Embracing Mobile App Evolution via Continuous Ecosystem Mining and Characterization

Haipeng Cai
haipeng.cai@wsu.edu
Washington State University, Pullman

ABSTRACT

While an indicator of its vibrancy, the rapid evolution of a mobile ecosystem also causes challenges to mobile software engineers in developing and maintaining quality products, and to users concerning the usability and security of resulting apps. In this context, it is crucially important to arm mobile software engineers with effective and practical tool support that is informed and enabled by a comprehensive understanding of the evolutionary dynamics of this ecosystem. Targeting Android, we envision to build an infrastructure that is capable of systematically and continuously mining a mobile software ecosystem. Using this infrastructure, we then perform large-scale longitudinal characterization studies of the ecosystem to understand its evolutionary dynamics with a focus on the behavioral evolution patterns of, and ecological interaction among, three ecosystem elements: the mobile platforms, user apps built on the platforms, and users associated with the apps (including end users and developers). Further, the characterization results enable proactive app quality and sustainable app security. We also report our current progress in this effort with initial results, and discuss risks and next steps.

CCS CONCEPTS

• **Software and its engineering → Software evolution**; **Maintaining software**;

KEYWORDS

Android, ecosystem, vision, characterization

ACM Reference Format:
Haipeng Cai. 2020. Embracing Mobile App Evolution via Continuous Ecosystem Mining and Characterization . In *IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems (MOBILESoft '20), October 5–6, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3387905.3388612

1 INTRODUCTION

The holistic mobile software *ecosystem*, including the run-time platform, apps, and user community, evolves rapidly to accommodate latest hardware advances and volatile end-user requirements. This is evidently true with Android, the long-dominating mobile operating system. As an example, the platform SDK of Android has experienced 28 API versions in less than ten years [31]. Any successful software is expected to evolve constantly [42, 43], so the rapid evolution of the Android ecosystem is a potential indicator of its vibrancy. Meanwhile, this rapid evolution also causes challenges to mobile software engineers in developing and maintaining quality app products, and hence challenges to users concerning the usability and security of resulting mobile software. Two prominent examples of such evolution-induced challenges, among many others, are the quick deterioration of security defense tools for Android [10, 14, 16, 29] and extensive compatibility issues with Android apps [23, 50].

To illustrate the first challenge, consider learning-based malware detection [47], a key mobile app security defense approach to the Android ecosystem. These detectors quickly became *outdated* due to the fast evolution of both the Android platform and its user apps. As an evidence, despite the existence of numerous Android malware detection techniques and tools, malware in the Android ecosystem still skyrockets with a new specimen emerging every 4.2 seconds [8].

A main reason that existing Android malware detection techniques do not sustain well lies in their failure to account for the evolutionary dynamics of the Android ecosystem [16]. Typically these techniques work by extracting certain sets of features from sample apps and then training a classifier based on the features. The evolution of Android itself and that of app development paradigms (e.g., how malware realizes security attacks), however, may largely impede or even render almost unusable the classifier in identifying new samples [10]. Retraining may not always be a solution either since new samples may not be as soon available as needed (e.g., for recognizing zero-day malware). For instance, a state-of-the-art malware detector, MamaDroid [39], can sustain reasonable accuracy for only one year—the technique degenerated to merely a random prediction (i.e., with 50% accuracy) when working on apps over two years newer than training samples. While not subject to exactly the same issue (known as concept drift in machine learning [34]), non-learning-based (e.g., signature-based) approaches are also vulnerable to the rapid evolution of the Android platform [13, 26, 47].

For the second challenge, consider the consequences of the Android ecosystem evolution on app development and testing. The Android platform evolves to harness the full potentials of new-generation hardware capabilities of the host device, while apps evolve to accommodate the evolution of Android platforms. This well-justifiable symbiosis results in issues including fragmentation and other complicated app/device compatibility issues [32, 48] which have become immediate barriers for app development, understanding, and testing. For example, as shown in a recent study on app compatibility issues [23], on overall average 15% of 62,894 sampled Android apps cannot be installed to, and 30–50% of installed apps cannot be normally executed on, one or more of eight Android platform versions involved in the study.

