Abstract

This study seeks to understand the relationship between three key input constructs in app development, namely code originality, interdependence with other apps, and app complexity and an outcome construct, app quality. This is a largely unexplored area of research in both the computer science and MIS fields. Source code of 100 Android apps was analyzed to determine originality of source code, and API usage, as well as output data on user reviews and number of downloads. App originality was found to have a negative association with user reviews, and a negative association moderated by app complexity with number of downloads. App interdependence was found to have a positive association with highly-downloaded apps compared against low-downloaded apps.

Keywords

Apps, code originality, code quality, software success

Introduction

With the rising popularity of smartphones, software developers’ attention has been drawn to mobile application (app) development. Successful apps generate profit for their developers both through sales and by collecting user information which can be resold (Chandy and Gu 2012). A recent increase in research on app-related topics has emphasized user experiences and perspectives, such as user perspectives of mobile app ratings (Khalid 2013), factors related to user reviews of low rated apps (Vasa et al. 2012), and the effects of rewarding user engagement on app quality (Claussen et al. 2013). However, there is little research on factors contributing to app quality and its relationship to app design. Since the mobile market is competitive, app development cycles are kept short so that product updates can be brought to market frequently (Franke and Weise 2011). In contrast to traditional development, app developers must deal with various mobile-related challenges such as signal strength and restricted user interfaces. Such circumstances demand specific approaches for mobile app development to assure high quality of the final product. Although software for mobile devices differs from its desktop counterpart, the improved capability of mobile devices has led to growing similarities between desktop and mobile programming techniques, resulting in similar source code. Thus, researchers have applied established desktop software concepts such as software quality to mobile application development (Franke and Weise 2011).

For this study, we advance this approach by extending the existing constructs of ‘code reuse’ and ‘software quality’ from the software development literature and apply them to mobile application development and quality. The development of apps is reliant on the integration of available resources through application programming interfaces (APIs). For example, if an app requires a touchscreen interface a developer will include an API call in the coding which allows the app to utilize an external touchscreen interface. The result should be more efficient coding and improved stability as the external touchscreen interface is...
already written, tested, and used in other apps. Using APIs that are less prone to faults than custom coding increases the expected benefit to app developers and users. We propose that app source code structure has an effect on app quality. We analyzed the source code of 100 Android apps and examined the relationship between source code structure and an app’s quality. Specifically, we analyzed the source code to understand how app code originality (the percentage of an app’s lines of code not related to external calls) and code interdependence (the number of API uses in an app) are related to the app quality. We argue that the complexity of code structure may affect these relationships. Thus, we postulate our research question as: ‘Does the structure of an app’s source code contribute to its quality?’

Literature Review

App Development

The inherent limitations of mobile devices – limited memory, storage, processing capacity, and display resources – require efficient, stable coding for apps (Syer et al. 2011). With limited research on app development in existence, the literature on software reuse provides context for research in this field. Software reuse, the process of developing software from existing software artifacts rather than constructing a new design (Krueger 1992), is one of the most desirable properties of high-quality software as it reduces costs, development time and developers’ cognitive efforts in the coding process (Dantas and Garcia 2010). There are various approaches to software reuse. McIlroy (1968) introduced the notion of source code components, which serve as building blocks in the development of a larger software system. Examples of such units in object-oriented programming languages include modules and classes (Krueger 1992). In app development, the notion of source code reuse is evolving. Where early apps were developed by phone manufacturers and network operators, the emergence of smartphones in the late 2000s has brought many independent developers into the market (Holzer and Ondrus 2009). Apps are developed on a mobile platform using a software development kit (SDK) which includes debuggers, libraries, and handset emulators (Holzer and Ondrus 2009). Once developed, apps are published on a marketplace portal for customers to download.

