused on dynamic analysis of mobile applications. Dynamic analysis refers to a set of techniques that monitor the behavior of a program while it is executed.AndroidRipper is an automated technique that test Android applications via their GUI [30]. An application's GUI is explored to construct the GUI tree of the corresponding application for testing purposes or exercising it in a structured manner. Joorbachi et al. [31] presents a similar tool for iOS called iCrawler, a reverse engineering tool for iOS mobile applications that uses a state-machine model. It is capable of automatically detecting the unique states of a mobile application. However, such approaches cannot be applied on large scale studies due to the limitations of available dynamic GUI reverse engineering techniques. Instead, similar to Shirazi et al. [29], we use static analysis for GUI reverse engineering.

2.3 Studying the UI of Mobile Applications

Nilsson suggests that experienced UI developers who want to start developing UIs for mobile applications should start with UI templates to develop their applications [18]. In this context, a pattern is a formalized description of a proven concept that expresses

non-trivial solutions to a UI design problem. The primary goal of patterns in general is to create an inventory of solutions to help UI designers resolve UI development problems that are common, difficult and frequently encountered [32].

Software Engineering [33] practices adopted patterns as a way to facilitate reuse of software. Software reuse has been studied widely in the literature. Hindle et al. investigated the naturalness of software [34]. They show that code is very repetitive, and in fact even more so than natural languages. They showed evidence that code reuse is a common practice in software engineering. Moreover, in a very large-scale study of code by Gabel and Su [35], they found that code fragments of surprisingly large size tend to reoccur. As a result, software reuse is a common practice in software engineering.

There has been some studies focusing on studying the extent of reuse in the source code of mobile application as well. Ruiz et al. show that on average 61% of all classes in each category of mobile applications occur in two or more applications [25]. Moreover, Chen et al. implement an approach to detect application clones on Android market to detect malwares [36]. Another aspect of a mobile application is its UI. Developers' negligence in the importance of UI design is one of the major reasons for users to abandon a task on mobile applications and switch to PC [22]. User interface designers also have noticed that certain design problems occurred over and over [18]. Having the same goal as software engineering studies for code reuse, we study the extent of reuse for UI in Android applications over 8 different categories (*i.e.,* Shopping, Health, Transportation, Travel, News, Weather, Finance, Social).

To the best of our knowledge, there have been no studies that investigate the effect of the UI design on user-perceived quality. The question is whether reusing

frequently used UI elements can lead to better user-perceived quality or not.

2.4 Summary

In this chapter we introduce the related work about emprical studies on mobile applications and UI of mobile applications.

Chapter 3

Processing Mobile Applications Extracted from Android Market

In this chapter, we introduce the overall procedure that processes the data from a mobile application store (*e.g.,* Android Market). We discuss the approach for calculating user-perceived quality, extracting the structure of mobile applications and identifying functionalities of mobile applications which play a vital role to conduct our studies. Finally, we describe our subject system (*i.e.,* Android Market).

3.1 Approach Overview

In this thesis, we investigate different aspects of UI design (*i.e.,* UI complexity and UI reuse) that may lead to a better user-perceived quality. To this end, there are three paramount factors that need to be calculated to conduct our studies, *i.e.,* calculating user-perceived quality, extracting the structure of mobile applications, and identifying functionalities of mobile applications. Figure 3.1 gives an overview of our overall approach.

Given a mobile application corpus crawled from a mobile application store (*e.g.,*

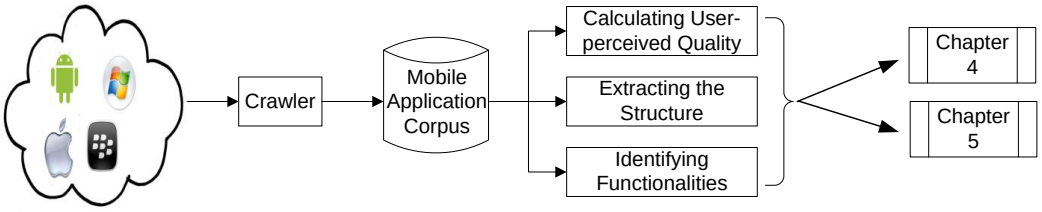

Figure 3.1: Overview of our data collection process.

Android Market), we calculate the followings:

3.1.1 Calculating User-perceived Quality

In Android market, there exists useful information for each application which reflects the experience of the users. Figure 3.2 shows the information available for eBay mobile application in Android market. Figure 3.2-A shows the rating for eBay application reported by Android market based on the rating of previous users. Figure 3.2-B shows the number of raters for this application. Figure 3.2-C shows the number of downloads which is a range. Here, we consider the higher bound as the representative of number of downloads for the eBay application. Finally, Figure 3.2-D shows a comment that has been written by a user about the eBay application.

As shown in Figure 3.2, users can rate applications from 1 to 5 (i.e., Low to High). The rating tends to reflect the user-perceived quality of applications, and give them a judgmental power to choose among the candidate applications. The rating of an application informs potential users about the experience of the earlier users.

However, Ruiz et al. [24] have shown that the rating of an application reported by Android Market is not solely a reliable quality measure. They found that 86% of the five-star applications throughout the Android Market in 2011 are applications

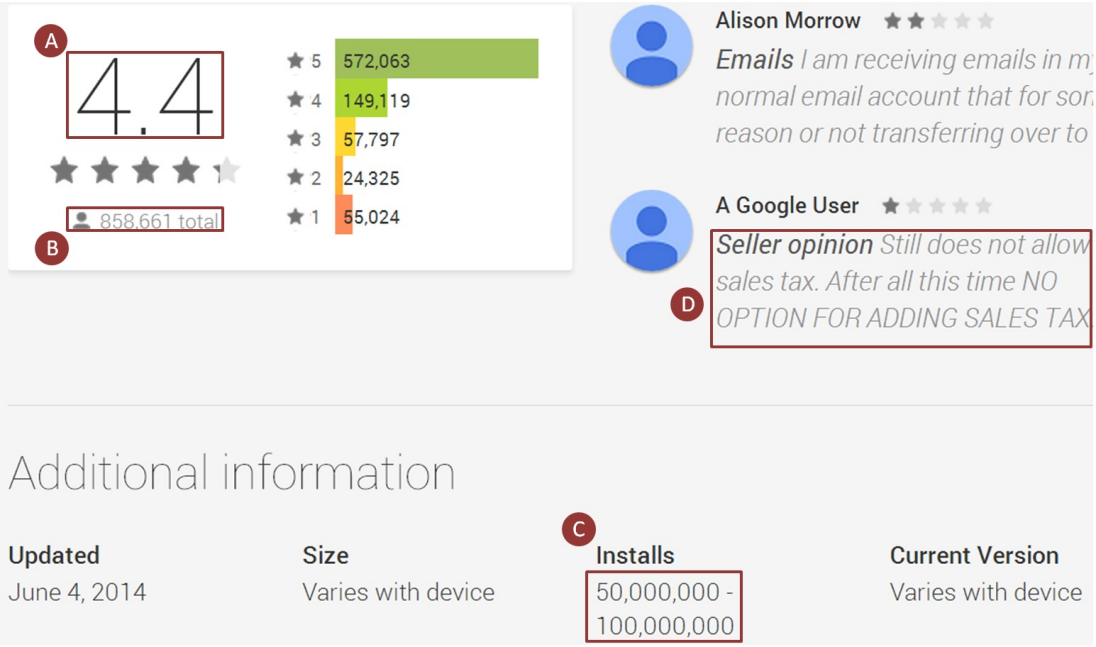

Figure 3.2: Information that are available for a mobile application in Android market.

with very few raters (less than 10 raters). Moreover, Harman et al. [26] show that the ratings have a high correlation with the download counts which can be deemed as a key measure of the successes for mobile applications. To overcome these challenges, we measure user-perceived quality by considering both rating and popularity factors (*i.e.*, the number of downloads and raters) using Equation (3.1):

$$UPQ(A) = (\frac{1}{n} * (\sum_{j=1}^{n} log(Q_j))) * Rating(A), \tag{3.1}$$

Where $UPQ(A)$ is the measured user-perceived quality for an application; A refers to an application; n is the total number of quality attributes (i.e., the number of downloads and raters) extracted from Android Market for A. Q_j shows a quality attribute. To normalize the value of quality attributes, we used log transform. $Rating(A)$ is the rating score extracted for A from the Android Market.

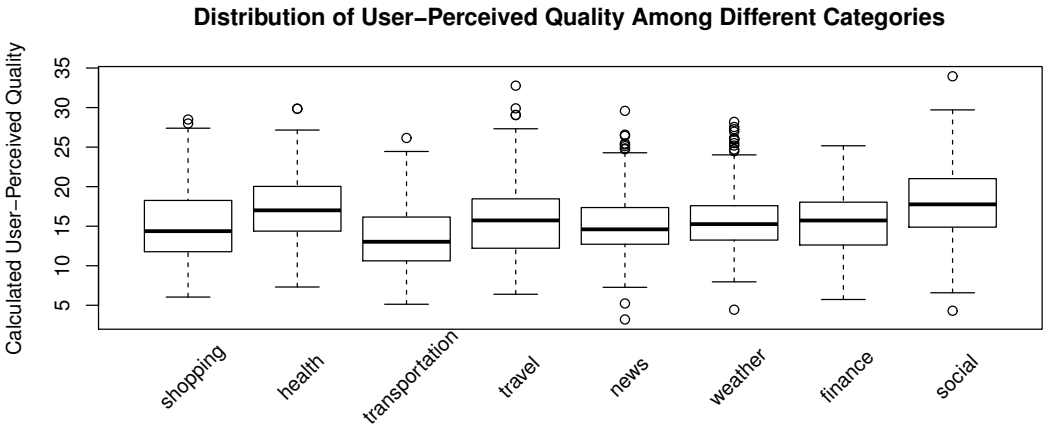

Figure 3.3: Distribution of user-perceived quality among different categories.

For instance, lets consider eBay and Amazon Android applications from the Shopping category. The rating scores reported by Android Market for each of these applications are 4.4 and 4.5. Relying on this rating, we can conclude that Amazon has a better user-perceived quality than eBay. However, if we go into more details and investigate the number of downloads and the number of raters for these applications, we can see that the number of downloads for eBay and Amazon are $1e + 08$ and $5e + 07$, respectively. The number of raters are 563,494 and 112,984, respectively. That is eBay has around 2 and 5 times more downloads and raters than Amazon, respectively. The value of $UPQ(A)$ for eBay and Amazon are 30.25 and 29.36. As a result, in total, eBay has a better user-perceived quality than Amazon.

Figure 3.3 shows the distribution of user-perceived quality calculated by Equation (3.1) among different categories in our corpus.

3.1.2 Extracting the Structure

In Android Market, each mobile application is shipped as an Android PacKage (APK) file. To analyze Android applications, we need to extract the content and the needed information from APK files. To decode an APK file, we use apktool [37], a tool for reverse engineering closed, binary Android applications. It decodes APK files almost to the original form and structure. It provides the source code of the application in an intermediate "Smali" format [38] which is an assembler for the dex format used in Android Java virtual machine implementation. The files resulted from reverse engineering on an APK file are used for our analysis to obtain insights about different aspects of UI design, namely UI complexity and UI reuse, and their effect on user-perceived quality of mobile applications.

3.1.3 Identifying Functionalities

To perform a fine-grained analysis, we study different aspects of UI design among activities with similar functionalities for mobile applications. We extract the functionalities of each mobile application using text mining techniques. We are interested in the keywords available in the source code or UI elements. For each activity, we extract contents, strings, labels and filenames associated to the source code of activities and their corresponding UI XML files. We use two different heuristics to extract the texts shown to a user from an activity: i) labels assigned to each element in the UI XML file, and ii) strings assigned from the source code. For the strings in UI XML files, each element in a UI XML file may contain an *android:text* label in which the value is a string shown to a user (see Figure 3.4). For the strings in source code of an activity, we search for *setText()* method call statements. This method specifies

```
<TextView android:text="@string/reminders", ...
```

Figure 3.4: Part of a Sample XML File

the human readable label of a UI element. We extract the human-readable label by analyzing the parameter values.

3.2 Subject Systems

In this thesis, we study 1,292 free Android applications distributed in 8 categories extract from Android Market (Google Play). Android Market (Google Play) started with 2,300 applications in March 2009. Currently there is more than 1 million applications in this market [39]. Android operating system has the highest market share among other competitors [40]. As a result, we decided to analyze the applications from this market. Moreover, we only study the free applications due to cost issues.

In Android Market, there are 34 different kinds of categories from which we analyze 8 different categories. The 8 different categories are: Shopping, Health, Transportation, News, Weather, Travel, Finance and Social. The intuition behind choosing these categories is that they encompass different functionalities of everyday use of mobile applications.

Table 3.1 shows descriptive statistics for different categories. In total, we study 1,292 free android applications crawled in the first quarter of 2013.

3.3 Summary

In this chapter, we describe the overview of our study design for extracting necessary information from Android Market to conduct our studies. At first, we calculate

Table 3.1: Summary of the characteristics of different categories

Category	# Applications	# Pages
Shopping	193	2,822
Health	286	4,129
Transportation	128	1,078
News	114	1,302
Weather	244	1,608
Travel	106	1,711
Finance	103	1,167
Social	118	1,107

user-perceived quality for each mobile application based on the availabe information such as: number of downloads and ratings in the Android Market. Then, to extract the source code and UI files for each mobile application from their APK files, we use an open source tool called apktool. Moreover, we also identify the consisting functionalities of mobile applications by using topic modeling techniques. Using these information, we investigate different aspects of UI design (*i.e.,* UI complexity and UI reuse) of mobile applications that may lead to a better user-perceived quality.

Chapter 4

An Exploratory Study on the Relation between User Interface Complexity and the Perceived Quality of Android Applications

In this chapter, we focus on UI complexity and its relation with the user-perceived quality of mobile applications. Moreover, we aim to derive guidelines for UI complexity by mining the available mobile applications on Android Market. It is possible to judge about user-perceived quality of mobile applications by studying statistics (*e.g.,* number of downloads and raters) on mobile stores [24]. To quantify UI complexity, we define seven metrics that can be calculated using static analysis. We calculate the metrics in two different granularities: i) *category*, and ii) *functionality* of mobile applications. A category reflects the purpose of a group of mobile applications (*e.g.,* Shopping or Health) extracted from mobile stores. A functionality defines a fine-grained capability of a mobile application (*e.g.,* Payment or Sign in). To quantify functionalities of mobile applications, we cluster their pages (activities) based on the topic similarity.

We examine our static analysis approach for quantifying UI complexity in the first

research question and discuss the relation between UI complexity and user-perceived quality in the following two research questions.

4.1 Motivation

User-perceived quality can be defined as the user's opinion of a mobile application. It can be quantified by the number of downloads and ratings in mobile stores. User-perceived quality can be influenced by UI design, stability and performance of applications [16]. The user-study conducted by Kane et al. [16] shows that user-perceived quality quantified by the amount of usage of mobile applications can vary among different categories due to the difficulties in text entry or poor output layouts. For example, most participants do not use smartphones for shopping activities. Furthermore, Karlson et al. [22] demonstrate that improper use of UI elements (*e.g.,* input and output) on mobile applications increases end-user frustration (*i.e.,* user-perceived quality). For example, the excessive use of input fields in mobile applications negatively affect user-perceived quality. Although mobile applications seem to be simple and easy to develop, these studies illustrate that designing UI for mobile applications is not a trivial task. To the best of our knowledge, there is no study providing empirical evidence that UI complexity has an impact on the user-perceived quality of mobile applications.

