An Exploratory Study on the Usage of Common Interface Elements in Android Applications

Seyyed Ehsan Salamati Taba[1], Iman Keivanloo[2], Ying Zou[2], and Shaohua Wang[1]

[1] *School of Computing, Queen's University, Canada*

[2] *Department of Electrical and Computer Engineering, Queen's University, Canada*

[1]{*taba, shaohua*}*@cs.queensu.ca,* [2]{*iman.keivanloo, ying.zou*}*@queensu.ca*

Abstract. The number of mobile applications has increased drastically in the past few years. A recent study has shown that reusing source code is a common practice for Android application development. However, reuse in mobile applications is not necessarily limited to the source code (*i.e.,* program logic). User interface (UI) design plays a vital role in constructing the user-perceived quality of a mobile application. The user-perceived quality reflects the users' opinions of a product. For mobile applications, it can be quantified by the number of downloads and raters. In this study, we extract commonly used UI elements, denoted as Common Element Sets (CESs), from user interfaces of applications. Moreover, we highlight the characteristics of CESs that can result in a high user-perceived quality by proposing various metrics. Through an empirical study on 1,292 mobile applications, we observe that (i) CESs of mobile applications widely occur among and across different categories; (ii) certain characteristics of CESs can provide a high user-perceived quality; and (iii) through a manual analysis, we recommend UI templates that are extracted and summarized from CESs for developers. Developers and quality assurance personnel can use our guidelines to improve the quality of mobile applications.

Keywords: mobile applications, common UI elements, user-perceived quality.

1 Introduction

Recently, mobile applications have become increasingly popular. The number of downloads of mobile applications on various platforms by the end of 2016 is predicted to be 211 billions [1], increased from 7 billion in 2009 [2]. This large number of downloads implies that a new software industry is emerging. There are various potential reasons for the rapid growth in the number of downloads of available mobile applications. One of them could be the ease of building new applications for mobile platforms [3]. A more fundamental reason that can explain this rapid growth might be the use of proven software engineering practices, such as code reuse [4]. To develop a simple mobile application, there are two things that should be developed: i) User Interface (UI), and ii) business logic (*i.e.,*

source code) of the application. Israel et al. [5] have shown that Java classes in mobile applications on the Android Market are reused significantly, which implies that many applications in the Android Market use very similar *logic*. As aforementioned, another important aspect of a mobile application is the user interface (*UI*).

Due to the limitations of mobile applications (*e.g.,* small screen size, network problems, and computational power [6]), developers should be more careful in designing their applications on mobile platforms than PCs [7]. Developers' negligence in the importance of UI designs is one of the major reasons for users to abandon a task on mobile applications and switch to PCs [7]. To design a UI for a mobile application, developers can adopt commonly used practices (*i.e.,* commonly used sets of UI elements) or standard templates that are used by other developers. However, this question still remains whether the use of commonly used practices (or standard templates) can impact the user-perceived quality of a mobile application. A user-perceived quality can be defined as a user's opinion of a mobile application. The perceived quality can be quantified by the number of downloads and raters in mobile stores [8]. According to the statistics, the user-perceived quality can be influenced by UI designs, stability and performance of applications [9].

In this paper, we are interested to know to what extent the use of commonly used practices (or standard templates) occurs in UI designs of mobile applications. To quantify the extent, we define a Common Element Set (CES), as a set of commonly used UI elements that appear in at least one other UI page. Moreover, we investigate whether there is a relation between using CESs and user-perceived quality of mobile applications. More specifically, we investigate whether using CESs within a category or across categories of the Android applications can impact the user-perceived quality. Furthermore, we go into more details, and investigate the characteristics of CESs related to the high user-perceived quality of the Android applications in each category. Through a manual analysis, we extract templates from the CESs, with the high user-perceived quality, extracted from the Shopping category for a number of functionalities. The UI templates are recurring solutions that solve common design problems. Recommending UI templates associated with the high user-perceive quality helps developers boost UI development for mobile applications. To the best of our knowledge, no one has focused on studying the usage of CESs and extracting UI templates with high user-perceived quality in mobile applications.

Using the user-perceived quality calculated from the statistics in the Android Market and the extracted CESs from 1,292 free Android applications in 8 different categories of Google Play Android Market (i.e., Shopping, Health, Transportation, Travel, News, Weather, Finance, and Social), we address the following research questions:

RQ1) ***To what extent commonly used UI element sets occur?***

We show that CESs widely occur within each category (*i.e.,* 60% of a UI XML file can be constructed by similar CESs that are repeated in other UI XML

files). Moreover, on average, the CESs occur in 23% of mobile applications across different categories.

RQ2) ***Do the usage of commonly used UI element sets impact the user-perceived quality within and across categories?***

We observe that CESs in UI designs of mobile applications can make a significant difference in the user-perceived quality of mobile applications. More specifically, we observe that in specific application categories (e.g., Health), the perceived quality of mobile applications with CESs in their UI designs is significantly higher than the ones without CESs. Therefore, developers should take into consideration the application domain during the process of a UI design.

RQ3) ***Do the usage of commonly UI element sets have an impact on the user-perceived quality of functionalities in mobile applications?***

We observe that the usage of CESs has an impact on the user-perceived quality of functionalities in mobile applications. In almost every functionality, to achieve the high user-perceived quality, it is better to use the CESs that are used in a few number of applications, or used in more UI XML files. Additionally, for certain functionalities, it is desirable to use CES having developer-customized UI elements to achieve the high user-perceived quality.

RQ4) ***Can we extract UI templates from commonly used UI element sets with a high user-perceived quality?***

Through a manual analysis, we extracted a set of UI templates associated with the high user-perceived quality (according to the findings in our empirical study in RQ1 to RQ3). Our extracted UI templates are meaningful and helpful.

The remainder of this paper is organized as follows. First, we explain the architecture of Android applications in Section 2. We describe the experimental setup of our study in Section 3, and report our findings in Section 4. In Section 5, we discuss threats to the validity. We summarize the related literature in Section 6. Section 7 concludes our work and outlines avenues for future works.

2 Background

In this section, we briefly talk about the architecture of Android applications. Typically, 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 content of an Android application, and is the file that Android devices use to install the Android application.

There are four types of application components, including activities, services, content providers and broadcast receivers, that are the essential building blocks of an Android application. Activities are used to implement user interface screens. Services implement background processes. Content providers and

broadcast receivers handle shared data and messages. Among the four listed Android application components, users only interact with activities. An Android application can consist of several activities. The guidelines [10] for Android developers recommend that an activity is a single, focused task that a user can do. Each activity represents a single-screen user interface (UI). As a result, only one activity can be in the foreground for users to interact with.

There are two ways to declare a UI layout for an activity: 1) Declaring UI layout elements in an XML file (aka., UI XML layout), or 2) Instantiating UI layout elements programmatically. Our premise in this work is towards the former approach since it is the recommended way by the Android design guidelines [10]. An XML layout defines a human-readable visual structure for a user interface. Applications using the latter way are excluded from our study, as our analysis and data gathering approach cannot handle them.

