World Journal of Engineering Research and Technology
WJERT
www.wjert.org SJIF Impact Factor: 3.419

ROLE OF SOFTWARE ENGINEERING IN MOBILE APPLICATIONS DEVELOPMENT

Sanjeev Kumar Punia* and Parveen Kumar

[1*]Department of Computer Science & Engineering, NIMS University – Jaipur.

Article Received on 21/07/2016 **Article Revised on 11/08/2016** **Article Accepted on 31/08/2016**

***Corresponding Author**
Sanjeev Kumar Punia
Department of Computer Science & Engineering,
NIMS University – Jaipur.

ABSTRACT

There has been tremendous growth in the use of mobile devices over the last few years. This growth has fueled the development of millions of software applications for these mobile devices often called as ‘***apps***’. Current estimates indicate that there are hundreds of thousands of mobile app developers. As a result, in recent years, there has been an increasing amount of software engineering research conducted on mobile apps to help such mobile app developers. In this paper, we discuss current and future research trends within the framework of the various stages in the software development life-cycle: requirements (including non-functional), design, development, testing and maintenance. As there are several non-functional requirements but we focus on the topics of energy and security in our paper. The mobile apps are not necessarily built by large companies that can afford to get experts for solving these two topics.

INDEX TERMS: Mobile apps, mining, app markets.

I. INTRODUCTION

In the context of this paper, a mobile app is defined as the “*application developed for the current generation of mobile devices popularly known as smart phones*”. These apps are often distributed through a platform specific and centralized app market and we sometimes refer to mobile apps simply as apps. In the past few years we are observing an explosion in the popularity of mobile devices and mobile apps. In fact, recent market studies show that the centralized app market for ***Apple’s platform*** (**iOS**) and ***Google’s platform*** (**Android**) have more than 1.5 million apps. These mobile app markets are extremely popular among developers due to the flexibility and revenue potential. Mobile apps also bring a whole slew of new challenges to software practitioners as challenges due to the highly-connected nature of these devices, the unique distribution channels available for mobile apps (app markets like **Apple's *App Store*** and **Google's *Google Play***).

Till date the majority of the software engineering research has focused on traditional "shrink wrapped" software such as *Mozilla Firefox*, *Eclipse* or *Apache HTTP*. However, recently researchers have begun to focus on software engineering issues for mobile apps. For example, the 2011 *Mining Software Repositories Challenge* focused on studying the Android mobile platform. Other work focused on issues related to code reuse in mobile apps, mining mobile app, testing mobile apps and teaching programming on mobile devices. Therefore, we feel it is a perfect time to reflect on the accomplishments in the area of Software Engineering research for mobile apps and to draw a vision for its future.

The purpose of this research paper is to serve as a reference point for mobile app work. We start by providing some background information on mobile apps and discuss the current state-of-the-art technology in the field relating to the software development phase as requirements, development, testing and maintenance as shown in Figure 1.

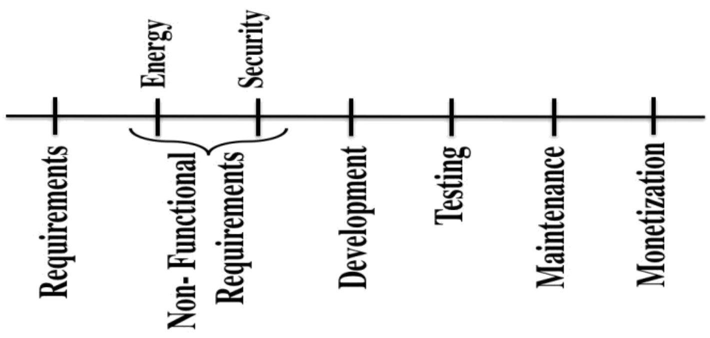


Fig 1: A framework for presenting the state-of-the-art in software engineering research for mobile apps.

We discussed about two non-functional requirements: ***energy use*** and ***security of mobile apps***. Finally, even though it is not a software development phases but we talk about the software engineering challenges recommendations for mobile apps. Along with a discussion of the state-of-the-art, we also present the challenges currently faced by the researchers/developers of mobile apps. Finally, we discuss our vision for the future of software engineering research for mobile apps and the risks involved.