In the early days of software development, software metrics are used to derive guidelines for programmers. For example, McCabe [41] defines a complexity metric for functions, and recommends a proper implementation should hold a value below 10. Such guidelines can be exploited either during the development process for on-the-fly recommendation or during the quality assurance process. There exist several

studies on the design patterns for UI development of mobile applications [18]. However, they do not provide a concrete number of appropriate UI complexity for mobile applications in order to achieve high user-perceived quality. In this study, we study the relation between UI complexity and user-perceived quality of mobile applications. Furthermore, we strive to provide guidelines for the proper amount of UI complexity that helps an application achieve high user-perceived quality. To this end, we focus on three research questions introduced in the following section.

4.2 Research Questions

In this section, we address the following research questions:

RQ4.1) Can our measurement approach quantify UI complexity?

A previous study by Kane et al. [23] shows that the user-perceived quality of some categories of mobile applications are higher than the others. Moreover, Karlson et al. [22] report that UI complexity can impact the user-perceived quality of mobile applications. The earlier aforementioned findings are based on user studies. Since we are aiming to derive guidelines, we need to study a large number of applications. Therefore, instead of user studies, we opt for a scalable measurement approach based on static analysis. As the first step in this study, we evaluate the practicability of our UI complexity metrics and measurement approach. We try to replicate the aforementioned findings by earlier user studies using our static analysis approach.

RQ4.2) Does UI complexity have an impact on the user-perceived quality within each category of mobile applications?

We found that the number of activities (pages) used as a quantifier of UI complexity for mobile applications has a relation with user-perceived quality. However, we did

not observe any relation between UI complexity and user-perceived quality for other studied quantifiers of UI complexity (*e.g.,* average number of inputs). Therefore, we cannot provide guidelines about the proper amount of usage of UI complexity metrics at the application level for mobile applications.

RQ4.3) Does UI complexity have an impact on the user-perceived quality of the functionalities in mobile applications?

We observe that there exists a relation between UI complexity measured in functionality level and user-perceived quality of application pages (activities) belong to a similar functionality. Specifically, we observe that activities with high user-perceived quality tend to be simpler in terms of UI complexity (*e.g.,* less number of inputs or outputs). Similar to other guidelines for software development (*e.g.,* Macabbe [41]), we also report the average number of usage of our metrics in applications with high user-perceived quality.

4.3 Study Design

In this study, we investigate whether UI complexity can affect the quality of android applications perceived by the users. The objective of our study is to derive guidelines for the proper amount of UI complexity of Android applications in a large scale study. To this end, we study 1,292 free Android applications distributed in 8 categories. Developers and quality assurance personnel can use our guidelines to improve the quality of mobile applications.

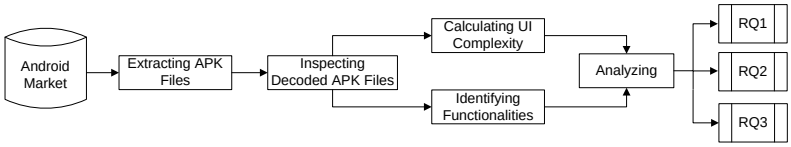

Figure 4.1: Overview of our data collection process.

4.3.1 Data Processing

Figure 4.1 shows an overview of our data processing steps in this study. As mentioned in chapter 3, we are calculating user-perceived quality, extracting the APK files, and identifying functionalities which are key factors to conduct our studies. To answer our three research questions in this study, we also need to calculate UI complexity. In the two following sections we describe how we calculate UI complexity.

Inspecting Decoded APK Files

As mentioned in chapter 1.1.3, for a standard activity (*i.e.,* a UI page in an Android application) there should exists two files: i) the source code of the activity which the path is indicated in the AndroidManifest.xml (manifest) file, and ii) the corresponding UI XML file. However, there does not exist any direct mapping between the source code and the UI of an activity. To measure UI complexity, we need to recover this linking. In this section, we discuss the heuristics for linking source code of activities to their corresponding UI XML files.

Given an application, we can extract the path to the source code of activities from the manifest file. To map the activities to their corresponding UI XML files, we use different heuristics. Similar to Shirazi et al.'s work [29], we parse the source code of an activity (*i.e.,* Smali file) to look for a call of the *SetContentView()* method, which

includes an ID to the corresponding UI XML file. However, this heuristic cannot map an activity to the corresponding XML file when there does not exist an ID for the corresponding XML file. For example, Figure 4.2 shows a part of the source code (Smali file) for an activity of eBay application. As marked in bold in Figure 4.2, the input argument to the *SetContentView()* is not an ID. It is the name of the UI XML file which is converted to an ID at run time. To overcome this issue, we trace IDs and names.

```
.line 149
sget v0, R$layout;->photo_manager_activity:I

invoke-virtual {p0, v0}, PhotoManagerActivity; -> setContentView(I)V
```

Figure 4.2: Part of a Sample Smali File

Calculating UI Complexity

We parse the UI XML files to calculate different UI metrics that is used to quantify UI complexity. We consider two sets of metrics in different granularities (*i.e.,* application and activity levels) as shown in Table 4.1. For the application level metrics, we compute the UI complexity metrics for each activity, and lift the metrics up to the application level by using the *average* values for ANI, ANO, ANE and *sum* for NA. To identify the elements as inputs and outputs, we categorize the elements shown in Table 4.2 as inputs and outputs. We use these input and output tags since such elements are frequently used in Android applications [29]. Table 4.3 shows the number of inputs, outputs and UI elements calculated in each category.

Table 4.1: Proposed Application and Activity Level Metrics

	Metric Names	Description
Activity Level	NI	Number of Inputs in an activity
	NO	Number of Outputs in an activity
	NE	Number of Elements in an activity
Application Level	ANI	Average Number of Inputs in an application
	ANO	Average Number of Outputs in an application
	ANE	Average Number of Elements in an application
	NA	Average Number of Activities in an application

Table 4.2: Input and Output Tags

	Element Names
Inputs	Button, EditText, AutoCompleteTextView, RadioGroup, CheckBox ToggleButton, DatePicker, TimePicker, ImageButton, RadioButton Spinner
Outputs	TextView, ListView, GridView, View, ImageView, ProgressBar GroupView

4.3.2 Analysis Methods

We investigate the impact of UI complexity on the user-perceived quality of Android applications. Kruskal Wallis test [42] is used to study whether there exists a difference in UI complexity between our 8 different categories of mobile applications. Kruskal Wallis test is a non-parametric test to assess whether two or more samples (*i.e.,* 8 in here) originate from the same distribution. We use Wilcoxon rank sum test [42] to compare the usage of UI complexity elements between applications with low and high user-perceived quality. The Wilcoxon rank sum test is also a non-parametric statistical test to assess whether two distributions have equally large values. In general, non-parametric statistical methods do not make assumptions about the distributions of assessed variables. The Kruskal Wallis test is an extension of the Wilcoxon rank sum test for more than two samples.

Table 4.3: Summary of the number of inputs, outputs, and UI elements of different categories

Category	# Inputs	# Outputs	# Elements
Shopping	12,529	25,058	68,468
Health	23,232	40,330	108,366
Transportation	5,603	7,718	22,991
News	4,725	7,407	23,507
Weather	6,713	38,659	84,739
Travel	7,164	15,210	38,285
Finance	5,989	12,899	33,818
Social	4,948	7,646	24,091

To extract the functionalities of mobile applications in a category, we use Latent Dirichlet Allocation (LDA) [43] to label a set of activities with a fine-grained functionality. LDA [43] is a topic model that generates a topic distribution probability for each document analyzed. A topic is a collection of frequently co-occurring words in the corpus. LDA can provide the two following information given a set of documents: (i) the topics that describe these documents, and (ii) for each document, the probability that a document will belong to a particular topic. Then, we use Wilcoxon rank sum test [42] to investigate whether UI complexity varies between activities with low and high user-perceived quality for each functionality.

4.4 Study Results

This section presents and discusses the results of our three research questions.

4.4.1 RQ4.1: Can our measurement approach quantify UI complexity?

Motivation.

Measuring the complexity of a UI is not a trivial task. As the first step, we evaluate if our UI complexity metrics and our measurement approach (*i.e.,* static analysis) can be used to quantify UI complexity. We want to answer this concern by testing whether our UI complexity metrics can testify hypotheses reported by previous user-studies. A user study by Kane et al. [23] has shown that user-perceived quality of some categories of mobile applications is lower than the others. For example, users are reluctant to use smartphones for shopping purposes. Earlier studies conjecture that the difference between user-perceived quality of mobile applications can be due to their UI complexity (*e.g.,* input and output challenges) [22]. As a result, we aim to find out whether we can make similar observations using our metrics and approach. If we provide evidence that our measured metrics for quantifying UI complexity can correlate with the findings of previous studies, we will conjecture that our proposed metrics can be used for studies on the UI complexity of mobile applications.

Approach.

For each APK file (application) in different categories. We use the approach mentioned in Section 4.3.1 to map the source code of activities to their corresponding UI XML files. Next, to quantify UI complexity within each category (see Table 4.1), we calculate four application level UI metrics (*i.e.,* ANI, ANO, ANE and NA). Finally, based on each metric, we observe whether the UI complexity is different between categories. We test the following null hypothesis among categories:

H_0^1: *there is no difference in UI complexity of various categories.*

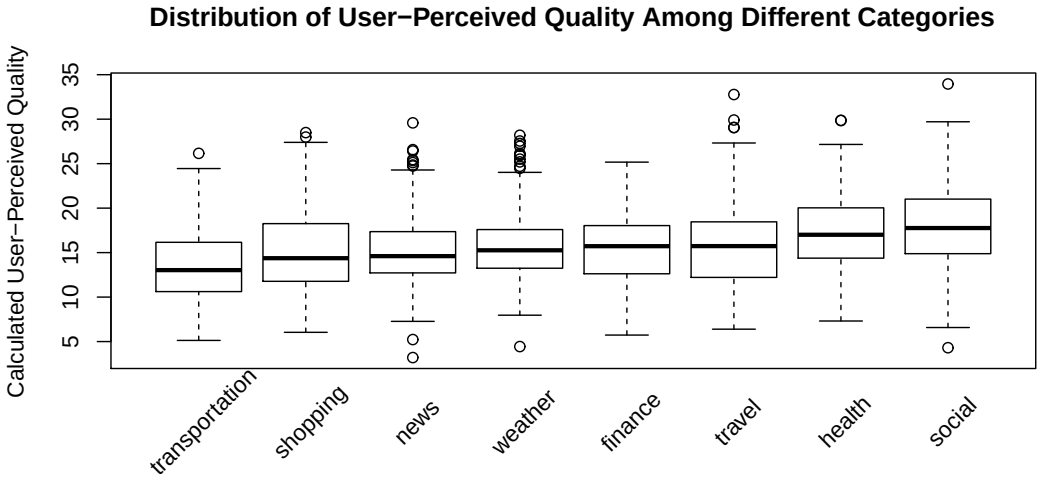

Figure 4.3: Sorted categories based on the distribution of their user-perceived quality.

We perform Kruskal Wallis test [42] using the 5% confidence level (*i.e.,* p-value $<$ 0.05) among categories. This test assesses whether two or more samples are originated from the same distribution. In other words, it is an extension of the Wilcoxon rank sum test applied to more than two groups.

To testify the previous findings by Kane et al. [23], We compute the Spearman correlation [42] between the user-perceived quality calculated by Equation (3.1) for each category and the ones reported by Kane et al.'s [23] study for each category. We found that there exists a 0.36 (*i.e.,* medium) correlation between these two groups. Therefore, we can replicate their study by using our method for calculating user-perceived quality for each category. We classify our categories based on their suggestion into two categories: i) applications that belong to the categories with high user-perceived quality (see Figure 4.3), and ii) the ones that belong to categories with low user-perceived quality (*i.e.,* Shopping, Transportation, news, and Weather). To decide

which categories are of high or low user-perceived quality. First, we sort the categories based on user-perceived quality. Then, we call the categories in the lower half the ones with low user-perceived quality, and the categories in the other half, categories with high user-perceived quality (see Figure 4.3). We investigate whether UI complexity is varied among these two groups. We test the following null hypothesis for these two groups:

H_0^2: *there is no difference in the UI complexity of applications related to categories with high and low user-perceived quality.*

We perform a Wilcoxon rank sum test [42] to evaluate H_0^2, using the 5% level (*i.e.*, p-value < 0.05).

Table 4.4: Kruskal-Wallis test results for application level UI metrics in different categories.

Metric	p-value
ANI	0.001148
ANO	<2.2e-16
ANE	<2.2e-16
NA	4.842e-05

Table 4.5: Wilcoxon rank sum test results for the usage of application level UI metrics in categories with high and low user-perceived quality.

Metric	p-value	ΔCliff
ANI	3.04e-6	-0.24
ANO	2.8e-10	-0.16
ANE	0.002	-0.12
NA	0.009	-0.07

Findings.

Our approach for quantifying UI complexity confirms the findings of previous studies. The Kruskal Wallis test was statistically significant for each application level UI metric between different categories (Table 4.4) meaning that there exists a significant difference in the UI complexity of various categories. Moreover, there also exists a difference between the UI complexity of applications related to categories with high and low user-perceived quality. As shown in Table 4.5, there exists a significant difference in UI complexity quantified by the four studied metrics that are used to quantify the applications in the categories of high and low user-perceived quality. Therefore, by quantifying UI complexity of mobile applications, we found the similar findings as the earlier user studies ([22], [23]) that UI complexity is important on user-perceived quality of mobile applications. Moreover, UI complexity varies among different categories. Therefore, our measurement approach based on static analysis can quantify UI complexity.

Our measurement approach can quantify UI complexity.

4.4.2 RQ4.2: Does UI complexity have an impact on the user-perceived quality within each category of mobile applications?

Motivation. In this research question, we quantify the user-perceived quality by the measures extracted from Android Market for each application (*i.e.,* application level metrics), and investigate whether UI complexity can play a vital role in the user-perceived quality of mobile applications. If we can show that the difference exists, we can guide developers about the proper UI complexity in the application level. For example, given an application, we can recommend the proper number of activities

(*i.e.,* pages) required to have a shopping application with high user-perceived quality.

Approach. Using the same approach used for **RQ4.1**, we collect the application level UI metrics (see Section 4.3.1) to quantify UI complexity. Next, to measure user-perceived quality, we extract the quality attributes (*i.e.,* the number of downloads, rating of applications, and the number of raters) for each application from Android Market (see Section 3.1.1). Finally, we investigate whether there exists a relation between UI complexity and the user-perceived quality of mobile applications within each category.

For each category of mobile applications, we want to investigate the trend of using application level UI metrics between applications with low and high user-perceived quality, and try to define guidelines about the usage of UI-complexity for high user-perceived quality applications. To define low and high quality applications, for each category, we sort the applications based on their calculated user-perceived quality (see Equation (3.1)). We break the data into four equal parts, and named the ones in the highest quartile, applications with high user-perceived quality, and the ones in the lowest quartile, applications with low user-perceived quality. Finally, we investigate whether there exists any statistically significant difference in UI complexity between applications with low and high user-perceived quality for each category. We test the following null hypothesis for each category:

H_0^3: *there is no difference in the UI complexity among applications with low and high user-perceived quality in each category.*

We perform a Wilcoxon rank sum test [42] to evaluate H_0^3. To control family-wise errors, we apply Bonferroni correction which adjusts the threshold p-value by dividing the number of tests. The number of tests is 32 since we have 8 different

categories and 4 different application UI metrics to quantify UI complexity. There exists a statistically significant difference, if p-value is less than 0.05/32=0.001. We also compute and report the difference between the average application UI metric in low and high quality applications. Finally, we report the average number of usage of application level UI metrics in applications with high user-perceived quality.

