Empir Software Eng (2017) 22:3088–3116
DOI 10.1007/s10664-017-9507-3

A study of the relation of mobile device attributes with the user-perceived quality of Android apps

Ehsan Noei[1] · **Mark D. Syer**[2] · **Ying Zou**[1] · **Ahmed E. Hassan**[2] · **Iman Keivanloo**[1]

Published online: 13 March 2017

Abstract The number of mobile applications (apps) and mobile devices has increased considerably over the past few years. Online app markets, such as the Google Play Store, use a star-rating mechanism to quantify the user-perceived quality of mobile apps. Users may rate apps on a five point (star) scale where a five star-rating is the highest rating. Having considered the importance of a high star-rating to the success of an app, recent studies continue to explore the relationship between the app attributes, such as User Interface (UI) complexity, and the user-perceived quality. However, the user-perceived quality reflects the users' experience using an app on a particular mobile device. Hence, the user-perceived quality of an app is not solely determined by app attributes. In this paper, we study the relation of both device attributes and app attributes with the user-perceived quality of Android apps from the Google Play Store. We study 20 device attributes, such as the CPU and the display size, and 13 app attributes, such as code size and UI complexity. Our study is based on data from 30 types of Android mobile devices and 280 Android apps. We use linear mixed effect mod-

Communicated by: Lin Tan

✉ Ehsan Noei
e.noei@queensu.ca

Mark D. Syer
mdsyer@cs.queensu.ca

Ying Zou
ying.zou@queensu.ca

Ahmed E. Hassan
ahmed@cs.queensu.ca

Iman Keivanloo
iman.keivanloo@queensu.ca

[1] Department of Electrical and Computer Engineering, Queen's University, Kingston, Canada

[2] School of Computing, Queen's University, Kingston, Canada

els to identify the device attributes and app attributes with the strongest relationship with the user-perceived quality. We find that the code size has the strongest relationship with the user-perceived quality. However, some device attributes, such as the CPU, have stronger relationships with the user-perceived quality than some app attributes, such as the number of UI inputs and outputs of an app. Our work helps both device manufacturers and app developers. Manufacturers can focus on the attributes that have significant relationships with the user-perceived quality. Moreover, app developers should be careful about the devices for which they make their apps available because the device attributes have a strong relationship with the ratings that users give to apps.

Keywords Mobile applications · Software attributes · Device attributes · User-perceived quality

1 Introduction

The number of mobile devices and mobile applications (apps) has increased by 550 % in the last four years (Lawson 2009; Stats 2016). Users not only use their mobile devices for phone calls, but also for entertainment, such as social networking and playing games. Online app stores, such as the Google Play Store (Google 2015b), use a star-rating mechanism to quantify the user-perceived quality of mobile apps. A user can give a one (i.e., the lowest) to five (i.e., the highest) star rating to each app. Due to the large number of apps in app stores, users rely on the rating information when choosing new apps to download (Martin et al. 2015, to appear; Bavota et al. 2015). Consequently, the published user-perceived quality information can affect the revenue of app developers (Costa-Montenegro et al. 2012; Kim et al. 2011). Therefore, it is critical to identify the attributes that have significant relationships with the user-perceived quality of mobile apps. Knowledge of such attributes helps mobile device manufacturers to focus on the device attributes that have the strongest relationships with the user-perceived quality as expressed through app reviews. Such knowledge also helps developers proactively quantify the expected user-perceived quality of apps prior to release. Developers can limit the availability of their apps to specific devices to reduce the risk of receiving low ratings from specific devices. Moreover, developers can prioritize their app testing efforts by focusing on certain devices (Khalid et al. 2014).

Recent studies mainly focus on the relationships between app attributes and the user-perceived quality (Martin et al. to appear; Taba et al. 2014; Linares-Vásquez et al. 2013; Kim et al. 2011). For instance, Taba et al. indicate that the User Interface (UI) complexity has a significant relationship with the user-perceived quality of apps (Taba et al. 2014). However, the observed user-perceived quality is the result of users' experience by running the apps on various mobile devices (Wasserman 2010). Therefore, we believe that the user-perceived quality of mobile apps is not determined solely by the app attributes. In this paper, we study the relationship between the device attributes and the user-perceived quality of Android apps. We structure our study along the following three research questions:

RQ1) How strong is the relationship between the user-perceived quality of apps and the devices on which they run?

The level of satisfaction with an app varies based on the device on which an app runs. Based on a study of data from 30 mobile devices and 280 Android apps, we find that users of various devices give varying star-ratings to the same set of apps.

RQ2) Which device attributes have the strongest relationship with the user-perceived quality?

We define 20 device attributes, such as memory capacity, display size, and battery size. We find that the user-perceived quality of an app has a significant relationship with several device attributes such as the Central Processing Unit (CPU), display, and Operating System (OS). For example, we find that users with devices with more powerful CPUs perceive apps to be of higher quality. Conversely, users with a newer version of the operating system perceive apps as being of lower quality.

RQ3) Do device attributes have a stronger relationship with the user-perceived quality than app attributes?

We define 13 app attributes, such as UI complexity and code size, in addition to the device attributes of **RQ2**. We find that app attributes, such as code size, have significant relationships with the user-perceived quality. However, some device attributes, such as the CPU, have a significantly stronger relationship with the user-perceived quality of apps than the majority of the app attributes. Unlike earlier attempts (e.g., Taba et al. 2014) that only consider the app attributes to judge the user-perceived quality of apps, we find that both app and device attributes have statistically significant relationship with the user-perceived quality of apps.

The contributions of this paper include: (a) we show that the user-perceived quality of mobile apps varies across different devices; (b) we find that device attributes, such as the CPU, have significant relationships with the user-perceived quality of mobile apps; and (c) we show that device attributes maintain their strong relationship with the user-perceived quality of mobile apps even after we control for the app attributes. Both app and device attributes have significant relationships with the user-perceived quality of mobile apps. However, some device attributes, such the display resolution, have a greater relationship with the user-perceived quality in comparison with some app attributes, such as the number of UI inputs.

Paper Organization The remainder of this paper is organized as follows. Section 2 presents our case study design. Section 3 discusses the details of our research questions and our findings. Section 4 lists the threats to the validity of this paper. Section 5 discusses prior related work. Finally, Section 6 concludes the paper.

2 Case Study Design

Figure 1 gives an overview of our case study design. First, we retrieve the user reviews, app details, Android PacKage (APK) files, and device attributes from the Google Play Store and the technical device-specific websites, including GSM Arena[1] and Phone Arena[2]. Technical device-specific websites provide the required information about the device attributes. For example, the details of device specifications, such as the capacity of RAM, are listed on

[1] http://www.gsmarena.com/

[2] http://www.phonearena.com/

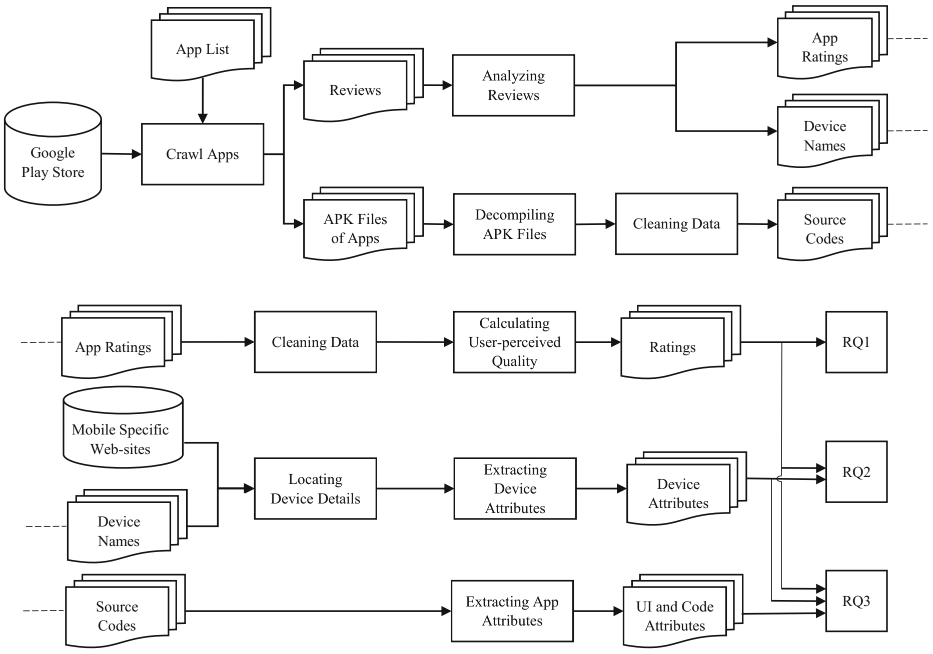


Fig. 1 An overview of the case study design

such websites. Then we apply various steps of data processing. The outcome of these steps is the required data for addressing the research questions.

2.1 Data Source

In our study, we consider the star-ratings that are listed in the Google Play Store as the user-perceived quality. While several mobile app platforms exist (e.g., iOS, BlackBerry 10, and Windows Phone), Android is by far the largest platform. Furthermore, the Android platform suffers from device fragmentation, where there are many different devices running many different versions of Android (Han et al. 2012).