II. BACKGROUND

Mobile apps have been around for a long time now. Back in the 1990s they were usually created by device manufacturers like Nokia and Motorola. These apps performed certain basic tasks. Later on, wireless service providers started making apps to differentiate the devices sold on their network to others. At the same time, third party companies started making apps for the mobile platforms like the ***Windows mobile OS*** and the ***Symbian OS***. These included games for the devices and other utility apps. However, there was no centralized place where end users could acquire these apps. The most modern iteration of the mobile apps started in 2007 when Apple announced the first generation of the ***iPhones***. At the same time Apple also announced the centralized market for mobile apps called the '***App Store***', but end users had to download all their apps. Soon after in 2008, Google deployed their own platform (Android) and their own app market the '***Android Market***' (which was later renamed as '***Google Play***'). Similar app markets were released for the mobile phone platforms developed by Microsoft and BlackBerry as well. The mobile app developers have a larger customer base to sell these apps in the markets. It is estimated that there are currently 2.6 Billion smart mobile phone users. An overview of the various stakeholders in the world of mobile apps is shown in Figure 2.

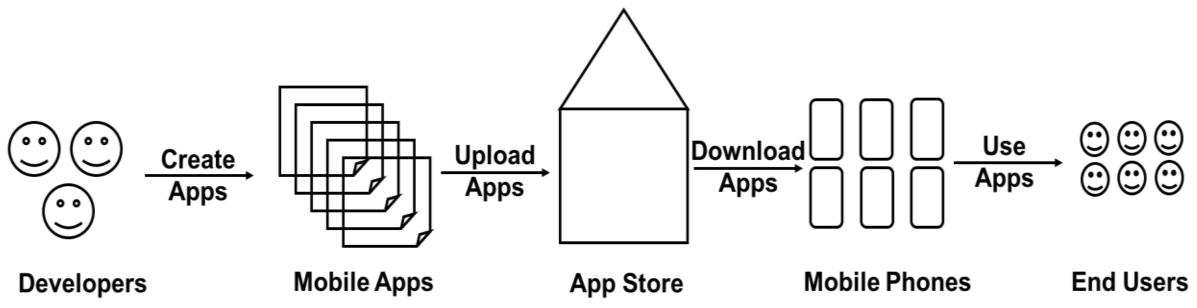


Fig. 2: Overview of the various stakeholders with respect to modern day mobile apps.

With the introduction of app markets for each platform, now developers have the ability to manage the distribution of their software through one centralized market for each platform. All big and small developers have the same app market thus making it an even playing field for anyone to succeed.

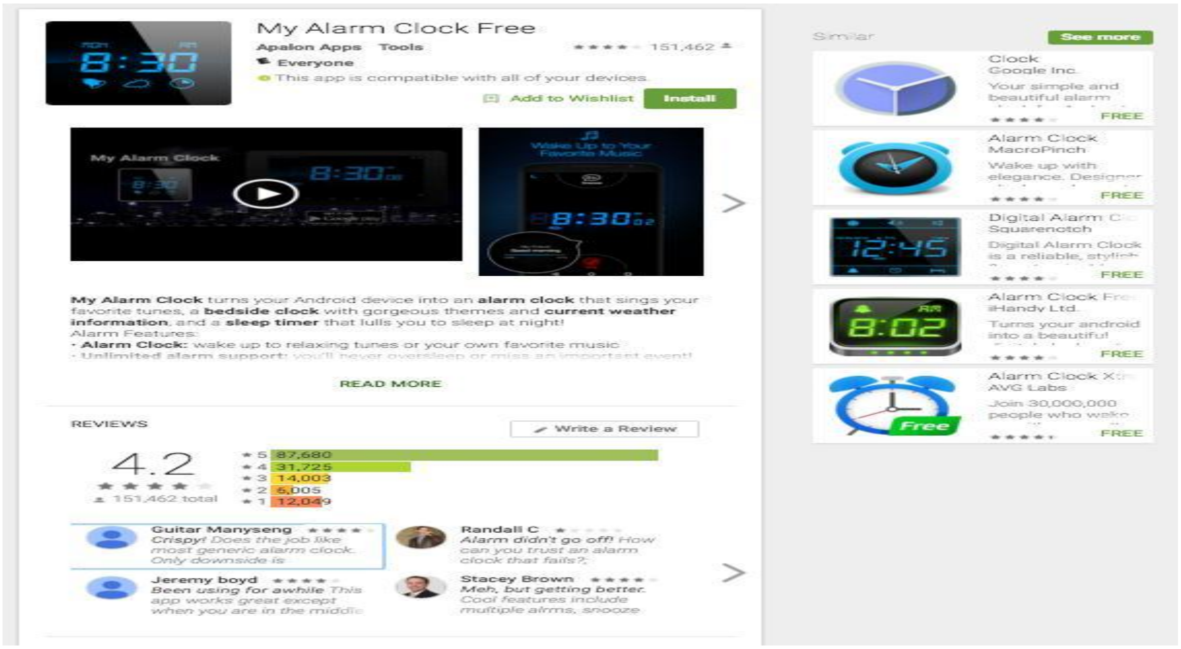
The app markets made it easy for the developers to upload their apps, manage updates to them and push the latest version seamlessly to the end users. Thus a combination of market potential, ease of use and democratized platform made it highly lucrative for developers to build mobile apps.