Table 4.6: Difference of usage of used UI metrics between the applications with low and high user-perceived quality in each category.

Category	ANI	ANO	ANE	NA
Shopping	↗ 2.35	↘ 5.65	↘ 16.03	↘ 41.24*
Health	↗ 1.91^{+}	↗ 5.64	↗ 14.25	↘ 29.2°
Transportation	↗ 2.55	↘ 6.40	↘ 15.82	↗ 8.82^{+}
News	↘ 4.25	↘ 6.06	↘ 18.28	↗ 16.02
Weather	↘ 2.01	↘ 12.16	↘ 24.08^{+}	↗ 8.86*
Travel	↗ 2.39	↘ 6.71	↘ 17.48	↘ 28.68
Finance	↘ 5.67	↘ 10.35	↘ 27.76	↘ 16.43
Social	↘ 2.40	↗ 4.51	↗ 14.42	↘ 19.56^{+}

Given a UI complexity metric, ↘ means that applications with low user-perceived quality have less of the UI complexity metric, and ↗ means that applications with low user-perceived quality have more of the UI complexity metric than the ones in high user-perceived quality.
($p<0.001/50^{\star}$; $p<0.001/10^{\circ}$; $p<0.001^{+}$)

4.4.3 Findings.

Only number of activities (NA) has a strong relation with the perceived quality of applications, but we cannot find a strong relation between the usage of (ANI, ANO, and ANE) and the quality of applications. Table 4.6 shows the difference in the usage of calculated UI metrics (*i.e.,* the quantifiers of UI complexity) between the applications with the low and high user-perceived quality in each category. For each cell of the table, we report three pieces of information. Let's consider the cell related to the Shopping category for NA (*i.e.,* Number of Activities) metric. First, there is a "↗" or "↘" sign which implies whether the difference for the corresponding metric

(*i.e.,* NA) between applications with low and high user-perceived quality is positive or negative. In this example, it is negative ("↘") which means that applications in the Shopping category with low user-perceived quality have less activities than the ones with high user-perceived quality. Moreover, we report the average usage of the corresponding metric (NA) for the applications with high user-perceived quality which is 41.24 in this example. Such average values can be used to derive software development guidelines (*e.g.,* McCabe [41]). It implies the average number of the activities for applications with high user-perceived quality. Last not the least, we report whether the difference in the usage of the corresponding metric (*i.e.,* NA) is statistically significant between applications of low and high user-perceived quality. In this example, the difference is statistically significant ($\star$).

The difference between low and high quality applications is significant for NA (*i.e.,* number of activities) in 5 cases among 8 different categories. It means that the number of activities for an application have a relation with its user-perceived quality in the five identified categories. As shown in Table 4.6, the difference between each category is varied. For example, in the Transportation and Weather categories, applications with low user-perceived quality have more activities than the ones with high user-perceived quality. However, this behavior is quite different in the Shopping category. For ANI, ANO and ANE in most cases, the difference is not statistically significant. As a result, it is not possible to provide guidelines for the usage of UI complexity elements in general for the application level. Our intuition is that we need to conduct a more fine-grained study since even in the same category, the purpose of the applications might vary considerably. For example, in the Shopping category, there exists some applications like eBay and Amazon which are fairly complex (*i.e.,*

consist of a large number of input and output elements, and a considerable number of activities). On the other hand, there also exists some discount applications in Shopping category which are simply calculating the price of an item after a discount. They are more simpler than eBay and Amazon. Therefore, in **RQ4.3**, we perform a more fine-grained analysis in the functionality level, and investigate whether we can provide guideline in the usage of UI complexity metrics in the functionality level.

We can successfully derive guidelines for the proper number of activities to achieve high user-perceived quality in each category.

4.4.4 RQ4.3: Does UI complexity have an impact on the user-perceived quality of the functionalities in mobile applications?

Motivation.

In **RQ4.2**, we could not find empirical evidence that UI complexity has a strong relationship with user-perceived quality of mobile applications, using a high level analysis. This originates from the fact that mobile applications are designed for different purposes even in the same category. To perform a more fine-grained analysis, we observe whether there is a relation between UI complexity and the user-perceived quality among various functionalities of mobile applications. If yes, we can provide guidelines to developers of the proper number of activity level UI metrics required to have a high quality functionality.

Approach.

For each application, we extract the corresponding activities and their UI XML layouts (see Section 4.3.1). Next, to label each activity with a fine-grained functionality,

we use LDA [43] which clusters the activities (documents) based on their functionalities (*i.e.,* topics). In other words, for each activity, we extract all the strings and labels shown to the users (see Section 3.1.3). We apply LDA to all the activities retrieved from the existing applications in a category to extract their corresponding functionalities.

Since mobile applications perform a limited number of functionalities, the number of topics (*i.e.,* K) should be small in our research context. As we are interested in the major functionalities of applications, we empirically found that $K = 9$ is a proper number for our dataset by manual labeling and analysis of randomly selected mobile applications. We use MALLET [44] as our LDA implementation, which uses Gibbs sampling to approximate the distribution of topics and words. We run the algorithm with 1000 sampling iterations, and use the parameter optimization in the tool to optimize α and β. In our corpus, for each category, we have n activities (extracted from the applications in the corresponding category) A = $\{a_1, ..., a_n\}$, and we name the set of our topics (*i.e.,* functionalities) F = $\{f_1, ..., f_K\}$. It is important to mention that these functionalities are different in each category, but the number of them is the same ($K = 9$). For instance, f3 in the Shopping category is about "*Login*" and "*Sign in*" functionality. However, in the Health category, it is about "*search*" and "*information seeking*" functionality. LDA automatically discovers a set of functionalities (*i.e.,* F), as well as the mapping (*i.e.,* θ) between functionalities and activities. We use the notation θ_{ij} to describe the topic membership value of functionality f_i in activity a_j.

To answer this research question, we go into a lower level of granularity and study **RQ4.2** from a different perspective (*i.e.,* functionality level). Each application (A)

is consisted of several activities ($\{a_1, a_2, ..., a_n\}$), and it has a user-perceived quality ($UPQ(A)$) calculated by Equation (3.1). To compute the user-perceived quality for each activity, we assign each activity the user-perceived quality obtained from the application that they belong to. As a result, all the activities from the same application acquire the same user-perceived quality. However, by applying LDA [43] each activity (document) acquires a weight of relevance to each functionality. Therefore, the user-perceived quality for an activity can originate from two sources: i) the user-perceived quality of its corresponding application, and ii) the probability that this activity belongs to a functionality. Moreover, we use a cut-off threshold for θ (*i.e.,* 0.1) that determines if the relatedness of an activity (document) to a functionality is important or not. A similar decision has been made by Chen et al. [45]. Therefore, we calculate the user-perceived quality for each activity as the following:

$$AUPQ(a_j) = \theta_{ij} * UPQ(a_j), \tag{4.1}$$

Where $AUPQ(a_j)$ reflects the activity level user-perceived quality for activity j (a_j); θ_{ij} is the probability that activity j (a_j) is related to functionality i (f_i); UPQ(a_j) is the user-perceived quality of the application which a_j belongs to it.

Similar to **RQ4.2**, but from a different perspective (*i.e.,* functionality level), we sort the activities based on the user-perceived quality for each functionality in each category. Then, we break the data into four equal parts, and named the ones in the highest quartile, activities with high user-perceived quality, and the ones in the lowest quartile, activities with low user-perceived quality. Finally, we investigate whether there exists any difference in the distribution of activity level UI metrics (*i.e.,* quantifiers of UI complexity in functionality level) between activities of low and

high user-perceived quality for each functionality in each category. To this end, we test the following null hypothesis for each activity level UI metric in each category for each functionality:

H_0^5*: there is no difference in UI complexity between activities with low and high user-perceived quality.*

We perform a Wilcoxon rank sum test [42] to evaluate H_0^5. To control family-wise errors, we apply Bonferroni correction which adjusts the threshold p-value by dividing the number of tests. The number of tests is 216 since we have 8 different categories and 3 different activity level UI metrics among 9 different functionalities. There exists a statistically significant difference, if p-value is less than 0.05/216=2.31e-04.

Findings.

There is a significant difference between UI complexity of activities with low and high user-perceived quality. Table 4.7 shows our findings for activity level UI metrics (*i.e.,* NI, NO, NE) which are the quantifiers of UI complexity in functionality level. For each cell of Table 4.7, we report three pieces of information. Let's consider the cell related to the Shopping category for the third functionality (*i.e.,* f3) which refers to "*Login*" and "*Sign in*" functionalities, for the NI (*i.e.,* Number of Inputs) metric. In this cell, first, there is a "↗" or "↘" sign which implies whether the difference for the corresponding metric (*i.e.,* NI) between activities with low and high user-perceived quality is positive or negative. In this example, it is positive ("↗") which means that activities for "*Login*" and "*Sign in*" functionality (f3) in the Shopping category with low user-perceived quality have more complexity for NI than the ones with high user-perceived quality. Moreover, we report the average usage of the corresponding

Table 4.7: Difference of usage of activity level UI metrics between the activities with low and high user-perceived quality for each functionality in each category.

		f1	f2	f3	f4	f5	f6	f7	f8	f9
Shopping	NI	↗2.23*	↗2.38*	↗3.92	↗2.83*	↘3.89	↗3.22*	↗2.53*	↘3.44*	↗2.25
	NO	↗4.01*	↗3.55	↗4.77*	↗4.92*	↘8.55	↗5.40*	↗4.28*	↗6.27*	↗3.57
	NE	↗11.13*	↗9.84*	↗16.17*	↗13.32*	↘22.80	↗15.71*	↗11.83*	↗16.90*	↗10.00*
Health	NI	↗2.92*	↗2.01*	↘2.57*	↗3.25*	↗2.41*	↗2.54*	↘2.55	↗3.23	↘2.46
	NO	↘4.20*	↘3.16*	↗3.22*	↗5.24*	↗2.70*	↗3.70*	↗3.78*	↗4.66*	↘3.11*
	NE	↗13.41*	↘13.43*	↗10.35*	↗14.55*	↗8.32*	↗10.89*	↗10.86*	↗13.77*	↘8.95*
News	NI	↗2.63*	↘2.36*	↘3.25	↗2.50*	↗2.21*	↗2.45*	↗3.26*	↗2.13*	↘2.47*
	NO	↘3.70*	↘3.15*	↗3.50*	↘3.03*	↗3.00*	↗3.91*	↗3.58*	↗2.51*	↗3.72*
	NE	↘11.66*	↘10.40*	↘11.94*	↘9.69*	↗10.06*	↗12.38*	↗12.59*	↗8.63*	↗12.75*
Transportation	NI	↘3.49*	↗4.17	↗3.11*	↘1.77*	↘3.83*	↗4.39	↗3.22*	↗4.09^{+}	↗3.78*
	NO	↗2.81*	↗5.07^{+}	↗2.84*	↗3.35*	↘3.49*	↘4.21*	↗4.85*	↗3.53*	↘5.25°
	NE	↗10.39*	↗15.91^{+}	↗8.96*	↗9.44*	↗12.83*	↘12.72*	↗13.34*	↗12.70*	↘12.98*
Weather	NI	↗2.08*	↘3.33	↗2.35	↗1.23*	↗2.87*	↘2.04	↗2.03*	↗3.78^{+}	↘3.29°
	NO	↘5.35*	↘6.17*	↗5.57*	↘11.77*	↘5.48*	↗4.23*	↗1.82*	↗6.38*	↗3.79*
	NE	↘10.92*	↘16.61*	↗14.03*	↘30.97*	↗14.15*	↗8.99*	↗5.92*	↗14.65*	↗11.51*
Travel	NI	↗3.36*	↘3.81	↗2.36*	↘3.52^{+}	↗3.51*	↘2.87*	↗3.64*	↗3.03*	↗2.41*
	NO	↗4.22*	↗5.64*	↗2.61*	↘5.66*	↗4.77*	↘4.03*	↗4.26*	↘3.57*	↘3.21*
	NE	↗12.94*	↗16.62*	↗7.78*	↘16.19*	↗14.52*	↘12.38*	↗11.94*	↘12.68*	↗10.46*
Finance	NI	↗3.38*	↘2.81*	↗3.97*	↘2.81*	↗2.40*	↗2.37*	↘4.14	↗4.02*	↘5.29
	NO	↗6.42*	↘4.59*	↘7.85*	↗6.30*	↗4.00*	↗4.01*	↗7.22*	↗5.98*	↘9.41°
	NE	↗15.90*	↘11.60*	↘21.00*	↗18.16*	↘11.24*	↗10.58*	↗18.24*	↗16.85*	↘24.77
Social	NI	↗2.77*	↘3.17*	↗2.48*	↘2.04*	↗2.96^{+}	↗3.02*	↗3.12*	↗2.06*	↘4.07
	NO	↗4.57*	↘3.81*	↗3.86*	↗2.56*	↗3.50*	↗3.86*	↗5.38*	↗2.55*	↘6.16
	NE	↗14.45*	↘14.10*	↗13.43*	↘9.39*	↗12.41*	↗14.36*	↗16.38*	↗9.35*	↘19.34

Given a UI complexity metric, ↘ means that activities with low user-perceived quality have less of the UI complexity metric, and ↗ means that activities with low user-perceived quality have more of the UI complexity metric than the ones in high user-perceived quality.
($p<0.0002/50^{\star}$; $p<0.0002/5^{\circ}$; $p<0.0002^{+}$)

metric (NI) for the activities with high user-perceived quality which is 2.23 in this example. Such average values can be used to derive software development guidelines (*e.g.,* McCabe [41]). Here, it implies the average number of the corresponding activity level UI metric for good quality activities. Finally, we report whether the difference in the usage of the corresponding metric (*i.e.,* NI) is statistically significant between low quality activities and high quality ones. In this example, the difference is statistically significant ($\star$).

As it can be seen from Table 4.7, we can reject H_0^5, and conclude that there exists a significant difference in UI complexity between activities with low and high user-perceived quality. Moreover, in most cases this difference is a positive number ("$\nearrow$") meaning that low quality activities tend to use more activity level UI metrics than the high quality ones. In other words, simpler activities in terms of our used activity level UI metrics may results in a better perceived quality by the users. Our guidelines can be exploited by developers to use the proper UI complexity required to have functionalities with high user-perceived quality.

Simpler activities in terms of UI complexity tend to have better user-perceived quality. We can successfully provide guidelines for the proper number of inputs, outputs and elements to achieve high user-perceived quality for each functionality in different categories.

4.5 Summary

We provided empirical evidence that UI complexity has an impact on user-perceived quality of Android applications. To quantify UI complexity, we proposed various UI

metrics with different granularities (*i.e.,* application and activity level). The highlights of our analysis include: i) We can quantify UI complexity based on our measurement approach (**RQ4.1**), ii) NA (number of activities) can have an impact on the user-perceived quality of an application. (**RQ4.2**), and iii) There is a significant difference between UI complexity of activities with low and high user-perceived quality. Activities with high user-perceived quality tend to use less activity level UI metrics (*i.e.,* simpler) than activities with low user-perceived quality. Moreover, we derive guidelines for the proper amount of UI complexity required to have functionalities with high user-perceived quality (**RQ4.3**).

Chapter 5

An Exploratory Study on Reused User Interface Elements of Andriod Applications

In this chapter, we discuss the second empirical study that is extended from our first empirical study. We focus on another aspect of UI design (*i.e.,* UI reuse). Fast development of mobile industry causes a need for the creation of reusable components, which would decrease the amount of time needed to build a mobile application. An important part of a mobile application is the user interface (UI). To expedite the development process of designing UIs, developers usually use previously used UI practices or templates [18].