Evidently, well-informed Android app development and testing strategies require an understanding of the evolutionary dynamics of the Android ecosystem, as well as needed for sustainable app security. We believe that a comprehensive understanding of this kind can be achieved and maintained through continuous

MOBILESoft '20, October 5–6, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7959-5/20/05…$15.00
https://doi.org/10.1145/3387905.3388612

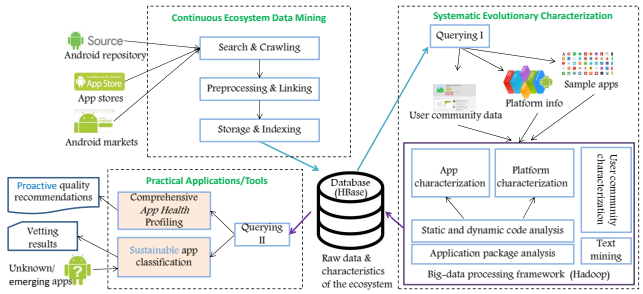

Figure 1: An overview of the envisioned approach.

ecosystem mining and characterization. The results would be a foundation for designing effective and practical tools that assist mobile app developers with developing quality and secure apps. To lay this foundation, we envision to create new knowledge about the Android ecosystem's evolutionary dynamics and, by exploiting the knowledge gained, to build a comprehensive semantic model of apps that measures their quality and guides the developers to create apps of proactive and sustaining quality and security.

Next, we outline our **vision** (§ 2) and justify **why it is new** (§ 3). Then, we present preliminary results on the envisioned approach and lay out the **next steps** in the proposed direction, while also discussing the **risks** with our approach (§ 4).

2 VISION

An overview of the envisioned approach is depicted in Figure 1. The continuous ecosystem mining infrastructure (§ 2.1) addresses data collection, storage, and analysis (e.g., computing app traits), offering high-efficiency data processing and access for data users (e.g., the evolutionary characterization study and applications/tools) through specialized database design and indexing scheme. Two different querying interfaces are dedicated for accessing raw ecosystem data (Querying I) and characterization results (Querying II). The underlying code analysis and non-code artifact processing are highly parallel, both between individual apps and platform versions and among the ecosystem elements. This is enabled by a unified big-data processing framework based on Hadoop customized for software data analytics. With these infrastructure facilities, the characterization (§ 2.2) focuses on empirical investigations, addressing key research questions to which the answers will lead to knowledge about the ecosystem evolutionary dynamics. The characterization results are then used to develop an app health profile that provides developers with recommendations toward proactive quality, and sustainable app security classification (§ 2.3). Note that the infrastructure facilities are spread out over the three technical components (marked by dashed rectangles) in the figure to better illustrate which parts of the infrastructure will be used where. Next, we outline the design of each of these three key components.

2.1 Continuous Ecosystem Data Mining

To enable an evolutionary study at an ecosystem level and a large scale, we first develop an infrastructure following a systems approach. This infrastructure provides basic data mining facilities that both extract app behavioral traits from multiple perspectives and garner user community data from various sources on the web. Efficient data harvesting and storage schemes are built in to facilitate continuous mining, so as to enable the collection of ecosystem data of maximal representativeness possible while the ecosystem evolves.

We mine three complementary and closely related categories of ecosystem data: *platform* (environment), *apps* (software), and *users* (human). The Android platform and its user applications constitute the complete Android software stack. Interacting with the stack are two groups of users: developers who directly interact with the platform and the end users who directly interact with app products. The interaction between developers and users is indirectly realized via the related apps. These three categories of data form a comprehensive sample description of the entire ecosystem of Android, and serve as the basis of ecosystem characterization. Since these categories are naturally distributed over different sources (top left of Figure 1), we need to discover the hidden links among them in order to study the ecosystem dynamics as a whole.

We focus on the evolution of **platforms** as reflected in the changing SDK/framework versions of Android. In particular, we target the Android source repository as the primary data source, available via the Android Open Source Project (AOSP) [3]. For each platform version, we mine varied kinds of metadata relevant to evolutionary dynamics, focusing on those about its SDK, API, permission systems, and targeted device specs, etc.

Our infrastructure mines large numbers of both benign and malicious **apps** of various kinds (e.g., functionality categories, malware families), including code and non-code artifacts of these apps. Benign apps are crawled from a large variety of sources, including alternative app stores in addition to the Google Play Store [30]. Malicious apps are collected from previously referenced data sources (e.g., [1, 2, 9, 37]). We also leverage web crawling to garner malware samples (especially emerging ones) in the wild.