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

3 Case Study Design

In this study, we investigate to what extent common element sets (CESs) occur in user interface designs of the Android applications. We examine the characteristics of CESs that may lead to the high user-perceived quality.

3.1 Data Collection

The Android Market (Google Play) started with 2,300 applications in March 2009. Currently there are more than 1 million applications in the market [11]. The Android operating system has the highest market share among other competitors [12]. As a result, we decided to analyze Android applications from the Android market. Moreover, we only study the free applications due to cost issues.

In the Android Market, there are 34 different kinds of categories from which we analyze 8 different categories: Shopping, Health, Transportation, News, Weather, Travel, Finance and Social. The intuition behind choosing these categories is that they encompass different functionalities of daily use of mobile applications. Table 1 shows the descriptive statistics of our study. In total, we study 1,292 free android applications crawled in the first quarter of 2013.

The Android Market allows users rate applications with stars from 1 to 5 (i.e., Low to High), and write reviews for applications. The ratings of an application can show the user-perceived quality of applications, and inform potential users about the experience of the earlier users. However, the rating of an application is not solely a reliable quality measurement, as 86% of the five-star applications throughout the Android Market in 2011 are applications with very few raters (less than 10 raters) [8]. Moreover, Harman et al. [13] show that the raters have

Table 1: Summary of the statistics of applications from different categories. Avg:Average, CES:Common Element Set.

Category	# Applications	# Pages	# CESs	Avg Size of CESs	Size of the largest CES
Shopping	193	2,822	6,622	7.85	14
Health	286	4,129	5,571	8.61	15
Transportation	128	1,078	2,341	7.90	13
News	114	1,302	2,212	7.57	13
Weather	244	1,608	1,968	4.56	9
Travel	106	1,711	2,843	16.93	32
Finance	103	1,167	2,792	9.38	17
Social	118	1,107	3,404	7.72	15

a high correlation with the number of downloads which can be deemed as a key measurement of a success for a mobile application. To overcome these challenges, we measure the user-perceived quality by considering both raters and popularity factors (*i.e.,* the number of downloads) using Equation (1) proposed in our prior study [14]. In this paper, we consider two types of quality attributes: the number of downloads and the number of raters.

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

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

For instance, the rating scores reported by the Android Market for eBay and Amazon Android applications are 4.4 and 4.5, respectively. Relying on these ratings, we can conclude that the Amazon application has the better user-perceived quality than the one of the eBay application. However, if we delve 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 the eBay application and the Amazon application are $1e + 08$ and $5e + 07$, respectively. The number of raters are 563,494 and 112,984, respectively. The eBay application has around 2 and 5 times more downloads and raters than the ones of the Amazon application, respectively. The value of $UPQ(A)$ for the eBay application and the Amazon application are 30.25 and 29.36. As a result, in total, the eBay application has the better user-perceived quality than the Amazon application. Although we use the number of downloads and raters in this paper for measuring the user-perceived quality, more factors (e.g., qualitative analysis of user comments) could be added to our formula to measure the user-perceived quality more accurately.

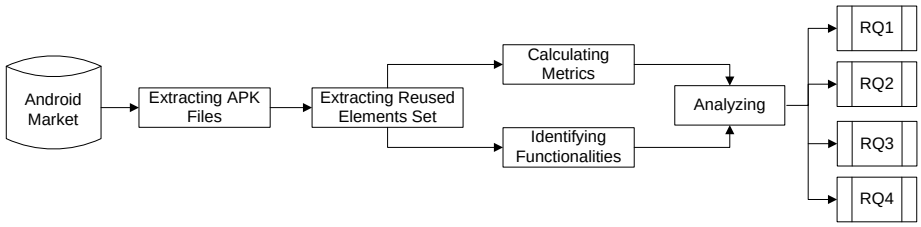

Fig. 1: Overview of our data collection process.

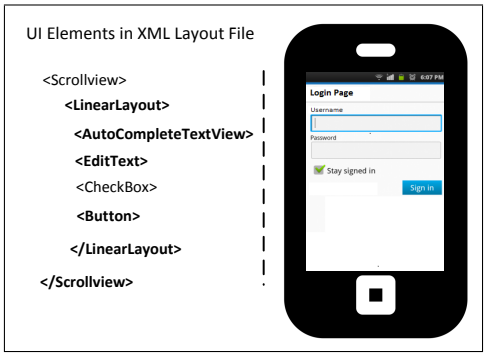

Fig. 2: A UI XML layout file with its corresponding UI elements.

3.2 Data Processing

Figure 1 shows the steps of our data processing. There are three major processing steps in our case study that are responsible for extracting raw data from compiled Android applications, discovering the *Common Element Sets* in UI designs, and finally quantifying the occurrences of CESs using a set of software metrics.

Extracting the Structure To analyze Android applications, we need to extract the contents and the needed information from APK files. To decode an APK file, we use apktool [15], a tool for reverse engineering closed, binary Android applications. The apktool decodes an APK file almost to the original form and structure. It provides the source code of the application in an intermediate "Smali" format [16] which is an assembler for the dex format used in the Android Java virtual machine implementation. The files resulted from the reverse engineering on an APK file are used for our analysis to obtain insights about different aspects of UI designs, namely UI complexity and occurrences of commonly used sets of UI elements, and their effect on the user-perceived quality of mobile applications.

Extracting Common Element Sets We define a Common Elements Set (CES) as a set of commonly used UI elements that occur, at least, more than

once in UI XML files. A Common Element Set (CES) specifies how certain UI elements should be used together. As mentioned in Section 2, for a standard activity (*i.e.,* a page that a user can see), there should exist two files: i) the source code of the activity (*i.e.,* the path of the source code 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.

In this paper, CESs are recommended as meaningful and reusable solutions for building mobile applications. As a result, the minimum number of UI elements in a CES should be at least three to be meaningful and reusable for developers. As shown in Figure 2, for the UI XML file, UI elements highlighted with bold style (*i.e.,* LinearLayout, AutoCompleteTextView, EditText, and Button) can be considered as a CES since this pattern appeared in other UI screens more than 3 times. For each category of mobile applications, we extract CESs using the Frequent Item-set Mining (FIM) algorithm proposed by Agrawal et al. [17]. Given a UI XML file (as shown in Figure 2), the FIM algorithm extracts all CESs whose frequencies are more than a predefined threshold (*i.e.,* named support).

However, FIM techniques unavoidably generate an exponential number of sub-patterns. To solve the shortcoming, Pasquier et al. [18] conduct the association mining to find frequent closed itemsets and their corresponding rules, instead of mining the complete set of frequent itemsets and their associations. In this context, an itemset (*i.e.,* a CES) is closed if none of its immediate supersets has the same support (*i.e.,* the number of UI elements in the CES) 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 reduces the redundant itemsets to be generated and it increases both efficiency and effectiveness of mining [19]. Table 1 shows the common element sets (CESs) extracted from different categories in this study.