To ensure compatibility, apps access a mobile device’s hardware using APIs provided by the platform. Through APIs, developers reuse commonly required functionality to speed up development (Ruiz et al. 2012; Syer et al. 2015). The source code of all apps on a given platform includes a certain number of API uses to ensure functionality with the platform. Consistent with McIlroy’s notion of source code component reuse, reusing API classes in app development creates advantages for developers and users. Mohagheghi et al. (2004) explain that since reused components are modified less, reuse of source code components reduces faults in the code and improves app reliability. Compared to traditional software, app design is characterized by shorter design life cycles, specific development tools, and increased use of APIs to generate improved efficiency and compatibility. The driver behind all of these aspects of app development is a single end: improved app quality for the end user. As such, it is important to build an understanding of the nature of app quality.

App Quality

While the literature has identified app quality as an important aspect of app development, there is little discussion of how to define and measure this construct. Two key themes emerge from the research: 1) assessment of app quality and 2) factors leading to app quality. To understand the link between app development and quality, a great deal of research has emphasized the user experience and perspective (e.g., Bavota et al., 2015; Guerrouj et al. 2015), which has led to the majority of work on app assessment relying on user perceptions. Much of this research emphasizes star ratings as an assessment of app quality (Stoyanov et al. 2015). For example, Khalid (2013) has studied user perspectives of mobile app ratings, Vasa et al. (2012) have examined user reviews of low rated apps, Fu et al. (2013) have introduced a method to analyze user ratings and comments in mobile app markets, and Claussen et al. (2013) have studied the effect of rewarding user engagement on app quality.

An alternate method to app evaluation is to assess the program from a technical perspective (i.e., evaluating platform components, programming, and technical quality). This, however, can be costly and
prone to error. Traditionally, the number of user downloads or installations has been used to evaluate the quality of free software, where more installations indicate more use and higher quality. This method can be misleading for apps. The number of installations double counts users if they install the app on different hardware (e.g., mobile phone and tablet) or if they install upgrades to the first version of the app which they already have. Also, a higher number of installations does not necessarily mean more usage of the app because not all users will use the installed apps (Claussen et al. 2013). Consistent with Claussen et al.’s (2013) view, we define app quality from the user’s perspective. That is, an app is perceived as high quality if it keeps users’ engagement and interest. Thus, we use app rating as a proxy for app quality and propose that app rating is an appropriate method to operationalize app quality. In the context of the Google Play store, users rate apps on a five-point rating scale. This scale is expected to represent relative app quality perceived by users. While app quality is established as an important construct in the mobile app literature, less research exists on factors which contribute to app quality. In a study of 154 free Android apps, Guerrouj et al. (2015) found that high churn (the rate at which users abandon an app) and higher volume of changed API classes are related to poor app ratings. Bavota et al.’s (2015) study of 5,848 free apps found that higher rated apps used APIs that were less prone to faults and changes. Software developers use existing software artifacts (e.g., source code, design models) to improve the quality of their software, reduce the cost associated with software development and save duplicate efforts (Luo et al. 2007). While the literature on app quality is sparse, existing work emphasizes an understanding of app quality as a user perception which is driven by development efficiency and reliability, particularly through the use of proven software artifacts. For the purpose of this study, our examination of the dependent variable, app quality, emphasizes the user’s perspective, with technical measures playing a secondary role.

**App Originality and Interdependence**

App originality relates to the existing but limited body of research on source code originality, most of which views the issue from a plagiarism lens (e.g., Ahtiainen et al. 2006; Luo et al., 2007). Despite the argument that code reuse can speed up the development cycle, some developers prefer to write original code. At the individual level, developers can be reluctant to adopt new APIs (McDonnell et al. 2013). In a survey of Android client code, they found that despite Android being a fast-developing platform, 28% of client code references were out-of-date within sixteen months. In contrast, developers can become frustrated with common obstacles caused by “one-size-fits-all” APIs, resulting in the need for customization (Wang and Godfrey 2013).