In this study, we measure to what extent the UI of different mobile applications are similar to each other within the same categories or across different categories. Moreover, we try to show whether building the UI of a mobile application based on previously used practices can affect the user-perceived quality of the mobile applications or not.

We investigate four research questions on the relation between UI design and user-perceived quality of mobile applications. Compared with the previous study discussed

in Chapter 4, we use the same study design process, but different approaches to identify UI design of mobile applications to carry out the study in this chapter.

5.1 Motivation

To develop a simple mobile application, there are two things that should be developed: i) User Interface (UI), and ii) logic (*i.e.,* source code) of the application. Israel et al. have shown that classes in the Android Market are reused significantly [25], which implies that many applications in the Android Market use very similar logic. Another important aspect of a mobile application is the user interface (UI). To the best of our knowledge no one has focused on studying the reuse of UI in mobile applications.

Due to the limitations of mobile applications (*e.g.,* small screen size, network problems and computational power [17]) developers should be more careful in designing their applications on mobile applications than PCs [22]. Developers' negligence in the importance of UI design is one of the major reasons for users to abandon a task on mobile applications and switch to PC [22]. To design UI for an application, developers usually adopt commonly used practices or standard templates [18]. However, the question still remains whether reusing previously used UI elements can impact the user-perceived quality of the applications.

We focus our study on 8 different categories of Android Market (i.e., Shopping, Health, Transportation, Travel, News, Weather, Finance, and Social). First, we investigate the extent of reuse for UI among Android mobile applications. To quantify the extent of reuse for UI, we define Reused Elements Set (RES), as a set of UI elements that appear in at least more than one UI page. Then, we investigate whether using RES within and across categories of Android applications can impact

the user-perceived quality. Furthermore, we go into more details, and investigate what characteristics of RES can provide high user-perceived quality in different functionalities of Android applications in each category. A functionality defines a fine-grained capability of a mobile application (e.g., Payment or Sign in). Finally, through a manual analysis we extract templates with high user-perceived quality from the RES extracted from the Shopping category for a number of functionalities. UI templates are recurring solutions that solve common design problems. UI templates are hidden (*i.e.,* a subset) in the RES. Recommending UI templates used in applications with high user-perceive quality can help developers to faster develop UIs with high user-perceived quality for mobile applications.

5.2 Research Questions

In this section, we address the following four research questions:

RQ5.1) To what extent UI elements set are reused?

We show that reuse widely occurs for RES within each category (*i.e.,* 60% of a UI XML file can be constructed by similar RES that are repeated in other UI XML files). Moreover, on average there exists a 23% reuse of RES across different categories.

RQ5.2) Does reused UI elements set impact user-perceived quality within and across categories?

RES that are reused in across categories can provide a significant difference in the user-perceived quality. However, it is category dependent whether it will provide better user-perceived quality or not. Within each category, RES with high user-perceived quality tends to be distributed in fewer applications. Moreover, using developer-customized UI elements may provide high user-perceived quality.

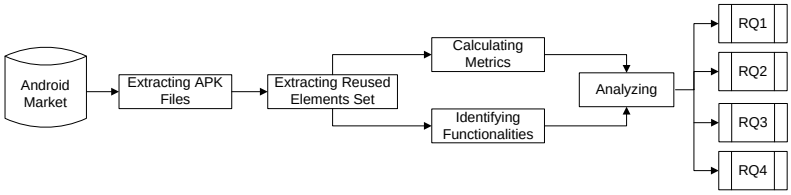

Figure 5.1: Overview of our data collection process.

RQ5.3) Does reused UI elements set have an impact on the user-perceived quality of the functionalities in mobile applications?

In almost every functionality, in order to achieve high user-perceived quality, it is better to use the RES that are reused in a few number of applications, or used in more UI XML files. Each application can consist of a number of UI XML files. Moreover, for some certain functionalities it is better to use developer-customized UI elements to achieve high user-perceived quality.

RQ5.4) Can we extract UI templates from reused UI elements sets of high user-perceived quality?

Yes, through a manual analysis, we can successfully recommend UI templates related to UI XML files of high user-perceived quality for certain functionalities in the Shopping category.

5.3 Study Design

In this study, we investigate to what extent RES (*i.e.,* Reused Elements Set) within and across categories exists, and examine what characteristics of RES may lead to a better user-perceived quality.

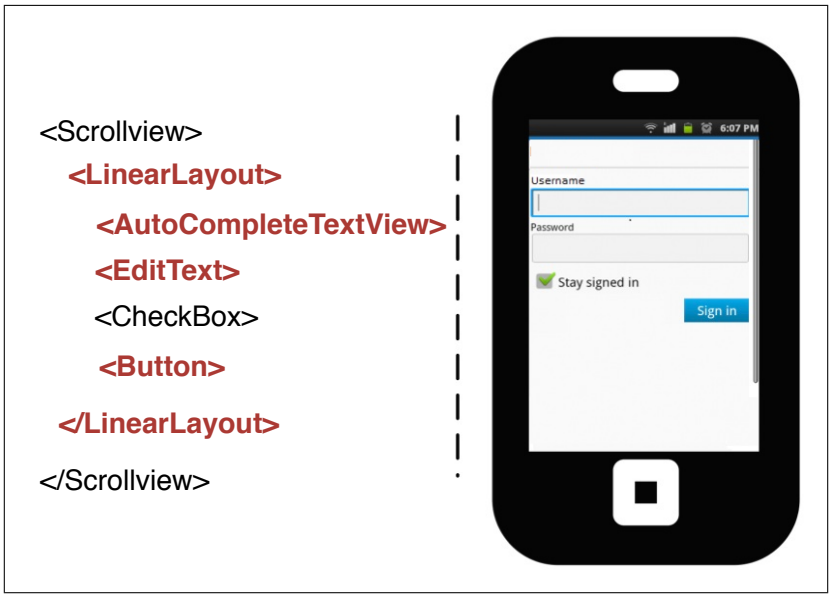

Figure 5.2: A UI XML file with its corresponding UI elements.

5.3.1 Data Processing

Figure 5.1 shows an overview of our data processing steps in this study. As mentioned in chapter 3, we calculate user-perceived quality, extract the structure of mobile applications, and identify functionalities which are key factors to conduct our study. To answer our four research questions in this chapter, we also need to calculate reused UI elements set (RES) to measure UI reuse and their characteristics. In the two following sections, we describe how we calculate RES and their different characteristics.

Extracting RES

A reused element set (RES) specifies how certain UI elements should be used together. As mentioned in chapter 1.1.3, for a standard activity (*i.e.,* a page that a user can see) there should exists two files: i) the source code of the activity which the path

is indicated in the AndroidManifest.xml (manifest) file, and ii) the corresponding UI XML file (*i.e.,* the UI of a page). This XML file contains the UI elements of the activity. In other words, each application consists of several activities, and each of them is linked to a UI XML file which contains UI elements for the corresponding activity.

We define RES (*i.e.,* Reused Elements Set) as a set of UI elements that are occurred at least more than one UI XML files, and it contains at least three UI elements. In this context, RES are recommended as meaningful and reusable solutions for building mobile applications. As a result, the minimum number of UI elements in a RES should be at least three to be meaningful and reusable for the developers. For each category of mobile applications, we extract RES by means of Frequent Item-set Mining (FIM) algorithm. FIM is proposed by Agrawal et al. [46]. FIM aims at finding regularities in a dataset. Given a UI XML file (as shown in Figure 5.2), FIM algorithm tries to extract all the RES whose frequency are more than a predefined threshold (*i.e.,* support). In this study, we consider a set of UI elements as RES (*i.e.,* Reused Elements Set), if they appear at least once in another UI XML file, and at least have 3 UI elements. As shown in Figure 5.2, for the UI XML file, UI elements marked in red (*i.e.,* LinearLayout, AutoCompleteTextView, EditText, and Button) can be considered RES since they have appeared in other UI XML files, and they are more than 3 UI elements.

However, FIM techniques unavoidably generate an exponential number of subpatterns. To solve the shortcoming, a solution is proposed by Pasquier et al. [47] instead of mining the complete set of frequent itemsets and their associations, association mining only needs to find frequent closed itemsets and their corresponding rules.

Table 5.1: Summary of the characteristics of different categories

Category	# Applications	# Activities	#RES
Shopping	193	2,822	6,622
Health	286	4,129	5,571
Transportation	128	1,078	2,341
News	114	1,302	2,212
Weather	244	1,608	1,968
Travel	106	1,711	2,843
Finance	103	1,167	2,792
Social	118	1,107	3,404

In this context, an itemset (*i.e.,* a RES) is closed if none of its immediate supersets has the same support (*i.e.,* number of UI elements in the RES) as the itemset. In this study, we consider the frequent closed itemsets instead of mining the complete set extracted by FIM techniques. An important implication is that mining frequent closed itemsets has the same power as mining the complete set of frequent itemsets, but it will reduce the redundant itemsets to be generated and it will increase both efficiency and effectiveness of mining [48].

Table 5.1 shows the reused UI elements set (RES) extracted from different categories in this study.

Calculating Metrics

We extract RES from each UI XML file of an application using the approach mentioned in section 5.3.1. To study different characteristics of the RES, and how they are distributed among different UI XML files, we compute three different metrics. Using these three metrics (*i.e.,* RDM, DRM, and RLM) we investigate whether there exists a significant difference in the characteristics of RES that are related to UI XML files of high and low user-perceived quality.

1. Reused elements set Distribution Metric (RDM)

Let A^i, $i \in \{1, 2, ..., n\}$ be the applications in a category (*e.g., Shopping*) of the Android Market, and let R^i, $i \in \{1, 2, ..., n\}$ be the list of RES (*i.e.,* Reused Elements Set) extracted among UI XML files for the applications in a category, *i.e.,* R^1 being the first extracted RES and R^n being the last one.

To count the number of UI XML files that are using a RES (*e.g.,* R^i), we define $NX(R^i)$. To count the total number of applications that are using the R^i, we define $NA(R^i)$.

We define the Reused elements set Distribution Metric (RDM) to capture the distribution of a set of reused UI elements among different applications withing the same category. To this end, we use Equation (5.1).

$$RDM(R^i) = \frac{NA(R^i)}{NX(R^i)}, \tag{5.1}$$

Where R^i is the i^{th} RES among all the RES found from UI XML files of the applications in a category.

Given a reused element set, RDM measures two things: (i) How many applications are using the reused element sets, and (ii) How many UI XML files use the reused element set in their structure. To illustrate RDM let's consider a reused element set (e.g., R^i), where i is the i^{th} RES among the RES found among the applications in a category. Let's assume that the number of times that R^i has occurred in different UI XML files of all applications in a category (*i.e.,* $NX(R^i)$) is 100. Moreover, let's assume that R^i has been distributed between 5 different applications. The value of $RDM(R^i)$ is 0.05. That is to say, the more a reused element set is distributed

among different applications, the higher RDM is. The more a reused element set is distributed among different UI XML files, the less RDM is.

2. Developer-customized Reused element set Metric (DRM)

In Android platforms, developers can create customized UI components and define their own UI elements [19]. To capture the number of customized UI elements by developers in a RES (*e.g.,* R^i), we define Developer-customized Reused element set Metric (DRM) for R^i following Equation (5.2).

$$DRM(R^i) = DUE(R^i), \tag{5.2}$$

Where $DUE(R^i)$ *is the number of Developer-customized UI elements in* R^i*.*

To detect developer-customized UI elements, we use two heuristics. First, we manually built a dataset consisting of prebuilt UI elements. Therefore, if a new UI element is not in this list, it would be recognized as a developer-customized UI element. Second, developer-customized tags have different naming convention in comparison with prebuilt UI elements. For example, "$com.ebay.android.widget.ExpandingImageView$" is a developer-customized UI element. Therefore, for a new UI element, we examine its naming format, and see whether they have different naming format in comparison to prebuilt UI elements.

Given a reused element set (e.g., R^i), to extract the number of Developer-customized UI elements we use $DUE(R^i)$ which indicates the total number of UI elements that are not provided by default in the android SDK, and defined by developers.

3. Reused elements set Length Metric (RLM)

Our last metric is the Reused elements set Length Metric (RLM), which captures the number of UI elements that are used in a reused element set. RLM is computed following Equation (5.3).

$$RLM(R^i) = RL(R^i), \tag{5.3}$$

Where $RL(R^i)$ is the number of UI elements in R^i.

Using these three metrics (*i.e.,* RDM, DRM, and RLM) we investigate whether there exists a significant difference in the characteristics of RES that are related to UI XML files of high and low user-perceived quality.

5.3.2 Analysis Methods

We investigate the extent of usage of RES. Moreover, we see what characteristics of RES can result in high user-perceived quality. Moreover, through a manual analysis, we aim to recommend templates from the extracted RES which can be used as standard templates for designing UI by developers. We use Wilcoxon rank sum test [42] to compare the distribution of different characteristics of RES between applications with low and high user-perceived quality.

5.4 Study Results

This section presents and discusses the results of our four research questions.

5.4.1 RQ5.1: To what extent UI elements set are reused?

Motivation.

According to Android guidelines [19], to develop a standard android application, developers should separate the UI design of the corresponding application from its logic. Prior research has shown that mobile applications reuse the classes in source code to a great extent which implies that the logic of mobile applications is widely reused [25]. However, no one has studied the reuse of the UI of mobile applications. Studying reused UI elements set (RES) has great value in mobile applications since they may convey good practices that are used frequently.

Approach.

In this research question, we want to analyze the extent of reuse present in the UI of mobile applications in the Android Market within each category. Moreover, we highlight whether different categories reuse RES from each other or not (*i.e.,* across categories). Therefore, in this research question we want to determine:

RQ5.1.a) What is the percentage of reuse of RES in Android Market within each category?

RQ5.1.b) Do RES occur across different categories?

For each category, we extract reused UI elements set (RES) based on the method described in Section 5.3.1. To measure the extent of usage of RES in Android mobile applications, we use similar metrics which were used in previous empirical studies in the status of code cloning in software systems [49], [50].

To answer RQ5.1.a, we use the two following metrics to measure the usage of RES in android applications for each category.

Percentage of UI Elements Reused (PER): The idea behind this metric is that what percentage of a UI XML file can be constructed from its corresponding RES. It is important to mention that one UI XML file can contain different RES since different parts of a UI XML file can be repeated in other UI XML files. As a result, this metric would be the union of the corresponding RES for a UI XML file divided by the total number of UI elements in it. For example, let's consider the UI XML file shown in Figure 5.2 which is constructed from 6 UI elements. It can be seen that this UI XML file has a RES consisting of four UI elements. As a result, PER for this UI XML file is 0.66, meaning that 66% of this UI XML file is constructed from RES that are occurred in other UI XML files. In each category, we calculate PER for each UI XML file. Then, we use the average of all calculated PER for each UI XML file to calculate PER for a category.

Files Associated with RES (FAR): While the above metric give the overall reusing statistic of RES for a subject category, it cannot tell us whether the RES are from some specific UI XML files, or scattered among many UI XML files all over a category. FAR provides these statistics for each category. We consider that a UI XML file is associated with RES if it has at least one reused UI element set that is used in at least one other UI XML file. For calculating FAR in a category, we divide the number of UI XML files that use at least one common RES by the total number of UI XML files in the category. For example, when FAR of a category x is 0.5, it means that 50% of the UI XML files in x use at least one RES.

We begin by looking at the overall reusing level of RES for mobile applications in Android Market. By using the two aforementioned metrics (*i.e.,* PER, FAR).