User community data mining concerns two human factors of the ecosystem: (1) attributes of app developers relevant to app usage and quality/security and (2) views of app users toward the apps they use. In particular, we mine all currently known Android markets (app stores), and visible online Android forums to be discovered through web crawling (e.g., [4, 5]). From these sources, we scoop various community data, including (1) app description and other metadata, (2) app ratings, reviews, and discussions by end users, and (3) reputation records, development history, and focused app domain of the relevant developer.

To enable a principled data management scheme, we store our ecosystem data in a distributed database, using Apache HBase (Hadoop database) [7] in order to seamlessly integrate with the Hadoop big-data processing framework [6] that we use for parallel characterization. A holistic database schema is utilized for effectively archiving the three categories of ecosystem data, and link them through association analysis for storage efficiency (e.g., an app is linked to associated user and developer data as well as the information about the platform it is supposed to run on). Both querying interfaces (Querying I and II of Figure 1) are based on SQL, given its high interoperability with various other languages.

In addition to mining the (raw) ecosystem data, we also build into the infrastructure common facilities that compute various characteristics of the data. As shown in Figure 1, we have dedicated static and dynamic code analysis to empower the computation of code-based characteristics, for both app and platform characterization. Non-code artifacts are dissected through application package analysis. We leverage recent advances and latest utilities in textual data mining and natural language processing (e.g., PyTorch [25] and OpenNLP [12]) to analyze the user community data (which are mostly textual). Our characterization analyses are designed such that they well suit the Map-Reduce computation paradigm, in a seamless synergy with parallel data accesses from our database (since HBase is the default

data storage substrate for Hadoop). These analyses are detailed below (Section 2.2).

2.2 Systematic Evolutionary Characterization

To gain a comprehensive understanding of the evolving ecosystem, we systematically characterize it concerning its behavioral traits and evolutionary patterns. The characterization is systematic as it addresses the characteristics of each of the three ecosystem data elements individually, as well as the interactions (e.g., via correlations) among them.

The rich user apps constitute a vital means by which the Android system interacts with its users, thus they are the *foremost source* for reasoning about the evolutionary dynamics of the entire ecosystem. Our **app characterization** captures app behaviors in three orthogonal *dimensions* and three complementary *views*. The three dimensions were discovered from an exploratory study [21]: structure, concerning the composition and functionality distribution of an app and its executions with respect to the use of different code layers: user code (*UserCode*), third-party libraries (*3rdLib*), and the SDK (*SDK*); communication concerning inter-component communication (ICC) within single apps and across multiple apps; and sensitive access concerning the production, consumption, and potential leakage of sensitive data in an app and its executions. The three views were recently investigated [22]: static code view concerning all classes and methods appeared in the app APK, which capture how the app code is structured; callsite view concerning the presence of methods and their enclosing classes based on the associated callsite covered, ignoring the frequency of each call, which capture the diversity of class/method invocations; and instance view concerning all instances of class/method invocations, which capture run-time app behaviors that are reflected by call frequencies. Finally, we consider non-code-based metrics for each app mined from the manifest, assets, and resource files in the app APK—for example, the distribution of permissions requested in installation time, configuration of Intent filters for each declared component, package naming pattern in relation to the names of components declared (in the manifest file), repackaging indicators (e.g., file extension and content mismatch), and obfuscation indicators (e.g., use of native code and dynamic code loading).

Our **platform characterization** targets historical versions of the Android SDK/framework. In addition to the platform characteristics that are implicitly covered in the proposed app characterization (e.g., calls to/from SDKs), we characterize the design of permissions as part of the platform, as opposed to the use of permissions in user apps. For instance, we study in each SDK version the permission categorization and distribution over different security-sensitivity levels, specification-wise association between permissions and APIs, and device permissions in relation to hardware features. We also characterize the API of each platform version, by analyzing the distribution of APIs over different functionality categories, API compatibility with SDK versions, and categorization of security-sensitive APIs. We further study across those versions the API updates, including addition, deletion/deprecation, and signature (e.g., parameters and visibility) changes. While permissions and APIs are essential parts of the platform SDK, we further characterize the SDK from a code structure perspective. Specifically, we analyze per SDK version the properties of the SDK hierarchy at the levels of packages, classes, and methods (e.g., the inclusion relationship between a package and its member classes), distribution of functional features over packages, classes, and methods, and third-party libraries used. Across SDKs, we study updates of packages, classes, interfaces, methods, and annotations.

