BECLoMA: Augmenting Stack Traces with User Review Information

Lucas Pelloni, Giovanni Grano, Adelina Ciurumelea, Sebastiano Panichella, Fabio Palomba, Harald C. Gall
University of Zurich,
Department of Informatics, Switzerland
lucas.pelloni@uzh.ch, {grano,ciurumelea,panichella,palomba,gall}@ifi.uzh.ch

***Abstract*—Mobile devices such as smartphones, tablets and wearables are changing the way we do things, radically modifying our approach to technology. To sustain the high competition characterizing the mobile market, developers need to deliver high quality applications in a short release cycle. To reveal and fix bugs as soon as possible, researchers and practitioners proposed tools to automate the testing process. However, such tools generate a high number of redundant inputs, lacking of contextual information and generating reports difficult to analyze. In this context, the content of user reviews represents an unmatched source for developers seeking for defects in their applications. However, no prior work explored the adoption of information available in user reviews for testing purposes. In this demo we present BECLOMA, a tool to enable the integration of user feedback in the testing process of mobile apps. BECLOMA links information from testing tools and user reviews, presenting to developers an augmented testing report combining stack traces with user reviews information referring to the same crash. We show that BECLOMA facilitates not only the diagnosis and fix of app bugs, but also presents additional benefits: it eases the usage of testing tools and automates the analysis of user reviews from the Google Play Store.**

***Index Terms*—Automated Software Testing, Mobile Applications, User Reviews Analysis**

I. Introduction

Mobile devices such as smartphones, tablets and wearables are currently used in many aspects of our everyday life [14]. Indeed, we witnessed a gradual shift to the so called *post-pc* era. Such a phenomenon caused the explosion of the app industry in the last decade [20]. For this reason, the mobile market is experiencing a high competition where developers, to stay ahead with their competitors and continue to gain and retain users, need to deliver high quality applications in short release cycles. Thus, to maximize app market success developers aim at attaining high quality software by revealing and fixing potential software bugs as early as possible [14]. As a natural consequence, in last years both researchers and pratictioners developed techniques and tools to automate the testing of mobile applications [8], [12], [13], [21]. Such tools aim to reveal unhandled exceptions while exercising the app under test (AUT) with input and system events. Usually, whether a crash occurs, they save the corresponding stack trace together with the sequence of events that led to the crash.

Unfortunately, such tools suffer of some limitations: (i) they are only able to detect bugs that cause unhandled exceptions, potentially missing those not raising any and (ii) they fail to generate input sequence that requires human intelligence or domain knowledge [12]; thus, they might fail to reveal particular buggy sequences that were encountered by the users. [5]. Moreover, the sequence of events recorded by these tools is often redundant, lacks of contextual information and is difficult to analyze [4], [11]. Therefore, developers might struggle to understand the root cause of the crashes highlighted by automated tools [13].

In the context of mobile development, recent work proved the usefulness of user review information for the planning of maintenance tasks, providing tools for achieving this goal [4], [15], [17], [19], [22]. Specifically, they show that feedback posted on mobile app stores represents an unmatched source for developers seeking for defects in their applications [6], [17], [19]. Despite such a huge source of available knowledge, no prior work explored the adoption of user review information in the context of automated testing of mobile application.

In this paper, we present BECLOMA (**B**ug **E**xtractor, **C**lassifier and **L**inker **o**f **M**obile **A**pps), a tool that enable the integration of user feedback in the testing process of mobile apps. BECLOMA automatically establishes links between stack traces and user reviews related to crashes or bugs. Then, it presents developers unique reports aiming at (i) augmenting the stack traces' information with the natural description of crashes provided by users and (ii) easing the fixing of a bug. Moreover, BECLOMA eases the setting and the execution of two of the most well-known Android testing tools, namely MONKEY and SAPIENZ. Such tools are used to reveal unhandled exceptions and therefore collect the correspondent stack traces. Further, BECLOMA incorporates a crawler able to mine user reviews from the Google Play Store. Then, using a Machine Learning (ML) classifier, it discerns only the user reviews discussing about crashes and bugs (worth to be linked to the stack traces).

Benefits for Developers. We envision BECLOMA to be helpful for developers testing their mobile apps. Indeed, understanding of the steps that led to a failure might be often troublesome with stack traces only. In this cases, user reviews can be a human readable companion for such traces. For instance, a review like *"...the app crashes when I press the messages button.."* linked to a specific stack trace immediately indicate developers where to investigate for the occurred fault.