Calculating Metrics We extract CESs from each UI XML file of an application using the approach mentioned in section 3.2. To study the different characteristics of CESs, and how they are distributed among different UI XML files, we compute a set of UI metrics. Using the metrics, we can investigate whether there exists a significant difference in the characteristics of CESs that are related to UI XML files with high or low user-perceived quality.

[Metric 1 - CDM] Common element set Distribution Metric

Let A^i, $i \in \{1, 2, ..., n\}$ be the applications in a category (*e.g., Shopping*) of the Android Market, and let C^i, $i \in \{1, 2, ..., n\}$ be a list of CESs (*i.e.,* Common Element Sets) extracted among UI XML files for the applications in a category, *i.e.,* C^1 being the first extracted CES and C^n being the last one.

We define the Common element set Distribution Metric (CDM) to capture the distribution of a set of commonly used UI elements (*i.e.,* Common Element

Sets (CESs)) among different applications within the same category. To this end, we use Equation (2).

$$CDM(C^i) = \frac{NA(C^i)}{NX(C^i)}, \tag{2}$$

Where C^i is the i^{th} CES among all the CESs found from UI XML files of the applications in a category. $NX(C^i)$ denotes the number of UI XML files that use a CES (e.g., C^i). $NA(C^i)$ is the total number of applications that use C^i.

Given a common element set, CDM measures two things: (i) How many applications use the common element set; and (ii) How many UI XML files use the common element set in their structure. To illustrate CDM, we consider a common element set (e.g., C^i), where i is the i^{th} CES among the CES found among the applications in a category. Let us assume that the number of times that C^i has occurred in different UI XML files of all applications in a category (*i.e.,* $NX(C^i)$) is 100. Moreover, we assume that C^i has been distributed among 5 different applications. The value of $CDM(C^i)$ is 0.05. That is to say, the more a common element set is distributed among different applications, the higher CDM is. The more a common element set is distributed among different UI XML files, the less CDM is.

There are two types of UI elements in an Android application. The Android standard UI elements are part of the core Android development kit providing the basic UI features such as a button and a text area. Additionally, the android platform allow developers to create customized UI components and define their own UI elements [10] by extending the stardard UI elements. To detect developer-customized UI elements, we manually build a dataset consisting of all Android standard UI elements. During the data processing phase, if we encounter a new UI element that does not exist in our dataset, it would be recognized as a developer-customized UI element. For example, "*com.ebay.android.widget .ExpandingImageView*" is a developer-customized UI element that is implemented in eBay app.

[Metric 2 - DCM] Developer-customized common element set Metric

To capture the number of customized UI elements by developers in a CES (*e.g.,* C^i), we define Developer-customized common element set Metric (DCM) for C^i following Equation (3).

$$DCM(C^i) = DUE(C^i), \tag{3}$$

Where C^i is the i^{th} CES among all the CESs found from UI XML files of the applications in a category. $DUE(C^i)$ is the number of Developer-customized UI elements in C^i.

[Metric 3 - CSM] Common element set Size Metric

Our last metric is the Common element set Size Metric (CSM), which captures the number of UI elements that are used in a common element set. CSM is computed following Equation (4).

$$CSM(C^i) = SL(C^i), \tag{4}$$

Where $SL(C^i)$ is the number of UI elements in C^i.

Using these three metrics (*i.e.,* CDM, DCM, and CSM), we investigate whether there exists a significant difference in the characteristics of CESs that are related to UI XML files with high or low user-perceived quality. Table 2 shows the summary of our three metrics for a CES: CDM, DCM, and CSM.

Table 2: Summary of our proposed metrics: CDM, DCM, and CSM.

Metric	Explanations
Metric 1. CDM: Common element set Distribution Metric	It captures the distribution of a CES among different applications within a category.
Metric 2. DCM: Developer-customized Common element set Metric	It captures the number of customized UI elements by developers in a CES.
Metric 3. CSM: Common element set Size Metric	It captures the number of UI elements in a CES.

Identifying Functionalities We extract the functionalities of each mobile application using text mining techniques. We are interested in the available keywords in the source code or UI elements. For each activity, we extract textual information, such as contents, strings, labels and filenames associated with the source code of activities and their corresponding UI XML layout files. We use two different heuristics to extract the textual description shown to a user from an activity: i) labels assigned to each element in the UI XML layout file, and ii) strings assigned from the source code. For the former, each element in a UI XML layout file may contain an *android:text* label in which the value is a string shown to a user (see Figure 3). For the latter, in the source code of an activity, we search for *setText()* method call statements. This method specifies the human readable label of a UI element. We extract the human-readable label by analyzing the parameter values.

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

Fig. 3: Part of a Sample XML File

Analysis Methods We use Wilcoxon rank sum test [20] to compare the distribution of different characteristics of CESs between applications with low and high user-perceived quality. The Wilcoxon rank sum test is a non-parametric statistical test to assess whether two independent distributions have equally large values. In general, non-parametric statistical methods do not make assumptions about the distributions of assessed variables.

To extract the high level functionalities of mobile applications in a category, we use a topic modeling technique, namely Latent Dirichlet Allocation (LDA) [21],

to label a set of activities with fine-grained functionality. The LDA generates a topic distribution probability for each document analyzed. A topic is a collection of frequently co-occurring words in the corpus. A topic modeling technique can provide the following information given a set of documents: (i) the topics contained in the documents; and (ii) for each document, the probability that a document belongs to a particular topic. Then, we use Wilcoxon rank sum test [20] to investigate whether the distribution of certain characteristics of CESs vary between activities with low and high user-perceived quality for each functionality.

4 Case Study Results

This section presents and discusses the results of our **four** research questions. We investigate the extent of usage of CESs and the characteristics of CESs that can result in the high user-perceived quality. Finally, through a manual analysis, we recommend UI design patterns from extracted CESs that can be reused as standard templates for UI designs by developers.

RQ1: To what extent commonly used UI element sets occur?

Motivation. According to the Android guidelines [10], in order to develop a standard android application, developers should separate the UI design of the corresponding application from its business logic. Prior research has shown that mobile applications reuse the classes in source code to a great extent which implies that the business logic of mobile applications is widely reused [5]. However, no one has studied the commonly used UI elements of mobile applications. Studying common UI element sets (CESs) has a great value in mobile applications since they may convey good practices that are used frequently.

Approach. In this research question, we analyze the extent of occurrences of CESs within each category. We highlight whether CESs occur across categories. In this research question we want to determine:

RQ1.a) What is the percentage of usage of CESs in Android applications within each category?

RQ1.b) Do CESs occur across different categories?

To answer **RQ1.a**, we use the following two metrics, adapted from earlier empirical studies in software systems [22][23], to measure the usage of CESs in android applications for each category.