With the increased use of smart phones and mobile apps by end users and development of these mobile apps by software developers, mobile apps became an obvious area for software engineering researchers to examine. One of the earliest software engineering papers on such mobile apps was the study of micro apps on the Android and BlackBerry platforms by Syer et al. and one of the earliest studies on the app markets was by Harman et al.

The increase in such software engineering studies on mobile apps are because of two reasons (1) the app markets are publicly available, it is now possible to mine the data relatively easily (although later in this section we explore where researchers faced trouble in getting this public data) (2) a variety of new types of data that were previous not available are now available and reliably well linked together. Some of these new types of data are discussed below (and a snapshot of the app store is in Figure 3 shown on the next page).

The review data is rich in what users want from the app - both features and bug fixes, along with praise for the features that they love. Therefore, such review data has now become a treasure of data for requirements engineering researchers. The reviews have a numeric rating, which are then aggregated to determine the overall rating of the app by making it easy for users to know if the past users thought an app was good or not.

Additionally, these numeric ratings also provided researchers with a clear way of knowing if an app is good or not and if the review by a user is overall of a complimentary or derogatory nature.



My Alarm Clock turns your Android device into an **alarm clock** that sings your favorite tunes, a **bedside clock** with gorgeous themes and **current weather information**, and a **sleep timer** that lulls you to sleep at night!
Alarm Features:
• **Alarm Clock:** wake up to relaxing tunes or your own favorite music
• **Unlimited alarm support:** you'll never oversleep or miss an important event!

4.2 ★★★★ 151,462 total

★ 5	87,680
★ 4	31,725
★ 3	14,003
★ 2	6,005
★ 1	12,049

Guitar Manyseng ★★★★
Crispy! Does the job like most generic alarm clock. Only downside is

Randall C ★
Alarm didn't go off! How can you trust an alarm clock that fails?;

Jeremy boyd ★★★★
Been using for awhile This app works great except when you are in the middle

Stacey Brown ★★★★
Meh, but getting better. Cool features include multiple alrms, snooze

Your simple and beautiful alarm

Wake up with elegance. Designer

Digital Alarm Clock is a reliable, stylish

Turns your android into a beautiful

Join 30,000,000 people who wake

Fig. 3: Snapshot of an app in the app market.

The app market also allows the developer to post release notes on each of the app's versions. Researchers are able to mine this information to determine how the apps are evolving.

Another piece of information available in the app store for each app is the contact information for the developer. Knowing the similarity between apps is further facilitated in the app markets by the category classification. Each app in the app store has to be classified in one of many predefined categories.

Often, we see that a software engineering research study is done on an **IDE** like ***Eclipse*** and another **OSS project** like the browser ***Firefox***. In the world of mobile apps if we conduct our research on only game apps then we can be more certain that our findings would apply to other game apps. Additionally, all these various data points are available for hundreds of thousands of apps in a public facing website making it a rich dataset for researchers to crawl.

Common Challenges

Firstly, app stores restrict public access to their data and typically only allow for access to a subset of all the user reviews. For example, in the case of the Google Play store, one can only access 500 reviews for an app.

Secondly, app stores do not provide the source code of the apps or any other associated artifacts like test code, design and requirement documents. Only the app binary and release notes are made available.

Finally, with respect to the release notes and the app binary, one can only get them for the latest release. There is no historical information that can be collected from the app store (except user reviews). The only way to gather historical information on the various releases of the mobile apps is to continuously mine the app stores at regular intervals (like daily or weekly basis).

