Open Research Online

The Open University's repository of research publications and other research outputs

Better Android Apps Using Android Vitals

Conference or Workshop Item

How to cite:

Harty, Julian and Müller, Matthias (2019). Better Android Apps Using Android Vitals. In: Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics (WAMA '19), 27 Aug 2019, Tallinn, Estonia, ACM, pp. 26–32.

For guidance on citations see [FAQs](#).

© 2019 The Authors

<https://creativecommons.org/licenses/by-nc-nd/4.0/>

Version: Accepted Manuscript

Link(s) to article on publisher's website:

<http://dx.doi.org/doi:10.1145/3340496.3342761>

Copyright and Moral Rights for the articles on this site are retained by the individual authors and/or other copyright owners. For more information on Open Research Online's data [policy](#) on reuse of materials please consult the policies page.

oro.open.ac.uk

Better Android Apps using Android Vitals

Julian Harty
julianharty@gmail.com

Commercetest Limited/Open University
High Wycombe, England

Matthias Müller
mueller@ist.tugraz.at

Graz University of Technology
Graz, Austria

ABSTRACT

Google provides Android Vitals, a set of reports and tools for Android Developers as part of Google Play Console. Android Vitals can help developers improve their Android apps after an app has been launched by providing information on how their app is performing in key areas such as battery use, performance, and stability (freezes and crashes). Android Vitals also provides various comparisons, including against global bad behavior thresholds, against various peer groups of apps, and across releases of this app.

Developers confirm Android Vitals notifies them of relevant problems and they found it valuable even if they also use crash reporting and mobile analytics.

The underlying data is used by Google to assess the relative quality of Android apps; and the perceived quality may materially affect the visibility of an app in the Google Play Store. Yet little is known about the tools.

This paper outlines various experiences from the developers' perspective of using Android Vitals with several popular Android apps to help open discussions and suggest further research areas. It introduces an open source project, created as part of our work, that enables developers to download pertinent data, particularly crash reports. The data can be analysed both by the development team and others. A particular benefit of this tool is to make the data available outside of the Google platform, which allows developers and (indirectly) researchers to develop additional analysis techniques not currently provided by the platform.

CCS CONCEPTS

• **Software and its engineering** → **Application specific development environments; Software testing and debugging; Empirical software validation.**

KEYWORDS

Android Vitals, App development, Feedback and reputation, Open-source, Quality of apps

ACM Reference Format:

Julian Harty and Matthias Müller. 2019. Better Android Apps using Android Vitals. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics (WAMA '19)*, August 27, 2019, Tallinn, Estonia. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3340496.3342761>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

WAMA '19, August 27, 2019, Tallinn, Estonia

© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-6858-2/19/08...\$15.00

<https://doi.org/10.1145/3340496.3342761>

1 INTRODUCTION

Android Vitals presents developer-centric information deemed 'vital' by Google for the viability of the developer's Android apps. Apps that perform well may be promoted by Google in the Play Store, increasing their popularity. Performant apps may decrease their rejection and/or abandonment rates, and increase their retention[6].

Android Vitals is possible because it utilizes Google software running on the Android platform that collects analytic data from devices based on usage by the crowd. The platform is the observer, and the measures are outside any particular app; instead, it measures the behaviours of all the apps installed on that device. From a developer's perspective, they do not need to modify their software or ask users for permission to obtain the information. Their app is also compared to several peer groups, and to blanket performance thresholds determined by Google.

Figure 1: Android Vitals data flows

Google analyses the data it collects from users' devices and also provides developers with data collected about their apps' behaviour, such as consumption of resources including power and networking. However there has been little opportunity for researchers to perform empirical research on Android Vitals, as they would need access to accounts for each app they wish to investigate.

As figure 1 shows, Android Vitals collects usage and run-time data from Android devices running Google Mobile Services¹, from users who agreed to provide diagnostic data. It is available to Android developers who have active apps in Google Play; these developers can provide granular access to other people who have a valid Google Account. Android Vitals is part of Google Play Console which also incorporates: release management (including pre-launch reports), store presence, user acquisition, and user feedback. Google Play Console links issues found in the pre-launch reports (that were found before the app was released) to failures that are discovered in production and reported in Android Vitals[7].