In this paper, we distinguish code originality from code reuse. Code originality refers to the proportion of an app’s source code which is unique to that app and is defined as “the ratio of the number of lines of code unique to the app to the total number of lines of code in the app.” Code originality is important for managing different aspects of the software development life cycle, such as evaluating asset reuse or tracking the source of code bugs (Luo et al. 2007). The use of existing code has been established as a core principle of developing reliable apps in a short amount of time, whereas customized code decreases reliability and performance, particularly when development and testing time is limited. That is, mobile applications with a lower proportion of original coding are anticipated to have better quality compared to those with more original lines of coding. Thus, we hypothesize that:

**H1:** Increased app originality will have a negative effect on app quality.

App interdependence is defined as the degree to which an app is dependent on other apps. As utilization of smartphones has increased in recent years, app development has become more challenging, thus requiring a broad set of resources and capabilities such as development platforms, software components, databases, skills and expertise. Since such resources are dispersed among various actors, development teams rely on external resources. We contend that a resource dependence theory lens can be applied to a software context, allowing exploration of the relationship between source code structure and app quality. Resource dependence theory is helpful in explaining how actors in an environment interact and manage their collective action (De Reuver and Bouwman 2008). Pfeffer and Salancik’s (1978) resource dependency theory posits that organizations need several resources (partly available in their environment) for their survival. These interdependencies can be symbiotic (actors need the same resources for survival) or competitive (actors require each other’s outputs for survival). Hence, organizations’ actions are influenced by other organizations in their environment on whose resources they are dependent. In applying this idea to app development, consider how development platforms and
software artifacts are critical resources upon which many app development teams are dependent. For instance, Android offers SDK Platform packages and APIs required for app development. App developers are dependent on these external resources to develop mobile applications. The majority of studies on software reuse agree that reuse of software components and artifacts contributes to the stability and quality of the software (e.g., Basili et al. 1996; Dantas and Garcia 2010; Lim 1994; Luo et al. 2007; Melo, 1996; Melo et al. 1998; Mohagheghi et al. 2004). A study of Android apps found, on average, 61% of classes in an app are reused by apps in the same domain (Ruiz et al. 2012); however, the relationship between such reuse and app quality has not been studied (Syer 2013). Melo et al. (1998) maintain that because repeated use of reusable components brings out any flaws in their design or implementation, and because the conception of these components is more careful, reusable components are better designed compared to custom-designed components. Reusable components thus serve as a safeguard in improving the software. As a result, reuse of software components is associated with a positive impact on overall software design quality. For the purpose of this study we define app interdependence as "the degree to which a mobile app is dependent on SDK platform packages." The construct of app interdependence conceptualizes the increased use of API calls in app development. We operationalize app interdependence as the number of API uses in an app’s design. For example, rather than designing a touchscreen interface, an app would use an API call to utilize an existing touchscreen interface. The app’s interdependence would be a measure of its use of the touchscreen interface and the number of times it calls on that interface using the API. Because the use of API calls increases app stability and reliability, and assuming that APIs are without defect, higher levels of interdependence should be associated with higher quality of produced application. Thus, we propose that:

H2: Increased app interdependence will have a positive effect on app quality.

A challenge with both our hypotheses is that apps can address a limitless range of purposes from everyday tools like a calculator to custom technical applications. Because of their breadth of application, they vary in terms of complexity. Apps are composed of many components which can make their structure complex. We contend that the complexity of an app is related both to our independent and dependent variables. Researchers use both static code attributes which measure the number of linearly-independent paths through the source code (the higher the number, the more complex the code) and size-based measures, which estimate structural complexity based on program control flow (Zhang et al. 2007). Software complexity measures are proven to influence software quality. For example, complexity can affect software run-time (Simon et al. 2015). By nature, a more involved technical app will not only require increased volume of code, it will also be more interdependent on other apps. While the interdependence increases reliability, the effect is offset by the complexity. Because such an app is more complex, it creates increased system demands, thus resulting in potentially reduced device and app performance compared with a simple app. Thus, we argue:

H3a: Increased app complexity will moderate the relationship between app originality and quality, compounding the negative effect of code originality on app quality.