III. REQUIREMENTS

A number of studies have focused on requirement extraction for mobile applications. Contrary to traditional work on software requirements, which mainly focused on analysis of the requirements and specifications document, the majority of mobile app-related studies leveraged app reviews posted by users to extract requirements. Galvis-Carreno and Winbladh extract topics from user reviews in order to revise requirements. They show that their automatically extracted requirements match with manually extracted requirements and use natural language processing (NLP) techniques to identify app features in the reviews and use sentiment analysis to determine how users feel about app features. They also compare their extracted features to manually extracted features and find that the extracted features are coherent and relevant to requirements evolution tasks.

Besides requirements extraction from the user reviews, there have been several studies on feature analysis. For example Rein and Munch present a case study for feature prioritization that extract the set of features from the release notes available in the app store for a large collection of apps. They found a mild correlation between the number of features in an app and the cost of an app.

A. Challenges and Future Directions

The fact that requirements are extracted from app reviews has its own challenges. In many cases and for many apps, there may not be enough user reviews or the quality of the reviews may be low. All of the aforementioned studies need a high quantity and quality of user reviews. Chen et al. have done some initial work in automatically identifying reviews that are informative. One interesting problem that has already been addressed by app markets like Google Play is the ability for the developer to reply to user reviews.

Therefore the natural directions of research in the area of requirements engineering are as follows: building NLP techniques that are not subject to the limitations in the user reviews (and exploiting the newly available knowledge bases), come up with sampling techniques that takes the sampling bias into account and building robust data collection tools that are able to collect a more complete set of reviews. All these research opportunities will allow us to mine requirements from the user reviews in a more efficient manner.

B. Risks

One of the risks involved in pursuing the above lines of research is that we may have reached the limits of NLP when analyzing poorly written user reviews. Another risk is that users may be prefer the features that they are provided before they ask for it and when the user complains about the features then it is already too late. The only solution might be to build an updated review system for the app stores that allows a better mechanism for feature requests from the users.

IV. ENERGY

Due to the fact that energy is a scarce resource for mobile devices, a plethora of studies have proposed ways to measure and save energy of mobile apps. One of the first works related to the measurement of energy of mobile applications is ***GreenMiner*** which is a dedicated hardware platform that enables measurement of energy consumption of mobile devices. In other work, Hao et al. propose a technique that leverages program analysis to provide per instruction energy modeling. They show that their approach that can estimate energy consumption within 10% of the ground truth for Android apps.

Pathak et al. proposed taxonomy of energy bugs based on more than 39,000 posts. They also propose a framework for the debugging of energy bugs on smart phones. Li et al. perform an empirical study on 405 apps to better understand energy consumption. They make several interesting findings as: 1) the majority of a mobile app's energy that is spent in the idle state and 2) networking is the component that is more resource heavy.

Linares-Vasquez et al. present an empirical study into the categories of API calls and usage patterns that consume high energy. The findings of the empirical study can help developers to reduce the energy consumption when using certain categories of Android APIs.

A. Challenges and Future Directions

The two main challenges in energy related research for mobile apps is not knowing what to measure for accurately identifying energy issues and then trying to fix the issues for the developers because current state-of-the-art tools are not easily accessible to developers. Therefore, we need good estimates of energy use. In fact, there has been very little work on even understanding how much developers know about energy bugs and which of their actions actually cause them. Knowing more about developer coding habits and which ones cause more energy bugs could be impactful research.

Future directions in energy research could be in the area of identifying practical ways in which energy usage can be improved in apps. Another potentially impactful area of energy research is trying to understand how and when our findings translate to other platforms. Currently most of the energy research is happening on the Android platform. For example will the same third party libraries have a similar impact on the Windows or BlackBerry platform? If not, then can we build tools that can make recommendations to developers who are building cross-platform apps?

B. Risks

One of the more practical risks for researchers who want to pursue this line of research is: access to the hardware that can measure power or settle for software models that can be inaccurate.

There are some initial solutions like *Green Miner* framework that are available for researchers to remotely access the hardware resources for energy measurements. Even when researchers have access, there exists the issue of sampling frequency. If the sampling frequency for energy measurement is longer than the time interval in which energy bugs occur then there is a strong chance that the results are not consistent.

V. SECURITY

A number of recent studies focused on the security of Android apps. However in security, there are two lines of research in the intersection of software engineering, mobile apps and security.