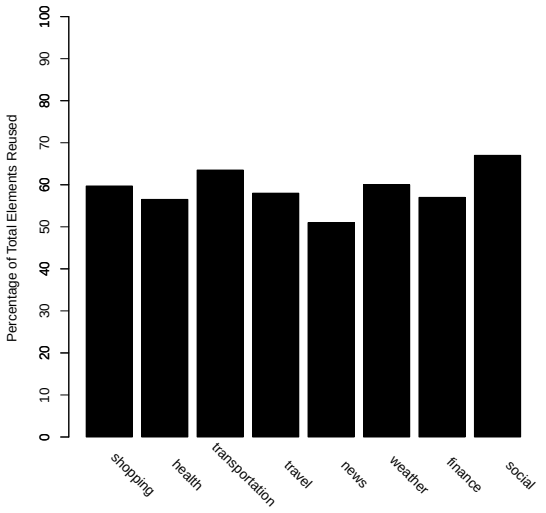

Figure 5.3: Percentage of Total Elements Reused in each Category

In RQ5.1.b, we are interested to know whether RES happen only withing categories or they are also available across categories. If yes, we can more generalize our conclusions that can be find in each category. To measure the similarity of the RES between two categories, we use the Jaccard Similarity Coefficient (JSC) [51] which is used for comparing the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets. JSC is calculated following Equation (5.4).

$$JSC(P1, P2) = \frac{|P1 \cap P2|}{|P1 \cup P2|}, \tag{5.4}$$

Where P1 and P2 are the sets of all RES extracted by the method mentioned in Section 5.3.1 from two categories (e.g., Shopping and Health).

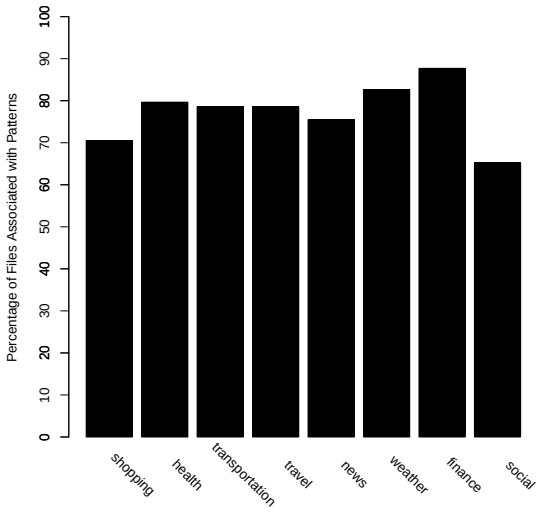

Figure 5.4: Percentage of Files Associated with RES in each Category

Findings.

Figure. 5.3 and Figure. 5.4 show the Percentage of Elements Reused (PER) and the percentage of Files Associated with RES (FAR) for each category. Our results suggest that on average 60% of a UI XML file can be constructed by similar RES that are repeated in other UI XML files (See Figure. 5.3). This high percentage of reuse originates from the fact that most UI XML files use some standard UI elements (*e.g.,* RelativeLayout, FrameLayout, and LinearLayout). Moreover, we found that on average 77% of the UI XML files in a category use at least one RES from other UI XML files in the corresponding category (See Figure. 5.4). This high percentage of reuse indicates that very few UI XML files are unique in the sense that they do not use any UI elements similar to other UI XML files. We observe that the Weather and Finance categories have the highest FAR. The reason is that for instance in the Weather category there exists several applications that have exactly the same UI, but

Table 5.2: Jaccard similarity measure for RES reused across different categories.

Categories	Shopping	Health	Transportation	Travel	News	Weather	Finance	Social
Shopping	100%							
Health	30.92%	100%						
Transportation	23.48%	23.16%	100%					
Travel	22.71%	23.16%	26.71%	100%				
News	21.22%	21.18%	24.73%	26.92%	100%			
Weather	25.51%	25.45%	25.16%	26.71%	24.73%	100%		
Finance	24.89%	23.79%	25.90%	24.55%	25.19%	25.90%	100%	
Social	25.84%	33.61%	24.91%	23.53%	26.82%	24.91%	23.66%	100%

have been made for different countries, and in different languages (e.g., the Houston and WLOX weather applications). As a result, FAR is higher in this category.

Table 5.2. reports the proportion of RES that are reused across different categories. Each cell of this category is the calculated JSC (see Equation 5.4) for the RES extracted from the corresponding two categories indicated by corresponding column and row (e.g., Shopping and Health). As shown in Table 5.2. there exists on average around 23% Jaccard similarity in the RES across different categories which means that similar patterns are reused across different categories.

RES are widely used within and across categories.

5.4.2 RQ5.2: Does reused UI elements set impact user-perceived quality within and across categories?

Motivation.

In **RQ5.1**, we show that RES widely occur within and across categories. Therefore, it seems that when the developers want to create a new mobile application in android platform for the first time, they prefer to reuse the existing solutions or RES. In

this research question, we want to show that whether exploiting previously used UI elements (RES) can provide a better user-perceived quality or not.

Approach.

In this research question we want to determine:

RQ5.2.a) Are RES from other categories can provide better user-perceived quality in a category?

RQ5.2.b) What kind of characteristics of a RES can provide high user-perceived quality within a category?

As mentioned in section 5.3.1, we extract RES from the UI XML files within all the applications in each category. Each application has a user-perceived quality ($UPQ(A)$) calculated by Equation (3.1), and it consists of several UI XML files. To compute the user-perceived quality for each UI XML file, we assign each UI XML file the user-perceived quality obtained from the application that those UI XML files belong to. As a result, all the UI XML files from the same application acquire the same user-perceived quality. In this research question, we focus on RES, and measure whether using a reused element set can provide good user-perceived quality or not. Therefore, we need to map each RES with a user-perceived quality. Each RES can be related to multiple UI XML files. To assign the calculated user-perceived quality for each UI XML file to each RES, we use the average of the calculated UPQ (See Equation 3.1) of the corresponding UI XML files.

To answer RQ5.2.a, we find the RES that are reused between each two (*i.e.,* source category and destination category) categories among all the 8 categories. In this question, we define the reused RES as the reused UI element sets that are shared

between Shopping (source) and Health (destination) categories, and we define the none reused RES as the UI element sets that are not reused in the destination (*i.e.,* Health) category.

We test the following null hypothesis for each source category and the other destination category:

H_0^1: *There is no difference in the user-perceived quality of the reused RES and none reused RES.*

We perform a Wilcoxon rank sum test [42] to evaluate H_0^1. To control family-wise errors, we apply Bonferroni correction which adjusts the threshold p-value by dividing the number of tests. The number of tests is 56 since we have 8 different categories as source categories and 8 different categories as destination categories. Moreover, we do not evaluate H_0^1 if the source and destination categories are the same. There exists a statistically significant difference, if p-value is less than 0.05/56=0.0007.

In RQ5.2.b, we are interested to know whether certain specifications of RES (see Section 5.3.1) can provide better user-perceived quality or not. To this end, we sort the RES based on their user-perceived quality for each category. Then, we break the data into four equal parts, and name the ones in the highest quartile, RES that are related to UI XML files with high user-perceived quality, and the ones in the lowest quartile, the RES that have low user-perceived quality. Finally, we observe whether there exists any difference in the computed metrics (*i.e.,* quantifiers of certain characteristics of RES) between RES that are related to UI XML files of high and low user-perceived quality. We test the following for each metric (*i.e.,* RDM, DRM, and RLM) in each category.

H_0^2: *there is no significant difference in certain specifications of RES between the*

ones related to UI XML files of high and low user-perceived quality.

We perform a Wilcoxon rank sum test [42] to evaluate H_0^2. Again, to control family-wise errors, we apply Bonferroni correction which adjusts the threshold p-value by dividing the number of tests. The number of tests is 24 since we have 8 different categories and three different metrics (*i.e.,* RDM, DRM, and RLM) to quantify different characteristics of RES. There exists a statistically significant difference, if p-value is less than 0.05/24=0.002.

Table 5.3: Difference of the user-perceived quality for the RES that are reused within each two pair of categories.

Source \ Destination	Shopping	Health	Transportation	Travel	News	Weather	Finance	Social
Shopping	-	↘ *	↘ ○	↘ ○	↘	↗	↘ *	↘ *
Health	↗ *	-	↗ *	↗ *	↗ *	↘	↗ *	↘
Transportation	↗	↘	-	↗ *	↗	↗ *	↘	↘
Travel	↗	↗	↘	-	↘ *	↘ *	↘ *	↘ *
News	↘ *	↘	↘ *	↘ *	-	↘ *	↘ *	↘
Weather	↘	↗ *	↘	↗ +	↗	-	↘	↗
Finance	↗ *	↗	↗+	↗	↗+	↗	-	↗+
Social	↗	↗	↗+	↘	↘+	↗ +	↗	-

Given source and destination categories, ↘ means that the RES that are shared between source and destination categories have less user-perceived quality than the RES that are not shared between these two categories, and it is vice-versa for ↗. ($p<0.0007/50^{\star}$; $p<0.0007/5^{\circ}$; $p<0.0007^{+}$)

Findings.

Each cell of Table 5.3 provides two types of information for the RES that are reused among each pair of categories (*i.e.,* the source and destination). Table 5.3 shows that whether there exists a significant difference in the user-perceived quality between the RES that are shared among the source (*i.e.,* Shopping) and the destination (*i.e.,* Health) categories, and the RES that are not shared with the destination (*i.e.,* Health) category in the source (*i.e.,* Shopping) category. For instance, let's consider the cell

in the first row for the second column of Table 5.3 in which the source category is Shopping, and the destination category is Health. In this cell, first, there is a "↗" or "↘" sign which implies whether the difference between the average user-perceived quality of the reused RES (*i.e.,* the reused UI element sets that are shared between Shopping (source) and Health (destination) categories), and the average user-perceived quality of the none reused RES (*i.e.,* the UI element sets that are not reused in the destination (*i.e.,* Health) category) is positive or negative. In this example, it is negative ("↘") which means that reused RES (*i.e.,* the UI element sets related to the ones that are shared between the Shopping and Health category) tend to have less user-perceived quality in average than the none reused RES ones (*i.e.,* the UI elements set that are not reused between the Shopping and Health categories). Moreover, we report whether the difference in the distribution of user-perceived quality for the RES that are used within the source and destination categories, and the ones that are not used in the destination category from the source category is statistically significant or not. In this example, the difference is statistically significant ($\star$).

As it can be seen from Table 5.3 we can reject H_0^1, and conclude that using the RES that are used in other categories can provide a significant difference within categories in certain cases. For example, if a developer wants to design an application in the Transportation category and uses the RES that are shared between Travel and Transportation categories may provide significant difference in the user-perceived quality. However, this difference can be negative or positive depending on the source and destination categories. Here, it is a positive meaning that the RES that are used in these two categories may provide higher user-perceived quality in the Transportation category.

RES that are reused across categories can provide a significant difference in the user-perceived quality. However, this behavior is category dependent.

Table 5.4: Difference in the usage of different characteristics of RES within various categories.

	RDM	DRM	RLM
Shopping	↘ *	↗ *	↗ *
Health	↘ *	↗ ∘	↘
Transportation	↘ ∘	↘	↘
News	↘ *	↘	↗ +
Weather	↘ *	↗	↘
Travel	↗ *	↗ *	↗ ∘
Finance	↗	↗ *	↘
Social	↘ *	↗ *	↗ *

Given a metric that quantifies the characteristics of RES, ↘ means that the distribution of such metric is lower in RES with high user-perceived quality compared to the ones with low user-perceived quality, and it is vice-versa for ↗.
($p<0.002/50^{\star}$; $p<0.002/5^{\circ}$; $p<0.002^{+}$)

Table 5.4 shows whether there exists any difference in the certain characteristics of RES that are related to UI XML files of high and low user-perceived quality. For example, let's consider the first cell for Shopping category in Table 5.4. Similar to Table 5.3, there is a "↗" or "↘" sign which shows that whether the average difference in the distribution of the corresponding metric (*i.e.,* here it is RDM) between the RES related to high and low user-perceived quality is positive or negative. In this example, it is negative ("↘") which means that RES with high user-perceived quality tend to have lower Reused elements set Distribution Metric (RDM) than the ones with low user-perceived quality. Moreover, we report whether the difference for the corresponding metric (*i.e.,* RDM) is statistically significant between RES related to UI XML files of high and low quality. In this example, the difference is statistically significant ($\star$).

As shown in Table 5.4, we can reject H_0^2, and conclude that in most categories RES

with high user-perceived quality tend to have lower Reused elements set Distribution Metric (RDM). In other words, RES with high user-perceived quality tend to be reused in fewer applications or to be reused in more UI XML files than the RES related to UI XML files with low user-perceived quality. Moreover, we can observe that RES with high user perceived quality tend to have higher Developer-customized Reused element set Metric (DRM) which means that they tend to have more developer-customized UI elements by the developers. As a result, being more unique may provide better user-perceived quality for a UI design.

RES with high user-perceived quality tend to be distributed in fewer applications. Moreover, using developer-customized UI elements may provide high user-perceived quality.

5.4.3 RQ5.3: Does reused UI elements set have an impact on the user-perceived quality of the functionalities in mobile applications?

Motivation.

As shown in **RQ5.2**, RES from certain categories, with certain specifications can improve user-perceived quality in general. However, this is a general conclusion, and it does not provide a practical guideline for developers. Each mobile application is designed with a purpose to fulfill some certain functionalities for its users. In this research question, we go into more details, and study whether there are certain characteristics (*i.e.,* see Section 5.3.1) of RES that result in high or low user-perceived quality in each functionality. If yes, we can highlight the characteristics of RES that can lead to high user-perceived quality in each functionality with a lower level of granularity.

Approach.

As mentioned in section 5.3.1, we extract the frequently used RES from the UI XML files in each category. Next, to compare the characteristics of RES of different UI XML files. We label each UI XML file with a fine-grained functionality, we use LDA [43] which clusters the UI XML files of each activity (documents) based on their functionalities (*i.e.,* topics). In other words, for each UI XML file, we extract all the strings and labels shown to the users (see Section 3.1.3). We apply LDA to all the UI XML files of activities retrieved from the existing applications in a category to extract their corresponding functionalities.

Since mobile applications perform a limited number of functionalities, the number of topics (*i.e.,* K) should be small in our research context. As we are interested in the major functionalities of applications, we empirically found that $K = 9$ is a proper number for our dataset by manual labeling and analysis of randomly selected mobile applications. We use MALLET [44] as our LDA implementation, which uses Gibbs sampling to approximate the distribution of topics and words. We run the algorithm with 1000 sampling iterations. The number of sampling iterations should be a trade off between the time taken to complete sampling and the quality of the topic model. In this study, we manually found that 1000 (*i.e.,* default value) is a good choice for the number of sampling iterations. Moreover, we use the parameter optimization in the tool to optimize α and β. In our corpus, for each category, we have n UI XML files (extracted from the applications in the corresponding category) $\chi = \{x_1, ..., x_n\}$, and we name the set of our topics (*i.e.,* functionalities) F = $\{f_1, ..., f_K\}$. It is important to mention that these functionalities are different in each category, but the number of them is the same ($K = 9$). For instance, f3 in the Shopping category is about

"*Login*" and "*Sign in*" functionality. However, in the Health category, it is about "*search*" and "*information seeking*" functionality. LDA automatically discovers a set of functionalities (*i.e.,* F), as well as the mapping (*i.e.,* θ) between functionalities and UI XML files. We use the notation θ_{ij} to describe the topic membership value of functionality f_i in UI XML file x_j.