H3b: Increased app complexity will moderate the relationship between app interdependence and quality, reducing the positive effect of code reuse on app quality.

Methodology and Measures

Data Collection

This study explores the relationship between two dimensions of code structure: (i) app originality and app interdependence, and (ii) app quality. We use a conservatively sized random sample of 100 apps drawn from an existing dataset of source code of 915 apps across 25 categories in the Google Play Store to test a
minimum increase in mean app rating of 0.25 stars. Each App from the Android platform is packaged as an Android Package file known as an APK. In order to study the class file from each APK file, the Java Archive Files (JARs) were extracted by: i) downloading the APK file associated with each app, ii) extracting the JAR files by extracting the Java bytecode and converting to JAR using the dex2Jar disassembler tool, iii) compiling the JAR files using a Bash script, iv) generating class files using Javascript to run the JClassInfo tool for each app, and, v) discarding all API classes outside the android packages.

For each app, the source code was analyzed using Dantas and Garcia’s (2010) metrics for code reuse. App originality was assessed as the percentage of lines of code which were not duplicated in the coding of the working app.

1. App interdependence was assessed as the number of API uses within an app’s coding.
2. App quality was assessed using two measures: the mean app score on a scale of one to five as assessed by user votes on the Google Play Store, and the number of downloads from the Google Play Store.
3. App complexity is a composite score which uses the number of files, classes, lines of code, API calls, as well as a range of functions defined by the Sonarqube complexity metric (Racodon 2015).

**Analytical Approach**

The relationships with app quality as measured by user reviews were tested using a simple linear regression model. Statistical significance was tested for $\alpha=0.05$. Models were tested to ensure compliance with the assumptions of a linear regression model and tests for collinearity and influential variables (i.e., extreme outliers) were conducted and adjusted for as discussed in the results section. Analysis of user downloads revealed a non-linear distribution. Accordingly, user downloads were categorized into one of three segments: low (less than 2,000,000 downloads), medium (2,000,000–9,999,999 downloads) and high (10,000,000 downloads or more). Because of the use of a discrete outcome variable, the relationship with app quality measured as the number of user downloads was modeled using a multinomial logistic regression. Again, statistical significance was tested for $\alpha=0.05$.

**Results and Discussion**

A descriptive analysis of the data revealed that while the User Rating variable followed a normal distribution, it was highly left-skewed. This phenomenon is not uncommon with 1-5 rating scales, particularly when users may be biased towards a high rating, as is the case with “star reviews” through marketplace websites. Given that a skewed distribution can be problematic for generating a predictive model, the rating variable was reversed (i.e., 0=very good, 4=very poor) and a log transformation was applied which, while diminishing the normality of the pattern, reduced the skewness from 0.942 to –0.334. This provides improved confidence in our regression results.

Further testing of the model assumptions suggested there may be collinearity between the complexity and interdependence variables. This is not surprising as they both use the number of API calls as part of their measurement, and is addressed in the models below. As discussed in the methodology section, the distribution of the Downloads variable was distributed into three general categories with a very wide standard distribution (Figure 5). As such, this variable was converted to a categorical variable with three values: low (less than 2,000,000 downloads), medium (2,000,000–7,999,999 downloads), and high number of downloads (8,000,000 or more downloads).

From these variables, eight models were produced and tested. Models one through four explore the relationship between the independent variables of interest and user reviews as a measure of app quality while models five through eight use downloads as a measure of app quality.

---

1 Based on a standard sample size formula (Rosner, 2015), given a population mean app rating of 4.14 and standard deviation of 0.327, where required sample size is 83.