The first line of research is in identifying vulnerabilities in apps. Chin et al. propose a tool called ***ComDroid***, which detects communication vulnerabilities. Sadeghi et al. proposed ***COVERT***, a technique that detects inter-app vulnerabilities. Potharaju et al. look at various attack strategies and defense techniques from plagiarized mobile apps. Jha in their PhD thesis cataloged a set of risks for mobile applications.

The second line of research is in finding malicious apps as Gorla et al. proposed the ***CHABADA*** tool which detects unexpected behavior of Android apps. *CHABADA* generates topics from app descriptions and compares the behavior of the app against its description. They find that *CHABADA* is effective in flagging 56% of malware without any knowledge of malware patterns.

Mudflow examines the sources and sinks of data flows and examines that Mudflow flags apps is malicious if their data flows deviate from the data flows in benign apps. Arzt et al. proposed the ***FlowDroid*** tool which performs static analysis of Android apps. ***Appscopy*** is a similar tool that detects Android malware through static analysis.

A. Challenges and Future Directions

The static analysis approaches suffer from a high rate of false positives. That issue however, may be less critical for mobile apps since they tend to be smaller in size. Other work depends on data provided by the app developers such as the app's description. Such approaches cannot guarantee to perform well for applications that do not have well documented descriptions.

Another challenge is the availability of data. Malicious code and vulnerabilities in code are ever changing (and at a great pace). However, there is a serious lack of malicious/vulnerable apps. Arzt et al. built a publicly available benchmark suite of malicious apps called ***Droid Bench***. This benchmark contains only 120 apps currently so there is a real need to bolster this benchmark with more data.

There are many directions of future research that is possible in this area. The most obvious of this is to advance the state-of-the-art in static analysis research. When it comes to malware research the ultimate goal is to build a lightweight static analysis tools that can be deployed at the app store and prevent malicious apps from being uploaded to the store. Another outcome of such research is to provide the end user with an easy to use approach to understand what the app is doing and if its behavior is abnormal.

B. Risks

Most of the work has been done for Android apps. This is mainly due to the fact that the Andorid platform is more open than other platforms e.g. ***iOS*** or ***BlackBerry***. Also the apps are written in Java for which there exist many compilers and static analysis tools. Performing our studies on Android causes a risk in terms of how applicable the proposed approaches would work for mobile apps from other platforms.

Another risk is to focus on reactive approaches for security in order to solve the current security issues and not focusing on preventive solutions. Focusing on reactive approaches is not just an issue with mobile apps but with all software. However, with mobile apps due to the speed at which they are evolving, this issue could be even more potent - as we may never catch up.

VI. DEVELOPMENT

As a bit of past work is done in the areas of requirements, energy, security, testing and maintenance for mobile apps, there has been very little work that has been done on actually developing the apps. Most of the work has been from the platform developers like Google and Apple in providing the development tools required for building the mobile apps.

In software engineering research, Syer et al. compared the source code of Android and BlackBerry applications along three dimensions source code, code dependencies and code churn. They find that BlackBerry apps are larger and rely more on third party libraries, whereas Android apps have fewer files and rely heavily on the Android platform. Hecht et al. proposed a tool called ***Paprika*** to study anti patterns in mobile apps using their byte code. Khalid et al. examined the relationship between warning from Find Bugs and app ratings. They find that certain warnings correlate with app ratings. Around the same time, Tillmann et al. developed a platform, ***TouchDevelop*** use to build mobile apps for the Windows Phone and help to novice developers with little to no experience in either software engineering or software development to build apps.

A. Challenges and Future Directions

With the popularity of all platforms increasing in the past few years, developers are tempted to develop the same app for multiple platforms (cross-platform development). In order to enable this, there are several frameworks like ***Sencha***, ***PhoneGap***, ***Appcelerator*** (some of the cross-platform development frameworks like *Cocos2d*, *Unity 3D* and *Corona* are specifically for games). The developer has to build the app to call the APIs present in these frameworks and at build time generated an app for each platform framework. All these frameworks have an adverse affect on both the performance of the app and its user interface due to their design. Very little research has been conducted to help developers to understand the costs and benefits of the various approaches of developing cross-platform apps.