Tool and Data Replication. The tool and the evaluation dataset are available on the BECLOMA website [1].

II. BECLoMA, Bug Extractor, Linked and Classifier of Mobile Apps

This section briefly describes the approach and technologies we employed. BECLoMA is a tool built with a set of different technologies: the core part is written in Java while the Machine Learning classifier for the user reviews (as well as the embedded SAPIENZ tool) is written in Python. The remaining of this section reports the main characteristics of BECLoMA, as well as details about its architecture and inner-working.

A. *Design Goals*

BECLoMA is designed with the main goal to link together the stack traces extracted from testing tools with the user reviews that describe with a high level language the occurred crash. In addition, BECLoMA eases the execution of testing tools (it supports at the moment MONKEY and SAPIENZ), providing to the users a friendly interface for the parameters tuning. Moreover, BECLoMA embeds a reviews miner and a classifier able to discern only the crash-related reviews. Since often developers might already have stack traces and reviews from different sources, the tool offers the support to the import of the two from external sources.

B. BECLoMA *Architecture*

The overall architecture of BECLoMA is depicted in Figure 1. Users interact with the tool through a *Presentation layer* that exposes the main features. In particular, the UI is built upon the Swing Java framework, being extremely portable on every kind of machine. The *Application Layer* is the core of BECLoMA. It contains the subsystems that implements the various features of the tool. Logically, the architecture can be divided into three main parts: (i) the *Testing Subsystem* that handles the execution of the testing experiments for the AUT on the physical or emulated devices; (ii) the *The Crawler Subsystem* that allows to mine user reviews from the Google Play Store and to discern the ones addressing bug or crashes (the ones to be linked); (iii) the *Linking* package, that implements the core feature of BECLoMA, *i.e.*, the link between the stack traces and the user reviews concerning the correspondent problem. The upcoming sections will detail such three components.

C. *The Testing Subsystem*

The aim of this subsystem is to facilitate the execution of MONKEY and SAPIENZ to the user. The tool's UI (Figure 2) allows an user-friendly specification of all the customizable parameters. The `ExperimentLauncher` class sets up the entire execution of the testing experiment. It manages an instance of `AppTester`; such class relies on the implementation of `Emulator` and `Tablet` in the `devices` package to perform all the preliminary operations on the target device (*e.g.*, install the AUTs). Once the target is ready, `AppTester` launches the execution of the desired testing tool (either MONKEY or SAPIENZ). It is worth to notice that BECLoMA supports also the multi-device testing execution. At the end of the testing process, the `CrashExtractor` class extracts and

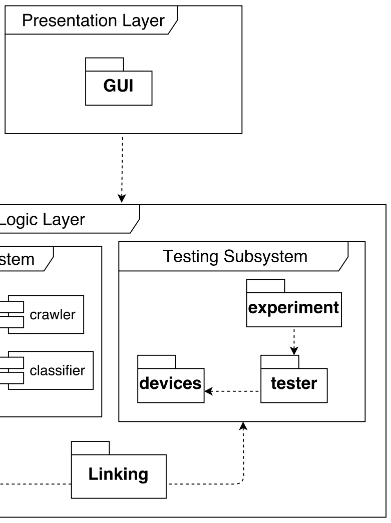
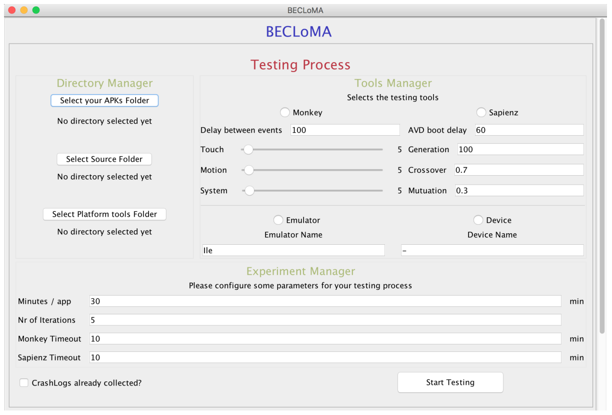


Fig. 1. BECLoMA's architecture



Directory Manager
Select your APKs Folder
No directory selected yet
Select Source Folder
No directory selected yet
Select Platform tools Folder
No directory selected yet