2 Results of model assumption tests are available from the authors.
App Quality as User Reviews

The first single variable model (Table 1) showed a statistically significant relationship (p=0.0117) between app originality and the log-reverse user review variable. An $R^2$ value of 0.0644 indicates the model explains 6.4% of the variance in the relationship between these two variables. The second model added app complexity as a moderator in the relationship between app originality and the log-reverse user review variable. In this model, complexity and the interaction term between complexity and originality were statistically insignificant. The third model showed a statistically insignificant relationship (p=0.7304) between app interdependence and the log-reverse user review variable. The fourth model showed added app complexity as a moderator in the relationship between app interdependence and the log-reverse user review variable. Again, all values were statistically insignificant.

<table>
<thead>
<tr>
<th>Model 1</th>
<th>Model 2</th>
<th>Model 3</th>
<th>Model 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>2.03369</td>
<td>2.2032</td>
<td>-0.45123</td>
</tr>
<tr>
<td>Originality</td>
<td>0.02337</td>
<td>0.02503</td>
<td>-5.97E-06</td>
</tr>
<tr>
<td>Complexity</td>
<td>-5.63E-08</td>
<td>1.81E-05</td>
<td>3.88E-07</td>
</tr>
</tbody>
</table>

Table 1. App Originality and Interdependence as Predictors of Log-Reverse User Reviews Models

App Quality as Number of Downloads

A multinomial regression model was used to compare the odds of an app having either a low, medium, or high number of downloads based on app originality (Table 2). The estimate of 0.00931 indicates that a one-unit increase in app originality results in a 0.00931 increase in the log odds of being in the medium download group versus the low download group. The model showed statistically insignificant results, though, and with a likelihood ratio chi-sq value of 1.2173, we can conclude that this model provides less explanation of app downloads than a model without originality as a predictor variable. Adding complexity

<table>
<thead>
<tr>
<th>Med Group vs. Low Group</th>
<th>Model 5</th>
<th>Model 6</th>
<th>Model 7</th>
<th>Model 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>-2.9608</td>
<td>-12.5495</td>
<td>-3.4789</td>
<td>-3.5213</td>
</tr>
<tr>
<td>Originality</td>
<td>0.00931</td>
<td>0.0999</td>
<td>0.00146 **</td>
<td>-3.00E-05</td>
</tr>
<tr>
<td>Complexity</td>
<td>-0.00036</td>
<td>4.46E-06</td>
<td>-3.00E-05</td>
<td>-7.47E-09</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>High Group vs. Low Group</th>
<th>Model 5</th>
<th>Model 6</th>
<th>Model 7</th>
<th>Model 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>4.9166</td>
<td>17.8028</td>
<td>-2.3792</td>
<td>-3.1039</td>
</tr>
<tr>
<td>Originality</td>
<td>-0.0696</td>
<td>-2.18E-01</td>
<td>0.00185</td>
<td>0.00047</td>
</tr>
<tr>
<td>Complexity</td>
<td>-0.00143 **</td>
<td>0.00106 **</td>
<td>0.000016 **</td>
<td>-3.14E-08</td>
</tr>
</tbody>
</table>

Table 2. App Originality and Interdependence as Predictors of Downloads
as a moderator to the relationship tested in Model 5 also produced statistically insignificant results for the odds of being in the medium downloads group over the low downloads group, but statistically significant results for being in the high downloads group over the low downloads group. The likelihood ratio produced a chi-sq value of 23.3876, which, with a corresponding p-value of 0.0007, indicates that this model is better than not having a model. Also, the originality parameters are both negative which, despite the limited sample size, suggests that H1 and H3a are correct. Adding complexity as a moderator to the relationship tested in Model 5 also produced statistically insignificant results for the odds of being in the medium downloads group over the low downloads group, but statistically significant results for being in the high downloads group over the low downloads group (Table 8). The likelihood ratio produced a chi-sq value of 23.3876, which, with a corresponding p-value of 0.0007, indicates that this model is better than not having a model. Also, the originality parameters are both negative which, despite the limited sample size, suggests that H1 and H3a are correct. A simple model between app interdependence and downloads resulted in positive parameters, which suggests that the higher an app's interdependence, the higher probability it will have of having medium or high downloads compared with low downloads. The parameters were statistically significant, and the model's likelihood ratio produced a p-value of 0.003, suggesting this model is better than no model. Thus, we have some support for H2. Lastly, adding moderation to Model 7 found negligible values for the interaction term between interdependence and complexity. Although the likelihood ratio test found the model better explains the outcome than no model (p=0.0021), a goodness of fit test demonstrates Model 7 is preferable to Model 8. This is likely due to the previously demonstrated collinearity between interdependence and complexity. Thus, we do not have support for H3b.