Each application is consisted of several UI XML files ($\{x_1, x_2, ..., x_n\}$), and it has a user-perceived quality ($UPQ(A)$) calculated by Equation (3.1). To compute the user-perceived quality for each UI XML file, we assign each UI XML file the user-perceived quality obtained from the application that those UI XML files belong to. As a result, all the UI XML files from the same application acquire the same user-perceived quality. However, by applying LDA [43] each UI XML file (document) acquires a weight of relevance to each functionality (*i.e.,* θ). Therefore, the user-perceived quality for a UI XML file can originate from two sources: i) the user-perceived quality of its corresponding application, and ii) the probability that the UI XML file belongs to a functionality. Moreover, we use a cut-off threshold for θ (*i.e.,* 0.1) that determines if the relatedness of a UI XML file (document) to a functionality is important or not. A similar decision has been made by Chen et al. [45]. Therefore, we calculate the user-perceived quality for each UI XML file (x_j) in a functionality as the following:

$$AUPQ(x_j) = \theta_{ij} * UPQ(x_j), \tag{5.5}$$

Where $AUPQ(x_j)$ reflects the user-perceived quality for UI XML file j (i.e., x_j); θ_{ij} is the generated probability by LDA that UI XML file j (x_j) is related to functionality i (f_i); $UPQ(x_j)$ is the user-perceived quality of the application which x_j belongs to it.

Each RES can be related to multiple UI XML files. To assign the calculated

user-perceived quality for each UI XML file to each RES, we use the average of the calculated AUPQ (see Equation (5.5)) of the UI XML files that are related to the RES.

To answer this research question, we sort the RES based on their user-perceived quality for each functionality in each category. Then, we break the data into four equal parts, and named the ones in the highest quartile, RES that are related to UI XML files with high user-perceived quality, and the ones in the lowest quartile, RES that are related to UI XML files with low user-perceived quality. Finally, we investigate whether there exists any difference in the distribution of the characteristics of RES (*i.e.,* see Section xx) between RES that are related to UI XML files of high and low user-perceived quality for each functionality in each category. To this end, we test the following null hypothesis for each metric (explain we mean characteristic metrics) in each category for each functionality:

H_0^3*: there is no difference in the defined characteristics between RES of UI XML files with low and high user-perceived quality.*

We perform a Wilcoxon rank sum test [42] to evaluate H_0^3. To control family-wise errors, we apply Bonferroni correction which adjusts the threshold p-value by dividing the number of tests. The number of tests is 216 since we have 8 different categories and 3 different calculated RES characteristics among 9 different functionalities. There exists a statistically significant difference, if p-value is less than 0.05/216=0.0002.

Findings.

Table 5.5 shows our findings for calculated metrics on RES, *i.e.,* Reused elements set Distribution Metric (RDM), Developer-customized Reused element set Metric

Table 5.5: Difference of usage of activity level UI metrics between the activities with low and high user-perceived quality for each functionality in each category.

		f1	f2	f3	f4	f5	f6	f7	f8	f9
Shopping	RDM	↘⋆	↘⋆	↘⋆	↘⋆	↘⋆	↘⋆	↘⋆	↘⋆	↘⋆
	DRM	↗⋆	↗	↗⋆	↗	↗⋆	↗⋆	↗+	↘⋆	↘
	RLM	↘	↘⋆	↘	↘⋆	↘+	↘	↘	↘⋆	↘⋆
Health	RDM	↘⋆	↘⋆	↘⋆	↘⋆	↘°	↘⋆	↘⋆	↘⋆	↗
	DRM	↘⋆	↘°	↘°	↘	↘⋆	↗°	↘	↘	↘
	RLM	↘⋆	↘	↗⋆	↗	↗	↘⋆	↗	↘⋆	↘
Transportation	RDM	↘⋆	↘⋆	↘⋆	↘⋆	↘⋆	↘+	↘+	↘⋆	↗
	DRM	↘°	↗⋆	↘	↘°	↗°	↘	↗	↘⋆	↘⋆
	RLM	↘	↗	↘	↘⋆	↗	↘	↗	↗⋆	↗
News	RDM	↘⋆	↘⋆	↘⋆	↘°	↗+	↗	↘⋆	↘⋆	↗
	DRM	↘⋆	↗	↗	↗⋆	↘	↘⋆	↘	↘	↘
	RLM	↗°	↗	↗	↗⋆	↗	↘+	↗	↗	↘
Travel	RDM	↘	↘⋆	↗	↘⋆	↘⋆	↘⋆	↘⋆	↘⋆	↘⋆
	DRM	↘	↗⋆	↗	↘	↗	↗	↘⋆	↘+	↘
	RLM	↘⋆	↗⋆	↘	↗	↘	↗	↘	↘	↗
Weather	RDM	↗	↘⋆	↗	↘	↘⋆	↘	↘	↘+	↗⋆
	DRM	↗⋆	↗	↘	↗	↗°	↗	↗	↗	↘
	RLM	↗	↘	↘	↗+	↗	↗	↘	↗	↘⋆
Finance	RDM	↘⋆	↘⋆	↘⋆	↘⋆	↘+	↗	↘⋆	↗	↗⋆
	DRM	↘⋆	↗	↗⋆	↗⋆	↗+	↗	↗	↘⋆	↗⋆
	RLM	↘°	↗+	↘	↗	↗	↘°	↘	↗⋆	↗
Social	RDM	↘⋆	↘⋆	↘⋆	↗	↘°	↘⋆	↘+	↗	↗°
	DRM	↘	↘⋆	↗⋆	↗°	↗	↗⋆	↘⋆	↘	↗
	RLM	↗	↘⋆	↗	↘	↗	↘	↘⋆	↘	↘

Given a metric that quantifies the characteristics of RES, ↘ means that the distribution of such metrics in RES with high user-perceived quality is less than the ones in low user-perceived quality, and it is vice-versa for ↗.
($p<0.0002/50^{\star}$; $p<0.0002/5^{\circ}$; $p<0.0002^{+}$)

(DRM), Reused elements set Length Metric (RLM) for each functionality in each category. For each cell of Table 5.5, we report two pieces of information. Let's consider the cell related to the Shopping category for the third functionality (*i.e.,* f3) which refers to "*Login*" and "*Sign in*" functionalities, for the RDM (*i.e.,* Reused elements set Distribution Metric). In this cell, there is a "↗" or "↘" sign which implies whether the difference for the corresponding metric (*i.e.,* RDM) between the

RES related to UI XML file of activities with low and high user-perceived quality is positive or negative. In this example, it is negative ("↘") which means that RES related to the UI XML file for "*Login*" and "*Sign in*" functionality (f3) in the Shopping category with low user-perceived quality have more complexity for RDM than the ones with high user-perceived quality. Moreover, we report whether the difference for the corresponding metric (*i.e.,* RDM) is statistically significant between UI design of UI XML files with low quality activities and high quality ones. In this example, the difference is statistically significant ($\star$).

As it can be seen from Table 5.5, we can reject H_0^3, and conclude that there exists a significant difference in certain characteristics of RES of UI XML files with low and high user-perceived quality. For example, in most cases this difference is a negative number ("↘") for RDM meaning that RES of UI XML files with low user-perceived quality tend to have less RES distribution than the high quality ones. In other words, better RES tend to be used in fewer applications or to be used in more UI XML files than the RES related to UI XML files with low user-perceived quality. Moreover, as demonstrated in Table 5.5 using developer-customized tags can have an impact on the user-perceived quality. For example, in the shopping category for "*Login*" and "*Sign in*" functionality (f3), if developers use developer-customized UI elements, they may end up having UI XML files with high user-perceived quality. Our guidelines can be exploited by developers to use the proper RES to have functionalities with high user-perceived quality.

In almost every functionality, to achieve high user-perceived quality, it is better to use the RES that are reused in a few number of applications, or used in more UI XML files. Moreover, for some certain functionalities it is better to use developer-customized UI elements to achieve high user-perceived quality.

5.4.4 RQ5.4: Can we extract UI templates from reused UI elements sets of high user-perceived quality?

Motivation.

In previous research questions, of the Android applications we analyzed, we show that RES are used widely within and across categories. Moreover, we recommend what characteristics of RES may lead to high user-perceived quality in each functionality. As a result, following these major characteristics in using a RES can help developers to have UI designs with high user-perceived quality.

In this research question, to provide more detailed and practical guidelines for the developers, we try to go beyond the characteristics of RES, and recommend UI templates of high user-perceived quality for a limited number of certain functionalities in the Shopping category. UI templates are standard and reusable solutions for building the UI of mobile applications, and they are underlying in RES. Therefore, there need to be an extra manual analysis to extract UI templates from RES. For instance, when a developer wants to design a *login* page for her application in the *Shopping* category, based on our guidelines, she knows what characteristics of a reused UI element set can help her to achieve high user-perceived quality (answered in **RQ5.3**). In addition, she needs to know which meaningful combination of UI elements (*i.e.,* UI template) can provide her high user-perceived quality in a certain (*e.g.,* login) functionality (the

aim of **RQ5.4**).

Approach.

RES encompass UI elements that are frequently used in UI XML files. However, not all the extracted RES can be considered as standard UI templates (*i.e.,* UI templates) that can be recommended to the developers. Therefore, through a manual analysis step, we extract the UI templates from the RES since it cannot be done automatically. To extract UI templates for certain functionalities, we need to focus on a more fine-grained functionality than the ones extracted in **RQ5.3**. In our previous research question, we discussed that to further analyze the characteristics of RES , we need to cluster the UI XML files since they encompass a variety of different functionalities. To this end, we used LDA [43] as our approach to extract high-level functionalities. However, each functionality that is produced by the LDA is a high-level functionality, and it contains a variety of sub-functionalities. For example, for the third functionality extracted in the Shopping category (*i.e.,* f3), Table 5.6 shows the words that describe (f3). As shown in Table 5.6, UI XML files that are related to f3 are about a high level functionality (*i.e.,* sign-in and login). However, it also contains some other sub-functionalities (*e.g.,* sharing a page). As a result, recommending a good UI template for a high-level functionality (*e.g.,* f3) does not make any sense since it may contain other sub-functionalities, and each sub-functionality should have its own UI template. For each high-level functionality, through a manual analysis we scan the name of the UI XML files associated with the high user-perceived quality RES, and extract more fine-grained functionalities (*i.e.,* sub-functionalities). Then, we observe whether the RES is eligible to be introduced as a UI template or not.

Table 5.6: List of related words for Login/Sign in functionality in the Shopping category

Functionality	Related Words
Login/Sign in	email, password, account, sign, login, share, facebook, save, deals register, address, find, home, send, enter, create, cancel, friends, store

To demonstrate the usefulness of our results, we did the manual analysis for Shopping category. For each high-level functionality extracted from **RQ5.3**, we extract the RES with high user-perceived quality. Then, we manually analyze the name of the associate UI XML files to the extracted RES, and identify which sub-functionality can be extracted using the extracted RES, and we further analyze the RES to study the candidates for UI templates.

Findings.

As demonstrated in Table 5.7, we manually extracted a limited set of lower level functionalities and their corresponding high user-perceived quality templates from the *Shopping* category. We also report the frequency of occurrence of each UI template among UI XML files. Moreover, in Appendix A, we provide screen-shots from the pages of mobile applications that are using the UI templates mentioned in Table 5.7. These UI templates can be exploited by the developers to design UI XML files of high user-perceived quality. For each sub-functionality, we provided a set of UI elements (*i.e.,* a UI template) that have been used frequently within the UI XML files of its corresponding sub-functionality. For example, for the *Sign in* sub-functionality from Table 5.7, we have recommended a UI template that has been used frequently (*i.e.,* 43 times) by high user-perceived quality UI XML files in this sub-functionality. A sample part of this UI template is shown in Figure 5.2 and Figure A.3. This UI

Table 5.7: UI templates for different sub-functionalities

Sub-functionality	UI Template	Frequency of Occurrence
User agreement	ScrollView, WebView, FrameLayout, TextView, LinearLayout,ProgressBar	21
Privacy Policy	View, ScrollView, Button, TextView, RelativeLayout, LinearLayout	26
Sign in	AutoCompleteTextView, CheckBox, RelativeLayout, LinearLayout, TextView, ScrollView	43
Suggest page	TabHost, ViewAnimator, ListView, Com, TextView, LinearLayout, ProgressBar, Button	16
Account Profile	TableRow, Com, ImageView, Button, TableLayout, LinearLayout, TextView	10
Date Time	DatePicker, TimePicker, ScrollView, ImageView, Button, RelativeLayout, TextView, LinearLayout	29
Photo preview	Gallery, Button, RelativeLayout, TextView, LinearLayout, ImageView	31
Configure photo	RequestFocus, EditText, FrameLayout, Scrollview, ImageView, Button, RelativeLayout, TextView, LinearLayout	27
Address entry	Br, LinearLayout, TextView, RelativeLayout, Button, ScrollView, EditText, Spinner	19
Shopping Cart	ImageButton, View, ProgressBar, Com, EditText, RelativeLayout, TextView, LinearLayout, ScrollView, Button, ImageView	12
Search	RequestFocus, View, ProgressBar, ListView, ScrollView, ImageView, RelativeLayout, TextView, LinearLayout, Com	33
Order detail	TableRow, ListView, ScrollView, TextView, TableLayout, Button, RelativeLayout, ImageView	16
Vote / Rate	RatingBar, Com, LinearLayout	37
Settings	View, CheckBox, LinearLayout, TextView, ProgressBar, EditText, Button	25

template includes an *AutoCompleteTextView* and a *TextView* elements for entering the username and password of the users. The *TextView* element displays text to the user and optionally allows them to edit it, and the *AutoCompleteTextView* is an editable *TextView* that shows completion suggestions automatically while the user is typing. This UI template shows that UI XML files of high user-perceived quality related to *Sign in* functionality use an *AutoCompleteTextView* element instead of a simple *TextView* element possibly for having an extra feature for auto completing the username. This UI template also contains a *CheckBox* element possibly for remembering the username or/and password. Moreover, it contains an *ScrollView* element to make the page scrollable in case the page does not fit in the screen of a smartphone, and also a *Button* element for submitting the username and password for the sign in procedure. Finally, this UI template contains two page formatting elements, a *LinearLayout* and a *RelativeLayout* elements. The *LinearLayout* arranges its elements in a single column or a single row. The *RelativeLayout* allows for relative positioning of its elements in relation to each other or the parent. As it can be seen these recommended UI templates can give developers an idea that what UI elements should be used together to achieve high user-perceived quality in designing a UI XML file for a certain functionality. It is also important to mention that when there exists a developer-customized UI element in a UI template, we have used *com* to this end.

Through a manual analysis, we can successfully recommend UI templates related to UI XML files of high user-perceived quality for certain functionalities in the Shopping category.

5.5 Summary

We provided empirical evidence that UI of Android applications are widely reused within and across different categories. To measure UI reuse, we proposed various metrics based on the occurrence of reused UI elements set (RES). We performed a detailed case study using 1,292 free Android applications distributed in 8 categories. The highlights of our analysis include: (i) Reuse in the UI of mobile applications widely occurs among and across different categories (**RQ5.1**); (ii) certain characteristics of reused UI elements set can provide high user-perceived quality (**RQ5.2**, **RQ5.3**). For instance, if developers use developer-customized UI elements in some functionalities it can improve the user-perceived quality; and (iii) through a manual analysis approach we recommended reusable UI templates (i.e., UI templates) for developers (**RQ5.4**).

Chapter 6

Conclusion

In this Chapter, we describe the contributions of this thesis. We also discuss the threats to validity for our empirical studies and explore the future work.

6.1 Contributions

In this thesis, we identify the relation between UI design and user-perceived quality. We focus on two aspects of UI design,*i.e.,* UI complexity and UI reuse. Based on our two empirical studies, we make the following contributions:

- We explore the relation between UI complexity and user-perceived quality. We define various metrics based on the number of inputs and outputs in a page to quantify UI complexity, namely number of inputs (NI), number of outputs (NO), and number of elements (NE).