Tools Manager
Selects the testing tools
Monkey
Sapienz
Delay between events: 100
AVD boot delay: 60
Touch: 5
Generation: 100
Motion: 5
Crossover: 0.7
System: 5
Mutuation: 0.3
Emulator
Device
Emulator Name: lle
Device Name: –

Experiment Manager
Please configure some parameters for your testing process
Minutes / app: 30 min
Nr of Iterations: 5
Monkey Timeout: 10 min
Sapienz Timeout: 10 min
CrashLogs already collected?
Start Testing

Fig. 2. BECLoMA's testing manager

combines all the accumulated crashes (composed by a stack trace and a log with a sequence of events). It also discard duplicates amongst such crashes. We define two crashes as duplicated if the two stack traces are exactly the same. Moreover, BECLoMA gives some statistics about the testing phase, like the time spent in the process, the number of total crashes collected and the number of not-duplicated ones.

D. *The Crawler Subsystem*

This subsystem basically fulfills two goals: (i) crawling the user reviews for the AUT from the Google Play Store, and (ii) discerning amongst them the ones concerning about crashes and bugs. To do this, BECLoMA relies on two external modules. The first one, (the `crawler` module in Figure 1) is a Java-built scraper tool that relies on PhantomJS[1] and Selenium[2] to navigate the Play Store and extract the reviews from the HTML [10]. Such crawler is executed in a separate

[1]http://phantomjs.org
[2]http://www.seleniumhq.org

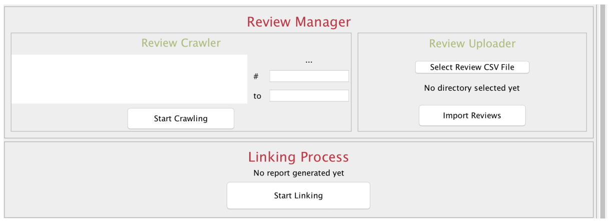


Fig. 3. BECLoMA's reviews and linking manager

process; this task is handled by the `ReviewCrawler` class in the `crawling` package.

Once the reviews for the AUTs are mined, BECLoMA uses a second external component (the `classifier` module in Figure 1) to discern the useful ones for the linking, addressing features-related and crash-related bugs (according to the taxonomy presented in [9]). Such component is built with Python and implemented with the `scikit-learn` [18] library. It uses a Gradient Boosted Classifier algorithm trained with a dataset of 6,600 user reviews manually labelled. As features, it relies on the *tf-idf* scores of the 1-grams, 2-grams and 3 gram-s computed on the terms for each reviews. It is worth to underline that the classifier has showed a precision, recall and F1 score of the 87%, the 88% and the 87% respectively. Similarly to the `crawler` component, also the `classifier` one is executed on a separate process; this execution is handled by the `PythonClassifierManager` class in the `clawling` package.

E. Linking

The linking between the stack traces and the user reviews represents the core element of BECLoMA. To this aim, BECLoMA preprocesses both the two sources; for the stack traces it takes into account only the name and the cause of the exception (*e.g.*, it removes the list of native methods —part of the Android SDK— in the exception). Therefore, the stack trace is *augmented* with the source code methods included in the stack trace itself. This step is needed to add contextual details from source code in order to produce better results for the linking.

Afterwards, both the user reviews and the stack traces *augmented* as explained above are preprocessed; this task involved the typical steps of correcting mistakes, expanding contractions, filtering nouns and verbs, removing common words, programming language keywords and stemming. Finally, the two resulting bag of words are compared using the Dice similarity coefficient [7], defined as follow:

$$\text{Dice}\left(review_j, crash_i\right) = \frac{\left|W_{review_j} \cap W_{crash_i}\right|}{\min\left(\left|W_{review_j}\right|, \left|W_{crash_i}\right|\right)}$$

where W_{review_j} is the set of words composing a user review j and W_{crash_i} is the one contained in an *augmented* stack trace i. The tool links stack traces and reviews having a Dice score higher than 0.5 (in a range between $[0, 1]$).

The overall approach in implemented in the `linking` package (see Figure 1). In detail, the `CrashLog` class takes care of selecting the relevant part of the stack trace, *i.e.*, the name, the cause of the exceptions and the signature of the methods that will be used to augment the stack itself. Therefore, the `Augmenter` class (i) parses the source code (that need to be provided by the user), (ii) extracts the content of the recorded methods, (iii) preprocesses the content of such methods, applying all the preprocessing steps and (iv) completes the whole bag of words representing the stack trace. The same preprocessing step is done for the user reviews by the `Reviews` class. At the end, BECLoMA implements in the `AsymmetricDiceIndex` class of the `linking.metrics` the calculation of the index.