We study the same set of user reviews that were retrieved by Khalid et al. using a Selenium-based crawler (Khalid et al. 2014; Selenium 2014). Selenium is a web automation and testing tool. The crawler extracts data, such as app names, reviews, devices that are used to review the apps, and the numerical star-rating of reviews. Khalid et al. selected 99 apps for their study (Khalid et al. 2014). However, our dataset extends their dataset to include 593 apps, 595,951 reviews, and 210 devices. This dataset contains reviews from 2009 to 2013. Unfortunately, the Google no longer reveals the name of the device that is associated with each posted review. Hence, we cannot update this unique dataset.

We extract the technical details of each device type from the GSM Arena and Phone Arena websites. Both websites contain technical articles and details about many of the industry's mobile devices. When the details of a particular device are not available through GSM Arena, we check the Phone Arena website to get the device details.

2.2 Data Processing

In this section, we discuss our data processing steps, such as the cleaning of the crawled data, and the extraction of app and device attributes.

2.2.1 Decompiling APK Files

Android apps are primarily written in the Java programming language using the Android Software Development Kit (SDK). To publish an Android app on an online app store (e.g., Google Play Store), the app has to be compiled into a single APK file. An APK file contains all the content that is required for both the installation and execution of an app. An APK file contains a manifest file, a meta-information directory, a Dalvik EXecutable (DEX) file, and a resource directory. The manifest file describes app-level meta-information, such as the title of the app or the permission list. The compiled source code of the app is stored in a DEX file. A DEX file represents the compiled code of Java classes using an intermediate language similar to Java bytecode. Unlike Java bytecode, DEX files are compiled to be consumed by Dalvik, which is a register-based virtual machine that is provided by the Android platform.

To extract the source code and the required information from APK files, we use (Dex2jar 2016) and Java Decompiler (Dex2ja 2016). To calculate the various app attributes, we decompile the DEX files into Java classes. Then, we use a Java Decompiler to convert the Java classes to Java source code. Finally, we use the decompiled Java files to measure the app attributes, such as the number of lines of code (LOC) and the number of methods. In our study, we use off-the-shelf decompilers to decompile APK files. The decompilers have some limitations, such as problems in dealing with variables, literals, and types (Miecznikowski and Hendren 2002). Therefore, we remove the apps that failed to decompile from our initial dataset.

2.2.2 Cleaning Dataset

Our initial dataset contains 595,951 reviews, 210 devices, and 596 apps. The Google Play Store only allows us to download the latest version of an app (Google 2015b). There are 194 apps for which we had not downloaded their APK files previously, and their older versions are no longer available. Therefore, we remove these 194 apps from our study. Moreover, we remove 22 apps that do not declare their UI using XML files from our study. We believe that the UI is an important factor in the user-perceived quality of apps because users have direct interaction with the UIs of the apps. Furthermore, we removed 100 paid apps from our dataset because we would need to pay to get access to their APK. We focus on free to download apps and not paid apps to reduce the influence of other confounding factors, such as the variance of user expectations based on the cost of an app (Harbach et al. 2014).

We also remove the devices whose attributes are not publicly available on the Internet. In addition, we choose the devices that have the most apps in common. To avoid skewing our findings due to apps with few reviews, we consider the largest subset of apps that have at least five reviews for each device in our dataset. We end up with 30 mobile devices, 280 Android apps, and 150,373 reviews in our dataset.

2.2.3 Extracting App Attributes

We describe the attributes of apps along three aspects: (a) source code, (b) user interface, and (c) miscellaneous. Source code attributes are classically- and extensively-studied attributes in software quality literature (i.e., number of methods, lines of code, McCabe's cyclomatic complexity and weighted method per class). Such attributes are used as control variables and are known to be correlated to the number of defects in an app (a measure of source code quality that may impact user-perceived quality). User interface attributes measure the complexity of the user interface (e.g., number of inputs) and are known to be correlated with the user-perceived quality of apps (Taba et al. 2014). User-reviews, such as "*Nice user interface - creative & comprehensive! Great for school closure info too!*" and "*This app has everything in a great user interface!*", also suggest that the complexity of the user interface has a relationship with user-perceived quality.

Source Code For each app, we count the number of methods and LOC in the decompiled Java files. LOC is a software attribute that is used to measure the size of a program. Larger programs in terms of lines of code may be harder to maintain (Li and Henry 1993). As a result, the size of an app might impact its user-perceived quality. We measure the number of methods as another attribute that captures the code size (Johnson and Foote 1988).

We calculate the complexity of each app using two measures: (a) McCabe's cyclomatic complexity (McCabe 1976), and (b) Weighted Method per Class (WMC) (Chidamber and Kemerer 1994). After calculating the McCabe's cyclomatic complexity for each method, we count the number of methods with a cyclomatic complexity value of more than ten. Coppick and Cheatham note that methods with cyclomatic complexity values of less than ten are considered simple while methods with cyclomatic complexity values of more than ten indicate more complex code (Coppick and Cheatham 1992). To find the WMC of each app, we measure the weighted average of the WMCs of each class of the app.

User Interface App components are the fundamental building blocks of an Android app. There are four types of app UI components: (a) activities, (b) services, (c) content providers, and (d) broadcast receivers (Google 2015a). An activity represents a single-screen UI. A service is to perform functions for remote processes or to perform long-running operations. A content provider manages a shared set of app data. A broadcast receiver is a component that responds to broadcast announcements (Google 2015a). Users interact with activities only. There are two ways to define a UI layout for an activity: (a) declaring UI design elements in an XML file; and (b) instantiating UI layout elements in the source code. Similar to earlier studies (e.g., Taba et al. 2014) on UI attributes of Android apps, we study the apps that follow the best practices for UI implementation approach as suggested by Google (2015c). With the best practices for UI implementation, developers need to declare the layouts of apps in XML files. We can then measure the UI complexity using static analysis (Nielson et al. 1999). We parse the XML files to calculate a set of UI attributes, as proposed by (Taba et al. 2014), including the number of inputs, outputs, and activities. We use the input and output tags that are listed in Table 1, to identify inputs and output elements in a UI page (Taba et al. 2014). As the XML files are represented in a tree-like data structure, we are interested in examining the relationship between the main characteristics of the UI trees and the user-perceived quality. Hence, we extracted the total number of leaves, nodes, and the layout depth of the UI trees of each app (Cormen et al. 2009).

Miscellaneous We consider the size of the APK files of an app since the size metric captures the overall complexity and richness of an app. An APK file contains various media and other resources that are associated with an app.

Table 1 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

We summarize our app attributes in Table 2, where the last column describes each attribute. Overall, we compute 13 attributes along the three aspects. Table 3 shows a brief overview and statistics of all of the computed app attributes.

2.2.4 Extracting Device Attributes

For each device, we extract information about their body (i.e., physical dimensions), display, platform, memory, camera, and battery. We extract 38 *raw attributes*. A raw attribute is referred to as an attribute without any processing on its value. All the 38 raw attributes are listed in Table 4. We remove device attributes that have the same value for all of the devices that we study because this information cannot be used for differentiating between devices. Furthermore, we focus on the device attributes that capture device abilities instead of other attributes, such as the current price of a device, since a device that is expensive today would be cheaper in the future. A user's experience with an app will remain the same independent of how much the user paid for the device. We end up with 20 device attributes that are summarized in Table 5.

We classify the various device attributes as (a) nominal variables that include two or more categories without considering the ordering among the categories; (b) ordinal variables that are similar to nominal, while considering the ordering among the variables; or (c) numeric

Table 2 The app attributes

Category	Attribute	Description
User Interface	Number of Leaves	The number of leaves in XML layout files.
	Number of Nodes	The number of nodes in XML layout files.
	UI Depth	The deepest tree in XML layout files.
	Number of Activities	The number of activities in the UI.
	Number of Inputs	The number of inputs in in the UI.
	Number of Outputs	The number of outputs in the UI.
	LOCUI	The number of lines of UI code.
Source Code	LOC	The number of lines of source code.
	Number of Methods	The number of methods in source code.
	WMC	The average of weighted method per class.
	Low McCabe	The number of methods with a cyclomatic complexity equal or less than 10.
	High McCabe	The number of methods with a cyclomatic complexity more than 10.
Miscellaneous	APK Size	The size of the downloadable APK installation file.

Table 3 Overview of the app attributes

Category	Attribute	Maximum	Minimum	Mean	Median
User Interface	LOCUI	4,222	4	605	371
	UI Leaves	4,222	2	605	160
	UI Nodes	1,846	1	264	183
	UI Depth	17	1	6	6
	Activity	495	1	49	34
	Input	769	1	52	22
	Output	1,375	1	152	79
Source code	LOC	607,458	315	140,541	137,598
	Code Method	54,444	40	12,162	11,692
	WMC	37	3	16	16
	Low McCabe	47,573	34	12,528	9,915
	High McCabe	1,661	0	380	315
Miscellaneous	APK Size	52,139	73	9,478	7,790

variables that can be expressed with real numbers. We also split the device attributes that consist of two properties into two different attributes. For instance, we split the device size into the height of the device and the width of the device. For boolean attributes, such as supporting OS upgrades or not, we convert them to nominal variables of 0 and 1. For attributes, such as GPU and CPU, we present them in order of computation power, based on the information that is available on the hardware review websites (e.g., GSM Arena).