Metric. [PCES]: Percentage of a UI XML file that can be constructed by CESs. The idea behind this metric is that what percentage of a UI XML file can be constructed from its corresponding CESs. One UI XML file can contain different CESs, since different parts of a UI XML file can be repeated in other UI XML files. As a result, this metric is the union of the corresponding CESs for a UI XML file divided by the total number of UI elements in it.

For example, the UI XML file shown in Figure 2 which is constructed from 6 UI elements, has a CES consisting of four UI elements. As a result, PCES for this UI XML file is 0.66, meaning that 66% of the UI XML file is constructed from CESs that are occurred in other UI XML files. In each category, we calculate PCES for each UI XML file. Then, we use the average of all calculated PCES for each UI XML file to calculate the PCES for a category.

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

To answer **RQ1.b**, we are interested to know whether CESs happen only within categories or they are also available across categories. If yes, we can generalize our conclusions that CESs can be found in each category. To measure the similarity between the CESs from every pair of two categories, we calculate the Jaccard Similarity Coefficient (JSC) [24] between the CESs from one category and the CESs from the other category. The Jaccard Similarity Coefficient is used for comparing the similarity and diversity of sample sets. The Jaccard coefficient measures the similarity between finite sample sets. JSC is calculated using Equation (5).

$$JSC(P1, P2) = \frac{|P1 \cap P2|}{|P1 \cup P2|}, \tag{5}$$

Where P1 and P2 are the sets of all CESs extracted by the method mentioned in Section 3.2 from two categories (e.g., Shopping and Health).

We are also interested to compare the chances of CESs occurring across categories against the chances of randomly chosen sets of UI elements occurring across categories. For a category, we randomly obtain 100 sets of UI elements. The size of a chosen set ranges from 3 to the size of the largest CES in the category. To compare the similarity between the random sets of UI elements from two categories, we use Equation (5).

Findings. Here, we present our empirical results for RQ1.a and RQ1.b.

Results for RQ1.a. Our results in Table 4 suggest that on average 60% of a UI XML file can be constructed by similar CESs that are repeated in other UI XML files (See Figure 4). Such a high percentage suggests that most UI XML files use some standard UI elements (*e.g.,* RelativeLayout, FrameLayout, and LinearLayout).

Results in Figure 5 show that, on average, 77% of the UI XML files in a category use at least one CES from other UI XML files in the corresponding

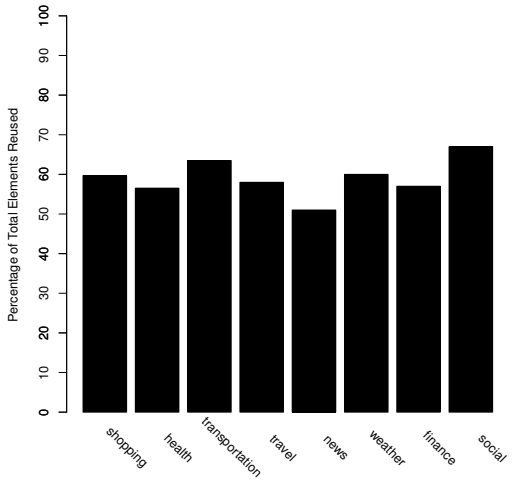

Fig. 4: Percentage of a UI XML file that can be constructed by CESs (PCES) in each category

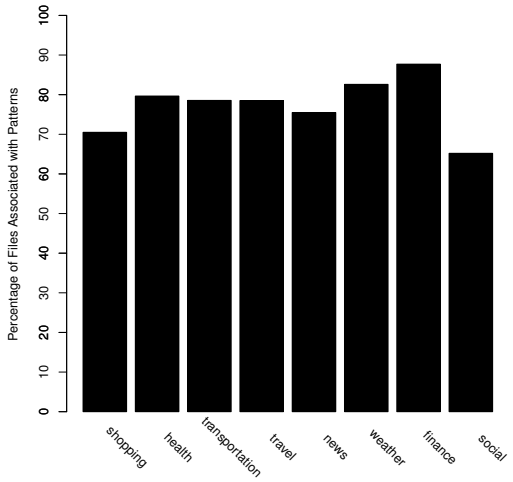

Fig. 5: Percentage of Files Associated with CESs in each category

category, which indicates that very few UI XML files are unique. We observe that the Weather and Finance categories have the highest FAC. The reason is that, for instance, in the Weather category there exist several applications that have exactly the same UI, but are made for different countries, and in different

Table 3: Jaccard similarities between CESs from a pair of 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%

Table 4: Jaccard similarity between random sets of UI elements from a pair of categories.

Categories	Shopping	Health	Transportation	Travel	News	Weather	Finance	Social
Shopping	100%							
Health	15.5%	100%						
Transportation	12.5%	12%	100%					
Travel	11%	11.5%	14%	100%				
News	10.5%	10%	13%	14.5%	100%			
Weather	13.5%	13%	13.5%	15%	14%	100%		
Finance	13%	13%	14%	13%	13.5%	14%	100%	
Social	13.5%	16%	13%	12.5%	12.5%	11%	12%	100%

languages (e.g., the Houston and WLOX weather applications). As a result, FAC is higher in this category.

Results for RQ1.b. Table 3 reports the proportion of CESs that occur across different categories. Each cell of this category is the calculated JSC (see Equation 5) for the CESs extracted from the corresponding two categories indicated by corresponding column and row (e.g., Shopping and Health). As shown in Table 3, there exist, on average, around 23% of CESs across different categories which means that similar patterns across different categories. Table 4 shows the proportion of random sets of UI elements that occur across different categories. As shown in Table 4, on average, only 10%-15% of the randomly obtained sets of UI elements occur across different categories.

Common UI Element Sets are widely used within and across categories. More specially, on average, 77% of in a category use, at least, one common UI element set, and 23% of the CESs from any two domains are similar.

RQ2: Do the usage of commonly used UI element sets impact the user-perceived quality within and across categories?

Motivation. In **RQ1**, we show that CESs widely occur within and across categories. It implies that the common element sets widely occur. In this research question, we want to show whether using common UI element sets (CESs) can provide better user-perceived quality or not.

Approach. In this research question, we focus on CESs, and measure whether using a common element set can provide good user-perceived quality or not. Therefore, we map each CES with user-perceived quality in the following steps:

S1. As mentioned in Section 3.2, we extract CESs from the UI XML files from all applications in each category. Each application has a user-perceived quality value ($UPQ(A)$) calculated by Equation (1), and consists of several UI XML files. To compute the user-perceived quality for each UI XML file, we assign each UI XML file with 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.
S2. Each CES can be related to multiple UI XML files. To assign the calculated user-perceived quality for each UI XML file to each CES, we use the average of the calculated UPQ (See Equation (1)) of the corresponding UI XML files.

Once we obtain the user-perceived quality values for CESs, we want to determine:

RQ2.a) Can CESs from other categories provide a better user-perceived quality in a category?

RQ2.b) What kind of characteristics of a CES can provide a high user-perceived quality within a category?