Discussion

The results advance the limited literature on app originality and interdependence, which has heretofore suggested that app design benefits from integration with existing apps and reuse of proven, stable design. This study has tested this proposition by seeking to understand the relationship between code structure and app quality. Four hypotheses were tested with mixed results, suggesting a more complex relationship than previously theorized.

The relationship between app originality and app quality was the best supported. Using user reviews as an operationalization of app quality, we found strong support for this relationship. When app quality was operationalized as user downloads, however, we found no relationship either between app originality and low to medium downloads or low to high downloads. However, as established in the review of the literature, user reviews are a better indicator of app quality. Given that model one demonstrated a negative relationship between the two variables, we have found support for H1. When app complexity was added as a moderator, results varied based on the outcome variables. No relationship was found in this context between app originality and user reviews. A negative relationship was found, however, between app originality and downloads when moderated by app complexity, but only when comparing low-download apps against high-download apps. This suggests that apps that receive a high number of downloads are more likely to have less original code and less complexity. Thus, we have partial support for H3a.

<table>
<thead>
<tr>
<th>Operationalization of App Quality</th>
<th>User Reviews</th>
<th>Downloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1 (-) App originality/quality</td>
<td>Supported</td>
<td>Not Supported</td>
</tr>
<tr>
<td>H2 (-) App interdependence/quality</td>
<td>Not Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>H3a (- moderation) App complexity on originality/quality</td>
<td>Not Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>H3b (- moderation) App complexity on interdependence/quality</td>
<td>Not Supported</td>
<td>Weak Support</td>
</tr>
</tbody>
</table>

Table 3. Summary of Results

The relationship between app interdependence and app quality, when operationalized as user reviews, was not supported, neither as a simple model nor with app complexity acting as a moderator. When app quality was measured as the number of downloads, however, we did find a significant relationship between the two. In both of these models, app interdependence is seen to have a positive effect on an app.
having a high rather than low number of downloads. Statistical significance was limited in the relationship between low- and medium-download apps. While there was some evidence that app complexity dampens the positive effect of app interdependence on the number of downloads, this model was not as effective as the simple model, and the interaction terms were statistically insignificant. Therefore, while we found partial support for H2, there is only weak support for H3b. A summary of the results is provided in Table 3.

In comparing the differing results for the two independent variables, the findings are interesting. On the one hand, apps with low originality receive higher scoring user reviews than those with high originality, but there is no significant difference in downloads between the two. This suggests that code originality does not make a difference to the purchasing decision, but does impact the user experience. The moderating effect of complexity, however, did generate a significant relationship with the number of downloads. We have theorized that app coding complexity is a suitable proxy for the complexity of an app’s function. Given the mobile nature and hardware constraints of devices utilizing apps, we would expect, in general, users to be more inclined to use simple rather than complex apps. The moderation effect of complexity on the app originality-app quality relationship supports this theory. If an app is more complex, it is less likely to be downloaded because it likely provides too much functionality than a user desires in a mobile context. In terms of user experience, however, there is no difference between complex and simple apps. In other words, if they have lower coding originality, the effect on user experience is the same regardless of app complexity.