Knowing how human users are involved in the Android ecosystem is essential for understanding its evolutionary dynamics. Our **user community characterization** examines the Android user-community dynamics in two dimensions, using information mainly from app stores and optionally from quality online user forums (e.g., [11]). The first is end-user dynamics, regarding the change patterns with four informative measures: usage statistics (e.g., number of downloads) of user apps in different functionality categories and platform versions, distribution of end user demographics in relation to mobile device configurations, distribution of end user ratings of apps across functionality categories in relation to platform versions, and end user review quality in relation to app quality ratings and security/privacy concerns. The second is developer dynamics, concerning how developer characteristics change over time via three measures: the distribution of developer demographics in relation to app functionality categories, developer productivity (e.g., numbers of apps released per year), and developer trustworthiness and reputation according to the quality/security distribution of associated apps (as per end-user ratings and reviews).

Importantly, we study the interplay among the three ecosystem data elements via an in-depth *co-evolution* study. The primary aim is to discover the co-change and association patterns across these different data elements/modalities. Thus, our **cross-cutting characterization** focuses on multivariate statistic analyses and association (co-change pattern) mining, studying, among many others: the safest mechanism for declaring and requesting permissions during app installation and run time, whether unusable Intent fields should be explicitly nullified or left unassigned by default, which security tips are only applicable to specific platform versions and which are more generally applicable, the prominent security characteristics of the apps developed by highly rated developers, etc. The main results are actionable guidelines and recommendations for app developers.

2.3 Proactive Applications/Tools

We develop two practical applications/tools by immediately utilizing the diverse ecosystem characteristics with respect to each individual app and findings on evolution patterns of different app groups (e.g., malware versus benign apps).

First, we develop a comprehensive semantic model for Android apps, called *app health profile*, to describe various quality indices about an app, along with a tool for constructing this profile. This health profile provides a central reference for app developers to understand potential quality issues of an app being developed, so as to guide the developers to achieve higher app quality in a *proactive* manner. The profile is comprehensive in that it includes diverse quantitative metrics, such as those on compatibility, vulnerabilities, usability, and predicted user rating, each considered a health index. These indices are also ranked based on the severity (a numeric score representing each index) of the respective quality flaw to facilitate prioritized inspection by the developers.

Second, we explore an evolution-based methodology for achieving *sustainable* app security, by developing a sustainable app classifier that can offer competitive accuracy without constant retraining. We show achieving high sustainability in app classifiers as a key way to contain the current unending surge of malware in the Android ecosystem. We first conduct a differential characterization to identify metrics that significantly differentiate behavioral characteristics of app groups of different security

categories (e.g., between benign apps and malware, or between different malware families). The resulting metrics are then used as features to train a classifier. We expect this evolution-informed classifier to achieve and sustain high classification accuracy over time, based on our prior works in this regard [17, 18, 29].

As illustrated in Figure 1, these applications/tools retrieve both the raw ecosystem data and the ecosystem characterization results, via dedicated database querying interfaces. Note that the lasting relevance and validity of the health profiling and the sustainable app security defense solution are enabled by continuous mining and automatic updating via the proposed infrastructure. This way, the proactive quality and sustainable security will be sustained over time, which is a key merit of our approach.

3 EXISTING STUDIES

The need for understanding Android evolution has been recognized in prior work, with varying focuses on the particular aspect of the evolutionary dynamics (e.g., permission [49] and API [41]). Yet, we are not aware of any previous *systematic* study that investigates the evolutionary dynamics of a mobile software ecosystem holistically.

Existing characterizations for mobile software have addressed resource usage [28], battery consumption [27], code reuse [44], and ICC robustness [38] in Android. Yet they did not study the *evolution* of these characteristics *over time*. Few prior research involves the evolution of Android for a relatively short term [36, 41, 51] or targets a single particular scope, such as API [35] and permission [49]. Researchers have studied the evolution of apps in terms of the presence of anti-patterns [33], by looking into a few thousands of versions of a hundred of sample apps. The study was restricted to *static* code (syntactic) traits, like other studies being static also. In addition, previous works mostly focused on malware only (e.g., [45, 51]) and/or coarse characteristics [40, 46].