To answer **RQ2.a**, we find the CESs that occur between each two categories (*i.e.,* source category and destination category) among all of the 8 categories.

We define two concepts: shared and non-shared CESs. A CES is a shared CES if the CES occur in both source and destination categories (*i.e., The source is Shopping and the destination is Health*). A CES is a non-shared CES if the CES is from the source category, but does not occur in the destination category.

We test the following null hypothesis in **RQ2.a** for each source category and the other destination category:

H_0^1: *There is no difference in the user-perceived quality of a shared CES and a none-shared CES.*

We perform a Wilcoxon rank sum test [20] to evaluate H_0^1. To control family-wise errors, we apply Bonferroni correction that adjusts the threshold p-value by dividing the number of tests. The number of tests is 56 since we have 8 different categories as the source categories and 8 different categories as the 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.

To answer **RQ2.b**, we are interested to know whether certain specifications of CESs (see Section 3.2) can provide better user-perceived quality or not. To this end, we sort the CESs based on their user-perceived quality for each category. Then, we break the data into four equal parts (*4 quartiles*): (1) Q1 has the CESs with the highest user-perceived qualities in the highest quartile; (2) Q2 has the CESs with the high user-perceived quality; (3) Q3 has the CESs with the low user-perceived quality; (4) Q4 has the CESs in the lowest quartile. Finally, we observe whether there exists any difference in the computed metrics (*i.e.,* quantifiers of certain characteristics of CESs) between CESs that are related to UI XML files with high and low user-perceived quality. We test the following null hypothesis for each metric (*i.e.,* CDM, DCM, and CSM) in each category.

H_0^2: *there is no significant difference in certain specifications of CESs between the ones related to UI XML files with the high and low user-perceived quality.*

We perform a Wilcoxon rank sum test [20] 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.,* CDM, DCM, and CSM) to quantify the different characteristics of CESs. There exists a statistically significant difference, if p-value is less than 0.05/24=0.002.

Table 5: The results of our Wilcoxon rank sum tests on the differences of CESs that occur within each pair of categories in the user-perceived quality. Only significant results with a p-value < 0.007 are reported.

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

Given the source and destination categories, ↘ means that the average user-perceived quality (UPQ) of the shared CESs, shared between source and destination categories, is less than the average UPQ of non-shared CESs, not used in the destination category. It is vice-versa for ↗.
($p<0.0007/50^{\star}$; $p<0.0007/5^{\circ}$; $p<0.0007^{+}$)

Findings. We present our findings for RQ2.a and RQ2.b.

Results for RQ2.a The results in Table 5 show that using the CESs that are used in other categories can provide a significant difference in the user-perceived quality (UQP) within categories in certain cases. Therefore, we can reject H_0^1.

More specifically, our results in Table 5 can help a developer to choose a proper CES. For example,

E1. If the developer wants to design an application in Health category (as a source category), he or she should choose the shared CESs having a "↗" with a smaller p-value in the following five categories (destination categories): Shopping, Transportation, Travel, News, and Finance. It is because the shared CESs in Health and any of the above destination category can make a positive impact on the UQP. In Table 5, given the source and destination categories, ↗ means that the average user-perceived quality (UPQ) of the shared CESs, shared between source and destination categories, is more than the average UPQ of non-shared CESs, not used in the destination category. It is vice-versa for ↘.
E2. If the developer wants to design an application in Social category (as a source category), he or she should not choose the shared CESs in News category (as a destination), but the categories of Transportation and Weather. It is because the shared CESs in Social and News can hurt the UQP.
E3. If the developer wants to design an application in Shopping, Travel, and News, he or she should have their own special UI designs instead of using shared CESs. It is because all of the shared CESs with a statistically significant difference have a "↘".

Overall, our results suggest that the shared CESs with a "↗" should be used by developers.

> *CESs used across categories can provide a significant difference in the user-perceived quality. However, this phenomenon is category dependent.*

Results for RQ2.b. Table 6 shows whether there exists any difference in certain characteristics of CESs with high user-perceived quality (UPQ) and CESs with a low UPQ. For instance, we take the first cell for Shopping category in Table 6 as an example. Similar to Table 5, there is a "↗" or "↘" sign which shows that whether the average difference in the distribution of the corresponding metric (*i.e.,* here it is CDM) between the CESs with a high UPQ and the CESs with a low UPQ is positive or negative. In this example, it is negative ("↘") which means that the CESs with a high UPQ tend to have a smaller value of Common element set Distribution Metric (CDM) than the ones with a low UPQ. Moreover, we report whether the difference for the corresponding metric (*i.e.,* CSM) is statistically significant between the CESs with a high UPQ and the CESs with a low UPQ. In this example, the difference is statistically significant ($\star$).

As shown in Table 6, we can reject H_0^2, and conclude that, in most categories, CESs with the high user-perceived quality tend to have a smaller value of Common element set Distribution Metric (CDM). In other words, CESs with the high user-perceived quality tend to be used in fewer applications or to be

Table 6: The results of our Wilcoxon rank sum tests on the differences in the usage of different characteristics, CDM, DCM, and CSM, of CESs within various categories. Only significant results with a p-value < 0.002 are reported.

	CDM	DCM	CSM
Shopping	$\searrow$ *	$\nearrow$ *	$\nearrow$ *
Health	$\searrow$ *	$\nearrow$ ∘	
Transportation	$\searrow$ ∘		
News	$\searrow$ *		$\nearrow$ +
Weather	$\searrow$ *		
Travel	$\nearrow$ *	$\nearrow$ *	$\nearrow$ ∘
Finance		$\nearrow$ *	
Social	$\searrow$ *	$\nearrow$ *	$\nearrow$ *

Given a metric that quantifies the characteristics of a CES, $\searrow$ means that the value of the metric for CESs with the high user-perceived quality is less than that of the metric for CESs with the low user-perceived quality. It is vice-versa for $\nearrow$.
($p<0.002/50^{\star}$; $p<0.002/5^{\circ}$; $p<0.002^{+}$)

used in more UI XML files of one application than the CESs with the low user-perceived quality. Overall, CESs with the high user-perceived quality (UPQ) do not distribute widely. It could be because the number of the mobile applications with a high UPQ is much smaller than that of mobile applications with a low (UPQ). Moreover, we can observe that CESs with a high user perceived quality tend to have a higher Developer-customized Common element set Metric (DCM) which means that they tend to have more developer-customized UI elements by the developers. In some categories including Shopping, Health, Travel, Finance, and Social, the personalization is desired by users. Therefore, the CESs with a high UPQ can have more developer-customized Common UI element sets. As a result, being more unique may provide the better user-perceived quality for a UI design.

CESs with the high user-perceived quality tend to be distributed in fewer applications.

RQ3: Do the usage of commonly UI element sets have an impact on the user-perceived quality of functionalities in mobile applications?

Motivation. As shown in **RQ2**, CESs from certain categories, with certain specifications can improve the user-perceived quality in general. 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 3.2) of CESs that result in high or low user-perceived quality in each functionality. If yes, we can highlight the characteristics of CESs that can lead to the high user-perceived quality in each functionality with a lower level of granularity.

