How Do Android Operating System Updates Impact Apps?

Guowei Yang
Texas State University
San Marcos, TX, USA
gyang@txstate.edu

Jeffrey Jones
Rowan University
Glassboro, NJ, USA
jeffjones2994@gmail.com

Austin Moninger
Rice University
Houston, TX, USA
acm9@rice.edu

Meiru Che
Concordia University Texas
Austin, TX, USA
meiru.che@concordia.edu

ABSTRACT

Mobile devices are practically ubiquitous in today's society. People are increasingly dependent on mobile devices, for uses such as computation, navigation, storing private information, and web browsing among others. Thus, developers are required to produce high quality mobile apps. However, mobile operating systems are frequently updated, which can affect the functionality of mobile apps and hinder developers' ability to consistently provide high quality apps across multiple operating systems versions. In this paper, we introduce a novel approach for automatically locating the part of Android apps that have been affected by an update of the underlying Android mobile operating system, and statistically analyzing the impact of the update. Preliminary evaluation shows that the overall impact of an operating system update is low.

CCS CONCEPTS

• **Software and its engineering → Software evolution**; *Software testing and debugging*; • **Human-centered computing →** *Mobile devices*;

KEYWORDS

Android, change impact analysis, mobile operating system, API

ACM Reference Format:
Guowei Yang, Jeffrey Jones, Austin Moninger, and Meiru Che. 2018. How Do Android Operating System Updates Impact Apps?. In *MOBILESoft '18: MOBILESoft '18: 5th IEEE/ACM International Conference on Mobile Software Engineering and Systems , May 27–28, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3197231.3197258

1 INTRODUCTION

Mobile devices are increasingly popular and are practically ubiquitous in today's society. Mobile devices have become an integral part of life. According to a recent report of the global popularity of using different digital devices in nine countries, smartphones were the most popular internet device, compared to tablet, laptop, and desktop [1]. In particular, in 7 out of the 9 studied countries, more than 75% of all respondents use smartphones. Additionally, more than 40% of all respondents use tablets in these 7 countries. People are increasingly dependent on mobile devices, for uses such as computation, navigation, storing private information, and web browsing among others. Thus, developers are required to produce high quality mobile apps in terms of portability, reliability and security.

Mobile operating systems are frequently updated. More than 60 versions of Android mobile operating system have been released since the release of first commercial version Android 1.0 in 2008 [2]. The frequent updates of mobile operating systems can affect the functionality of mobile apps and hinder developers' ability to consistently provide high quality apps across multiple operating systems versions. A simple way of ensuring high quality mobile apps is through testing. In recent years, a great deal of research has been performed to improve the reliability of mobile apps by applying automatic testing [4, 5, 8, 15, 16, 19]. However, without knowing the impact of changes in mobile operating system, developers have to perform all tests for mobile apps with the new version of the operating system, which can be very expensive.

In this paper, we introduce a novel approach for change impact analysis of Android operating system. Our approach leverages Android API differences report, and statistically analyzes the use of these APIs in apps' source code. It automatically locates the portions of Android apps that have been affected by an update of the underlying Android operating system, and statistically analyzes the impact of the update in different granularities. Experiments based on 21 real-world Android apps were conducted to evaluate the effectiveness of our approach, and the preliminary results show that the overall impact of an operating system update is low.

2 APPROACH

2.1 Framework

Given that Android operating system is updated from an old version $v0$ to a new version $v1$ and the source code of an Android app that uses $v0$, our approach analyzes the impact of the operating system update on the app. We aim to provide an informative analysis of portions of code in the app that are impacted by an operating system update.

We leverage the Android API differences report to determine which parts of the Android libraries (the underlying operating system) have been modified from one API level to the next. API levels 19 through 26 are available online as API differences reports. For example, the API differences report for update from 24 to 25 is available at https://developer.android.com/sdk/api_diff/25/changes.html. The differences report lists elements including packages, classes, methods, and fields, in the libraries that are *added, changed, and removed*, respectively. For the impact analysis, the *added* elements are ignored as those elements are not used in the app and would have no effect on the app's functionality. If the *changed* or *removed* elements are used in the app, the app's functionality would probably be affected. Thus, we focus our analysis based on these two types of elements.