In contrast, our vision is to examine a comprehensive set of app characteristics, including code features, non-code artifacts, and run-time behaviors. Moreover, our study covers two other integral elements of the holistic ecosystem: platforms and user community. These elements have not yet been studied in relation to app characteristics.

4 CURRENT RESULTS AND NEXT STEPS

We have developed a toolkit for systematic app characterization [19, 20]. Using this toolkit, we sampled at least 1,000 benign apps and malware from each of eight past years, and characterized their behavioral evolution in terms of the proposed *structure* metrics on code layer interaction in the *instance* view [15]. Figure 2 depicts part of the results from this preliminary study, which revealed several interesting patterns of app evolution. First, both benign apps and malware had decreasing calls within user code and increasing calls within the SDK, with more abrupt changes seen in malware. Second, over time, malware almost constantly had most of its calls to SDK launched from various third-party libraries, while most of the calls targeting SDK in benign apps were launched from the same code layer. Third, generally the evolution of malware was much less predictable than that of benign apps.

These results have immediate implications for app quality. Steady drop in user-code involvement in app executions indicates the promise of prioritizing user code in app testing and security defense, assuming that the framework itself is secure and less defective. This strategy may be increasingly justifiable given the increasing portion of app executions being carried out through calls within the SDK and shrinking interaction between user code and libraries. Using the app characterization results, we have developed a prioritized

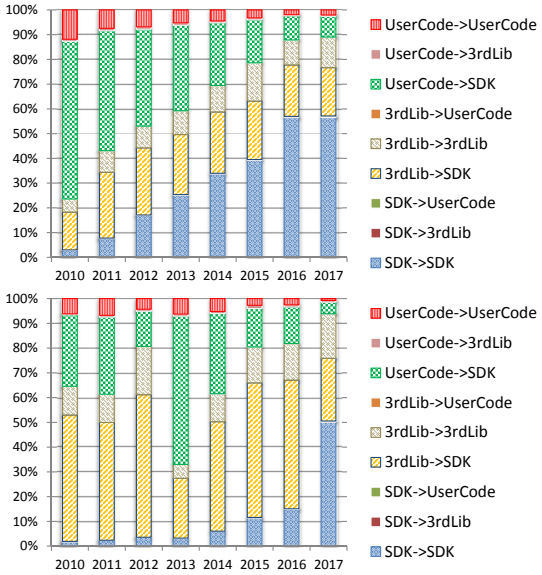

Figure 2: Example preliminary results on our app evolution characterization: cross-code-layer calling relationship distribution (y axis) of benign apps (top) versus that of malware (bottom) over eight historical years (x axis).

security analysis for inter-app communication risk analysis using a big-data processing framework, and demonstrated the usefulness of these results for sustainable app security by developing a malware detector based on the differential characteristics between malware and benign apps [17, 18].

Our immediate next step is to implement the utilities needed for platform and user community characterization, reusing relevant static and dynamic code analysis tools developed for the app characterization. We will then perform respective characterization studies and cross-cutting characterization as outlined earlier, and based on the empirical findings and observations distill practical and actionable recommendations (e.g., general coding guidelines, configuration rules, and testing caveats for app development and maintenance). We will also enhance the tool for health profile construction for a given app, by integrating our prior work on computing some of the health indices (e.g., incompatibility detection and repair [24]). By incorporating platform and user community characterization results, we expect to enhance our sustainable app security classification approach, on top of the current, preliminary technique based on app characteristics only. Finally, we envision our approach to be deployed as a service to perform *continuous* mining and characterization of the Android ecosystem in an autonomous manner.

There are several sources of risks with our approach. First, in addition to known challenges (e.g., scalability, precision) to app analysis needed for app characterization, our ecosystem mining infrastructure may suffer from scarcity of data sources (e.g., developer data and app reviews) and inaccuracy (e.g., incompleteness) of data crawling. Second, for characterizations, a main risk lies in the possible insufficient capabilities of underlying (e.g., text mining, natural language processing) techniques, which would result in unreliable results. Finally, although our preliminary results showed its merits, our approach might not generalize well into the future of the ecosystem evolution (e.g., the patterns we observed in the studied history may not continue to hold beyond).

REFERENCES

[1] 2016. MalwareDB. http://thezoo.morirt.com/.

[2] 2017. Android Malware. https://github.com/ashishb/android-malware.

