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

Seyyed Ehsan Salamati Taba[1], Iman Keivanloo[2], Ying Zou[2], Joanna Ng[3], Tinny Ng[3]

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

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

[3] *IBM Toronto Lab, Markham, Ontario, Canada*

taba@cs.queensu.ca, {iman.keivanloo, ying.zou}@queensu.ca, {jwng, tinny.ng}@ca.ibm.com

Abstract. The number of mobile applications has increased drastically in the past few years. 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) barriers (*i.e.,* input or output challenges) can affect the user-perceived quality of mobile applications. In this paper, we explore the relation between UI complexity 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 through an empirical study on 1,292 mobile applications in 8 different categories.

1 Introduction

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 [1] such as: checking e-mails or browsing the web. Due to the limitations of smartphones (*e.g.,* small screen size, network problems and computational power) developers should be more careful in designing their applications on smartphones than PCs. Developers' negligence in the importance of UI design is one of the major reasons for users to abandon a task on smartphone and switch to PC [2].

User-perceived quality can be defined as user's opinion of a mobile application. It can be quantified by the number of downloads and ratings in mobile stores. It is important to mention that based on our definition user-perceived quality has no relation with usability in this context. The studies conducted by Karlson et al. [2] and Kane et al. [3] demonstrate that improper use of UI elements (*e.g.,* input and output) on mobile applications increases end-user frustration. 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.

Software metrics are widely used to derive guidelines for programmers. For example, McCabe [4] 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 [5]. 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 paper, 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. We define seven UI complexity metrics that can be calculated using static analysis. We calculate the metrics in two different granularities: i) *category*, and ii) *functionality* of 1,292 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). We observe that there exists a relation between UI complexity and user-perceived quality of application pages (activities) belong to a similar functionality. UI complexity is dependent on the corresponding functionality. Activities with high user-perceived quality tend to be simpler in terms of UI complexity in general.

2 Background

In this section, we briefly talk about the architecture 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 content of an Android application.

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. Among those, users only interact with activities. An Android application consists of several activities. 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 (standard), or ii) Instantiating UI layout elements programmatically. Our premise in this work is towards the former approach since it is the recommended way by Android design guidelines [6]. Applications using the latter way 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 root directory. It contains meta-data information of an application (*e.g.,* the path to the source code of activities, permissions).

3 Study Design

3.1 Data Collection

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

Table 1. Summary of the characteristics of different categories

Category	# Applications	# Activities	# Inputs	# Outputs	# Elements
Shopping	193	2,822	12,529	25,058	68,468
Health	286	4,129	23,232	40,330	108,366
Transportation	128	1,078	5,603	7,718	22,991
News	114	1,302	4,725	7,407	23,507
Weather	244	1,608	6,713	38,659	84,739
Travel	106	1,711	7,164	15,210	38,285
Finance	103	1,167	5,989	12,899	33,818
Social	118	1,107	4,948	7,646	24,091

Extracting User-Perceived Quality. In Android Market, users can rate applications from 1 to 5 (i.e., Low to High), and write comments. The rating reflects the user-perceived quality of applications.However, Ruiz et al. [7] 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 with very few raters (less than 10 raters). Moreover, Harman et al. [8] show that the ratings have a high correlation with the download counts which is a key measure of the success 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 (1):

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

Where $UPQ(A)$ is the measured user-perceived quality for an application; A refers to an application; n is the total number of 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.

3.2 Data Processing

Extracting APK Files. To extract the content and the needed information from APKs, we use apktool [9], 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 [10] which is an assembler for the dex format used in Android Java virtual machine implementation.

Inspecting Decoded APK Files. Given an activity, there does not exist any direct mapping between its source code and its UI page. To measure UI complexity, we need to recover this linking.

Given an application, we extract the path to the source code of activities from the manifest file. To map the activities to their corresponding XML layouts, similar to Shirazi et al.'s work [11], 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 layout file. However, this heuristic cannot map an activity to the corresponding XML layout file if the input argument to this method is the name of the UI XML layout file. To overcome this issue, we trace both IDs and names.