To determine how Android operating system is used in an app, we must have access to the app's source code. The *import* statements in the source code provide an insight into how the Android libraries are used within the application, along with how often they are used. For example, if a package is imported across multiple user-defined classes, it is used more often. Import statements are not perfect for analyzing how much of a package or class is used. One class could be imported but the code might not use a single method from it, while another could be imported with its methods used many times. However, we treat it as an efficiency trade-off because for large code base, analyzing how methods and fields were used would have large time costs.

Taking as input an Android app's source code and the API level that the app is being updated to, our approach analyzes the app for all the imported libraries and checks the Android API differences report for packages and classes that are in the app's source code to determine how the operating system update impacts the app. The algorithmic complexity is based around the number of difference on the API Differences Report (differences between packages, classes, methods, and fields).

2.2 Assumption

Note that our framework currently only considers "direct" impact of changed APIs, and may not account for "indirect" impact in cases where an element depends on another element in the operating system. To further illustrate this, consider two methods A and B, where A is invoked in B and A is modified in some way. If A is not used in the app while B is used in the app, our framework would consider there is no impact from the change; however, we can see that A would indirectly impact the app through method B. Our current framework does not have the capability to track or detect that dependency.

Since an exhaustive analysis of dependency between different elements in operating system would be too time-consuming, we conducted a manual (but not exhaustive) analysis of method calls and class usage to estimate this threat. We randomly selected two classes `android.platform.build.AutoGenTestConfigUnittests` and `android.os.core.PropFile` from Android OS version 8.1 (API 27), and manually checked where these two classes are used in the operating system. We found that `AutoGenTestConfigUnittests` is used once in Android Build System and `PropFile` is used once in Android Core. Thus, we assume that the indirect impact is not significant and leave the extension of our framework to include indirect impact analysis as future work.

2.3 Implementation

We implemented our approach using a series of scripts that use an application's source code and recursively finds all of the user-defined classes. From these user-defined classes, it traverses through to find the Android libraries being used. Once this process is completed, the tester must specify which API level the application is being updated to. Then, our framework web scrapes the corresponding Android Differences Report to find if the packages being used in the application have been changed or removed. If this is the case, it dives deeper to the class level to check again if the classes used in the code have been changed or removed. Once the updated libraries have been found, our framework will report the affected user defined classes with their changed or removed package and class. For each affected class, the framework outputs the changed and removed methods and fields that could potentially be used in the code.

The scripts are composed of a Bash script along with a group of Python scripts. The Bash script allows us to easily scan for the application's import statements within the Unix Terminal. Python was selected to analyze the API Differences Report because Python has a few helpful web scraping libraries such as `requests` and `Beautiful Soup`.

The framework also reports a small set of statistics that give a numerical value to the amount that the application is impacted by the update to the operating system. These statistics can be used to help developers estimate how significant the change impact of an operating system update on an app, and how much effort is needed to ensure the correctness of the impacted components of the app. As an application with a lower impact would need less time and effort for testing than an application with a higher impact, the numerical characterization is very valuable knowledge for developers because it will save them time and money by being able to run fewer test cases, and in turn spend more time developing and less time testing.

3 EVALUATION

3.1 Experimental Setup

In order to evaluate the effectiveness of our framework, we conducted experiments based on real-world Android apps, which were selected according to the following 3 criteria.

(1) Available on the Google Play Store [1]. This allows us to track the last time the application was updated as well as the number of times it has been downloaded. Also, these are real-world applications that anyone can download.
(2) Code is open sourced and available through GitHub [2]. This allows to easily obtain the source code in order to run our program on the application.
(3) Has over 10,000 installs. This ensures that the app is somewhat widely used. If an app like this had poor functionality after an operating system update, many users would be affected.