B. Risks

With the fast platforms evaluation, it may be very difficult to build a static solution for cross-platform development. Additionally, there are hardware and app market policy mismatches that have to be taken care of. The study of the cross-platform development issue may be difficult because of linking the apps across the app markets. In some cases, mobile app developers may obfuscate their apps by making their development as a challenge since one would need to deal with the obfuscation of the code before being able to study the app.

VII. TESTING

There is a wide range of developed techniques that helps mobile app developers improve the testing of mobile applications specially by trying to improve UI and system testing coverage. Hu et al. propose the ***Monkey tool***, which automates the GUI testing of Andorid apps. Monkey generates random events, instruments the apps and analyzes traces that are produced from the apps to detect errors. Another tool proposed by Machiry et al. is ***Dynodroid*** which dynamically generates inputs to test Android apps. Contrary to *Monkey*, *Dynodroid* enables the testing of UI and system events. Due to this difference, the authors showed that *Dynodroid* can achieve 55% higher test coverage compared to *Monkey*.

Mahmood et al. presented the ***EvoDroid*** tool, which combines program analysis and evolutionary algorithms to test Andorid apps. The authors show that *EvoDroid* can outperform *Monkey* and *Dynordoid* by achieving coverage values in the range of 70-80%. They propose MonkeyLab, which mines recorded executions to guide the testing of Android mobile apps.

A. Challenges and Future Directions

One of the biggest challenges that researchers face in their current line of research on automated tests for mobile apps is that they are not able to achieve high code coverage. This is partially because of the inability to produce a wide range and variety of inputs and partially because of apps that are designed for user input which cannot be automatically generated. Often the automated testing tools are unable to proceed down a certain execution path due to the inability to generate inputs and therefore cannot test anything further along that execution path. Therefore research in generating a wider range of input that can mimic a human could have a great impact on automated mobile app testing tools.

Another challenge is that often researchers build tools that work on the app binary since that is the only thing to which they have access. The availability of more *OSS apps* could yield in more robust tools as one repository of *OSS apps* is the ***F-Droid repository***. A repository of *OSS apps* with the corresponding app binaries made available as a benchmark suite could greatly help researchers in the state-of-the-art in app testing.

However, with the increased success of multiple platforms there is now a large amount of cross-platform apps. Additionally, in all the platforms the apps need to run on different hardware with different versions of the OS. Thus even if the app is tested on one device, there is no guarantee that it may work on another device.

However, these problems are not entirely new. In the past software developers have had to develop for the PC/Mac/Linux platforms with varying hardware. Joorabchi et al. describes a tool, ***CHECKCAMP*** that tests the inconsistencies between *iOS* and *Android versions* of mobile apps by extracted abstract models. Such a step study in right direction is one area of research with the potential for high impact of cross-platform testing.

B. Risks

One of the big risks in pursuing the above line of research is that researchers may not have access to all the various devices and/or platforms. Additionally, there is no easy way to identify cross platform apps from the app stores. So far, there has been no effort to build such a database of cross platform apps that researchers can analyze.

VIII. MAINTENANCE

The area of software maintenance is one of the most researched areas in Software Engineering. However, due to the fact that mobile apps is a young subarea within SE, the maintenance of mobile applications remains to be largely undiscovered. Moreover, since mobile apps are different, the studies related to the maintenance of mobile apps tend to focus on issues that have not been traditionally studied in past software maintenance studies. For example, most mobile apps display advertisements, and as has been shown in prior studies, these advertisements require a significant amount of maintenance. A number of prior studies investigated the maintenance of mobile apps from different perspectives.

Mojica-Ruiz et al. compared the extent of code reuse in the different categories of Android applications. They find that approximately 23% of the classes inherit from a base class in the Android API and 27% of the classes inherit from a domain specific base class.
Furthermore, they find that 217 mobile apps are completely reused by another mobile app. They also compare mobile apps to larger “traditional” software systems in terms of size and time to fix defects. They find that mobile apps resemble Unix-utilities, i.e., they tend to be small and developed by small groups. They also find that mobile apps tend to respond to reported defects quickly.

Another line of work examined Android-related bug reports. Bhattacharya et al. study 24 mobile Android apps in order to understand the bug fixing process. He detected and characterized performance bugs among Android apps.

A. Challenges and Future Directions

Some of the challenges in maintenance research for mobile apps is that often there is a lack of historical data. The software maintenance research community has greatly benefited from openly available artifacts like source control and bug repositories of OSS projects. They now have a large trove of data to evaluate their hypotheses on. Such a support has spurred an increased level of research in software maintenance as evidenced by the number of research publications on it. However, for the most part there are not many OSS mobile apps as discussed in the previous section.