Calculating Metrics. We parse the XML layout 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 2. 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. We categorize the elements as inputs and outputs as shown in Table 3. We use input and output tags listed in Table 3 since such elements are frequently used in Android applications [11].

Table 2. 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

Extracting Functionalities We extract the functionalities of each mobile application using text mining techniques. For each activity, we extract contents, strings, labels and filenames associated to the source code of activities and their corresponding UI XML layout 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 layout file, and ii) strings assigned from the source code. Finally, we use LDA [12] to automatically extract the functionalities in each category.

Table 3. Input and Output Tags

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

4 Study Results

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

RQ1: 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 different studies. A user study by Kane et al. [3] 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. 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), we use the approach mentioned in Section 3.2 to map the source code of activities to their corresponding UI XML layout files. Next, to quantify UI complexity within each category (see Table 2), 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.*

We perform Kruskal Wallis test [13] 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.

To testify the previous findings by Kane et al. [3], we classify our categories based on their study into two categories: i) applications that belong to the categories with high user-perceived quality, and ii) the ones that belong to categories with low user-perceived quality (*i.e.,* Shopping, Health, Travel, Finance, Social). Then, we investigate whether UI complexity is different 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 [13] to evaluate H_0^2, using the 5% level (*i.e.*, p-value < 0.05).

Table 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 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	1.23e-11	-0.21
ANO	1.927e-10	-0.19
ANE	0.001	-0.10
NA	0.007	-0.05

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) 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 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 ([2], [3]) that UI complexity is important on user-perceived quality of mobile applications, and it varies among different categories. Therefore, our measurement approach based on static analysis can quantify UI complexity.

RQ2: Does UI complexity have an impact on the user-perceived quality of the functionalities in mobile applications?

Motivation. Mobile applications have a lot of variety even in the same category. To perform a fine-grained analysis, we cluster the activities based on their functionalities. We investigate 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 3.2). Next, to label each activity with a fine-grained functionality, we use LDA [12] 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 3.2). 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 [14] as our LDA implementation. We run the algorithm with 1000 sampling iterations, and use

the parameter optimization provided by 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}. These functionalities are different in each category, but the number of them is the same ($K = 9$). For instance, f1 in the Shopping category is about "*Login*" and "*Sign in*" functionality. However, in the Health category, it is about "*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.

Each application (A) is consisted of several activities ($\{a_1, a_2, ..., a_n\}$), and it has a user-perceived quality calculated by Equation (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. All the activities from the same application acquire the same user-perceived quality. However, by applying LDA [12] each activity 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 to a functionality is important. A similar decision has been made by Chen et al. [15]. We calculate the user-perceived quality for each activity as the following:

$$AUPQ(a_j) = \theta_{ij} * UPQ(a_j), \tag{2}$$

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.

For each functionality, we sort the activities based on the user-perceived quality. 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. We test the following null hypothesis for each activity level UI metric in each category for each functionality:

H_0^3: there is no difference in UI complexity between activities with low and high user-perceived quality.

We perform a Wilcoxon rank sum test [13] 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 (*i.e.,* 216). 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. For each cell of Table 6, we report three pieces of information. Let's consider the cell related to the Shopping category for the first functionality (*i.e.,* f1) which refers to "*Login*"

Table 6. Average usage of activity level UI metrics in the activities with low and high user-perceived quality for each functionality in each category. ($p<0.0002/50^{\star}$; $p<0.0002/5^{\circ}$; $p<0.0002^{+}$)

		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

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 related to this functionality (f1) 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 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 [4]). Here, it implies the average number of the corresponding activity level UI metric for high 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 6, we can reject H_0^3, and conclude that there exists a significant difference in UI complexity between activities with low and high user-perceived quality. Furthermore, we observe that UI complexity is dependent on the corresponding functionality and category. For some functionalities higher UI complexity can result in a better user-perceived quality. However, in some cases this relation is quite different. In most cases this difference is a positive number ("↗") 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.