Approach. To answer this research question, we first need discover functionalities of an application. Then, we need link a user perceived quality value with a CES in a functionality.

Discovering functionalities. For each UI XML file, we extract all of the strings and labels shown to the users (see Section 3.2). Each file with a set of extracted strings and labels becomes a *document*. We apply LDA on all of the *documents* from the existing applications in a category.

Since mobile applications usually 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 [25] as our LDA implementation, which uses Gibbs sampling to approximate the distribution of topics and words. We run the LDA with 1000 sampling iterations. The number of sampling iterations should be a trade off between the time taken to the 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. To compare the characteristics of CESs extracted from different UI XML files, we label each UI XML file with a fine-grained functionality.

Mapping the user-perceived quality to a CES for a functionality. 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 topics (*i.e.,* F), as well as the mapping (*i.e.,* θ) between the LDA topics and the UI XML files. We use the notation θ_{ij} to describe the topic membership value of a topic f_i in a UI XML file x_j.

The UPQ of a UI XML file from a functionality can originate from two sources: (1) the user-perceived quality of its corresponding application, and (2) the probability that the UI XML file belongs to a functionality. Applying the LDA [21] on UI XML files (documents) acquires a weight of relevance to each functionality (*i.e.,* θ). 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. [26]. Therefore, we calculate the user-perceived quality for each UI XML file (x_j) in a functionality as the following:

$$AUPQ(x_j) = \theta_{ij} * UPQ(x_j), \tag{6}$$

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

Each CES can be related to multiple UI XML files. Each XML file in a functionality has an AUPQ. To assign a user-perceived quality (UPQ) value to a functionality related to a CES, we calculate the average AUPQ (see Equation (6)) of the UI XML files related to the functionality and the CES.

We sort the CESs 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, CESs that are related to UI XML files with the high user-perceived quality, and the ones in the lowest quartile, CESs that are related to UI XML files with the low user-perceived quality. Finally, we investigate whether there exists any difference in the distribution of the characteristics of CESs (*i.e.,* see Section 3.2) between the CESs with the high UPQ and the ones with the low UPQ for each functionality in each category. To this end, we test the following null hypothesis for each metric in each category for each functionality:

H_0^3: there is no difference in the defined characteristics (our defined three metrics: CDM, DCM, and CSM) between the CESs with a low UPQ and the ones with a high UPQ for a functionality.

We perform a Wilcoxon rank sum test [20] 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 7 shows our findings for the calculated metrics on CESs with a statistically significant difference, *i.e.,* Common elements set Distribution Metric (CDM), Developer-customized Common element set Metric (DCM), Common elements set Size Metric (CSM) for each functionality in each category. Based on the results in Table 7, we can reject H_0^3, and conclude that there exists a significant difference in certain characteristics of CESs related to the UI XML files with a low UPQ and the ones with a high UPQ.

For each cell of Table 7, we report two pieces of information. For example, in the cell at the row of CDM in the Shopping category and the column of the f3 referring to "*Login*" and "*Sign in*" functionalities, there is a "negative" ("$\searrow$") that means that CESs, related to the UI XML file for "*Login*" and "*Sign in*" functionality (*i.e.,* f3) in the Shopping category with a low UPQ, have more complexity for CDM than the ones with a high UPQ, with a statistically significant difference ($\star$).

We make the following observations:

[O1.] In most cases, the difference is a negative number ("$\searrow$") for CDM, meaning that CESs from the UI XML files with a low UPQ tend to have a less CES distribution than the ones with a high UPQ. In other words, to develop a functionality, better CESs tend to be used in fewer applications or to be used in more UI XML files than the CESs from the UI XML files with a low UPQ.

Table 7: The results of Wilcoxon rank sum tests on the differences of usage of three metrics between the activities with low user-perceived quality (UPQ) and the ones with high UPQ for each functionality in each category. Only significant results with a p-value < 0.0002 are reported.

		f1	**f2**	**f3**	**f4**	**f5**	**f6**	**f7**	**f8**	**f9**
Shopping	CDM	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$
	DCM	$\nearrow^{\star}$		$\nearrow^{\star}$		$\nearrow^{\star}$	$\nearrow^{\star}$	$\nearrow^{+}$	$\searrow^{\star}$	
	CSM		$\searrow^{\star}$		$\searrow^{\star}$	$\searrow^{+}$			$\searrow^{\star}$	$\searrow^{\star}$
Health	CDM	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\circ}$	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$	
	DCM	$\searrow^{\star}$	$\searrow^{\circ}$	$\searrow^{\circ}$		$\searrow^{\star}$	$\nearrow^{\circ}$			
	CSM	$\searrow^{\star}$		$\nearrow^{\star}$			$\searrow^{\star}$		$\searrow^{\star}$	
Transportation	CDM	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{+}$	$\searrow^{+}$	$\searrow^{\star}$	
	DCM	$\searrow^{\circ}$	$\nearrow^{\star}$		$\searrow^{\circ}$	$\nearrow^{\circ}$			$\searrow^{\star}$	$\searrow^{\star}$
	CSM				$\searrow^{\star}$				$\nearrow^{\star}$	
News	CDM	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\circ}$	$\nearrow^{+}$		$\searrow^{\star}$	$\searrow^{\star}$	
	DCM	$\searrow^{\star}$			$\nearrow^{\star}$		$\searrow^{\star}$			
	CSM	$\nearrow^{\circ}$			$\nearrow^{\star}$		$\searrow^{+}$			
Travel	CDM		$\searrow^{\star}$		$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$
	DCM		$\nearrow^{\star}$					$\searrow^{\star}$	$\searrow^{+}$	
	CSM	$\searrow^{\star}$	$\nearrow^{\star}$							
Weather	CDM		$\searrow^{\star}$			$\searrow^{\star}$			$\searrow^{+}$	$\nearrow^{\star}$
	DCM	$\nearrow^{\star}$				$\nearrow^{\circ}$				
	CSM				$\nearrow^{+}$					$\searrow^{\star}$
Finance	CDM	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{+}$	$\nearrow$	$\searrow^{\star}$		$\nearrow^{\star}$
	DCM	$\searrow^{\star}$		$\nearrow^{\star}$	$\nearrow^{\star}$	$\nearrow^{+}$			$\searrow^{\star}$	$\nearrow^{\star}$
	CSM	$\searrow^{\circ}$	$\nearrow^{+}$				$\searrow^{\circ}$		$\nearrow^{\star}$	
Social	CDM	$\searrow^{\star}$	$\searrow^{\star}$	$\searrow^{\star}$		$\searrow^{\circ}$	$\searrow^{\star}$	$\searrow^{+}$		$\nearrow^{\circ}$
	DCM		$\searrow^{\star}$	$\nearrow^{\star}$	$\nearrow^{\circ}$		$\nearrow^{\star}$	$\searrow^{\star}$		
	CSM		$\searrow^{\star}$					$\searrow^{\star}$		

Given a metric that quantifies a characteristic of a CES, $\searrow$ means that the value of such a metric for the CES with a high UPQ is less than the ones with a low UPQ, and it is vice-versa for $\nearrow$.
($p<0.0002/50^{\star}$; $p<0.0002/5^{\circ}$; $p<0.0002^{+}$)

[**O2.**] As demonstrated in Table 7, using developer-customized elements in CESs for a few functionality in several categories, such as Social, can have an impact on the user-perceived quality. For example, in the shopping category for "*Login*" and "*Sign in*" functionality (*i.e.,* f3), if developers use CESs having developer-customized UI elements, they may have UI XML files with the high user-perceived quality. Our guidelines can be exploited by developers to use the proper CESs to have functionalities with high user-perceived quality.