In comparison, the results for app interdependence are more difficult to interpret. We would expect to see similar results to the app originality model. Instead, we find no significant relationship between app interdependence and user reviews and a significant relationship between app interdependence and the number of downloads. Yet app interdependence is an aspect of code structure which should largely be invisible at the time of purchase (i.e., in its relationship to the number of downloads), yet viscerally experienced by the user post-purchase (i.e., in its relationship to user reviews). This can be explained in the collinearity between interdependence and complexity. As argued above, app code complexity is something the user would be expected to see in some form during the purchase process because it is a reflection of the breadth of an app’s functionality. In essence, then, the measure of app interdependence in this study is to some degree measuring app complexity. This highlights the difficulty of operationalizing coding constructs through analyses of source code. Future studies will want to further explore the relationship between app interdependence and app quality and whether, perhaps, app originality is a sufficient gauge of app interdependence, or if more complex measures of interdependence are required.

**Contributions**

Results from this study contribute to both practice and theory. Most importantly, this is the first study to test the relationships between app originality and app quality as well as app interdependence and app quality. In doing so, it extends theory from the existing literature on software development and builds two new constructs in relation to app development: originality and interdependence. As the literature on mobile application quality lacks theoretical foundations, the study provides important insight into developing a theoretical framework for app development. Further, it demonstrates that existing conclusions regarding source code originality from the software development literature also apply to app development.

Second, the study advances a new platform in design study. To the best of our knowledge, this is the first study that incorporates mobile app data in the design application. Mobile apps research is a new avenue in software engineering where user reviews, ratings, number of downloads and source code are analyzed to determine the factors that influence the success of an app. However, with this new approach, we are able to study the importance of app originality and app interdependence. It thus serves as a guideline to design successful apps.

Last, the study provides an important insight into the app development process, namely, that developers can build higher quality apps by reducing the proportion of original code in their app design. Thus, the more an app uses previously proven code through API calls, and by extension, other apps, the higher its quality will be. This relationship was not found to be moderated by app complexity, which further suggests that app developers should ensure complex apps draw on functionality from other established apps to ensure stability and thus maintain high user ratings regardless of the degree of complexity.
**Limitations and Future Research**

The results of the study are subject to several limitations of the study design and data set. Of greatest importance is the operationalization of interdependence. On the one hand, the definition used to assess the construct included some overlap with the definition of app originality, whereby both utilized number of API calls as part of their respective measurements. The differing results, however, suggest that the two definitions were sufficiently distinct from one another. The collinearity between interdependence and complexity, however, suggests that future research should further refine the definitions of each. Given the limited discussion of these constructs in existing literature, this provides an important realm of theoretical development.

Second, we were limited by the data sample size. Although chosen to ensure sufficient power to test statistical significance, when the download variable was converted into the three categories, the middle category was relatively small. Again, future studies will want to expand the size of their datasets, as well as integrate data from other operating systems into the data pool.

Third, the process of decompiling from JAR files to source code files has its disadvantages. In the event of transformation, the Java decompiler produces many errors, such as duplicate variables, empty classes, and incomplete source code (i.e., obfuscated code). Obfuscated code in software development refers to machine code language that is difficult for humans to understand. As such, we used the Java decompiler to de-obfuscate the code and reconstruct the “original” source code. Thus, it is not completely reliable to use the decompiled source code for analysis to make quality predictions.

Last, previous studies have identified that reuse of software components is only useful if software stability is not compromised (Figueiredo et al. 2008). In this research we assumed that the software components (e.g., APIs) were stable and reliable, thus improving the overall quality of the apps. Future studies could strengthen the arguments by integrating an API stability variable, as well as a measure of API code reliability as moderating constructs. Further, the ‘stability’ construct has been discussed in the literature as a proxy for application quality. Future research can include this construct as a moderator between code originality and quality, or as a metric to analyze the quality of applications.

**REFERENCES**