[1]https://play.google.com/store
[2]https://github.com/

App Name	Number of Installs	Category	Date Last Updated
Freeciv	500,000 - 1,000,000	Game	May 10, 2014
ownCloud	100,000 - 500,000	Productivity	May 29, 2017
FanFiction Reader	100,000	Books/Reference	August 14, 2016
iFixit	1,000,000 - 5,000,000	Books/Reference	February 18, 2016
2048	1,000,000 - 5,000,000	Game	September 6, 2016
OsmAnd	5,000,000 - 10,000,000	Maps/Navigation	August 2, 2017
SageMath	10,000 - 50,000	Education	October 9, 2014
Google IO	500,000 - 100,000	Books/Reference	May 16, 2017
Lightning Web Browser	500,000 - 1,000,000	Communication	July 9, 2017
Firefox Focus	1,000,000 - 5,000,000	Communication	July 28, 2017
KeePassDroid	1,000,000 - 5,000,000	Tools	January 6, 2017
Avare	100,000 - 500,000	Maps/Navigation	July 31, 2017
Barcode Scanner	100,000,000 - 500,000,000	Shopping	September 16, 2016
ConnectBot	1,000,000 - 5,000,000	Communication	February 21, 2017
Brave Browser	500,000 - 1,000,000	Communication	July 21, 2017
Mozilla Stumbler	50,000 - 100,000	Tools	May 10, 2017
BOINC	100,000 - 500,000	Education	July 3, 2016
Prey	1,000,000 - 5,000,000	Tools	August 2, 2017
Minimal To Do	10,000 - 50,000	Productivity	September 23, 2015
GLtron	10,000 - 50,000	Game	September 9, 2014
Ringdroid	50,000,000 - 100,000,000	Multimedia	December 2, 2016

Table 1: Android apps selected for evaluation.

These criteria made sure that the apps we were analyzing provided useful and meaningful data. Table 1 lists the apps that were selected for the experiments. For each app, it shows its name, number of installs, category, and the date when it was last updated. In total, 21 apps from 9 categories were selected. Most of the apps are widely used and well maintained, e.g., the app "Barcode Scanner" has $100 - 500$ million installs and $623,602$ reviews, and was last updated on September 16, 2016.

The experiments were run on Mac OS Sierra version 10.12.6 with a 2.5GHz Intel Core i5 processor and 16GB 1600 MHz DDR3 memory.

3.2 RESULTS

Table 2 shows the impact of Android operating system update on apps. For each selected app, the table shows its name and version, the update of operating system in terms of change in API level, the number and percentage of packages in the app's code that are impacted by the change, and the number and percentage of packages in the app's code that are impacted by the change. Overall, we find that the impact of an operating system update is low. For example, only 5.18% packages and 0.42% classes in `ownCloud` are impacted by the update of operating system from API 24 to API 25. If the impact is high, developers may benefit from running their entire test suite but our results indicate that it would be unnecessary, and selectively running part of the tests may save a lot of time and effort.

Even though the overall impact is low, larger scale apps are impacted less by operating system update than smaller scale apps, which makes sense because the Android libraries would make up a less significant portion of a larger scale apps. This means that it would be much more beneficial for our framework to be used on larger scale apps in order to focus on testing small impacted portions of code. Whereas with smaller apps, running the entire test suite may be a better option.

In addition, the more fine-grain the analysis is, the more accurate our change impact analysis would be and the less the impact would be. In our framework, we have implemented the class-level and package-level impact analyses. We find that the impact calculated by a fine-grain analysis (class-level) is always less than the impact calculated from a coarse-grain analysis. For example, there are 21.35% classes vs. 52.94% packages that were impacted for `Minimal To Do`. There is still room for delving deeper by determining how methods are impacted, which would involve more cost and is left for future work.

4 RELATED WORK

Change impact analysis [7] is an important and well studied problem in software evolution [6, 9, 12, 13, 17, 18, 20]. However, little work has been done on change impact analysis for Android apps.

Do et al. [10, 11] proposed an approach named Redroid for Regression Test Selection for Android apps. Redroid leverages the combination of static impact analysis and dynamic code coverage, and identifies a subset of test cases for re-execution on the modified version of Android apps. Li et al. [14] proposed ATOM, for automatic maintenance of GUI test scripts for evolving mobile apps. ATOM uses an event sequence model (ESM) to abstract possible event sequences in a GUI and a delta ESM to abstract the changes made to a GUI, and updates the test scripts that were written for the base version app, based on the ESM for the base version and the delta ESM for the changes introduced by the new version. GreenAdvisor [3] is a tool for analyzing the impact of software evolution on energy consumption. It compares the system call logs two consecutive versions of Android apps and predicts how the