2.2.5 Calculating the User-perceived Quality

To identify the user-perceived quality of an app, we use the star-ratings that are given by the users of each app. Table 6 provides an overview of our data. The columns in Table 6

Table 4 List of raw device attributes

Attributes		
2G network support	3G network support	SIM support
Announcement date	Status of availability	Dimensions
Weight	Size of display	Multitouch display
Display colors	Display protection	Loudspeaker
3.5mm jack	Vibration	Internal RAM
GPRS support	Hot-spot support	EDGE support
WLAN support	Bluetooth support	NFC
USB support	Primary camera details	Video recording support
Secondary camera support	OS version	Upgradable OS
CPU	GPU	Accelerometer support
Compass support	Messaging features	Browser features
GPS	Java support	Battery
Standby	Talk time	

Table 5 The device attributes

Category		Attribute	Description
Body	Dimension	DeviceHeight	Height of a device in centimeters.
		DeviceWidth	Width of a device in centimeters.
		DeviceThickness	Thickness of a device.
Display	Colors	DisplayColors	Range of colors that are supported by the screen.
	Size	DisplayInch	Display size in inches.
	Resolution	ScreenWidth	Display width in pixels.
		ScreenHeight	Display height in pixels.
		DisplayPPI	Pixels per inch is a measurement of the pixel density (resolution).
Platform	OS	OSVersion	Initial Android version.
		OSUpgradeable	Indicates whether any OS upgrade is announced by manufacturers after the initial release.
	CPU	CPU	How powerful the CPU is.
	GPU	GPU	How powerful the GPU is.
Memory	Intenal RAM	InternalRAM	Capacity of the internal RAM.
	Internal ROM	InternalROM	Capacity of the internal ROM.
Camera	Camera	CameraMegaPixel	Resolution of the primary camera in megapixels.
		Video	Resolution of recording video.
Battery	Battery Size	Battery	Battery size in ampere-hour.
	Talk time	Talktime	The officially-quoted longest time that a single battery charge lasts when a user is constantly talking on a device.
	Standby Time	Standby	The officially-quoted longest time that a single battery charge lasts when a device is constantly connected to the GSM network, but is not in active use.
Date	Release Date	Announced	The announcement date of device.

show the number of reviews per device as well as the mean, median, standard deviation, and skewness of the star-ratings, respectively.

3 Approach and Results

This section presents the approach and results of our three research questions. For each research question, we present the motivation behind the question, our analysis approach and a discussion of our findings.

Table 6 The number, mean, median, standard deviation (SD) and skew (Sk) of the user-perceived quality of each device (sorted by the number of reviews)

Device	#Reviews	Mean	Median	SD	Sk
Galaxy S3	22,022	3.4	4.0	1.5	−0.4
Galaxy S2	18,898	3.3	4.0	1.5	−0.4
Galaxy S	13,417	3.2	3.0	1.5	−0.3
Nexus 7	6,322	3.3	4.0	1.5	−0.3
Galaxy Nexus	5,764	3.0	3.0	1.5	−0.1
Droid RAZR	6,892	3.2	3.0	1.5	−0.2
Evo 4G	7,121	3.1	3.0	1.5	−0.1
Optimus One	6,503	3.3	4.0	1.5	−0.3
Galaxy Note	4,776	3.4	4.0	1.5	−0.5
Galaxy Y	5,224	3.6	4.0	1.4	−0.7
Droid X	4,670	3.0	3.0	1.5	0.0
Desire HD	4,211	3.2	3.0	1.5	−0.2
Galaxy Ace	4,390	3.4	4.0	1.5	−0.5
One X	2,889	3.3	4.0	1.5	−0.3
Sensation 4G	2,824	3.2	3.0	1.5	−0.2
EVO 3D	2,582	3.2	3.0	1.5	−0.2
Droid Bionic	2,946	2.8	3.0	1.6	0.2
Nexus S	2,175	3.1	3.0	1.5	−0.1
Desire	2,715	3.1	3.0	1.5	−0.1
Thunderbolt	2,312	3.1	3.0	1.5	−0.1
Galaxy Tab 10.1	2,067	3.3	4.0	1.5	−0.4
Droid	2,699	3.0	3.0	1.4	0.1
Droid Incredible	2,459	3.0	3.0	1.5	0.0
Galaxy Tab 10.1	2,338	3.4	4.0	1.5	−0.5
Droid X2	2,048	3.0	3.0	1.5	0.0
myTouch 4G	1,913	3.2	3.0	1.5	−0.3
Wildfire S	2,169	3.3	4.0	1.5	−0.3
Galaxy Mini	2,171	3.5	4.0	1.4	−0.6
Droid II	1,991	3.0	3.0	1.5	0.0
Incredible 2	1,865	3.1	3.0	1.5	−0.1

RQ1) How strong is the relationship between the user-perceived quality of apps and the devices on which they run?

Motivation Users can express the user-perceived quality of an Android app through the star-rating mechanism on a scale of one to five (Google 2015d). For example, a five star-rating means that a user believes that the app is of the highest quality. Earlier studies (e.g., Taba et al. 2014) report that app attributes have significant relationships with the user-perceived quality. In this research question, we explore if the user-perceived quality of an app is associated with the devices of the users who rated the app.

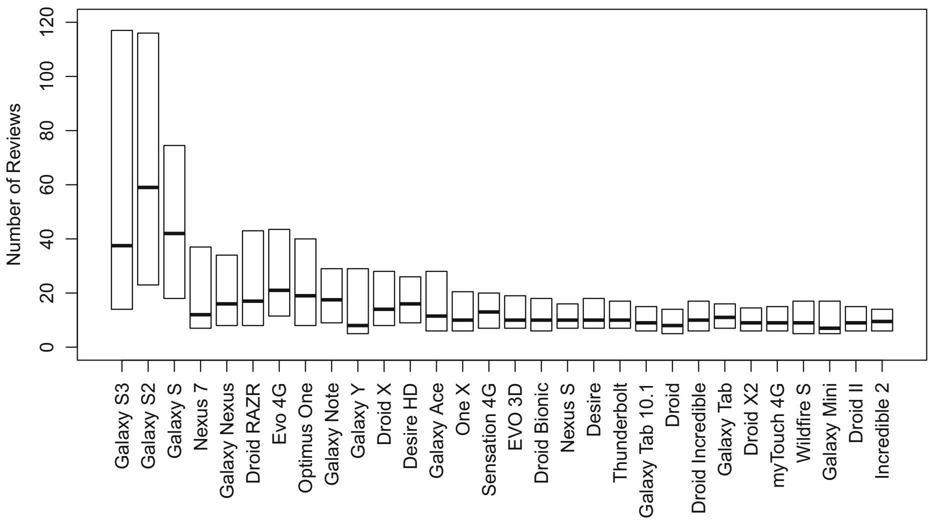


Fig. 2 A boxplot showing the number of reviews for each device

Approach To address this research question, we measure the user-perceived quality as observed by the users of the 30 mobile devices in our dataset. Figure 2 shows the distribution of the number of reviews per app for each device. As shown in Fig. 2, the *Galaxy S* series devices have more reviews per app than other devices in our dataset. Other devices have a smaller number of reviews per app. We compare the ratings of the same set of apps as rated by the owners of each device type. As a null hypothesis, we suppose that the distributions of ratings for the same set of apps running on different devices are the same. We use the Kruskal-Wallis test to check this hypothesis. The Kruskal-Wallis is a non-parametric method for testing whether samples originate from the same distribution. The test does not assume a normal distribution of the residuals as it is a non-parametric method. The Kruskal-Wallis extends the Mann-Whitney U test to test more than two groups (Kruskal and Wallis 1952). Hence, it shows whether different device types have the same distribution of ratings given the same set of apps. We can reject the null hypothesis if the observed p-value is less than 0.05. Hence, we would say that the differences in the app ratings for each device are significant and that users perceive different levels of quality for the same set of apps among at least one device type. We also use Dunn's test (Dunn 1964) to compare the user-perceived quality of each pair of devices and determine how many pairs of devices have statistically significant differences in the distributions of their user ratings. Finally, we draw boxplots to examine the distribution of the user-perceived quality for each device.