Most of the current research is based on the data available in the app markets. Therefore, with limited fine grained commit level information it is difficult to conduct maintenance research.

One interesting line of future research is in estimating the maintenance cost for a mobile app. While traditional maintenance issues like bug localization may not be an issue due to the small size of the apps, mobile app developers would like to be able to triage features and bugs from the user reviews.

Finally as mentioned in Section IX, there are several companies that collect operational data from mobile apps that have been installed on millions of devices. Most of these companies provide the app developers with the data and some rudimentary analysis on them. There is a wide variety of reliability and performance problems that can be solved by building tools and approaches that mine such operational data.

B. Risks

From past research we have seen that mobile apps are small and have very quick release cycles. With such rapid release it may be the case that the maintenance effort might overlap a lot with the evolution effort. Hence, it may not be easy to identify costs pertaining to maintenance.

Additionally, the varieties of apps are far more than the variety of successful desktop applications. For example, a small recipe app like AllTheCooks and a large application like Microsoft Excel are equally popular but they may have completely different maintenance efforts. Thus the issue of placing the results in the right context becomes paramount. Therefore, it is highly recommended to keep track of the app domain when conducting maintenance case studies.

IX. MONETIZATION

Some of the successful gaming apps (like Angry Birds and Candy Crush) and productivity apps (like Microsoft Excel) are produced in established software development companies with large teams. However, from past work, we know that successful apps can be developed by one or two core developers. In such apps where the development organization is small, often the developers will also have to make several engineering decisions that could affect their bottom line. Therefore software engineering researchers have examined how we can provide data to mobile app developers to make decisions in a more careful fashion. Some of these research studies are presented below.

Past research has found that ratings and downloads are often very highly correlated. Additionally, Kim et al. found ratings to be one of the key determinants in a user's purchase decision of an app. However, Ruiz et al. examined the rating system in Google Play and found that the current rating system of cumulative averages across all versions of an app makes the ratings sticky and thus does not encourage developers to improve their app.

While ratings may not be a sufficient condition for more downloads, it may be a necessary condition. With more downloads, the developers stand to increase their revenues as well. This is because mobile apps are just monetized through in app advertising. The app itself is given at no cost. If more users use an app, more ads are shown to users, and more revenues are generated for the app. A more detailed overview of the various stakeholders with respect to mobile ads can be found in the work. In the past, we have found the number of advertising networks that a developer connects to do not impact the user perception of an app. There were apps that used as many as 28 different ad libraries, and still had a good user rating. This was probably because even though the developers connect too many different networks through many different libraries they still were displaying just one ad at a time.

Thus we recommended, that a mobile app developer could add as many ad libraries as they wanted without impacting the rating. However, we found that including particular ad libraries could affect the rating. This was because, the ad libraries were being intrusive, and the user perceived the app to be intrusive as well. Hence, we recommended that the developer be careful about what ad networks they were connecting too. This study is a good example of how software engineering researchers could make software related recommendations that could improve the monetization strategies of an app. Currently there are also several analytics companies that provide valuable usage data to developers for improving their monetization strategies. They track downloads of apps and how the apps are being used, when users purchase things from the app etc. These companies are able to track such user data, by incorporating tracking libraries in the mobile devices. By this information developers are able to make smarter data driven decisions with respect to making their app more successful.

However, most of these recommendations are more from a marketing perspective than software engineering perspective.

A. Challenges and Future Directions

Even though, it finally comes down to the amount of money made through an app in most cases, we as software engineering researchers care more about what makes an app successful. Success can mean different things to different people.

It could mean more downloads, it could mean driving more users to a business that is outside the mobile space, and it could mean just recognition by means of having a high rating. Thus success is not just one fixed measure but one from a set of possible measures depending on the context.

Depending on the choice of success measure, researchers can then come up with various hypotheses for what factors could affect this success measure. By gathering a set of possible factors (independent variables), and the success measure (dependent variable) for a large collection of apps from the app store, we can then model the data to see when an app can be successful. We can also see what factors are most related to the success measure and then carry out controlled experiments to see how far the correlations translate to causation.