F. BECLoMA *at Work*

To generate links, BECLoMA needs to have both stack traces and user reviews for the AUT. Both the artifacts can be mined by BECLoMA (with the testing and crawler components described above) or imported. Figure 2 shows the UI that manages such process. The user has to specify some mandatory directories, like the one where the apks are stored, the one that contains the source code for the AUT (needed for the linking phase) and the Android directory in the operative system. Then, it is possible to tune parameters regarding the events to feed to the AUT as well parameters about the SAPIENZ genetic algorithm. It is worth to notice that BECLoMA works with both emulator and physical devices. However, due to SAPIENZ constraints, the tool needs an *Android SDK: API 19* version. The tool allows also the import of crash logs previously collected or gathered from other sources.

BECLoMA offers the same import option as well for the user reviews. In such a case, the user selects a csv file and clicks on the *Import Reviews* button (see Figure 3). In the alternative scenario, the user mines the reviews from the Google Play Store using the BECLoMA's built-in crawler. He simply has to list in the text box of the *Review Manager* section the ids of the apps he wants to mine the reviews for; then, he needs to click on the *Start Crawling* button to start the process.

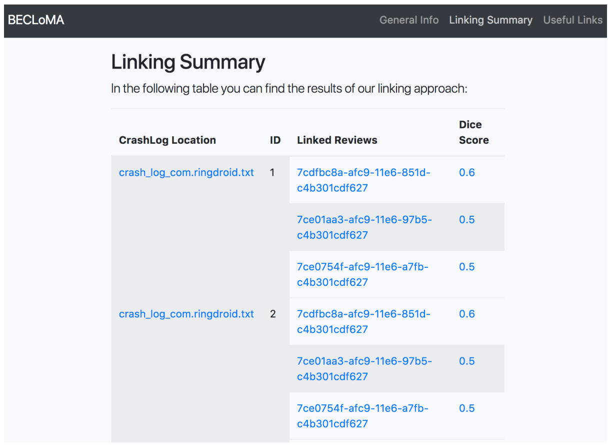
When both the stack traces and the user reviews are available, the *linking* feature can be used (the *Linking Process* box in Figure 3). BECLoMA generates an HTML report (depicted in Figure 4) where, for each AUT, the stack traces are linked to the correspondent user reviews.

III. EVALUATION

With the aim of evaluating the BECLoMA capabilities in linking user review information and stack traces, we setup an empirical study targeting the following research question:

- **RQ**$_1$: *What is the accuracy of* BECLoMA *in linking crash-related user review and crash reports?*

Our study involved 8 Android applications whose characteristics are reported in Table I. It is important to note that we needed to limit the number of apps considered because of the amount of manual analysis requested to build a golden set of



Linking Summary

In the following table you can find the results of our linking approach:

CrashLog Location	ID	Linked Reviews	Dice Score
crash_log_com.ringdroid.txt	1	7cdfbc8a-afc9-11e6-851d-c4b301cdf627	0.6
		7ce01aa3-afc9-11e6-97b5-c4b301cdf627	0.5
		7ce0754f-afc9-11e6-a7fb-c4b301cdf627	0.5
crash_log_com.ringdroid.txt	2	7cdfbc8a-afc9-11e6-851d-c4b301cdf627	0.6
		7ce01aa3-afc9-11e6-97b5-c4b301cdf627	0.5
		7ce0754f-afc9-11e6-a7fb-c4b301cdf627	0.5

Fig. 4. BECLoMA's HTML report

TABLE I
Subset of Apps Selected for the Study

Application	**Category**	**Cras.**	**LOC**	**Reviews**	
				Total	Crash
com.amaze.filemanager	Tools	7	39K	1,438	28
com.danvelazco.fbwrapper	Social	4	2.5K	1,900	252
com.eleybourn.bookcatalogue	Productivity	1	16K	677	11
com.evancharlton.mileage	Finance	2	9.8K	1,064	39
com.fsck.k9	Communication	1	52K	2,895	106
com.ringdroid	Video P.& Ed.	4	3.9K	2,363	84
cri.sanity	Communication	1	8K	695	11
org.liberty.android.fantastischmemo	Education	4	30K	264	3
Total		**24**	**162.2K**	**11,296**	**534**

links to compare with the output of BECLoMA (more details later in this section).