There is a loose hierarchy of information in Android Vitals, allowing developers to drill-down to obtain more details. Table 1

¹<https://www.android.com/gms/>

Table 1: Conceptual 'levels' of Android Vitals

Level	Source	Page name
0	Google Play Console	AppListPlace
1	[app] Dashboard	AppDashboardPlace
2	Android Vitals overview	AppHealthOverviewPlace
3	Details of a 'Bad behavior' ¹	AppHealthDetailsPlace
3	Tables of crashes & ANRs[7]	AndroidMetricsErrorsPlace
4	Cluster details	AndroidMetricsErrorsPlace... &clusterName=...

Figure 2: Crash rate for the Android Vitals Chemistry & Physics simulations app

indicates conceptual levels for various aspects of Android Vitals for a developer's account.

Android Vitals becomes increasingly useful as the volumes of user-sessions, and [paradoxically] problems, increase. Essentially it is driven by the usage of Android apps by end-users who also opt to automatically share their usage and diagnostic data[7]. Google Play Console includes a statistics tool which can incorporate data from its various tools, including data from Android Vitals, into custom reports. The reporting capabilities are useful despite being limited. For example, as figure 2 indicates, we observed the crescendo of crashes after a new release of one of the apps, and could plot up to 6 items when graphing the Android versions.

Based on the experience of the authors and others in industry, comprehensive testing - particularly of Android apps - is impractical given the many factors outside the control of the development team. Kong *et al* notes the complexity and variety of research into a subset of the challenges, *i.e.* automated testing[12]. Even very diligent teams do not find all the bugs in their app, and other teams often rely on more perfunctory testing which may miss many more issues. In contrast, actual use by end users may well expose flaws and failures in the app compared to testing by the developers.

Android Vitals may help developers use a heuristic based on **relative correctness**[5] where they compare the crashes and 'Application Not Responding' (ANR) rates across the releases of their apps in use. A potential process could be:

- Developers release one or more versions of their app. They use Android Vitals to track the relative crash rate per release. They identify flaws in the app from the crash clusters and other Android Vitals metrics and decide which ones to address.
- Developers make changes to the app. These may include updates to third-party libraries, to error handling, and/or improvements to the implementation. The app is uploaded to the app store and released to end users. Test releases may also help; however they're unlikely to provide the volumes or variety that production use does, so the more compelling results are likely to come from production.
- Through ongoing and regular checks of the active releases of the app, developers can track changes to (a) individual crash clusters and (b) to aggregate crash rates to assess the relative correctness.

Google offers a confidence interval estimate for several measurements. For example, how crashes vary on an ongoing basis depending on usage among other factors. These intervals seem quite broad, as indicated in Figure 4, where they span approximately 5% for crash rates in the 3% to 4% range *i.e.* they are larger than the figures being quoted. Therefore some improvements might be lost in the variations and/or the confidence interval, nonetheless egregious error rates are easy to detect, as Figure 2 indicates, and similarly significant improvements may also be visible using Android Vitals.

Application failures: crashes and freezes (ANR's) are clustered together in Android Vitals and available for developers to see online as figure 3 shows. Android Vitals tracks the frequency of each cluster and estimates how many users were affected. As an example the values for this crash cluster are: 49 reports in the last 7 days, and 473 reports affecting 207 users for the lifetime of this cluster.

2 RELATED WORK

Android Vitals was first examined by Harty[10] using a case study as well as identifying various flaws in the reports.

App Stores and their ecosystems have been studied and researched extensively; for instance in 2016, Martin *et al.* performed a comprehensive survey of research related to the app store ecosystem [14]. Harman *et al.* introduces two relevant topics: app store analysis app testing and optimisation[9], yet does not discuss possible relationships between these topics. Sibaihin *et al.* discusses alpha and beta testing by humans using facilities provided by the app store[1]; their paper does not cover pre-launch testing, other release management tools, or Android Vitals.

Debugging of failures found in the field has a rich history, for instance the work of Jin and Orso in 2012[11], albeit outside the domain of mobile apps. The concept of 'Debugging without Testing'[5] uses Relational Mathematics rather than data from the field.