There has been some recent initial work in this direction where Bavota et al. looked at the impact of using certain APIs on the ratings and model a set of factors (like size of app, complexity of app and its UI, quality of the library code etc.) against the ratings. They were able to find that there is initial evidence that high rated apps have a certain DNA.

In the future, we need to come up with more such factors to be evaluated, and strengthen our current findings with user studies. These factors can be derived through mobile app user and developer surveys.

B. Risks

While there is tremendous potential in determining under what circumstances an app will be successful, there are certain risks too. It will be easy to misinterpret the correlation in the data that we gather as causation. We need to be rigorous in controlling for other factors like category of apps, date apps were released, platform they run on etc. We also need to identify these factors based on common sense intuition and motivation based on previous work.

The paid monetization model is the traditional model, where the app developer sells the app directly to the user for a monetary price. There are has not been much software engineering research on the freemium/paid apps since they are more difficult for researchers to get access. By these limitations we simply do not know how results would generalize. Another challenge caused by the lack of access to historical data is the fact that success of an app can change overtime.

X. CONCLUSIONS

In conclusion, we believe that due to popularity of mobile apps, and the impact that research can have on developers from both small and large organizations, combined with the abundance of publicly available data, interesting research opportunities still left to be explored, and a vibrant community being built around it, software engineering research for mobile apps is a great place for young researchers to start.

REFERENCES

1. Syer and Ahmed E. Hassan “Examining the relationship between bugs warnings and end user ratings: A case study on 10,000 Android apps” in proceedings of the 56th international conference on software engineering, May 2015.
2. Mark Harman, Yue Jia and Yuanyuan Zhang “Test app store mining and analysis: Messenger for app stores” in proceedings of the 9th working conference on mining software repositories, switzerland, June 2012.
3. Galvis Carreno and Kristina Winbladh “Analysis of user comments: an approach for software requirements evolution” in proceedings of international conference on software engineering, 2013.
4. Rein and Munch “Explaining software defects using topic models” in proceedings of the 9th IEEE working conference on mining software repositories, 2009.
5. Gianpaolo Chen and Leandro Pinto “Self motion: a declarative language for adaptive service-oriented mobile apps” in proceedings of the ACM SIGSOFT 20th international symposium on the foundations of software engineering, 2012.
6. Shuai Hao, William G. J. Halfond and Ramesh Govindan “Estimating mobile application energy consumption using program analysis” in proceedings of the 35th international conference on software engineering, May 2013.

7. Pathak and Ahmed E. Hassan "Explaining software defects using topic models" in proceedings of the 9th IEEE working conference on mining software repositories, 2014
8. Geoffrey Li and Laurence Duchien "Tracking the software quality of android applications along their evolution" in 30th IEEE/ACM international conference on automated software engineering, 2015
9. Linares and David Wagner "Analyzing inter-application communication in Android" in proceedings of the 9th international conference on mobile systems, applications and services, 2011
10. Sadeghi and Andreas Zeller "Checking app behavior against app descriptions" in proceedings of the 36th international conference on software engineering, 2007
11. Vitalii Potharaju and Eric Bodden "Mining apps for abnormal usage of sensitive data" in 37th IEEE/ACM international conference on software engineering, Florence, Italy, 2015
12. Ajay Kumar Jha "A risk catalog for mobile applications Interface" Journal of computer engineering, science and technology, 2010.
13. Berg Insight "The mobile application market Online:http://www.berginsight.com/ReportPDF/ProductSheet/biapp1ps.pdf" 2013.
14. Steven Arzt, Eric Bodden, Jacques Klein and Patrick McDaniel "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle analysis for Android apps" in proceedings of the 35th ACM SIGPLAN conference on programming language design and implementation, 2008.
15. Hammad Khalid, Emad Shihab and Ahmed E. Hassan "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 foundations of software engineering, 2012.
16. Tillmann and Abram Hindle "Green mining: A methodology of relating software change to power consumption" in proceedings of the 9th IEEE working conference on mining software repositories, 2012.
17. Cuixiong Hu and Iulian Neamtiu "Automating GUI testing for android applications" in proceedings of the 6th international workshop on automation of software testing, 2014.
18. Y. Machiry and H. Leung, "Empirical analysis of software engineering for predicting high and low severity faults", IEEE transaction on software engineering, 32: 771-789, 2006.
19. R. S. Pressman, "Software engineering – a practioners approach" Fourth edition, Mc. Graw Hill International Edition., 1997.