- We show that (i) our measurement approach of UI complexity correlates with the earlier user studies on UI complexity [22], [23]; (ii) the number of pages (activities) of mobile applications can have an impact on the user-perceived

quality of applications; and (iii) UI complexity has a strong relationship with user-perceived quality of the functionality of mobile applications.

- We study the extent of reuse in the UI of mobile applications. Moreover, we observe whether exploiting reused UI elements can improve user-perceived quality or not. To this end, we propose various metrics based on the characteristics of reused UI elements set (RES), namely Reused elements set Distribution Metric (RDM), Developer-customized Reused element set Metric (DRM), and Reused elements set Length Metric (RLM).

- We observe that (i) reuse in the UI of mobile applications widely occurs among and across different categories; (ii) certain characteristics of frequently reused UI elements can provide high user-perceived quality; and (iii) through a manual analysis approach we recommend reusable UI templates with high user-perceived quality for developers. Developers and quality assurance personnel can use our guidelines to improve the quality of mobile applications.

6.2 Threats to Validity

We now discuss the threats to validity of our study following common guidelines for empirical studies [52].

6.2.1 Construct Validity

In this thesis, it is mainly due to measurement errors. Szydlowski et al. discuss the challenges for dynamic analysis of iOS applications [53]. They mention that these challenges are mostly user interface driven. That is, most iOS applications

are making heavy use of event driven graphical user interfaces. Therefore, launching an application and executing it for a given span of time might not be sufficient to collect all execution paths in an application. Due to such challenges, we were not able to use dynamic analysis to reverse engineer the UI of mobile applications for a large scale study. Also a similar static analysis approach (see Section 3.1.2) has been used satisfactorily by Shirazi et al. in a similar study on the user interfaces of mobile applications [29]. Moreover, there are two ways to declare a UI layout for an activity in Android architecture: i) Declaring UI elements in an XML file and ii) Instantiating layout elements programmatically. In this study, our premise is towards the UI elements that are declared in a UI XML file since it is the recommended approach by Android guidelines [19].

6.2.2 Internal Validity

In this thesis, it is mainly due to our selection of subject systems, tools, and analysis method. The accuracy of apktool impacts our results since the extracted activity and XML files are provided by apktool.

The choice of the optimal number of topics in LDA is a difficult task [54]. However, through a manual analysis approach, we found that in all categories there exist at least 9 common functionalities.

We conduct our studies over 1,292 mobile applications in 8 different categories. We chose these categories since they contain both categories with high and low user-perceived quality, and they are from different domains. Also, we attempt to provide all the necessary details to replicate our study. In this thesis, we excluded an important category of mobile applications named Gaming. The reason is that Gaming

applications tend to use graphical engines and other techniques for building the UI. As a result, our approaches for extracting the UI of such applications are not mature enough to handle these type of applications.

To quantify UI complexity, we use the number of inputs and outputs in a page. However, there are other factors (*e.g.,* screen size, font size, and font color) that can be added to more accurately measure UI complexity.

To measure the difference in the distribution of various metrics (*e.g.* number of inputs) in applications with low and high user-perceived quality (see Table 4.6), we have only reported the difference between the average of these two groups (shown by "↗" or "↘" symbols). Although for most of these groups there exists a significant statistical difference in their distribution. We think other statistical analysis (*i.e.,* effect size analysis [42]) can be used to show how different these two distributions are. Through the coarse of our analysis, we observed that in most cases the distribution of these two groups (*i.e.,* applications related to low and high user-perceived quality) we can find significant p-values. However, the effect size for these two groups are not too different, and we could not see how big is the difference between these two distributions.

6.2.3 External Validity

We try to study several mobile applications (1,292) from different categories. Our study analyzes free available mobile applications in 8 different categories of the Android Market. To find out if our results apply to other mobile stores and mobile platforms, we need to perform additional studies on those environments. Also, in order to generalize our results, we require an analysis of a more extended number of

applications across more categories available in Android Market.

6.3 Future Work

In future work, we plan to replicate the studies in this thesis on more mobile applications from other categories existing on Android Market such as Business, Personalization, and Medical. Moreover, we should investigate whether our findings are consistent among other platforms such as iOS, BlackBerry, and Windows Phone. Moreover, we also plan to consider additional aspects of mobile applications such as GPS and Camera, and observe how their usage can affect the user-perceived quality of mobile applications.

Bibliography

[1] Seyyed Ehsan Salamati Taba, Iman Keivanloo, Ying Zou, Joanna Ng, and Tinny Ng. An exploratory study on the relation between user interface complexity and the perceived quality of android applications. In *International Conference on Web Engineering (ICWE), Late Breaking Result*, 2014.

[2] Amy K. Karlson, Brian R. Meyers, Andy Jacobs, Paul Johns, and Shaun K. Kane. Working Overtime: Patterns of Smartphone and PC Usage in the Day of an Information Worker. In *Proceedings of the 7th International Conference on Pervasive Computing*, Pervasive '09, pages 398–405, Berlin, Heidelberg, 2009. Springer-Verlag.

[3] Apple Application Store. `www.apple.com/iphone/from-the-app-store/`, July 2014.

[4] Google Application Store. `http://play.google.com`, July 2014.

[5] Windows Phone Store. `www.windowsphone.com/store`, July 2014.

[6] BlackBerry Application World. `http://appworld.blackberry.com/`, July 2014.

[7] Amazon Application Store. `www.amazon.com/appstore`, July 2014.

[8] Nokia Application Store. `http://store.ovi.com/`, July 2014.

[9] Samsung Application Store. www.windowsphone.com/store, July 2014.

[10] Sam Costello. How Many Apps Are in the iPhone App Store? http://ipod.about.com/od/iphonesoftwareterms/qt/apps-in-app-store.htm, July 2014.

[11] MG Siegler. There is a Great Future in iPhone Apps. http://venturebeat.com/2008/06/11/analyst-theres-a-great-future-in-iphone-apps/, July 2014.

[12] Gartner. Mobile App Store Downloads. http://www.gartner.com/newsroom/id/2153215, July 2014.

[13] Mark Wilcox. Two Important App Market Trends to Watch in 2013. http://www.developereconomics.com/two-important-app-market-trends-to-watch-in-2013/, July 2014.

[14] Gartner. Number of Downloads for Mobile Applications. http://www.gartner.com/newsroom/id/2592315, June 2014.

[15] Forbes. Application Store Forecasts. http://www:forbes:com/sites/louiscolumbus/2013/06/09/roundup-of-mobile-apps-app-store-forecasts-2013/, July 2014.

[16] Google. Android Applications Quality. http://developer.android.com/distribute/googleplay/quality/core.html, Feb 2014.

[17] Luca Chittaro. Distinctive aspects of mobile interaction and their implications for the design of multimodal interfaces. *Journal on Multimodal User Interfaces*, 3(3):157–165, 2010.

[18] Erik G. Nilsson. Design Patterns for User Interface for Mobile Applications. *Journal on Advances in Engineering Software*, 40(12):1318–1328, December 2009.

[19] Google. Android UI Guidelines. `http://developer.android.com/guide/developing/building/index.html`, July 2014.

[20] Apple. iOS UI Guidelines. `https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/index.html`, July 2014.

[21] Microsoft. Windows Mobile UI Guidelines. `http://msdn.microsoft.com/en-us/library/bb158602.aspx`, July 2014.

[22] Amy K. Karlson, Shamsi T. Iqbal, Brian Meyers, Gonzalo Ramos, Kathy Lee, and John C. Tang. Mobile Taskflow in Context: A Screenshot Study of Smartphone Usage. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 2009–2018, New York, NY, USA, 2010. ACM.

[23] Shaun K. Kane, Amy K. Karlson, Brian R. Meyers, Paul Johns, Andy Jacobs, and Greg Smith. Exploring Cross-Device Web Use on PCs and Mobile Devices. In *Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction: Part I*, INTERACT '09, pages 722–735, Berlin, Heidelberg, 2009. Springer-Verlag.

[24] Israel Jesus Mojica Ruiz. Large-scale empirical studies of mobile apps. Master's thesis, Queen's University, 2013.

[25] IJ.M. Ruiz, M. Nagappan, B. Adams, and AE. Hassan. Understanding reuse in the Android Market. In *Program Comprehension (ICPC), 2012 IEEE 20th International Conference on*, pages 113–122, June 2012.

[26] M. Harman, Yue Jia, and Yuanyuan Zhang. App store mining and analysis: MSR for app stores. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 108–111, June 2012.

[27] A Shabtai, Y. Fledel, and Y. Elovici. Automated Static Code Analysis for Classifying Android Applications Using Machine Learning. In *Computational Intelligence and Security (CIS), 2010 International Conference on*, pages 329–333, Dec 2010.

[28] R. Minelli and M. Lanza. SAMOA – A Visual Software Analytics Platform for Mobile Applications. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, pages 476–479, Sept 2013.

[29] Alireza Sahami Shirazi, Niels Henze, Albrecht Schmidt, Robin Goldberg, Benjamin Schmidt, and Hansjörg Schmauder. Insights into Layout Patterns of Mobile User Interfaces by an Automatic Analysis of Android Apps. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '13, pages 275–284, New York, NY, USA, 2013. ACM.

[30] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, and Atif M. Memon. Using GUI Ripping for Automated Testing of Android Applications. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, pages 258–261, New York, NY, USA, 2012. ACM.

[31] M.E. Joorabchi and A Mesbah. Reverse Engineering iOS Mobile Applications. In *Reverse Engineering (WCRE), 2012 19th Working Conference on*, pages 177–186, Oct 2012.

[32] Asa Granlund, Daniel Lafrenière, and David A Carr. A Pattern-Supported Approach to the User Interface Design Process, 2001.

[33] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[34] Abram Hindle, Earl T. Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. On the Naturalness of Software. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 837–847, Piscataway, NJ, USA, 2012. IEEE Press.

[35] Mark Gabel and Zhendong Su. A Study of the Uniqueness of Source Code. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '10, pages 147–156, New York, NY, USA, 2010. ACM.

[36] Kai Chen, Peng Liu, and Yingjun Zhang. Achieving Accuracy and Scalability Simultaneously in Detecting Application Clones on Android Markets. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 175–186, New York, NY, USA, 2014. ACM.

[37] APKtool. `http://code.google.com/p/android-apktool/`.

[38] smali. `http://code.google.com/p/smali/`.

[39] AppBrain. Number of Android Applications. `http://www.appbrain.com/stats/number-of-android-apps`, July 2014.

[40] Richard Padilla. Different Mobile Platform's Market Share. `http://www.macrumors.com/2014/01/27/iphone-share-strong-android-lead/`, July 2014.

[41] T.J. McCabe. A Complexity Measure. *Software Engineering, IEEE Transactions on*, SE-2(4):308–320, Dec 1976.

[42] David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, 4 edition, 2007.

[43] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet Allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[44] Andrew Kachites McCallum. MALLET: A Machine Learning for Language Toolkit. 2002.

[45] Tse-Hsun Chen, Stephen W Thomas, Meiyappan Nagappan, and Ahmed E Hassan. Explaining Software Defects Using Topic Models. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 189–198. IEEE, 2012.

[46] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining Association Rules Between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pages 207–216, New York, NY, USA, 1993. ACM.

[47] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *Proceedings of the 7th International Conference on Database Theory*, ICDT '99, pages 398–416, London, UK, UK, 1999. Springer-Verlag.

[48] Jian Pei, Jiawei Han, Runying Mao, et al. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets. In *ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, volume 4, pages 21–30, 2000.

[49] Chanchal K. Roy and James R. Cordy. Are Scripting Languages Really Different? In *Proceedings of the 4th International Workshop on Software Clones*, IWSC '10, pages 17–24, New York, NY, USA, 2010. ACM.

[50] R. Al-Ekram, C. Kapser, R. Holt, and M. Godfrey. Cloning by Accident: an Empirical Study of Source Code Cloning Across Software Systems. In *Empirical Software Engineering, 2005. 2005 International Symposium on*, Nov 2005.

[51] Paul Jaccard. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1902.

[52] Robert K Yin. *Case study research: Design and methods*, volume 5. Sage, 2009.

[53] Martin Szydlowski, Manuel Egele, Christopher Kruegel, and Giovanni Vigna. Challenges for Dynamic Analysis of iOS Applications. In *Proceedings of the 2011 IFIP WG 11.4 International Conference on Open Problems in Network Security*, iNetSec'11, pages 65–77, Berlin, Heidelberg, 2012. Springer-Verlag.

[54] S. Grant and J.R. Cordy. Estimating the Optimal Number of Latent Concepts in Source Code Analysis. In *Source Code Analysis and Manipulation (SCAM), 2010 10th IEEE Working Conference on*, pages 65–74, Sept 2010.

Appendix A

Sample Screen-shots for Retrieved UI Templates

In this section, we provide screen-shots from the pages of mobile applications that are using the UI templates mentioned in Table 5.7.

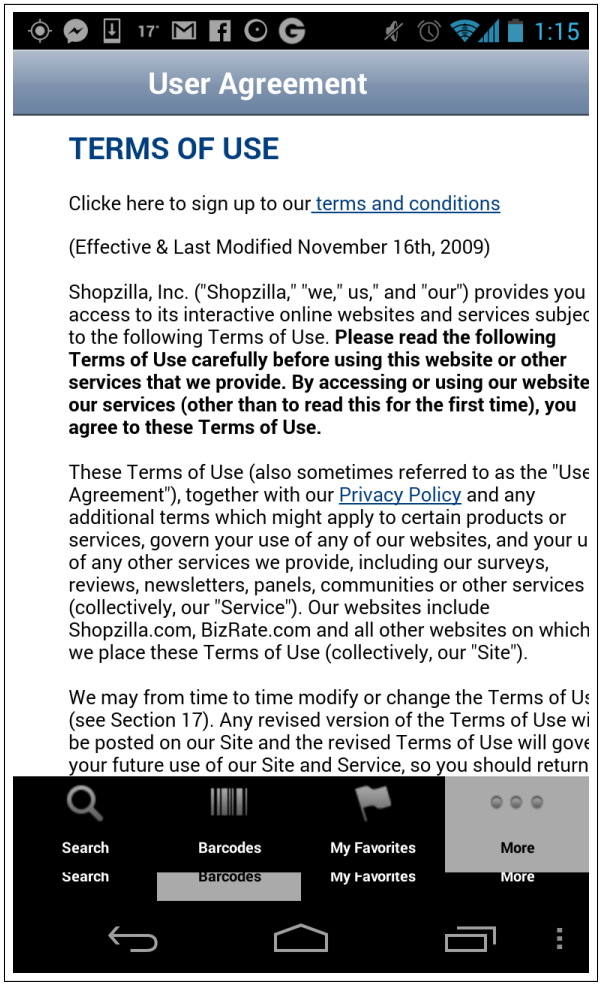

Figure A.1: Screen-shot of the User Agreement page of the Shopzilla application.