Findings **The user-perceived quality of the same set of apps changes per device.** The results of the Kruskal-Wallis test show that the distribution of the user-perceived quality for the same set of apps of at least one device is significantly different with a p-value of less than 2.2e-16 across the 30 devices. By using Dunn's test on the distribution of user-perceived quality across each pair of devices, we find that 81 % of the device pairs show a statistically significantly different distribution of user-perceived quality across the same set of apps. In addition, by calculating the Cliff's delta effect size (Cliff 1993), we find that

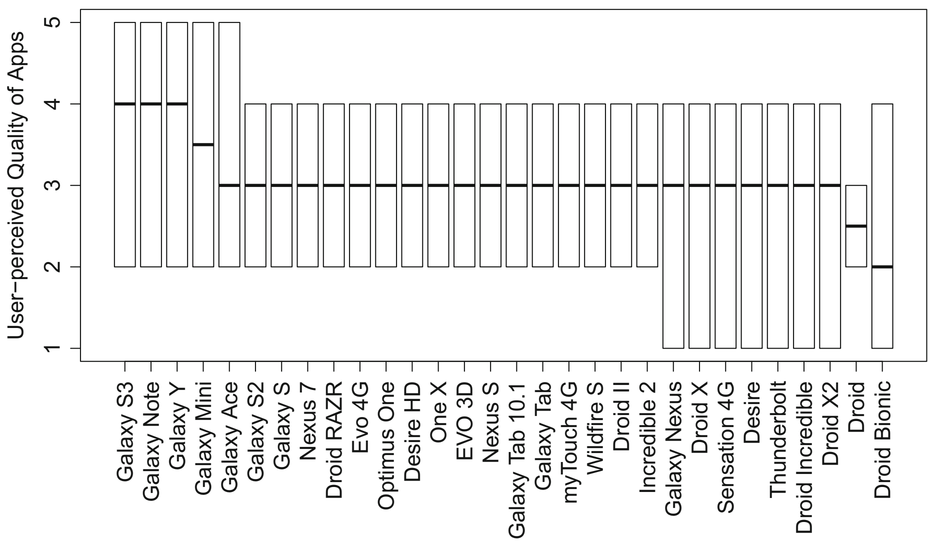


Fig. 3 A boxplot showing the user-perceived quality of apps per device

42 % of the device pairs show at least a small (effect size) difference. Figure 3 shows that the distributions of the user-perceived quality vary across devices. Hence, we conclude that not only do the apps themselves play a crucial role in the user-perceived quality (Taba et al. 2014), but also that the devices on which the apps are running play a significant role in the user-perceived quality.

The user-perceived quality of the same set of apps varies across devices. User of some devices perceive apps to be of higher quality in comparison with users of other devices.

RQ2) Which device attributes have the strongest relationship with the user-perceived quality?

Motivation Our findings in **RQ1** motivate us to investigate the relationships between the mobile devices and the user-perceived quality. In the first research question, we found that users of some devices perceive apps to be of higher quality compared to users of other devices for the same set of apps. However, a mobile device has several device attributes (e.g., CPU, internal RAM, and screen size). In this research question, we investigate whether different device attributes have statistically significant relationships with the user-perceived quality. Our goal is to find the device attributes that have a significant relationship with the user-perceived quality. The outcome of this research question helps both manufacturers and mobile app developers. Mobile device manufacturers can figure out the device attributes that have the most significant relationships with the user-perceived quality as estimated by their users. For instance, manufacturers can decide whether it is better to allocate more resources to advance the screen resolution or the battery capacity for their next generation devices. In addition, mobile app developers can understand the device attributes that have the most significant relationships with the user-perceived quality. Based on such knowledge, app developers might determine the devices for which they would like to make their app available, since the Google Play Store permits developers to limit an app to specific device types.

Approach We study the relationships between the user-perceived quality of an app and 20 device attributes, such as the display size and the internal RAM. We build a linear mixed effect model to study the device attributes that have the most significant relationships with user-perceived quality. The independent variables of our model are the various device attributes, and the dependent variable is the app rating. Table 5 shows the complete list of the device attributes that we study including a brief description of these attributes. We also normalize the independent variables so we can safely compare the coefficients.

Model Construction We identify the correlated variables since having correlated variables negatively affects the stability of our models and prevents us from understanding the full impact of each variable (i.e., attribute) (Harrell 2001). We use variable clustering analysis to build a hierarchical overview of the correlations between the explanatory variables (Harrell 2015). For sub-hierarchies of explanatory variables with correlation $|\rho| > 0.7$, we select only one variable from the sub-hierarchy for inclusion in our model. We pick the most simple and straightforward attribute. Figure 4 shows the hierarchical clustering of variables according to Spearman's $|\rho|$. The Spearman correlation evaluates the monotonic relationship between the continuous or ordinal variables. In Fig. 4, the horizontal line shows the threshold from which the variables are highly correlated with each other. From each sub-hierarchy of correlated variables, we select a representative independent variable to build our final model. We select the attributes that are simplest and easiest

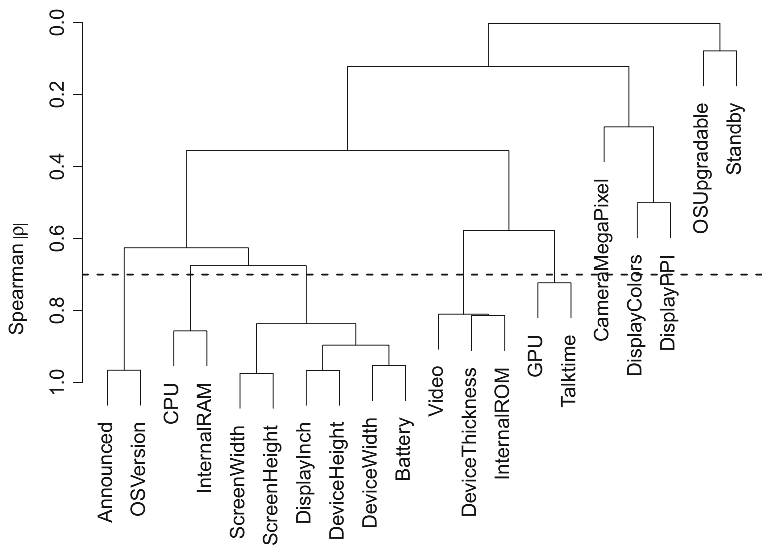


Fig. 4 Hierarchical clustering of device attributes according to Spearman's $|\rho|$

to compute and interpret. We select the following variables after the correlation analysis: OSVersion, CPU, DisplayInch, InternalROM, GPU, CameraMegaPixel, DisplayColors, DisplayPPI, OSUpgradable, and Standby.

Model Building We use a mixed effects model to investigate the relationships between the device attributes and the user-perceived quality of apps. A mixed effects model is a model that contains both fixed effects variables and random effects variables (Faraway 2005). Fixed effects variables are variables with constant coefficients and intercepts for every single observation while random effects variables vary across individual observations. Mixed effects models explain the relationships between a dependent variable and independent variables that are grouped according to one or more grouping factors. The models assume different intercepts for each group (Zuur et al. 2009).

Equation 1 shows a general equation for a mixed effects model with a grouping factor g. In Eq. 1, Y_g denotes the dependent variable; X_i represents the independent variables; $\beta_i + \theta_{1g}$ and β_i show the coefficients of each X_i; ϵ_g indicates the errors; β_0 demonstrates the constant intercept; and θ_{0g} shows the intercepts that vary across each group. If θ_{1g} equals zero, then only the intercepts can vary; if θ_{0g} equals zero, then only the coefficients can vary; and if both θ_{0g} and θ_{1g} equal zero, then the model would become a fixed effects linear model (as is commonly used is empirical software engineering research (Martin et al. to appear)).

$$Y_g = \beta_0 + \theta_{0g} + \sum_{i=1}^{m} \beta_i X_i + \sum_{i=m}^{n} (\beta_i + \theta_{1g}) X_i + \epsilon_g \tag{1}$$

We use mixed effects models, by letting the independent variables have constant coefficients (i.e., the independent variables are fixed effects), but have variable intercepts (i.e., the intercept is a random effect). We group our independent variables according to each device and app. In other words, we build a mixed effects model with ratings as the dependent variable, the 20 device attributes as the independent variables, and we add a random effect for every app and device type. Equation 2 shows the equation of our model. In Eq. 2, Y_g denotes the user-perceived quality of apps; X_i represents the independent variables; β_i show the coefficients of each X_i; and θ_g shows the intercepts that vary across each app and device type.

$$Y_g = \beta_0 + \theta_g + \sum_{i=1}^{n} \beta_i X_i + \epsilon_g \tag{2}$$

In contrast, traditional linear models are fixed effects models that have one or more fixed effects and an error ϵ (Draper et al. 1966). In mixed models, we can add one or more random effects. Mobile apps may behave differently on devices with similar device attributes. Likewise, mobile apps with the same app attributes may not behave similarly on the same mobile devices. Hence, we model the individual differences by assuming different random intercepts for each app and device. We can assign a different intercept for each app and device, and the mixed model can estimate these intercepts (Winter 2013). The random effects give structure to the error ϵ. In the case of our model, we add a random effect for each app and device type.

In mixed effects models, the marginal R^2 describes the proportion of variance that is explained by the fixed variables and the conditional R^2 describes the proportion of variance that is explained by both the fixed and random variables (Nakagawa and Schielzeth 2013). The marginal R^2 of the model is 0.010 in this research question, but the conditional R^2 is 0.190. The higher value of the conditional R^2 indicates that the proportion of variance that is explained by the fixed variables is much less than the proportion of variance that is

Table 7 Linear mixed effects model results based on device attributes, sorted by $p - value$

Attribute	Pr(>F)	Signif.	$\tilde{\chi}^2$	Estimate	Effect
OSVersion	0.0007	***	27.07	1.410e-01 ± 0.153	↘
DisplayPPI	0.0035	**	8.55	7.476e-02 ± 0.026	↘
CPU	0.0162	*	17.21	7.661e-02 ± 0.126	↗
Standby	0.0263	*	4.94	3.011e-04 ± 0.001	↗
CameraMegaPixel	0.0907	+	2.86	1.385e-02 ± 0.008	↘
InternalROM	0.0915	+	2.85	1.482e-02 ± 0.009	↘
GPU	0.2865		1.14	1.945e-02 ± 0.018	↘
DisplayInch	0.4197		0.65	3.020e-02 ± 0.037	↗
DisplayColors	0.7548		0.56	4.225e-02 ± 0.096	↗
OSUpgradable	0.9335		0.01	4.075e-03 ± 0.049	↘

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '+' 0.1 ' ' 1

explained by both fixed and random variables. This higher value indicates that the modeling of the random effects significantly helps to explain the user-perceived quality (indicating that such type of mixed effects modeling is needed for our dataset).