[3] 2017. Android Open Source Project. https://source.android.com/.

[4] 2017. AndroidCentral. https://www.androidcentral.com/.

[5] 2017. AndroidForums. https://androidforums.com/.

[6] 2017. Apache Hadoop. https://hadoop.apache.org//.

[7] 2017. Apache HBase. https://hbase.apache.org//.

[8] 2017. Mobile malware growth. https://www.gdatasoftware.com/blog/2017/04/29666-malware-trends-2017.

[9] 2017. VirusShare.com. http://www.virusshare.com/.

[10] Kevin Allix, Tegawendé F Bissyandé, Quentin Jérome, Jacques Klein, Yves Le Traon, et al. 2016. Empirical assessment of machine learning-based malware detectors for Android. *Empirical Software Engineering* 21, 1 (2016), 183–211.

[11] androidappsreview.com. 2017. Android Apps Reivew. http://www.androidappsreview.com/.

[12] Apache. 2018. OpenNLP. https://opennlp.apache.org/.

[13] Vitalii Avdiienko, Konstantin Kuznetsov, Alessandra Gorla, Andreas Zeller, Steven Arzt, Siegfried Rasthofer, and Eric Bodden. 2015. Mining apps for abnormal usage of sensitive data. In *Proceedings of IEEE/ACM International Conference on Software Engineering*. 426–436.

[14] Haipeng Cai. 2020. Assessing and improving malware detection sustainability through app evolution studies. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 29, 2 (2020), 1–28.

[15] Haipeng Cai, Xiaoqin Fu, and Abdelwahab Hamou-Lhadj. 2020. A Study of Run-time Behavioral Evolution of Benign versus Malicious Apps in Android. *Information and Software Technology* (2020), 106291.

[16] Haipeng Cai and John Jenkins. 2018. Towards sustainable android malware detection. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. 350–351.

[17] Haipeng Cai, Na Meng, Barbara Ryder, and Danfeng Yao. 2017. *DroidCat: Unified Dynamic Detection of Android Malware*. Technical Report TR-17-01. http://hdl.handle.net/10919/77523.

[18] Haipeng Cai, Na Meng, Barbara Ryder, and Danfeng (Daphne) Yao. 2019. DroidCat: Effective Android Malware Detection and Categorization via App-Level Profiling. *IEEE Transactions on Information Forensics and Security (TIFS)* 14, 6 (2019), 1455–1470.

[19] Haipeng Cai and Barbara G Ryder. 2017. Artifacts for Dynamic Analysis of Android Apps. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 659–659.

[20] Haipeng Cai and Barbara G Ryder. 2017. DroidFax: A toolkit for systematic characterization of Android applications. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 643–647.

[21] Haipeng Cai and Barbara G Ryder. 2017. Understanding Android application programming and security: A dynamic study. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 364–375.

[22] Haipeng Cai and Barbara G Ryder. 2020. A Longitudinal Study of Application Structure and Behaviors in Android. *IEEE Transactions on Software Engineering* (2020).

[23] Haipeng Cai, Ziyi Zhang, Li Li, and Xiaoqin Fu. 2019. A large-scale study of application incompatibilities in Android. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 216–227.

[24] Malinda Dilhara, Haipeng Cai, and John Jenkins. 2018. Automated detection and repair of incompatible uses of runtime permissions in Android apps. In *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*. 67–71.

[25] Facebook. 2018. PyTorch. https://pytorch.org/.

[26] Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan. 2015. Android security: a survey of issues, malware penetration, and defenses. *IEEE Communications Surveys & Tutorials* 17, 2 (2015), 998–1022.

[27] Denzil Ferreira, Anind K Dey, and Vassilis Kostakos. 2011. Understanding human-smartphone concerns: a study of battery life. In *Pervasive Computing*. 19–33.

[28] Denzil Ferreira, Vassilis Kostakos, Alastair R Beresford, Janne Lindqvist, and Anind K Dey. 2015. Securacy: an empirical investigation of Android applications' network usage, privacy and security. In *Proceedings of ACM Conference on Security & Privacy in Wireless and Mobile Networks*. 1–11.

[29] Xiaoqin Fu and Haipeng Cai. 2019. On the deterioration of learning-based malware detectors for Android. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 272–273.

[30] Google. 2017. Official Android app stores. https://play.google.com/store/apps?hl=en.