App Name	App Version	API Update	# Impacted Packages	% Impacted Packages	# Impacted Classes	% Impacted Classes
Freeciv	2.6.0	21->22	8	33.33	0	0.00
ownCloud	2.5.0	24->25	10	5.18	3	0.42
Fanfiction Reader	1.56	24->25	9	9.57	2	0.71
iFixit	2.9.3	21->22	17	16.35	11	2.80
2048	2.08	24->25	8	44.44	1	4.17
OsmAnd	3.0	23->24	44	9.69	80	5.19
SageMath	1.0.1	21->22	18	23.08	15	5.81
GoogleIO	5.1.5	25->26	34	12.78	58	6.15
Lightning Web Browser	4.5.1	26->27	26	22.41	26	6.99
Firefox Focus	4.0	26->27	26	19.40	23	4.99
KeePassDroid	2.2.1	23->24	31	28.70	36	8.80
Avare	7.8.6	25->26	30	21.28	55	9.87
Barcode Scanner	1.9.8	25->26	27	23.68	44	10.38
Connectbot	1.9.2	25->26	27	21.60	48	11.16
Brave Browser	1.0.4	22->23	20	22.47	45	11.94
Mozilla Stumbler	1.8.6	23->24	34	23.13	70	13.21
BOINC	1.0	23->24	24	42.86	30	15.23
Prey	1.7.9	25->26	28	27.72	52	15.38
Minimal To Do	1.2	23->24	18	52.94	19	21.35
GLtron	1.1.2	25->26	12	57.14	13	26.00
Ringdroid	2.7.4	25->26	15	68.18	26	31.33

Table 2: Impact analysis results.

energy-consumption profile of the new version will compare to that of the previous version.

While the previous work focuses on impact of changes in Android apps on the functionality or energy consumption of Android apps, this work focuses on impact of changes in Android operating system on the functionality of Android apps.

5 CONCLUSIONS

In this paper, we presented an approach that automatically identifies the portions of Android app code that have been affected by an update of the underlying operating system. When the impact is small, the application of test selection and generation based on the impact results has a great potential to save the cost and effort involved in ensuring high quality apps after an operating system update. We hope our work paves the way to automating test case selection and generation for Android as well as other mobile operating systems across operating system versions. It is also important to look further into the possibility of dependencies between APIs and potentially develop a framework that can take those into account in future.

ACKNOWLEDGMENTS

This work was funded in part by the National Science Foundation under Grant Nos. CCF-1464123, CNS-1659807, and CRI-1305302.

REFERENCES

[1] 18 December 2017. International Communications Market Report 2017. Ofcom. (18 December 2017).

[2] 21 January 2018. Android version history. https://en.wikipedia.org/wiki/Android_version_history. (21 January 2018).

[3] Karan Aggarwal, Abram Hindle, and Eleni Stroulia. 2015. GreenAdvisor: A tool for analyzing the impact of software evolution on energy consumption. In *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015.* 311–320.

[4] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, and Atif M. Memon. 2012. Using GUI Ripping for Automated Testing of Android Applications. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012).* ACM, New York, NY, USA, 258–261. https://doi.org/10.1145/2351676.2351717

[5] Saswat Anand, Mayur Naik, Mary Jean Harrold, and Hongseok Yang. 2012. Automated Concolic Testing of Smartphone Apps. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE '12).* ACM, New York, NY, USA, Article 59, 11 pages. https://doi.org/10.1145/2393596.2393666

[6] Taweesup Apiwattanapong, Alessandro Orso, and Mary Jean Harrold. 2005. Efficient and Precise Dynamic Impact Analysis Using Execute-after Sequences. In *Proceedings of the 27th International Conference on Software Engineering (ICSE '05).* ACM, New York, NY, USA, 432–441. https://doi.org/10.1145/1062455.1062534

[7] Robert S. Arnold. 1996. *Software Change Impact Analysis.* IEEE Computer Society Press, Los Alamitos, CA, USA.