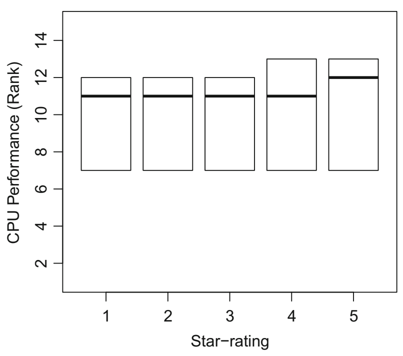
The significant variables are marked with asterisks in the output of the model (Table 7) using the ANOVA test (Pinheiro et al. 2007). These variables have $Pr(> |F|)$ less than 0.05. $Pr(> |F|)$ is the p-value that is associated with the F-statistic. The upward arrows indicate the variables with a positive relationship with the user-perceived quality of an app and the downward arrows indicate the variables with a negative relationship. The values of the $\tilde{\chi}^2$ (Chi-Squared) show if a model is statistically different from the same model in the absence of a given independent variable according to the degrees of freedom in our model.

Findings **Device attributes, such as the CPU and the display, have significant relationships with the user-perceived quality of apps.** Table 7 displays the results of our model. We identify that the OS version, the CPU, the display density (resolution), and the standby time have significant relationships with the user-perceived quality of an app. The CPU and the standby time have positive relationships with the user-perceived quality of apps, while the display density and the OS version have an inverse relationship with the user-perceived quality. First, we discuss the attributes with a positive relationship with the user-perceived quality in order of their statistically significant relationship. Second, we discuss the attributes with a negative relationship with the user-perceived quality in order of their statistically significant relationship.

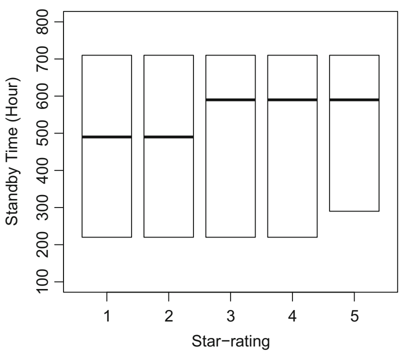
Device attributes with a positive relationship with the user-perceived quality of Apps

Figure 5 shows the values of the attributes that have a positive relationship with the user-perceived quality of apps. In this figure, we use boxplots to present the star-ratings that are associated with the values of these attributes. As we can see in Fig. 5a and b, the reviews that are associated with higher CPU performance (i.e., the CPU's rank in benchmark performance tests) and more standby time tend to be associated with higher star-ratings.

We find that powerful CPUs produce a higher user-perceived quality. Typically, the CPU is responsible for computations on devices, so it is reasonable that the CPU has a significant



(a) A boxplot showing CPU performance compared to user-perceived quality.



(b) A boxplot showing standby time compared to user-perceived quality.

Fig. 5 A boxplot showing standby time compared to user-perceived quality

effect on the user-perceived quality. As noted in Table 7, CPU relates to the user-perceived quality with $\tilde{\chi}^2 = 17.21$ (with 7 degrees of freedom) and $p - value = 0.0162$.

Standby is an upfront-derived specification that may not directly impact the user-perceived quality. Standby time quantifies the longest time that a battery charge will last when the phone is constantly connected to the network but is not in active use. One reason for standby having a positive relationship with the user-perceived quality could be the energy consumption of apps. One of the main energy consuming features of mobile apps is network usage (Li et al. 2014; Balasubramanian et al. 2009; Li and Halfond 2014). Li et al. report that the network is the most energy consuming component among all of the device components (Li et al. 2014). Moreover, Kaup and Hausheer indicate that the perceived quality can be affected by the data rates in the networks and the energy consumption (Kaup and Hausheer 2013). Hence, if an app over-consumes energy, it would create an inconvenient user experience for the users of devices with low standby time, and we can understand that the users who can use their phones for a longer time may run the apps with a higher quality than the users who need to recharge their batteries more often. Some user reviews support this claim, such as *"Why is this app using 7 % of my battery when not in use???"* and *"It is ok, but is takes up 18mb, not 4.5mb. Also, it massively drains the battery even when not in use and the phone is switched off. Uninstalled"*.

Device attributes with a negative relationship with the user-perceived quality of Apps

Table 7 shows that having a higher version of the installed OS has a significant inverse relationship with the user-perceived quality of apps. Moreover, we can see that the ability of the device to upgrade to a higher version of the OS has an inverse relationship with the user-perceived quality of apps too. One of the reasons for this observation can be API updates. Updating the Android OS requires the apps to update their API usage (Syer et al. 2011). Hence, new app bugs might emerge. These bugs are likely to impact the user-perceived quality negatively as McDonnell et al. show that a noticeable amount of apps are out-of-date (McDonnell et al. 2013). McDonnell et al. also mention that many developers are not fast enough in adapting new APIs, and they often fail to catch up with the pace of

Android API evolution. As Table 7, OS version relates to the user-perceived quality with $\tilde{\chi}^2 = 27.07$ (with 8 degrees of freedom) and $p - value = 0.0007$. Some user complaints, such as *"...since upgrading to Android 4.2.1, the messages will not update correctly displaying the..."* and *"After upgrade to ice cream sandwich, won't sync with Android Motorola bionic..."*, support this observation.

The display density has an inverse relationship with the user-perceived quality (i.e., the higher the density, the lower the user-perceived quality). Android has APIs that allow developers to precisely control the layout resources of apps for different screen sizes (Google 2015e). To further investigate the relationship between the display density and the user-perceived quality, we investigate the apps to see whether they have controlled their layout resources. 75, 81, and 93% of the apps in our dataset have a specific UI declaration for low-density, medium-density, and high-density screens, respectively. In addition, 63 and 99% of apps do not have a specific UI declaration for extra-high-density and extra-extra-high-density screens. Therefore, all the apps are not perfectly compatible with all the devices with different screen specifications. As examples of user reviews to support this observation, a user says that *"Scrolling and loading are much improved, but please update display size for larger phones."* and another review states that *"...says my resolution isn't high enough, then won't let me tap a single button. Bad app."*.

Device attributes, such as the CPU, have significant relationships with the user-perceived quality. However, having a better characteristics of an attribute, such as a higher display resolution, does not necessarily have a positive relationship with the user-perceived quality of apps.

RQ3) Do device attributes have a stronger relationship with the user-perceived quality than app attributes?

Motivation In **RQ2**, we explored the relationship between the device attributes and the user-perceived quality. However, the user-perceived quality reported by a user reflects his experience with using a mobile app. Therefore, we cannot ignore the role of the app itself on the overall user-perceived quality. In this research question, we study and compare the relationship between both the device and app attributes and the user-perceived quality. We aim to investigate whether app attributes have more significant relations with the user-perceived quality than device attributes. The results of this research question may be beneficial to app developers. They can decide the devices on which they would like to make their app available. The Google Play Store permits developers to specify the availability of an app for each specific device (Google 2015b). By limiting an app to specific devices, app developers can reduce the number of negative reviews, such as reviews from possibly underpowered devices. The developers can also devote additional resources to testing their apps on certain devices (Khalid et al. 2014).

Approach We study the relationship between the user-perceived quality and both the device and app attributes. We include the app attributes, such as the number of UI inputs in addition to device attributes. We include UI attributes because some device attributes, such as display size, have an inverse relationship with the user-perceived quality. Moreover, users

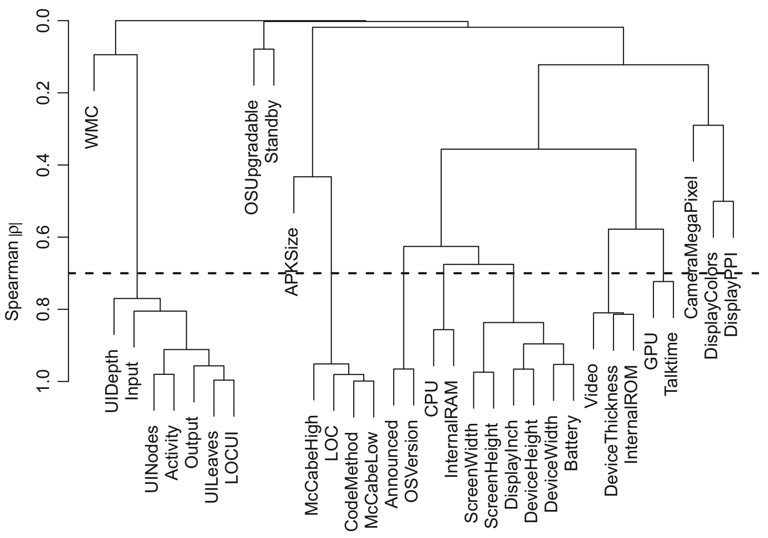


Fig. 6 Hierarchical clustering of app and device attributes according to the Spearman's $|\rho|$

have interaction with the UI elements of apps. Table 2 shows the list of the app attributes that we study including a brief description of the attributes.