> *In almost every functionality, to achieve the high user-perceived quality, it is better to use the CESs that are used in a few number of applications, or used in more UI XML files.*

RQ4: Can we extract UI templates from commonly used UI element sets with a high user-perceived quality?

Motivation. In the previous research questions, we show that CESs are used widely within and across categories. The characteristics of CESs can make an impact on the user-perceived quality. To utilize our findings in the previous RQ, practical guidelines using CESs are needed for developers. For example, when a developer wants to design a *login* page for her application in the *Shopping* category, based on the guidelines, the developer can know which characteristics of a UI element set can help achieve a high UPQ (answered in **RQ3**). In addition, the developer needs to know which meaningful combination of UI elements (*i.e.,* a UI template) can provide the high user-perceived quality in a certain (*e.g.,* login) functionality (the aim of **RQ4**).

Approach. To provide more detailed and practical guidelines for developers, we go beyond the characteristics of CESs, and extract UI templates with a high UPQ for a limited number of functionalities. UI templates, summarized from CESs, are standard and reusable solutions for building UIs of mobile applications.

To extract UI templates for certain functionalities, we focus on a more fine-grained functionality than the ones extracted in **RQ3**. As shown in our previous research question, to further analyze the characteristics of CESs, we cluster the UI XML files since they encompass a variety of different functionalities. To this end, we used LDA [21] 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 8 shows the words that describe f3. As shown in Table 8, UI XML files related to f3 are about a high level functionality (*i.e.,* sign-in and login). However, it also contains other sub-functionalities (*e.g.,* sharing a page). As a result, recommending a good UI template for sub-functionalities is necessary. 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 CES, and extract more fine-grained functionalities

(*i.e.,* sub-functionalities). Then, we observe whether the CESs are eligible to be introduced as a UI template or not.

Table 8: 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 **RQ3**, we extract the CESs with the high user-perceived quality. Then, we manually analyze the name of the associate UI XML files, and identify which sub-functionality can be extracted using the extracted CESs, and we further analyze the CESs to study the candidates for UI templates.

Findings. As demonstrated in Table 9, we successfully 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 occurrences of each UI template among UI XML files. For each sub-functionality, we provide 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. Our 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.

For example, for the *Sign in* sub-functionality from Table 9, we recommend a UI template that has been frequently (*i.e.,* 43 times) used by high user-perceived quality UI XML files in this sub-functionality. A sample part of this UI template is shown in Figure 2. Our recommended UI template includes the following elements for entering the username and password:

- An *AutoCompleteTextView* element is an editable *TextView* that automatically shows completion suggestions while the user is typing.
- A *TextView* element displays a text to the user and optionally allows them to edit it.

Our recommended UI template shows that UI XML files with the high user-perceived quality related to the *Sign in* functionality use an *AutoCompleteTextView* element instead of a simple *TextView* element 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;
- an *ScrollView* element to make the page scrollable in case that the page does not fit in the screen of a smart phone;

Table 9: 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

– a *Button* element for submitting the username and password for the sign in procedure;
– two page formatting elements:
  • A *LinearLayout* element arranges its elements in a single column or a single row
  • A *RelativeLayout* element allows for relative positioning of its elements in relation to each other or the parent.

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

5 Threats to Validity

We discuss the threats to validity of our study following common guidelines for empirical studies [27].

5.1 Construct Validity

In this paper, it is mainly due to the measurement errors. Szydlowski et al. [28] discuss the challenges for dynamic analysis of iOS applications. They mention that these challenges are mostly user interface driven. That is, most iOS applications make 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 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 the 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 [10].

5.2 Internal Validity

The internal threats are mainly from our selection of subject systems, tools, and analysis method. The accuracy of using Apktool for parsing APKs into XML files impacts our results. However, the Apktool is widely used in the academic research. The choice of the optimal number of topics in LDA is a difficult task [30]. However, through a manual analysis approach, we found that in all categories there exist at least 9 common functionalities. We choose 1, 292 mobile applications from 8 different categories, due to the variety in user-perceived qualities and domains

of applications. We excluded gaming applications, as the gaming applications tend to use graphical engines and other techniques for building the UIs. As a result, our approaches for extracting the UIs of such applications are not mature enough to handle these type of applications. We provide all the necessary details to replicate our study.

5.3 External Validity

Though we have studied 1,292 free mobile applications from different categories on the Android market, to generalize our results on other mobile stores or mobile platforms, it is encouraged to perform additional studies on those environments. Moreover, an analysis on more applications across more categories in Android Market is always desired.

6 Related Work

In this section, we introduce the research on (1) mining mobile applications; and (2) studying the UIs of mobile applications.

6.1 Mining Mobile Applications

Some research has been proposed to analyze mobile applications. For example, Harman et al. [13] propose the App Store Mining, and discuss factors of success for mobile applications. Their results show that there is a strong correlation between the customer ratings and the number of downloads. In this paper, we use this study as a motivation for defining factors that make an influence on the user-perceived quality of mobile applications. Shabtai et al. [31] conduct a formal study on Android APK files. The machine learning methods are applied 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 [32], a new tool 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 insights 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 UIs of mobile applications to assess the quality (*i.e.,* user-perceived quality). Shirazi et al. [29] analyze the 400 most popular free Android applications to gain insights of UIs of mobile applications. However, they did not provide any evidence that there exists a relation between different characteristics of UIs (*e.g.,* UI complexity) and user-perceived quality of mobile applications. In this paper, we analyze the source code and the relation between the constructed UI elements of mobile applications and the user-perceived quality of mobile applications.

Some other research is on dynamic analysis of mobile applications. Dynamic analysis refers to a set of techniques that monitor the behaviors of a program while it is being executed.For example, AndroidRipper, an automated technique, tests Android applications via their GUI [33]. 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. [34] 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 and computing resources. Instead, similar to Shirazi et al. [29], we use a static analysis for GUI reverse engineering.

6.2 Studying the UI of Mobile Applications

Studying the UIs of mobile applications has raised interests in the research community. For example, Nilsson suggests that experienced UI developers who want to start developing UIs for mobile applications should start with UI templates to develop their applications [35]. 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 [36].