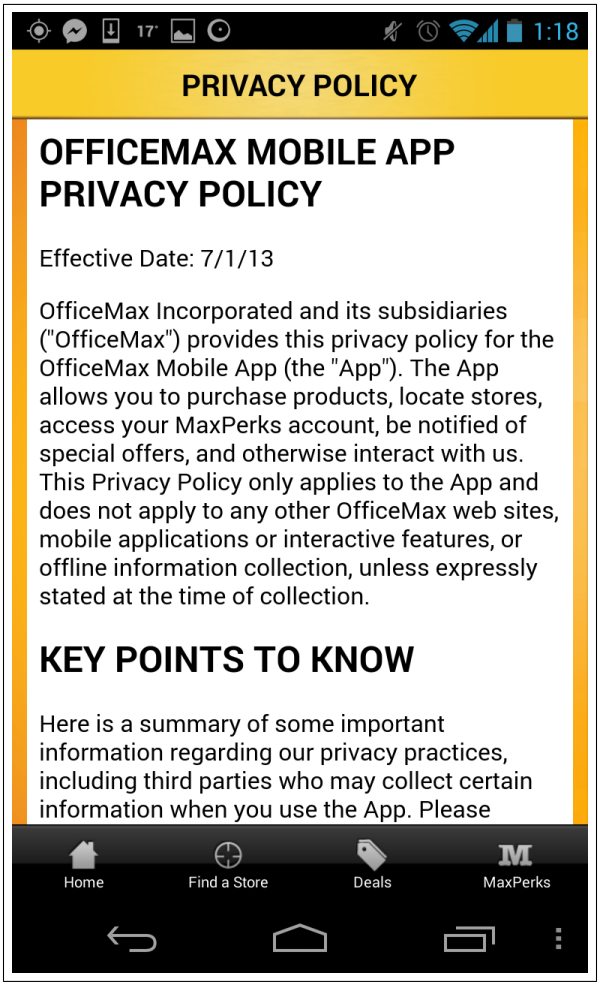

Figure A.2: Screen-shot of the Privacy Policy page of the OfficeMax application.

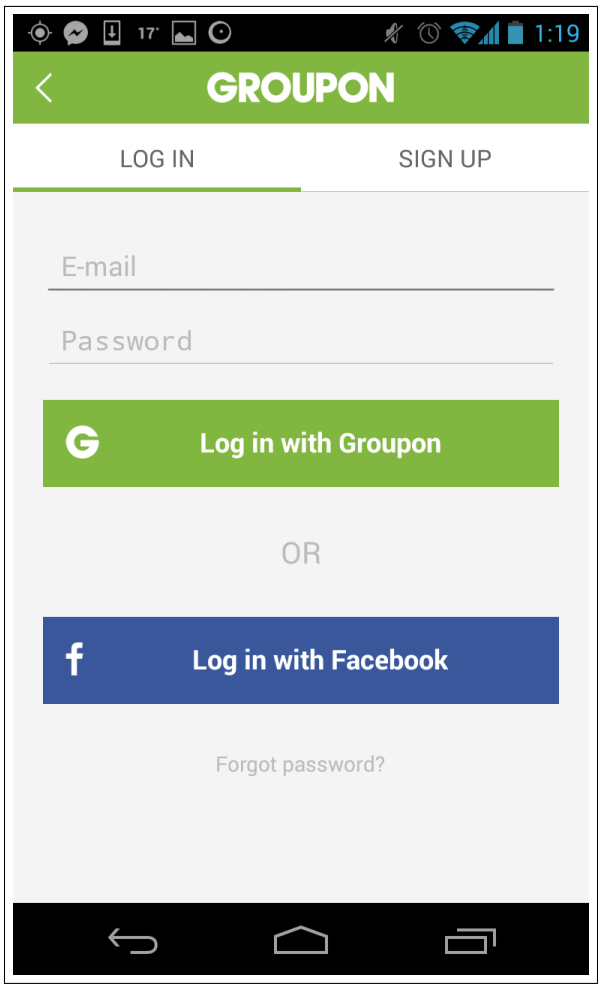

Figure A.3: Screen-shot of the Sign in page of the Groupon application.

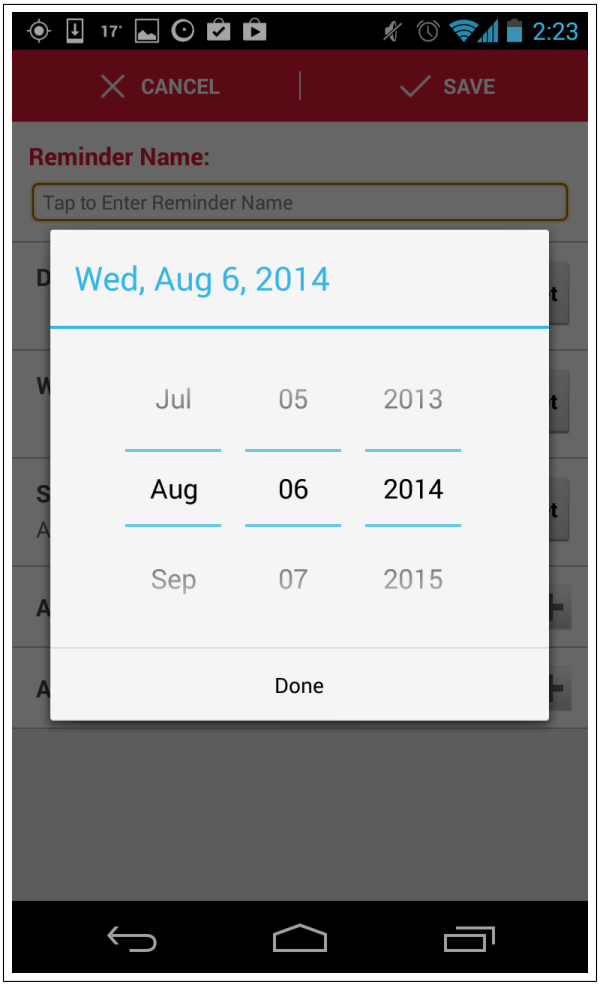

Figure A.4: Screen-shot of the Date Time page of the Walgreen application.

Figure A.5: Screen-shot of the Photo Preview page of the Walgreen application.

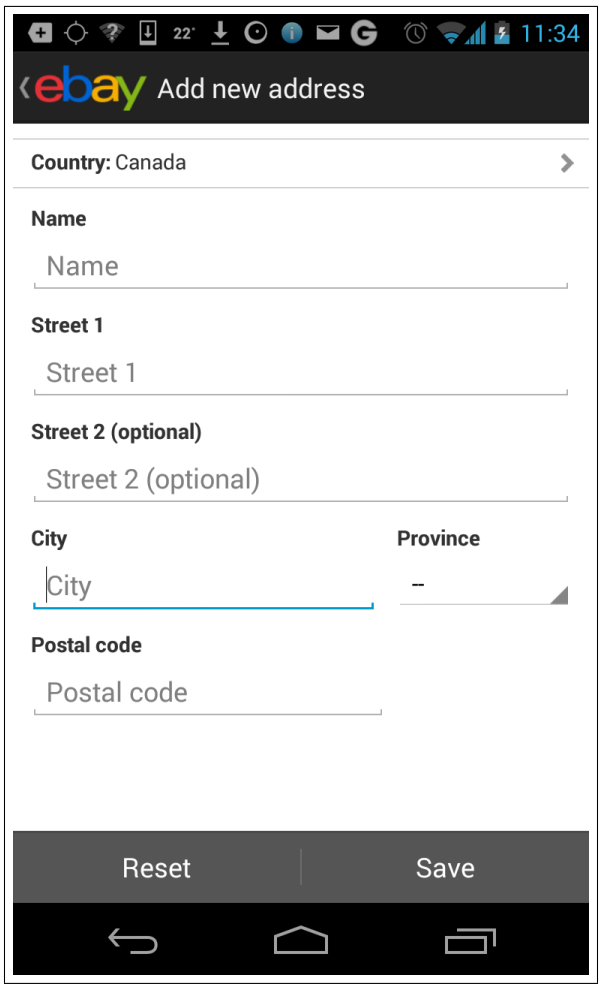

Figure A.6: Screen-shot of the Address Entry page of the eBay application.

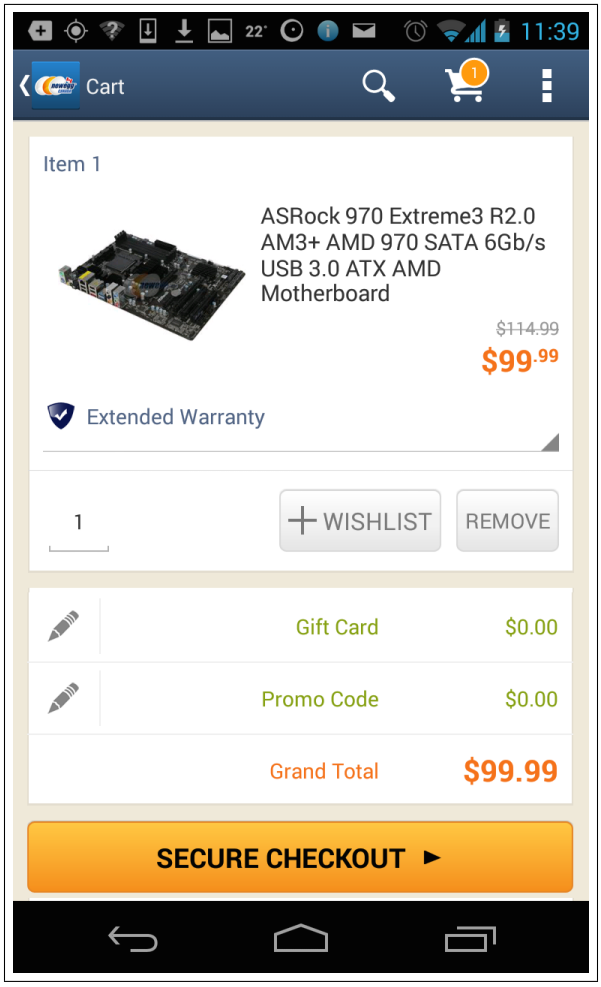

Figure A.7: Screen-shot of the Shopping Cart page of the Newegg application.

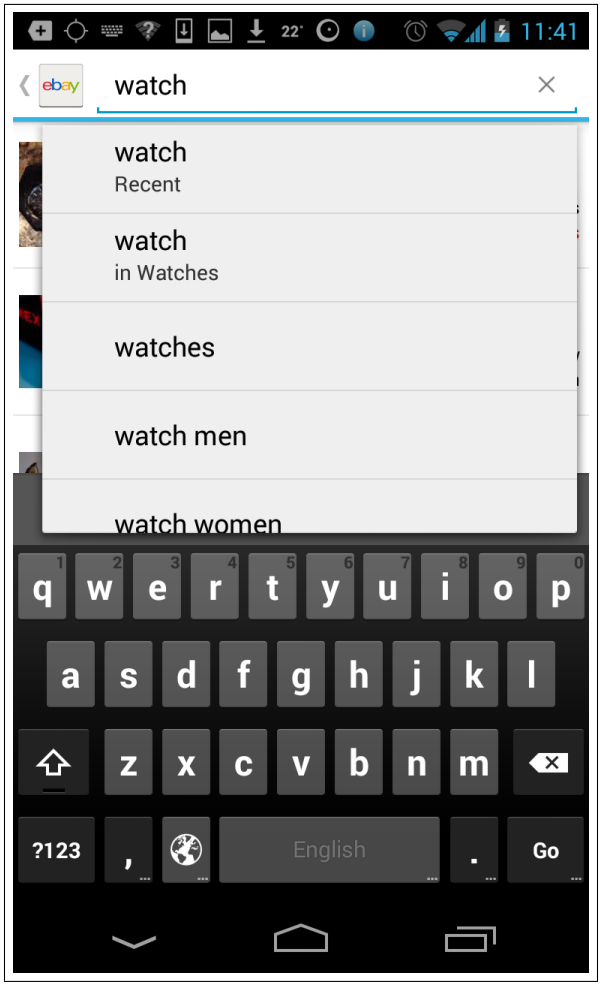

Figure A.8: Screen-shot of the Search page of the eBay application.

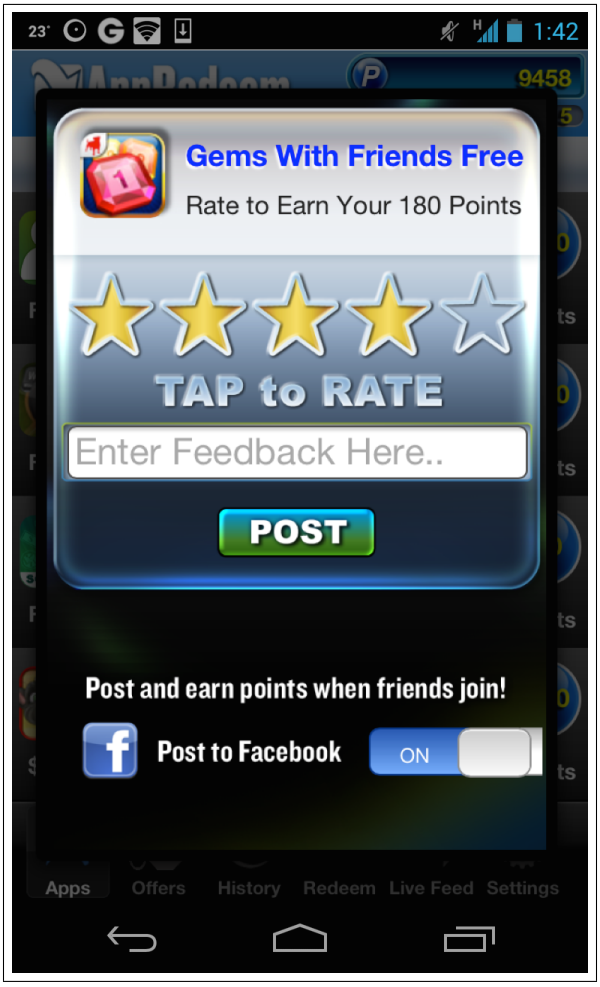

Figure A.9: Screen-shot of the Rating page of the AppRedeem application.

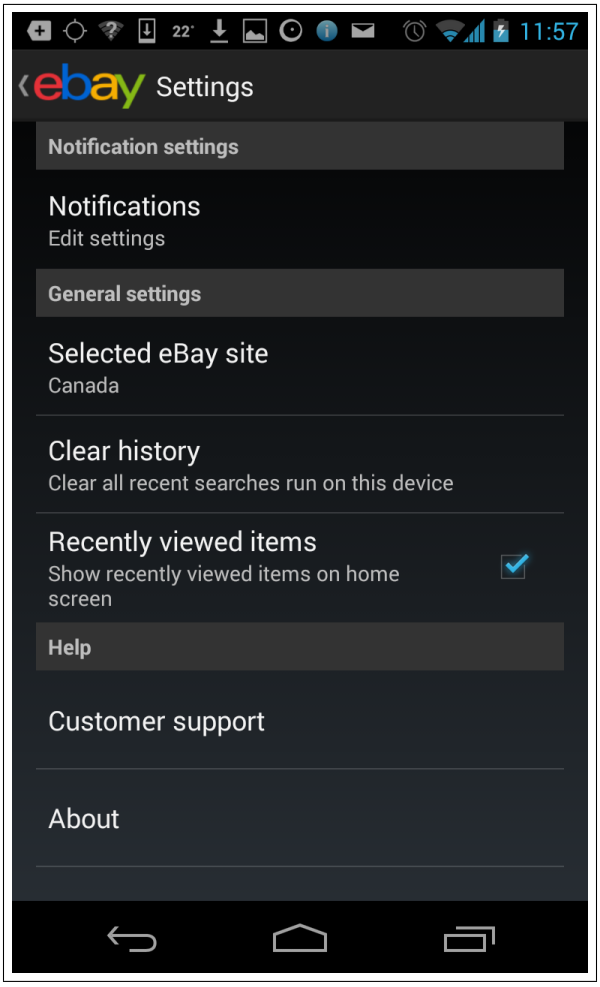

Figure A.10: Screen-shot of the Setting page of the eBay application.