Model Construction Figure 6 shows the result of the hierarchical clustering of variables according to Spearman's $|\rho|$. Based on Fig. 6, all the UI attributes, such as the number of inputs and outputs, the number of activities, and the lines of UI code (LOCUI), are correlated with each other. We select the total number of inputs as the representative attribute for the group that contains UI complexity attributes. From the other groups, we select WMC, OSUpgradable, Standby, APKSize, LOC, OSVersion, CPU, DisplayInch, InternalROM, GPU, CameraMegaPixel, DisplayColors, and DisplayPPI.

Model Building To determine the device and app attributes that have statistically significant relationships with the user-perceived quality, we build a linear mixed effects model (Faraway 2005) similar to **RQ2**, but with independent variables that are a combination of both the device and app attributes. The marginal R^2 of this model is 0.013 and the conditional R^2 is 0.191. The higher value of the conditional R^2 indicates that the proportional of variance that is explained by the fixed variables is much less than the proportional of variance that is explained by both fixed and random variables.

Findings **Both the app and the device attributes have significant relationships with the user-perceived quality of apps.** Table 8 shows the mixed effects model. We note that LOC has a significant relationship with the user-perceived quality. This observation could be the result of the relationship between the app maintainability and the code size (Li and Henry 1993; Albrecht and Gaffney Jr 1983). We find that LOC has a relationship with the user-perceived quality with $\tilde{\chi}^2 = 36.90$ and $p - value = 4.892e - 5$.

We also note that the WMC has a significant relationship with the user-perceived quality. One possible explanation for this observation could be that a larger development effort is required to maintain complex apps (Banker et al. 1993; Bandi et al. 2003).

Table 8 Linear mixed effects model results based on both device and app attributes, sorted by $p - value$

Category	Attribute	Pr(>F)	Signif.	$\tilde{\chi}^2$	Estimate	Effect
App	LOC	1.326e-09	***	36.90	3.343e-02 ± 0.0050	↘
	WMC	4.892e-05	***	16.51	1.129e-02 ± 0.0003	↘
	Input	0.0205	*	5.37	2.150e-02 ± 0.0093	↗
	APKSize	0.2710		1.21	5.619e-03 ± 0.0051	↗
Device	OSVersion	0.0005	***	27.78	1.406e-01 ± 0.1253	↘
	DisplayPPI	0.0025	**	9.13	7.688e-02 ± 0.0254	↘
	CPU	0.0136	*	17.73	7.638e-02 ± 0.1258	↗
	Standby	0.0276	*	4.86	2.971e-04 ± 0.0001	↗
	CameraMegaPixel	0.0778	+	3.11	1.438e-02 ± 0.0008	↘
	InternalROM	0.0829	+	3.00	1.516e-02 ± 0.0087	↘
	GPU	0.2876		1.13	1.932e-02 ± 0.0182	↘
	DisplayInch	0.4301		0.62	2.939e-02 ± 0.0372	↗
	DisplayColors	0.7300		0.63	4.632e-02 ± 0.0952	↗
	OSUpgradable	0.9071		0.01	5.666e-03 ± 0.0486	↘

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘+’ 0.1 ‘ ’ 1

Figure 6 shows the correlated attributes. From the app category, we find that all the UI attributes are correlated with each other. The APKSize and LOC are correlated, but not highly, because an APK file contains all the content that is required for both the installation and the execution of an app. An APK file also includes all the resources of an app such as graphical images. From the device category, we find that the internal RAM is highly correlated with the CPU, which in turn has a positive relationship with the user-perceived quality of apps.

Recalling the observations of **RQ2** and the result of this RQ, we can see that DisplayInch has a positive relationship with the user-perceived quality of apps. On the other hand, we can see that the number of UI inputs has a significant relationship with the user-perceived quality of apps. We also explore the relationship between the number of UI inputs and the display size using a nested mixed effects model (Pinheiro and Bates 2006). The results of the nested mixed effects model are in line with our findings. Users may perceive apps to be of higher quality with more number of UI inputs, and having a larger size screen could be an asset.

We notice that LOC (an app attribute) and OS version (a device attribute) both have significant relationships with the user-perceived quality. We find that other device attributes, such as the CPU and the screen resolution, play a significant role in the user-perceived quality. In other words, the characteristics of both the apps and the devices on, which they run, have statistically significant relationships with the user-perceived quality of the apps.

App developers should be more concerned about both the attributes of their own apps, and the attributes of the devices that are available in the market. For example, app developers can make their apps available to the devices that could deliver a higher user-perceived quality based on the attributes of the devices in the market. Moreover, app testers can prioritize their testing activities. In addition, mobile device manufacturers should consider the app attributes that have the most significant relationships with the user-perceived quality of apps.

Both the app and the device attributes have significant relationships with the user-perceived quality. However, some device attributes, such as the CPU and the display resolution, have relationships with the user-perceived quality more significantly than other app attributes, such as the number of UI inputs.

4 Threats to Validity

In this section, we discuss the threats that may influence the validity of our findings.

4.1 Construct Validity

Threats to construct validity concern whether theoretical constructs are measured and interpreted correctly (Shull et al. 2007).

4.1.1 Measuring the User-perceived Quality

Other measures may be used to capture the user-perceived quality. For example, the number of downloads shows the interest of users in the app. Harman et al. show that there is a strong relationship between the number of downloads and star-ratings (Harman et al. 2012). Therefore, we believe that the star-rating is a good measure of user-perceived quality. Nevertheless, future research might consider other measures to operationalize the concept of user-perceived quality.

4.1.2 Measuring Time and Location Dependent Attributes

The attributes of a particular device may be dependent on the age and location of the device. For example, the device attributes for battery (e.g., battery size, talk time, and standby time) degrade over time; as the device ages, the talk time and standby time decreases. Device attributes for network support (e.g., whether the device supports 2G, 3G, or 4G connectivity) may not be available in practice if such networks are not accessible from the user's locations. In this study, we consider the attributes that are available in a non-controlled empirical study.

4.1.3 Defining the UI Elements

We follow previous work (e.g., Taba et al 2014) and best practices (Google 2015c) for defining UI elements. With the best practices for UI implementation, developers need to declare the layouts of apps in XML files. An XML layout defines a human-readable visual structure for a user interface. In our initial dataset, only 7 % of the apps do not follow best practices to declare the UI elements.

4.1.4 Other Possible Attributes

As the first study of its kind that investigates device attributes, we expect that future studies will examine other (sub)attributes. For example, the manufacturer of a mobile device can be considered as another attribute in our models. We did explore this attribute in the models for RQ2 and RQ3. However, we found the manufacturer attribute to be statistically trivial.

4.2 Internal Validity

Threats to internal validity concern our selection of subject systems and analysis methods (Shull et al. 2007).

4.2.1 Precision of the Device Attributes

We extracted the device attributes as precise as possible. For ordinal attributes, such as CPU, we carefully ordered them based on the information available on the Internet.

4.2.2 Selecting the Apps

We select a set of apps with at least five reviews for each of the 30 devices from our dataset. The threshold of five reviews protects our results from being skewed by the set of apps and devices that have a small number of reviews.

4.2.3 Measuring the App Attributes

The app attributes that we study may correspond to a different version than the one for which the app rating was intended to. For example, a user may have multiple devices and has not kept them up to date, or has updated to a newer version, but mentally reviews based on previous versions. Unfortunately, we cannot find the actual versions of the apps that each user has reviewed based on those versions of the apps. Future studies should investigate this issue further.

4.3 External Validity

Threats to external validity concern the possibility of generalizing our results.

4.3.1 Ecosystem

The apps that we study are all from the Google Play Store. Therefore, our results may not generalize to other ecosystems, such as the iTunes store (Apple 2015). However, the variances across device attributes in such a store are not as large as the Google Play Store. Future studies should investigate other ecosystems and stores.

4.3.2 Paid Versus Free Apps

Our study is performed on free apps. Hence, our results may not generalize to paid apps. We focus on free to download apps as it reduces the impact of other confounding factors, such as the variance of user expectations based on the cost of an app (Harbach et al. 2014).

4.3.3 *Age of our Dataset*

The outdated dataset might affect the generalizability of the results to reviews occurring after 2013. However, the key message of our work is that there exist some apps where the user-perceived quality has significant relationships with both the device attributes and the app attributes.

4.3.4 *Type of the Apps*

Intuitively, certain types of apps, in particular games which are traditionally hardware taking, may be more dependent on device attributes. To investigate this issue, we add an interaction term to allow the coefficients in our model to vary depending on whether we are modeling the user-perceived quality of a game app or a non-game app. We found that the interaction terms for GPU and OSVersion are statistically significant. Hence, games might be more dependent on the GPU and the OSVersion of devices than other apps. However, we were not able to improve on our model fit. Therefore, we further built two separate models: (a) one model for game apps, and (b) another model for non-game apps. Again, we were not able to improve our model fit indicating that a *general* model of user-perceived quality performed better than more specific models, such as a model that is built only based on game apps. Our inability to improve on our model fit may be due to (a) some games might be more dependent on device attributes than other games (e.g., a complex graphics-intensive game, such as Clash of Clans, compared to a simpler game, such as Sudoku), or (b) user-perceived quality does not vary by app type.