5 Threats to Validity

We now discuss the threats to validity of our study following common guidelines for empirical studies [16].

Construct validity threats concern the relation between theory and observation. They are mainly due to measurement errors. Szydlowski et al. discuss the challenges for dynamic analysis of iOS applications [17]. They mention that these challenges are user interface driven. Due to such challenges, we were not able to use dynamic analysis for mobile application UI reverse engineering for a large scale study. In this study, our premise is based on the UI elements declared in XML files since it is the recommended approach by Android guidelines [6].

Threats to internal validity concern 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 this tool. Moreover, the choice of the optimal number of topics in LDA is a difficult task. However, through a manual analysis approach, we found that in all categories there exist at least 9 common functionalities.

Conclusion validity threats concern the relation between the treatment and the outcome. We paid attention not to violate assumptions of the constructed statistical models; in particular we used non-parametric tests that do not require any assumption on the underlying data distribution.

Reliability validity threats concern the possibility of replicating this study. Every result obtained through empirical studies is threatened by potential bias from data sets [18]. To mitigate these threats we tested our hypotheses 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.

Threats to external validity concern the possibility to generalize our results. We try to study several mobile applications (1,292) from different categories. Our study analyzes free (as in "no cost") 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.

6 Conclusion

In this paper, 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. Then, we performed a detailed case study using 1,292 free Android applications distributed in 8 categories, to investigate the impact of UI complexity on user-perceived quality of mobile applications. The highlights of our analysis include: i) We can quantify UI complexity based on our measurement approach (**RQ1**) and ii) 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 (**RQ2**).

In future work, we plan to replicate this study on more categories existing on Android Market. Moreover, we should investigate whether our findings are consistent among other platforms (iOS and BlackBerry).

References

1. A. K. Karlson, B. R. Meyers, A. Jacobs, P. Johns, and S. K. Kane, "Working overtime: Patterns of smartphone and pc usage in the day of an information worker," in *PerCom*, 2009.
2. 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.
3. S. K. Kane, A. K. Karlson, B. R. Meyers, P. Johns, A. Jacobs, and G. Smith, "Exploring cross-device web use on pcs and mobile devices," in *HCI: part I*, 2009.
4. T. McCabe, "A complexity measure," *Software Engineering, IEEE Transactions on*, vol. SE-2, no. 4, pp. 308–320, 1976.
5. E. G. Nilsson, "Design patterns for user interface for mobile applications," *Advances in Engineering Software*, vol. 40, no. 12, pp. 1318–1328, 2009.
6. "Android guidelines," Apr 2014. [Online]. Available: http://developer.android.com/guide/developing/building/index.html
7. I. J. Mojica Ruiz, "Large-scale empirical studies of mobile apps," Master's thesis, Queen's University, 2013.
8. M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: Msr for app stores," in *MSR*, 2012.
9. "apktool." [Online]. Available: http://code.google.com/p/android-apktool/
10. "smali." [Online]. Available: http://code.google.com/p/smali/
11. 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.
12. 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.
13. D. J. Sheskin, *Handbook of parametric and nonparametric statistical procedures*. crc Press, 2003.
14. A. K. McCallum, "Mallet: A machine learning for language toolkit," 2002. [Online]. Available: http://mallet.cs.umass.edu
15. T.-H. Chen, S. W. Thomas, M. Nagappan, and A. E. Hassan, "Explaining software defects using topic models," in *MSR*, 2012.
16. R. K. Yin, *Case study research: Design and methods*. Sage, 2009, vol. 5.
17. M. Szydlowski, M. Egele, C. Kruegel, and G. Vigna, "Challenges for dynamic analysis of ios applications," in *iNetSec*, 2012.
18. T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *Software Engineering, IEEE Transactions on*, vol. 33, no. 1, pp. 2–13, Feb 2007.