Data collected by Google from devices can be complemented by other approaches, for instance where information is mined from reviews to guide testing of Android apps. The work of Grano *et al.*[8] matched reviews to crash stack traces found using two automated testing tools: Sapienz[13] and Android Monkey[4]. The crashes they found were triggered by automated tests rather than by usage of the apps by end users. Potentially the crash reports presented in

¹Details of critical issues identified by Android Vitals https://support.google.com/googleplay/android-developer/answer/7385505?#review_metric_page

Figure 3: A Crash Cluster reported in Android Vitals

Android Vitals could be used as a complementary source of stack traces using a similar approach to theirs.

3 AIMS

The authors wanted to investigate Android Vitals and related tools provided in the Google Play Console. We chose to compare the experiences of several development teams and obtain data, with their permission, from their Android Vitals.

As several features of Android Vitals are only available for apps with significant volumes of data, our research involves a combination of direct research for Android apps the authors co-develop, with discussions with development teams for other Android apps. Those developers also provided various information from their Android Vitals accounts. For our projects (Julian: the Kiwix apps, Matthias: PocketPaint as part of the Catroid project) we had direct access to other developers and Android Vitals for our apps. For other apps, the developers were interviewed using a combination of in-person discussions, video conferencing, with follow-up work using email. Developers provided screenshots of content from their Android Vitals accounts. Each developer also confirmed that they, and where appropriate their organisations, were happy for us to use and share their contributions.

The first author also co-developed software, called Vitals Scraper [18] that downloads details of the crashes for a given combination of Android developer account and application id (to also support developers who have more than one organization and/or more than one app in the app store). Data is retrieved from the web page AndroidMetricsErrorsPlace and includes the contents of each

crash cluster, as shown in figure 3. The scraper pages through the individual reports of the failure and downloads the device model, Android version and stack trace. This software has been made available as an open source project. It works as a 'screen scraper' and has been implemented using an embedded web browser where the user of the tool needs to authenticate using their Google account in order to access any of the data, so it relies on the security and access mechanisms used by Google.

Each crash report is downloaded for the maximum period supported by Android Vitals (to maximise the raw data that is collected) and stored using the JSON format as a single file. It runs at a measured pace to avoid any limits imposed within the Google Play Console service.

The resulting JSON file can be shared with interested researchers to analyse it and using the file can decouple access to Google Play and Android Vitals from access to the data. As Google only retains details of the crashes for approximately 60 days, these files would need to be generated and stored on a regular basis (at least every 60 days) to enable longer-term analysis.

4 FINDINGS

4.1 Grouping Data is Insightful

For three of the five applications listed in table 3, the crash rate increases with newer Android versions. The Chemistry & Physics simulations app has a slightly higher crash rate for Android 8.1 than Android 9, yet both are significantly higher than the crash rate reported for Android 7. The causes for the crash rates of the Moonpig app are provided by one of their developers later in this paper.

Table 2: WikiMed (French)

(a) By Android version			
Android version	Impacted sessions	Crash-free sessions	Number of sessions
8.1	1.33%	98.67%	2K
8	1.05%	98.95%	2K
(b) By app version			
App version	Impacted sessions	Crash-free sessions	Number of sessions
1182450	1.34%	98.66%	4K
182450	1.15%	98.85%	3K

4.2 Gaps in the Data

Tables 2a and 2b represent the equivalent reports in Android Vitals for the WikiMed in French Android app; and present two different views of the crashes - there seems to be a data loss of approximately 3,000 sessions in the 'By Android version' table. It would be useful to understand reasons for the differences and a possible area for future research.

Figure 4 happens to capture a fairly frequent occurrence where Google's graphs have gaps in them. In the statistics section of

Figure 4: Example of the Google's confidence interval for crashes

Google Play Console there is some data for these dates so the cause of the gap is hard to fathom for users of these reports.

4.3 When does Android Vitals Provide Reports

As Google only provides various reports in Android Vitals once they decide enough data exists to preserve the privacy of end users, Android Vitals provides little for developers of less popular apps.