Software Engineering [37] 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 [38]. 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 [39], 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 [5]. Moreover, Chen et al. implement an approach to detect application clones on Android market to detect malwares [40]. 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 [7]. User interface designers also have noticed that certain design problems occurred over and over [35]. 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 using frequently used UI elements can lead to better user-perceived quality or not.

7 Conclusion

In this paper, we perform a detailed case study using 1,292 free Android applications distributed in 8 categories to investigate the relations between commonly used UI element sets, denoted as common element sets (CESs), and user-perceived qualities of mobile applications. We propose various metrics for CESs to study the characteristics of CESs.

Through our empirical results, we observe the following findings::

- Commonly used UI sets of Android applications are widely used within and across different categories (**RQ1**);
- Certain characteristics of CESs can provide high user-perceived quality (**RQ2**, **RQ3**);
- Through a manual analysis approach, we recommend standard and reusable UI templates (i.e., a set of CESs with a high user-perceived quality) for developers (**RQ4**).

In future work, we plan to replicate our study on more mobile applications from other categories existing on the Android Market, other platforms.

References

1. "Number of free and paid mobile app store downloads worldwide from 2011 to 2017," 2015. [Online]. Available: http://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/
2. "Number of downloads for android apps," June 2014. [Online]. Available: http://www.gartner.com/newsroom/id/2592315
3. "Googles do-it-yourself app creation software," July 2014. [Online]. Available: http://www.nytimes.com/2010/07/12/technology/12google.html?_r=0
4. V. R. Basili and H. D. Rombach, "Support for comprehensive reuse," *Softw. Eng. J.*, vol. 6, no. 5, pp. 303–316, Sep. 1991. [Online]. Available: http://dx.doi.org/10.1049/sej.1991.0032
5. I. Ruiz, M. Nagappan, B. Adams, and A. Hassan, "Understanding reuse in the android market," in *Program Comprehension (ICPC), 2012 IEEE 20th Int. Conf. on*, 2012, pp. 113–122.
6. L. Chittaro, "Distinctive aspects of mobile interaction and their implications for the design of multimodal interfaces," *J. on Multimodal User Interfaces*, vol. 3, no. 3, pp. 157–165, 2010.
7. A. K. Karlson, S. T. Iqbal, B. Meyers, G. Ramos, K. Lee, and J. C. Tang, "Mobile taskflow in context: A screenshot study of smartphone usage," in *SIGCHI*, 2010.
8. I. J. Mojica Ruiz, "Large-scale empirical studies of mobile apps," Master's thesis, Queen's University, 2013.
9. "Android apps quality," Feb 2014. [Online]. Available: http://developer.android.com/distribute/googleplay/quality/core.html
10. "Android guidelines," Feb 2014. [Online]. Available: http://developer.android.com/guide/developing/building/index.html
11. "Number of android apps," Feb 2014. [Online]. Available: http://www.appbrain.com/stats/number-of-android-apps

12. "Different mobile platform's market share," Feb 2014. [Online]. Available: http://www.macrumors.com/2014/01/27/iphone-share-strong-android-lead/
13. M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: Msr for app stores," in *MSR*, 2012.
14. S. E. S. Taba, I. Keivanloo, Y. Zou, J. Ng, and T. Ng, "An exploratory study on the relation between user interface complexity and the perceived quality," in *Web Engineering*. Springer, 2014, pp. 370–379.
15. "apktool." [Online]. Available: http://code.google.com/p/android- apktool/
16. "smali." [Online]. Available: http://code.google.com/p/smali/
17. R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '93. New York, NY, USA: ACM, 1993, pp. 207–216. [Online]. Available: http://doi.acm.org/10.1145/170035.170072
18. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules," in *Proceedings of the 7th International Conference on Database Theory*, ser. ICDT '99. London, UK, UK: Springer-Verlag, 1999, pp. 398–416. [Online]. Available: http://dl.acm.org/citation.cfm?id=645503.656256
19. J. Pei, J. Han, and R. Mao, "Closet: An efficient algorithm for mining frequent closed itemsets," 2000, pp. 21–30.
20. D. J. Sheskin, *Handbook of parametric and nonparametric statistical procedures*. crc Press, 2003.
21. D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the J. of machine Learning research*, vol. 3, pp. 993–1022, 2003.
22. C. K. Roy and J. R. Cordy, "Are scripting languages really different?" in *Proceedings of the 4th International Workshop on Software Clones*, ser. IWSC '10. New York, NY, USA: ACM, 2010, pp. 17–24. [Online]. Available: http://doi.acm.org/10.1145/1808901.1808904
23. 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, pp. 10 pp.–.
24. P. Jaccard, "Étude comparative de la distribution florale dans une portion des Alpes et des Jura," *Bulletin del la Société Vaudoise des Sciences Naturelles*, vol. 37, pp. 547–579, 1901.
25. A. K. McCallum, "Mallet: A machine learning for language toolkit," 2002. [Online]. Available: http://mallet.cs.umass.edu
26. T.-H. Chen, S. W. Thomas, M. Nagappan, and A. E. Hassan, "Explaining software defects using topic models," in *MSR*, 2012.
27. R. K. Yin, *Case study research: Design and methods*. Sage, 2009, vol. 5.
28. M. Szydlowski, M. Egele, C. Kruegel, and G. Vigna, "Challenges for dynamic analysis of ios applications," in *iNetSec*, 2012.
29. A. Sahami Shirazi, N. Henze, A. Schmidt, R. Goldberg, B. Schmidt, and H. Schmauder, "Insights into layout patterns of mobile user interfaces by an automatic analysis of android apps," in *SIGCHI*, 2013.
30. S. Grant and J. Cordy, "Estimating the optimal number of latent concepts in source code analysis," in *SCAM*, 2010.
31. A. Shabtai, Y. Fledel, and Y. Elovici, "Automated static code analysis for classifying android applications using machine learning," in *CIS*, 2010.
32. R. Minelli and M. Lanza, "Samoa – a visual software analytics platform for mobile applications," in *ICSM*, 2013.

33. D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon, "Using gui ripping for automated testing of android applications," in *ASE*, 2012.
34. M. Joorabchi and A. Mesbah, "Reverse engineering ios mobile applications," in *WCRE*, 2012.
35. E. G. Nilsson, "Design patterns for user interface for mobile applications," *Advances in Engineering Software*, vol. 40, no. 12, pp. 1318–1328, 2009.
36. sa Granlund, D. Lafrenire, and D. A. Carr, "A pattern-supported approach to the user interface design process," 2001.
37. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.
38. A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu, "On the naturalness of software," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 837–847. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337322
39. M. Gabel and Z. Su, "A study of the uniqueness of source code," in *SIGSOFT FSE'10*, 2010, pp. 147–156.
40. K. Chen, P. Liu, and Y. Zhang, "Achieving Accuracy and Scalability Simultaneously in Detecting Application Clones on Android Markets," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 175–186. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568286