[8] Tanzirul Azim and Iulian Neamtiu. 2013. Targeted and Depth-first Exploration for Systematic Testing of Android Apps. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA '13).* ACM, New York, NY, USA, 641–660. https://doi.org/10.1145/2509136.2509549

[9] Krishnendu Chatterjee, Luca de Alfaro, Vishwanath Raman, and César Sánchez. 2010. Analyzing the Impact of Change in Multi-threaded Programs. In *Proceedings of the 13th International Conference on Fundamental Approaches to Software Engineering (FASE'10).* Springer-Verlag, Berlin, Heidelberg, 293–307. https://doi.org/10.1007/978-3-642-12029-9_21

[10] Quan Do, Guowei Yang, Meiru Che, Darren Hui, and Jefferson Ridgeway. 2016. Redroid: A Regression Test Selection Approach for Android Applications. In *The 28th International Conference on Software Engineering and Knowledge Engineering, SEKE 2016, Redwood City, San Francisco Bay, USA, July 1-3, 2016.* 486–491.

[11] Quan Do, Guowei Yang, Meiru Che, Darren Hui, and Jefferson Ridgeway. 2016. Regression Test Selection for Android Applications. In *Proceedings of the International Conference on Mobile Software Engineering and Systems (MOBILESoft '16).* ACM, New York, NY, USA, 27–28. https://doi.org/10.1145/2897073.2897127

[12] Malcom Gethers, Bogdan Dit, Huzefa Kagdi, and Denys Poshyvanyk. 2012. Integrated Impact Analysis for Managing Software Changes. In *Proceedings of the 34th International Conference on Software Engineering (ICSE '12).* IEEE Press, Piscataway, NJ, USA, 430–440. http://dl.acm.org/citation.cfm?id=2337223.2337274

[13] Andrew V. Jones. 2016. Addressing the Regression Test Problem with Change Impact Analysis for Ada. In *Proceedings of the 21st Ada-Europe International Conference on Reliable Software Technologies — Ada-Europe 2016 - Volume 9695.* Springer-Verlag New York, Inc., New York, NY, USA, 61–77. https://doi.org/10.1007/978-3-319-39083-3_5

[14] Xiao Li, Nana Chang, Yan Wang, Haohua Huang, Yu Pei, Linzhang Wang, and Xuandong Li. 2017. ATOM: Automatic Maintenance of GUI Test Scripts for

Evolving Mobile Applications. In *2017 IEEE International Conference on Software Testing, Verification and Validation, ICST 2017, Tokyo, Japan, March 13-17, 2017*. 161–171.

[15] Aravind Machiry, Rohan Tahiliani, and Mayur Naik. 2013. Dynodroid: An Input Generation System for Android Apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2013)*. ACM, New York, NY, USA, 224–234. https://doi.org/10.1145/2491411.2491450

[16] Riyadh Mahmood, Nariman Mirzaei, and Sam Malek. 2014. EvoDroid: Segmented Evolutionary Testing of Android Apps. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*. ACM, New York, NY, USA, 599–609. https://doi.org/10.1145/2635868.2635896

[17] Andy Maule, Wolfgang Emmerich, and David S. Rosenblum. 2008. Impact Analysis of Database Schema Changes. In *Proceedings of the 30th International Conference on Software Engineering (ICSE '08)*. ACM, New York, NY, USA, 451–460. https://doi.org/10.1145/1368088.1368150

[18] Xiaoxia Ren, Fenil Shah, Frank Tip, Barbara G. Ryder, and Ophelia Chesley. 2004. Chianti: A Tool for Change Impact Analysis of Java Programs. In *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA '04)*. ACM, New York, NY, USA, 432–448. https://doi.org/10.1145/1028976.1029012

[19] Wei Yang, Mukul R. Prasad, and Tao Xie. 2013. A Grey-box Approach for Automated GUI-model Generation of Mobile Applications. In *Proceedings of the 16th International Conference on Fundamental Approaches to Software Engineering (FASE'13)*. Springer-Verlag, Berlin, Heidelberg, 250–265. https://doi.org/10.1007/978-3-642-37057-1_19

[20] Lingming Zhang, Miryung Kim, and Sarfraz Khurshid. 2012. FaultTracer: A Change Impact and Regression Fault Analysis Tool for Evolving Java Programs. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE '12)*. ACM, New York, NY, USA, Article 40, 4 pages. https://doi.org/10.1145/2393596.2393642