Based on Android Vitals reports for Kiwix custom apps, we infer that few apps with less than 20,000 total installs will have any detailed reports; WikiMed in Spanish has 5,373 active installs and has only one report, for crash rate 'By app version'. None of the apps with lower active install counts have any reports. Instead, the Android vitals overview page says "Google doesn't yet have enough data to surface Android vitals for your app."

Even for more active apps results are hard to predict. WikiMed in Arabic with 13,484 active installs has a crash rate of 2.21% but lacks the 'By Android version' breakdown of crashes; whereas WikiMed in French with 13,027 active installs has a crash rate of 1.23% and has a partial set of results in 'By Android version'.

4.4 Triaging Crash Clusters

Borrowing from the concept of triage in medicine, in our discussions we discovered developers also triage the crash clusters into one of three groups:

- Crashes they know they cannot address practically. For example: PocketPaint has crashes that affect a single user who continues to use a much earlier release of the app. Although the crash happens frequently for that user, it has been addressed in more recent releases and once the user upgrades they should have an improved user-experience.
- Crashes they already know how to address. For example:
 - Kiwix: numerous crashes in the 'download manager' led to two independent replacements being written.
 - Moonpig: crashes reported in RoboSpice. The team plan to replace the library and modify the app to use a newer, actively supported library.
- Crashes they see value in investigating further. For example: crashes reported in several of the kiwix apps for `org.kiwix.kiwixmobile.utils.files.FileUtils.contentQuery`. They

Table 3: Sessions impacted by crashes

Android Version					Overall	App
6.0.1	7	8	8.1	9	Crashes	
0.42%	1.43%	3.48%	3.48%	6.49%	4.05%	Kiwix ¹
	0.45%	0.75%	0.89%	1.52%	1.07%	WikiMed (en) ²
		2.69%	4.07%	3.77%	3.45%	PhET Simulations ³
	0.06%	0.14%	0.09%	0.93%	0.61%	Moonpig ⁴
		1.38%	1.51%	2.16%	1.66%	Pocket Paint ⁵

¹ <https://play.google.com/store/apps/details?id=org.kiwix.kiwixmobile>

² <https://play.google.com/store/apps/details?id=org.kiwix.kiwixcustomwikimed>

³ Chemistry & Physics simulations app <https://play.google.com/store/apps/details?id=org.kiwix.kiwixcustomphet> using material from <https://phet.colorado.edu/>

⁴ <https://play.google.com/store/apps/details?id=com.commonagency.moonpig.uk>

⁵ <https://play.google.com/store/apps/details?id=org.catrobat.paintroid>

occur frequently, are important to address, yet the team do not yet know enough to deal with the problems.

5 VIGNETTES

To highlight the benefits of Android Vitals from the perspective of developers, we provide examples from developers from several popular Android apps. The apps are Pocket Paint², a simple drawing app, and Moonpig³, an app that enables customers to select, customize, and buy greetings cards. Some additional examples for apps published by the Kiwix project are available in a previous paper [10].

5.1 Pocket Paint

Pocket Paint is a simple open source drawing app developed by the non-profit Catrobat project. First released in 2013, the project is published as standalone app with over 40,000 active users and also included as a library within the visual programming app *Pocket Code*. The open source project is primarily driven by students who are gaining real-world experience as part of their university education [15]. Therefore, the project not only addresses general qualitative requirements for published apps, but has also a "best practice" focus on agile software development and quality.

Although Catrobat follows a test-driven development (TDD) approach, users regularly report bugs, often specific to Android-versions or devices. As a result, in addition to automated tests, intensive manual testing has been performed on various devices before releases. Nevertheless, releases often included bugs that eluded the testing process. Some only occurred in certain languages or on specific devices. Thus, although the project is test driven and manual testing has been performed by a multitude of developers on different devices, certain bugs only emerged post-release.

As an example, a release in early 2019 resulted in a significant increase of crashes caused by a new bug. The developers then noticed the pre-launch report provided by the Google Developer Console, would actually have identified this bug before the production release. However, it was not considered in the release process at that time. Consequently, new versions now only get released after a pre-launch report is available and positive. Also non-urgent releases,

²<https://catrobat.at/pp>

³<https://www.moonpig.com/uk/>

that do not include security or important fixes, are rolled out in stages. This allows the development team to learn about the app's vitality early, before problems affect the entire user-base.