4.3.5 *Discussion of our Findings*

For each research question, we dug into the data to provide some context to our findings (e.g., the maintainability of the large codes). We are not claiming an exhaustive list of relationships between the suggested causes and the user-perceived quality.

4.4 Reliability

We provide the dataset and the scripts that we used to do the analyses at Noei et al. (2016).

5 Related Work

In this section, we summarize the related work along three research directions: (a) user studies to determine the attributes that influence the user-perceived quality; (b) empirical studies to determine the app attributes that have relationships with the user-perceived quality; and (c) information extraction from reviews, such as generating feature requests from the user comments.

5.1 User Studies of User-perceived Quality

Researchers have explored the user-perceived quality based on user studies. Ickin et al. focus on measuring user experience for a set of mobile apps that are used on a daily basis (Ickin et al. 2012). Ickin et al. observe that factors, such as interface design, app performance, and user lifestyle, influence the user-perceived quality. Hassenzahl and Tractinsky show the attributes of the system and the context within which an interaction occurs influence the user-perceived quality (Hassenzahl and Tractinsky 2006). Korhonen et al. survey the literature and conclude that the mobile device, user's task, and social context are the most significant attributes affecting the user-perceived quality (Korhonen et al. 2010). Our work is an empirical study based on the data gathered from 280 apps and 30 devices and 150,373 users' reviews over a four year period, from 2009 to 2013. Our study covers a longer period in comparison with user studies. Moreover, the results of prior user studies have been inferred from a limited group of users and a fewer number of apps. Nevertheless, we believe that there is a need for additional user studies to further investigate many of the observations in our paper.

5.2 Empirical Studies of App Attributes

Recent empirical studies on mobile apps mainly focus on examining app attributes that affect the user-perceived quality (Martin et al. to appear). Nevertheless, we study the device attributes in addition to the app attributes. Taba et al. studied the relationship between UI complexity and user-perceived quality of Android apps (Taba et al. 2014). They observe that the UI complexity has a relationship with the user-perceived quality of apps. Linares-Vásquez et al. find that the low quality APIs affects the ratings of apps (Linares-Vásquez et al. 2013). Kim et al. discuss how ratings influence the users' decision to download apps (Kim et al. 2011). Khalid et al. use mobile app reviews to help determine whether developers can use a small subset of devices to reduce the cost of testing their apps (Khalid et al. 2014). Shabtai et al. apply machine learning techniques on Android apps using app attributes and categorize Android apps into two categories of games and tools (Shabtai et al. 2010). Bavota et al. indicate that the user-perceived quality of an app is related with the fault- and change-proneness of the APIs used by such an app (Bavota et al. 2015). Syer et al. studied the defects in source code and the relationship between app attributes and source code quality (Syer et al. 2013, 2014). However, the authors did not consider the user-perceived quality. Such work has not studied device and app attributes together to see the relative contribution of these attributes on modeling the user-perceived quality of apps. One of the contributions of our work is to show such a comparison between app and device attributes. Otherwise, we would not know which one aspect (app or device) has a greater contribution. Our work is the first to ever uncover the impact of device attributes, such as CPU, OS Version, and screen resolution on the user-perceived quality (even after controlling for app attributes).

5.3 Information Extraction

Prior work attempts to extract valuable knowledge from reviews using various techniques, such as natural language processing and latent semantic analysis, to facilitate mobile app development. Pagano and Maalej study reviews from iOS apps to determine potentials for the requirements engineering processes (Pagano and Maalej 2013). Iacob and Harrison, and Galvis Carreño and Winbladh extract feature requests from the user reviews (Iacob and Harrison 2013; Galvis Carreño and Winbladh 2013). Moran et al. present a solution to automate the completion of the reproduction steps when reporting a bug (Moran et al. 2015). None of the earlier work has compared the relationship between both app and device attributes and the user-perceived quality of apps. In this paper, we find that both app and device attributes have significant relationships with the user-perceived quality, and both are needed to have a comprehensive understanding about the user-perceived quality of an app.

6 Conclusion

In this paper, we study the relation of the device attributes and app attributes with the user-perceived quality. Our analysis is based on 30 Android mobile devices, 280 Android apps, and 150,373 reviews. We find that the user-perceived quality of an app varies across devices. Device attributes, such as the CPU, display, and OS, have significant relationships with the user-perceived quality. Nevertheless, having higher specifications for a particular device attribute, such as a higher display resolution, does not always have a positive relationship with the user-perceived quality of the apps. Both app and device attributes play an important role in the user-perceived quality of the apps that we studied.

Like other empirical studies, our study has several limitations (cf. Section 4). Nevertheless, we wish to emphasize that we do not claim that our results will necessarily generalize to all Android mobile devices and all Android apps. Instead, the key message of our work is that the user-perceived quality of some apps is impacted by both the attributes of the device on which the apps are being used, and the attributes of the apps. Hence, we recommend that app developers carefully consider this fact when enabling which devices can download their app through the app market. Furthermore, our results highlight the rather complex nature of the star-ratings which empirical research has started to explore in-depth in recent years (Martin et al. to appear). The limited access to information about the model of the device which submitted a particular app is another bias (Martin et al. 2015) that should be carefully considered in future studies of mobile app reviews. Future studies should also explore other aspects and attributes, such as contextual conditions (e.g., the location of users). We believe that such aspects may also exhibit relationships with the user-perceived quality of apps.

References

Albrecht AJ, Gaffney Jr JE (1983) Software function, source lines of code, and development effort prediction: a software science validation. IEEE Trans on Softw Eng 9(6):639–648

Apple (2015) itunes. https://www.apple.com/itunes/

Balasubramanian N, Balasubramanian A, Venkataramani A (2009) Energy consumption in mobile phones: a measurement study and implications for network applications. In: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement, ACM, pp 280–293

Bandi RK, Vaishnavi K, Turk DE (2003) Predicting maintenance performance using object-oriented design complexity metrics. IEEE Trans Softw Eng 29(1):77–87

Banker RD, Datar SM, Kemerer CF, Zweig D (1993) Software complexity and maintenance costs. Commun ACM 36(11):81–94

Bavota G, Linares-Vasquez M, Bernal-Cardenas CE, Penta MD, Oliveto R, Poshyvanyk D (2015) The impact of api change-and fault-proneness on the user ratings of android apps. IEEE Trans Softw Eng 41(4):384–407

Chidamber SR, Kemerer CF (1994) A metrics suite for object oriented design. IEEE Trans Softw Eng 20(6):476–493

Cliff N (1993) Dominance statistics: Ordinal analyses to answer ordinal questions. Psychol Bull 114(3):494

Coppick JC, Cheatham TJ (1992) Software metrics for object-oriented systems. In: Proceedings of the 1992 ACM annual conference on communications, ACM, pp 317–322

Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to algorithms. MIT press

Costa-Montenegro E, Barragáns-Martínez AB, Rey-López M (2012) Which app? a recommender system of applications in markets: implementation of the service for monitoring users' interaction. Expert Syst Appl 39(10):9367–9375

Dex2jar J (2016) Java decompiler. http://jd.benow.com/

Draper NR, Smith H, Pownell E (1966) Applied regression analysis, vol 3. Wiley, New York

Dunn OJ (1964) Multiple comparisons using rank sums. Technometrics 6(3):241–252

Faraway JJ (2005) Extending the linear model with R: generalized linear, mixed effects and nonparametric regression models. CRC press

Galvis Carreño LV, Winbladh K (2013) Analysis of user comments: an approach for software requirements evolution. In: Proceedings of the 35th international conference on software engineering, IEEE, ICSE '13, pp 582–591

Google (2015a) Application fundamentals. http://developer.android.com/guide/components/fundamentals.html/

Google (2015b) Google play store. http://play.google.com/

Google (2015c) Layouts. http://developer.android.com/intl/ru/guide/topics/ui/declaring-layout.html/

Google (2015d) See your app's ratings & reviews. https://support.google.com/googleplay/android-developer/answer/138230

Google (2015e) Supporting multiple screens. http://developer.android.com/guide/practices/

Han D, Zhang C, Fan X, Hindle A, Wong K, Stroulia E (2012) Understanding android fragmentation with topic analysis of vendor-specific bugs. In: Proceedings of the 19th working conference on reverse engineering, IEEE, pp 83–92

Harbach M, Hettig M, Weber S, Smith M (2014) Using personal examples to improve risk communication for security & privacy decisions. In: Proceedings of the 32nd annual ACM conference on human factors in computing systems, ACM, pp 2647–2656

Harman M, Jia Y, Zhang Y (2012) App store mining and analysis: Msr for app stores. In: Proceedings of the 9th IEEE working conference on mining software repositories, IEEE, MSR '12, pp 108–111

Harrell (2015) Harrell. http://cran.r-project.org/web/packages/Hmisc/index.html

Harrell FE (2001) Regression Modeling Strategies: With Applications to Linear Models, Logistic Regression, and Survival Analysis. Springer

Hassenzahl M, Tractinsky N (2006) User experience - a research agenda. Behav Inform Technol 25(2):91–97