[31] Google. 2018. Android History. https://www.android.com/history/.

[32] Dongjie He, Lian Li, Lei Wang, Hengjie Zheng, Guangwei Li, and Jingling Xue. 2018. Understanding and detecting evolution-induced compatibility issues in Android apps. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 167–177.

[33] Geoffrey Hecht, Omar Benomar, Romain Rouvoy, Naouel Moha, and Laurence Duchien. 2015. Tracking the software quality of android applications along their evolution (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 236–247.

[34] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting concept drift in malware classification models. In *PROCEEDINGS OF THE 26TH USENIX SECURITY SYMPOSIUM (USENIX SECURITY'17)*. USENIX Association, 625–642.

[35] Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. 2013. API change and fault proneness: a threat to the success of Android apps. In *Proceedings of ACM International Symposium on the Foundations of Software Engineering*. 477–487.

[36] Martina Lindorfer, Matthias Neugschwandtner, Lukas Weichselbaum, Yanick Fratantonio, Victor van der Veen, and Christian Platzer. 2014. Andrubis - 1,000,000 Apps Later: A View on Current Android Malware Behaviors. In *Proceedings of International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*. 3–17.

[37] Martina Lindorfer, Matthias Neugschwandtner, Lukas Weichselbaum, Yanick Fratantonio, Victor Van Der Veen, and Christian Platzer. 2014. Andrubis–1,000,000 apps later: A view on current Android malware behaviors. In *Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS), 2014 Third International Workshop on*. 3–17.

[38] Amiya K Maji, Fahad A Arshad, Saurabh Bagchi, and Jan S Rellermeyer. 2012. An empirical study of the robustness of inter-component communication in Android. In *Proceedings of Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 1–12.

[39] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. 2017. MAMADROID: Detecting Android Malware by Building Markov Chains of Behavioral Models. In *Proceedings of Network and Distributed System Security Symposium*.

[40] William Martin, Federica Sarro, Yue Jia, Yuanyuan Zhang, and Mark Harman. 2016. A survey of app store analysis for software engineering. *IEEE Transactions on Software Engineering* (2016).

[41] Tyler McDonnell, Bonnie Ray, and Miryung Kim. 2013. An empirical study of API stability and adoption in the Android ecosystem. In *Proceedings of IEEE International Conference on Software Maintenance*. 70–79.

[42] Vaclav Rajlich. 2006. Changing the paradigm of software engineering. *Commun. ACM* 49, 8 (2006), 67–70.

[43] Vaclav Rajlich and Prashant Gosavi. 2004. Incremental change in object-oriented programming. *Software, IEEE* 21, 4 (2004), 62–69.

[44] Israel J Mojica Ruiz, Meiyappan Nagappan, Bram Adams, and Ahmed E Hassan. 2012. Understanding reuse in the Android market. In *Proceedings of IEEE International Conference on Program Comprehension*. 113–122.

[45] Aubrey-Derrick Schmidt, Hans-Gunther Schmidt, Leonid Batyuk, Jan Hendrik Clausen, Seyit Ahmet Camtepe, Sahin Albayrak, and Can Yildizli. 2009. Smartphone malware evolution revisited: Android next target?. In *Proceedings of IEEE International Conference on Malicious and Unwanted Software*. 1–7.

[46] Kimberly Tam, Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, and Lorenzo Cavallaro. 2017. The evolution of Android malware and Android analysis techniques. *Comput. Surveys* 49, 4 (2017), 76.

[47] Darell JJ Tan, Tong-Wei Chua, Vrizlynn LL Thing, et al. 2015. Securing Android: a survey, taxonomy, and challenges. *Comput. Surveys* 47, 4 (2015), 1–45.

[48] Lili Wei, Yepang Liu, and Shing-Chi Cheung. 2016. Taming Android Fragmentation: Characterizing and Detecting Compatibility Issues for Android Apps. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. 226–237.

[49] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos. 2012. Permission evolution in the Android ecosystem. In *Proceedings of Annual Computer Security Applications Conference*. 31–40.

[50] Ziyi Zhang and Haipeng Cai. 2019. A look into developer intentions for app compatibility in Android. In *2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 40–44.

[51] Yajin Zhou and Xuxian Jiang. 2012. Dissecting Android malware: Characterization and evolution. In *Proceedings of IEEE Symposium on Security and Privacy*. 95–109.