In the following subsections, we (i) report the data extraction and analysis process followed to answer our **RQ**$_1$ and (ii) discuss the results of our experiment.

A. Data Extraction and Analysis

To conduct the empirical evaluation of the proposed tool, we needed to mine data about crash-related user reviews and crashes. To this aim, we ran BECLoMA on each of the considered app. The crawling mechanism extracted a total of $11,296$ user reviews; of them, 534 were classified as issues likely related to crashes and were, therefore, used as basis for the linking process. On the other hand, the testing component of BECLoMA detected 24 unique crashes occurring in the considered apps.

To properly assess the extent to which our tool correctly retrieve links between the two sources of information, a golden set of links to compare with the output of tool was required. To avoid any bias, we asked to an external inspector having 2 years of experience in Android development to build such a golden set. Specifically, we provide her with (i) the stack traces arose from the execution of the testing tools, (ii) the logs of the executed events that led to the crashes, (iii) the `apk` and the source code of the apps in the dataset, and (iv) the set of crash-related user reviews.

TABLE II
Performance of the Experimented Linking Approaches (Recall and F1 Score are computed only for three of the subject apps)

App	**Dice Linking**		
	Precision	Recall	F1 Score
com.amaze.filemanager	67%	57%	62%
com.danvelazco.fbwrapper	62%	68%	65%
com.ringdroid	64%	60%	62%
com.eleybourn.bookcatalogue	100%	66%	80%
com.evancharlton.mileage	100%	100%	100%
org.liberty.android.fantastischmemo	100%	100%	100%
cri.sanity	-	-	-
com.fsck.k9	-	-	-
Average	**82%**	**75%**	**78%**

The golden set creation process was basically composed of three phases. In the first, the inspector had to re-run the stored sequences of events for the collected crashes in order to reproduce them with the aim of understanding their dynamics. She relied on a Monkey feature to perform this task. Once she had understood the issue leading to the crash, she inspected the source code of a certain app trying to figure out its behavior in proximity of the piece of code that threw the exception. Finally, she analyzed the set of provided reviews, linking them, whenever possible, to one of the stack traces. Once the golden set was built, we evaluated BECLoMA's performances using two widely-adopted metrics, *i.e.*, recall and precision [2].

$$recall = \frac{|Cor \cap Det|}{|Cor|}\% \qquad precision = \frac{|Cor \cap Det|}{|Det|}\%$$

where Cor and Det represent the set of actual links (those belonging to the golden set) and the set of links detected by BECLoMA, respectively. As an aggregate indicator of precision and recall, we report the F1 Score, defined as the harmonic mean of precision and recall:

$$F1\text{-}score = 2 * \frac{precision * recall}{precision + recall}\%$$

B. Experimental Results

Table II reports the performance achieved by BECLoMA on the set of mobile apps considered. It is worth noting that in two cases, *i.e.*, Sanity and K9, the golden set did not report any link between user reviews and crashes; thus, we could not compute precision and recall. Looking at the results, we observed that our tool achieved an average precision of 82% and a recall of 75% (F1 Score = 78%). This result confirms previous findings in the field [15], which highlighted how the Dice index represents one of the most suitable tools to cope with texts composed of few words such as user reviews and crashes. However, it is also interesting to note that the recall value reports how almost 18% of the correct links are not properly identified by BECLoMA. As part of our future research agenda, we plan to establish the usefulness of different approaches (*e.g.*, textual-based techniques such ad

LDA [3], [16]) as additional/alternative to the Dice linking currently exploited.

To show the potential of our tool, in the following we discuss an example of link found by BECLOMA on the `com.amaze.filemanager` app. Specifically, in the user review reported below the user complained about a failure happening immediately after releasing the keyboard - used for searching files in the app.

"Crashing when I release the keyboard!!! Pls fix!".

Our tool linked this review with the crash reports presented below, which report a `NullPointerException` arising in the `DialogUtils` class.

```
Long Msg: java.lang.NullPointerException
at com.afollestad.materialdialogs.util.DialogUtils.hideKeyboard(
      DialogUtils.java:226)
at com.afollestad.materialdialogs.MaterialDialog.dismiss(
      MaterialDialog.java:1810)
...
at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(
      ZygoteInit.java:795)
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:611)
at dalvik.system.NativeStart.main(Native Method)
... 3 more
```

This example clearly show how user review can be successfully exploited to augment crash reports with contextual information useful for developers to comprehend the cause of a failure in a quicker and more efficient manner.