Also, crash-reports are now actively used to identify new issues. However, to optimize the app, tools are used to shrink and obfuscate releases, preventing Google Vitals from providing deep insights without the required mapping file. Future work is intended to provide this mapping, and also to use more sophisticated tools, to augment the data provided by Google's developer console.

This case outlines the importance of analyzing an app's quality both before and after a release. Even a large set of automated tests and performing manual testing is insufficient. The diverse market situation, e.g. the different devices and Android versions, increases the need to support developers through tools that provide further information about an app's health. In particular, analyzing Google Play Console supports developers in various ways:

- Identifying crashes in pre-launch reports before the app is launched. Furthermore these crashes are discovered without needing users' feedback.
- By providing detailed information to help localize issues to Android version, device or language, which may help to find and fix specific bugs faster.

5.2 Moonpig

Moonpig is a business that relies on the internet to enable customers to order custom greetings cards and related items. Their Android app has approximately 130,000 active installs and is actively developed and enhanced. The app combines several generations of their architecture and uses various third-party libraries. The development team use Firebase Crashlytics and Google Analytics for diagnostics in addition to the information available in Android Vitals; and estimate they use Android Vitals approximately 30% of their time to identify flaws and issues related to their Android app.

As Table 3 shows their app is the most reliable of those included here, however the crash rate on Android 9 (0.93%) is at least six times higher than for other versions of Android and is the main reason why the overall crash rate is 0.61% rather than around 0.1%.

For the Moonpig app, a major cause of the higher crash rate on Android 9 is their use of an older third-party library RoboSpice⁴. Behavioural changes in Android 8 and more recently in Android 9 meant this library was no longer viable and this open source library was archived in January 2018. One of the main reasons given by the lead maintainer was changes in Android 8.0[16]; and unsurprisingly the library was not developed with Android 9.0 in mind.

Documentation for Android 8.0 explained key behavioral changes in background processing⁵. However, for the Moonpig Android app the crash rate remains low on Android 8.0 (0.14%) and 8.1 (0.09%) yet increased several fold to 0.93% on Android 9 devices. According to Jakob Durstberger one of the Moonpig Android developers: "As far as we are aware it is related to an issue on Android 9 that triggers an `IllegalStateException` when launching a service after resuming". He provided a reference to the underlying issue on StackOverflow⁶.

⁴<https://github.com/stephanenicolas/robospice>

⁵<https://developer.android.com/about/versions/oreo/android-8.0-changes#background> and <https://developer.android.com/about/versions/oreo/background>

⁶<https://stackoverflow.com/questions/52013545/android-9-0-not-allowed-to-start-service-app-is-in-background-after-onresume>

He also commented "*what you found in relation to Android 8 and RoboSpice is the reason why we are still targeting SDK 26.*"

6 DISCUSSION

For different reasons the crash rates are higher for newer versions of Android for the various apps sampled. Android Vitals provides reports and graphs that can help teams identify patterns in problems so they can then triage and address those deemed viable to fix. Once the fixes have been incorporated into the codebase the app can be tested by the team and automatically as part of the pre-launch reports service Google provides in Google Play Console. Once the revised app has been released the developers can monitor the effects of their changes using Android Vitals. They may also see additional effects emerge as measured through a combination of the installs and uninstalls tracked by Google Play Console. As Google states "*fixing issues and preventing bad behaviors can lead to a better user experience, higher ratings, and more retained installers*"[6].

6.1 Crash reporting versus Android Vitals

Android Vitals offers value to developers who do not use crash reporting libraries. It may also have a role to play even for developers who already incorporate and use crash-reporting in their Android apps.

The majority of Android apps include at least one crash reporting library; the most popular are Firebase and Crashlytics (AppBrain statistics are: Firebase used in 54.48% of apps and 74.90% of installs, and Crashlytics used in 14.41% of apps and 25.09% of installs[2]). The Exodus Privacy project⁷ audits Android apps and also finds these libraries are in the majority of the apps they have analysed (Google Firebase Analytics in 67% of apps and Google CrashLytics in 46% of apps).