Iacob C, Harrison R (2013) Retrieving and analyzing mobile apps feature requests from online reviews. In: Proceedings of the 10th working conference on mining software repositories, IEEE, MSR '13, pp 41–44

Ickin S, Wac K, Fiedler M, Janowski L, Hong JH, Dey AK (2012) Factors influencing quality of experience of commonly used mobile applications. IEEE Commun Mag 50(4):48–56

Dex2ja J (2016) Java decompiler. http://jd.benow.com/

Johnson RE, Foote B (1988) Designing reusable classes. Object-oriented Program 1(2):22–35

Kaup F, Hausheer D (2013) Optimizing energy consumption and qoe on mobile devices. In: Proceedings of the 21st IEEE international conference on network protocols, IEEE, pp 1–3

Khalid H, Nagappan M, Shihab E, Hassan AE (2014) Prioritizing the devices to test your app on: a case study of android game apps. In: Proceedings of the 22nd ACM SIGSOFT international symposium on the foundations of software engineering, pp 370–379

Kim HW, Lee H, Son J (2011) An exploratory study on the determinants of smartphone app purchase. In: Proceedings of the 11th international decision science institute and the 16th Asia pacific decision sciences institute joint meeting

Korhonen H, Arrasvuori J, Väänänen-Vainio-Mattila K (2010) Analysing user experience of personal mobile products through contextual factors. In: Proceedings of the 9th international conference on mobile and ubiquitous multimedia, ACM, New York, NY, USA, MUM '10, pp 11:1–11:10

Kruskal WH, Wallis WA (1952) Use of ranks in one-criterion variance analysis. J Am Stat Assoc 47(260):583–621

Lawson S (2009) Android market needs more filters, t-mobile says. http://www.pcworld.com/article/161410/article.html

Li D, Halfond WG (2014) An investigation into energy-saving programming practices for android smartphone app development. In: Proceedings of the 3rd international workshop on green and sustainable software, ACM, pp 46–53

Li D, Hao S, Gui J, Halfond WG (2014) An empirical study of the energy consumption of android applications. In: Proceedings of the 30th IEEE international conference on software maintenance and evolution, IEEE, pp 121–130

Li W, Henry S (1993) Object-oriented metrics that predict maintainability. J Syst Softw 23(2):111–122

Linares-Vásquez M, Bavota G, Bernal-Cárdenas C, Di Penta M, Oliveto R, Poshyvanyk D (2013) Api change and fault proneness: a threat to the success of android apps. In: Proceedings of the 9th joint meeting on european software engineering and foundations of software engineering, ACM, ESEC/FSE 2013, pp 477–487

Martin W, Harman M, Jia Y, Sarro F, Zhang Y (2015) The app sampling problem for app store mining. In: Proceedings of the 12th working conference on mining software repositories, IEEE, pp 123–133

Martin W, Sarro F, Jia Y, Zhang Y, Harman M (to appear) A survey of app store analysis for software engineering. IEEE Trans Softw Eng

McCabe TJ (1976) A complexity measure. IEEE Trans Softw Eng SE 2(4):308–320

McDonnell T, Ray B, Kim M (2013) An empirical study of api stability and adoption in the android ecosystem. In: Proceedings of the 29th IEEE international conference on software maintenance, IEEE, pp 70–79

Miecznikowski J, Hendren L (2002) Decompiling java bytecode: problems, traps and pitfalls. In: Horspool R (ed) Proceedings of the compiler construction, lecture notes in computer science, vol 2304. Springer, Berlin Heidelberg, pp 111-127

Moran K, Linares-Vásquez M, Bernal-Cárdenas C, Poshyvanyk D (2015) Auto-completing bug reports for android applications. In: Proceedings of the 10th joint meeting on european software engineering and foundations of software engineering, ACM, New York, NY, USA, ESEC/FSE 2015, pp 673–686

Nakagawa S, Schielzeth H (2013) A general and simple method for obtaining r2 from generalized linear mixed-effects models. Methods Ecol Evol 4(2):133–142

Nielson F, Nielson HR, Hankin C (1999) Principles of Program Analysis. Springer-Verlag New York, Inc., Secaucus, NJ, USA

Noei E, Syer MD, Zou Y, Hassan AE, Keivanloo I (2016) A study of the relation of mobile device attributes with the user-perceived quality of android apps, http://sailhome.cs.queensu.ca/replication/mobile_device_attributes/

Pagano D, Maalej W (2013) User feedback in the appstore: an empirical study. In: Proceedings of the 21st IEEE international requirements engineering conference, IEEE, pp 125–134

Pinheiro J, Bates D (2006) Mixed-effects models in S and S-PLUS. Springer Science & Business Media

Pinheiro J, Bates D, DebRoy S, Sarkar D et al. (2007) Linear and nonlinear mixed effects models. R package version 3:57

Selenium (2014) Selenium - web browser automation. http://seleniumhq.org/

Shabtai A, Fledel Y, Elovici Y (2010) Automated static code analysis for classifying android applications using machine learning. In: Proceedings of the 2010 international conference on computational intelligence and security. IEEE, pp 329–333

Shull F, Singer J, Sjøberg DI (2007) Guide to Advanced Empirical Software Engineering. Springer-Verlag New York, Inc., Secaucus, NJ, USA

Stats A (2016) Number of android applications. http://www.appbrain.com/stats/number-of-android-apps

Syer MD, Adams B, Zou Y, Hassan AE (2011) Exploring the development of micro-apps: a case study on the blackberry and android platforms. In: Proceedings of the 11th IEEE international working conference on source code analysis and manipulation, IEEE, pp 55–64

Syer MD, Nagappan M, Hassan AE, Adams B (2013) Revisiting prior empirical findings for mobile apps: an empirical case study on the 15 most popular open-source android apps. In: Proceedings of the conference of the center for advanced studies on collaborative research, pp 283–297

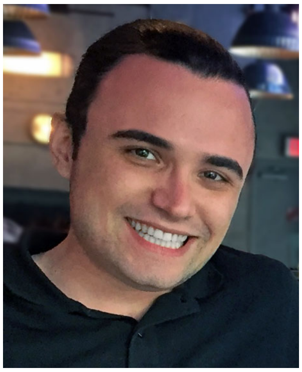
Syer MD, Nagappan M, Adams B, Hassan AE (2014) Studying the relationship between source code quality and mobile platform dependence. Softw Qual 23(3):485–508

Taba SES, Keivanloo I, Zou Y, Ng JW, Ng T (2014) An exploratory study on the relation between user interface complexity and the perceived quality. In: Proceedings of the 14th international conference on Web engineering, pp 370–379

Wasserman AI (2010) Software engineering issues for mobile application development. In: Proceedings of the FSE/SDP workshop on future of software engineering research, ACM, New York, NY, USA, FoSER '10, pp 397–400

Winter B (2013) A very basic tutorial for performing linear mixed effects analyses. arXiv:13085499

Zuur A, Ieno EN, Walker N, Saveliev AA, Smith GM (2009) Mixed effects models and extensions in ecology with R. Springer Science & Business Media



Ehsan Noei is a Ph.D. candidate at Queen's University. He received his M.Sc. in Software Engineering from Sharif University of Technology. His primary research interests include software engineering, software re-engineering, and mining software repositories. More about Ehsan is available online at http://www.noei.net/.

Mark D. Syer is a Data Scientist at Maple Leaf Sports and Entertainment. He received his Ph.D. and M.Sc. from the Software Analysis and Intelligence Lab at Queen's University. His research interests include software and systems engineering, software performance engineering, mining software repositories and empirical software engineering. He is also committed to the rigorous application of statistics and machine learning to software engineering and systems research. His work has been published at premier venues such as ICSE and ICSME and in major journals such as TSE, EMSE and ASE. His industrial experience includes improving the scalability, reliability and performance of large-scale software systems at SAP, BlackBerry and Dragonfly IT. Early tools and techniques developed by him are already used in daily industrial practice.

Ying Zou is a Canada Research Chair in Software Evolution. She is an associate professor in the Department of Electrical and Computer Engineering and cross-appointed to the School of Computing at Queen's University in Canada. She is a visiting scientist of IBM Centers for Advanced Studies, IBM Canada. Her research interests include software engineering, software re-engineering, software reverse engineering, software maintenance, and service-oriented architecture.

Ahmed E. Hassan is the Canada Research Chair (CRC) in Software Analytics, and the NSERC/BlackBerry Software Engineering Chair at the School of Computing at Queen's University, Canada. His research interests include mining software repositories, empirical software engineering, load testing, and log mining. Hassan received a Ph.D. in Computer Science from the University of Waterloo. He spearheaded the creation of the Mining Software Repositories (MSR) conference and its research community. Hassan also serves on the editorial boards of IEEE Transactions on Software Engineering, Springer Journal of Empirical Software Engineering, and PeerJ Computer Science. Contact ahmed@cs.queensu.ca. More information at: http://sail.cs.queensu.ca/.

Dr. Iman Keivanloo worked as a Postdoctoral researcher in the Department of Electrical and Computer Engineering at Queens University, Canada. He received his PhD from Concordia University, Montreal, Canada in 2013. His research focuses on the area of source-code search and recommendation. He has published over 20 papers in major refereed international journals, conferences and workshops. He has also served as a committee member on international conferences and workshops in the area of clone detection and program comprehension.