To conclude, the empirical experiment revealed preliminary evidence that BECLOMA could be useful in assisting developers during the debugging and evolution of mobile apps.

IV. DEMO REMARKS

In this demo, we present BECLOMA, an automated tool able to augment stack traces with contextual information extracted from user review. We described the architecture and inner-working of the tool; at the same time, we reported its empirical evaluation conducted on 8 Android apps, that showed how BECLOMA is highly accurate when linking crash reports and crash-related user reviews. We plan to integrate our tool in ANDROID STUDIO, *i.e.*, the most common Android's Integrated Development Environment (IDE). Moreover, we want to extend its functionalities to cluster different stack traces that might depend on the same bug or prioritize the arose failures taking into account the user feedback.

ACKNOWLEDGEMENTS

Pelloni, Grano, Ciurumelea, Panichella, and Gall acknowledge the Swiss National Science Foundation (SNSF) under project named "SURF-MobileAppsData" (Project no. 200021_166275) and the Swiss Group for Software Engineering (CHOOSE). Palomba acknowledges the support of the SNSF Project named "Data-driven Contemporary Code Review" (No. PP00P2_170529).

REFERENCES

[1] becloma-website. http://becloma.info/.

[2] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[4] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang. Ar-miner: Mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 767–778, New York, NY, USA, 2014. ACM.

[5] W. Choi, G. Necula, and K. Sen. Guided gui testing of android apps with minimal restart and approximate learning. *SIGPLAN Not.*, 48(10):623–640, Oct. 2013.

[6] A. Ciurumelea, A. Schaufelbuhl, S. Panichella, and H. C. Gall. Analyzing reviews and code of mobile apps for better release planning. In *SANER*, pages 91–102. IEEE Computer Society, 2017.

[7] L. R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[8] Google. Android monkey. https://goo.gl/P7jFeF.

[9] G. Grano, A. Ciurumela, S. Panichella, F. Palomba, and H. C. Gall. Exploring the integration of user feedback in automated testing of android applications. In *2018 IEEE 25rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*.

[10] G. Grano, A. Di Sorbo, F. Mercaldo, C. A. Visaggio, G. Canfora, and S. Panichella. Android apps and user feedback: A dataset for software evolution and quality improvement. In *Proceedings of the 2Nd ACM SIGSOFT International Workshop on App Market Analytics*, WAMA 2017, pages 8–11, New York, NY, USA, 2017. ACM.

[11] M. E. Joorabchi, A. Mesbah, and P. Kruchten. Real challenges in mobile app development. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 15–24, Oct 2013.

[12] A. Machiry, R. Tahiliani, and M. Naik. Dynodroid: An input generation system for android apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 224–234. ACM, 2013.

[13] K. Mao, M. Harman, and Y. Jia. Sapienz: Multi-objective automated testing for android applications. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ISSTA 2016, pages 94–105, New York, NY, USA, 2016. ACM.

[14] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman. A survey of app store analysis for software engineering. *IEEE Transactions on Software Engineering*, PP(99):1–1, 2016.

[15] F. Palomba, P. Salza, A. Ciurumelea, S. Panichella, H. Gall, F. Ferrucci, and A. De Lucia. Recommending and localizing change requests for mobile apps based on user reviews. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, 2017.

[16] A. Panichella, B. Dit, R. Oliveto, M. D. Penta, D. Poshyvanyk, and A. D. Lucia. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pages 522–531, 2013.

[17] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall. How can i improve my app? classifying user reviews for software maintenance and evolution. In *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, ICSME '15, pages 281–290, Washington, DC, USA, 2015.

[18] Scikitlearn. Gradient boosting classifier. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html.

[19] A. D. Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall. What would users change in my app? summarizing app reviews for recommending software changes. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016*.

[20] Statista. Number of available apps in the play store. https://goo.gl/QSG43J, Mar. 2017.

[21] T. Su, G. Meng, Y. Chen, K. Wu, W. Yang, Y. Yao, G. Pu, Y. Liu, and Z. Su. Guided, stochastic model-based gui testing of android apps. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 245–256, 2017.

[22] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta. Release planning of mobile apps based on user reviews. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 14–24, New York, NY, USA, 2016. ACM.