Android Vitals obtains data about apps at a system level, including recording 'Application Not Responding' (ANR) errors, background network, and battery usage. These data are not available from the popular crash reporting libraries mentioned above.

6.2 Which Apps can Android Vitals Best Help?

Our limited insight into Android Vitals already indicates that reports are only provided when there is sufficient data collected to 'prime the pump'. It may be possible to estimate how many apps of those in Google Play Store are likely to have enough volumes of usage data. Google makes various recommendations for developers on how to apply the results Android Vitals reports⁸ however the developers can't do much until Android Vitals actually shows them the data.

Data provided by AppBrain[3] was used to estimate the populations of apps that are not likely to generate enough data to see various reports in Android Vitals.

Developers see the Active Installs⁹ in Google Play Console rather than total installs (the count presented to Android users in Google Play Store). The Active Install count is around 20% to 30% of the total install count for various apps used in our research e.g. the active installs would be around 20k for an app that shows at having 100,000+

⁷<https://exodus-privacy.eu.org/>

⁸<https://developer.android.com/distribute/best-practices/develop/android-vitals>

⁹ ("Installs on Active Devices (devices online in the past 30 days with this app installed.)

[total] installs to end users in Google Play. The threshold for when there is enough data for Google to provide a report depends on various factors, so the total installs is a proxy measurement and imperfect.

By combining our case studies with AppBrain's download statistics to 19th June 2019[3], shown in figure 5, of the total populations of app developers we estimate:

- 3% to 4% (those with 100,000 to 500,000 total downloads) will get limited value as at least one report will be provided.
- < 1% (those with 500,000 to 1,000,000 total downloads) will get some value as many of the reports will be provided, but not all.
- 1% (those with > 1,000,000 total downloads) will get extensive value as most/all the reports will be provided.

Download distribution of Android apps

Figure 5: ©AppBrain: Download distribution of Android apps (June 2019)

6.3 Maintaining and Extending Vitals-Scraper

The vitals-scraper[18] program needed to be modified on several occasions as Google revises and updates the Google Play Console website. The current functionality retrieves statistics and stack traces for crashes as that was sufficient for our initial research. It also retrieves data for the maximum range Android Vitals provides: 60 days to maximise the data collected for analysis. Adding additional functionality, for instance to perform a small pilot download of the last 24 hours of data, and capabilities, for instance to download ANR errors or summary statistics, could make the scraper more useful.

6.4 Using Android Vitals to Complement Various Approaches

Android Vitals may complement other research, suggestions include: automated crash detection and reproduction; crowdsourced bug reproduction; and assessing the performance and reliability of research into Search Based Testing tools e.g. Sapienz[13] and others.

7 CONCLUSION AND FUTURE WORK

Android Vitals is an example of a software tool that is effectively part of the platform, or ecosystem. Developers see value in the information it provides; yet it only serves a minority of the developer-base, and we can see problems and gaps in the data and reports it does provide. Also, details of errors are only provided in the GUI which would be too time-consuming for developers to read or use online. Our tool enables crashes to be downloaded and the data used both within development teams *and provided developers share the data with researchers*.

This paper presents a snapshot of exploratory research into the potential for using Android Vitals to assist development teams to measure aspects of how their apps perform for end users; and to use those measurements to help assess whether their changes improve or degrade the performance and reliability of the app for those users. The crash reports may also help development teams to identify some causes of run-time failures. The knowledge can help them deal with the issues and improve their communication with current and potential users. Dealing with issues includes bug investigation and mitigation; communication includes updating the app description with known issues and any workarounds as well as being able to provide relevant updates to users who report issues, etc.

Our work is limited to the apps we work on and the development teams we can ask for their data and perspectives. More contributions may help increase the insights and validity of research. How can we and others scale the work to learn more about the behaviours and utility of Android Vitals?

We hope this paper will provide some insights into the potential of Android Vitals for practitioners and stimulate discussion in the research community. Suggested topics include ways Android Vitals and the data it makes available might help developers understand how their Android apps are performing in order to improve the development and testing of these apps. Perhaps it would be viable to correlate crashes reported in Android Vitals with the contents of user reviews, for instance by re-purposing the work of ReCDroid[19] or BECLoMA[17]? Note: unlike the approach used in BECLoMA, developers can download reviews for their apps and would not need to use a crawler to obtain the reviews.

The authors aim to use Android Vitals to help understand and improve Pocket Paint, and potentially other apps, through a longitudinal study over a series of releases of the app.

ACKNOWLEDGMENTS

Jakob Durstberger and the team at Moonpig - and Thomas Schwengler of the Catdroid project team - for sharing their experiences of using Android Vitals. Joseph Reeve for leading the development of the Vitals Scraper and Yijun Yu for reviewing drafts of the paper.

REFERENCES

- [1] Afnan AlSubaih, Federica Sarro, Sue Black, Licia Capra, and Mark Harman. 2019. App Store Effects on Software Engineering Practices. <https://doi.org/10.1109/TSE.2019.2891715>
- [2] AppBrain. 2019. Android app download statistics on Google Play. <https://www.appbrain.com/stats/libraries/tag/crash-reporting/android-crash-reporting-libraries>
- [3] AppBrain. 2019. Android app download statistics on Google Play. <https://www.appbrain.com/stats/android-app-downloads>

- [4] Android Developers. 2019. UI/Application Exerciser Monkey. <https://developer.android.com/studio/test/monkey>
- [5] Wided Ghardallou, Nafi Diallo, Ali Mili, and Marcelo F Frias. 2016. Debugging without testing. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 113–123. <https://doi.org/10.1109/ICST.2016.12>
- [6] Google. 2019. Use Android vitals to improve your app's performance, stability, and size. <https://developer.android.com/distribute/best-practices/develop/android-vitals>
- [7] Google. 2019. View crashes & application not responding (ANR) errors. Retrieved June 11, 2019 from <https://support.google.com/googleplay/android-developer/answer/6083203>
- [8] Giovanni Grano, Adelina Ciurumelea, Sebastiano Panichella, Fabio Palomba, and Harald C Gall. 2018. Exploring the integration of user feedback in automated testing of android applications. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 72–83.
- [9] Mark Harman, Afnan Al-Subaihini, Yue Jia, William Martin, Federica Sarro, and Yuanyuan Zhang. 2016. Mobile App and App Store Analysis, Testing, and Optimization. In *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 243–244.
- [10] Julian Harty. 2019. Google Play Console: Insightful Development using Android Vitals and Pre-Launch Reports. In *2019 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE. <http://oro.open.ac.uk/61066/>
- [11] Wei Jin and Alessandro Orso. 2012. BugRedux: reproducing field failures for in-house debugging. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 474–484.
- [12] Pingfan Kong, Li Li, Jun Gao, Kui Liu, Tegawendé F Bissyandé, and Jacques Klein. 2018. Automated testing of android apps: A systematic literature review. *IEEE Transactions on Reliability* 99 (2018), 1–22.
- [13] Ke Mao, Mark Harman, and Yue Jia. 2016. Sapienz: Multi-objective automated testing for Android applications. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ACM, 94–105.
- [14] 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* 43, 9 (2016), 817–847.
- [15] Matthias Müller, Christian Schindler, and Wolfgang Slany. 2019. Engaging Students in Open Source: Establishing FOSS Development at a University. In *Proceedings of the 52nd Annual Hawaii International Conference on System Sciences*. 7721–7730.
- [16] Stéphane Nicolas. 2018. README for RoboSpice. Retrieved June 12, 2019 from <https://github.com/stephanenicolos/robospice/blob/release/README.md>
- [17] Lucas Pelloni, Giovanni Grano, Adelina Ciurumelea, Sebastiano Panichella, Fabio Palomba, and Harald C Gall. 2018. Becloma: Augmenting stack traces with user review information. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 522–526.
- [18] Joseph Reeve and Julian Harty. 2019. commercetest/vitals-scraper. Retrieved July 01, 2019 from <https://github.com/commercetest/vitals-scraper>
- [19] Yu Zhao, Tingting Yu, Ting Su, Yang Liu, Wei Zheng, Jingzhi Zhang, and William GJ Halfond. 2019. Recdroid: automatically reproducing android application crashes from bug reports. In *Proceedings of the 41st International Conference on Software Engineering*. IEEE Press, 128–139.