ANDROR2: A Dataset of Manually-Reproduced Bug Reports for Android apps

Tyler Wendland*, Jingyang Sun†, Junayed Mahmud,‡ S. M. Hasan Mansur‡, Steven Huang†,
Kevin Moran‡, Julia Rubin†, Mattia Fazzini*
*University of Minnesota, MN, USA; wendl155@umn.edu, mfazzini@umn.edu
†University of Bristish Columbia, BC, Canada; sunjingy@ece.ubc.ca, huang145@student.ubc.ca, mjulia@ece.ubc.ca
‡George Mason University, VA, USA; jmahmud@gmu.edu, smansur4@gmu.edu, kpmoran@gmu.edu

***Abstract*—Software maintenance constitutes a large portion of the software development lifecycle. To carry out maintenance tasks, developers often need to understand and reproduce bug reports. As such, there has been increasing research activity coalescing around the notion of automating various activities related to bug reporting. A sizable portion of this research interest has focused on the domain of mobile apps. However, as research around mobile app bug reporting progresses, there is a clear need for a manually vetted and *reproducible* set of real-world bug reports that can serve as a benchmark for future work. This paper presents ANDROR2: a dataset of 90 manually reproduced bug reports for Android apps listed on Google Play and hosted on GitHub, systematically collected via an in-depth analysis of 459 reports extracted from the GitHub issue tracker. For each reproduced report, ANDROR2 includes the original bug report, an apk file for the buggy version of the app, an executable reproduction script, and metadata regarding the quality of the reproduction steps associated with the original report. We believe that the ANDROR2 dataset can be used to facilitate research in automatically analyzing, understanding, reproducing, localizing, and fixing bugs for mobile applications as well as other software maintenance activities more broadly.**

I. INTRODUCTION

Software maintenance activities are known to be generally time consuming, so much so that prior studies have illustrated they can often comprise more than half of the development effort for a given software project [1]. While developers carry out a vast array of maintenance activities, perhaps no artifact is more central to a wider variety of maintenance tasks than the bug reports filed in issue tracking systems.

A sizable portion of the research around automating activities related to bug report management has focused upon the domain of mobile applications [2], [3], [4], [5], as mobile devices and apps continue to grow in their ubiquity and popularity (e.g., 3 million apps on Google Play [6]), and developers require new techniques and tools to cope with challenges related to rapidly evolving and fragmented devices [7], [8] and a growing user base [9]. However, most existing studies on automating bug management activities for mobile applications are limited by the lack of a reliable and systematically created dataset, and tend to be evaluated against a manually curated sets of bug reports [2], [3], [4]. This complicates measuring the progress of automation in this research area due to the difficulty to compare techniques developed for similar maintenance tasks.

To help address these current limitations of research on automated management of mobile app bugs, this paper introduces the ANDROR2 dataset [10]. ANDROR2 consists of 90 manually-reproduced bug reports, collected via an in-depth analysis of 459 issues systematically mined from the bug tracking system of Android apps hosted on GitHub and available on the Google Play Store [6]. ANDROR2 includes 23 reports describing a failure that manifests as a crash and 67 reports detailing a non-crashing failure. Each bug report in the ANDROR2 dataset was manually verified to be fully reproducible by at least two authors of this paper; the dataset includes both executable `.apk` files and Android device configurations that allow for the reproduction of each reported bug. Furthermore, given that prior work illustrated the criticality of reproduction steps (S2Rs) to the quality of bug reports [11], [2], for each included bug, we offer an extensive analysis of the S2Rs and include structured data representation of information related to the number of steps, issues with the reported steps (e.g., missing information), as well as the setup or environmental constraints required for the bug to manifest. Finally, we include automated scripts that fully reproduce the failures described in the reports.

We believe that this dataset will support future work and studies related to automating various bug reporting activities. In particular, by including both the (often) flawed user reported S2Rs, as well as "ground truth" sets of reproduction steps, we believe that the dataset can directly support future work on bug report quality assessment and reproduction. Additionally, we describe how ANDROR2 and its intermediate artifacts can be used and extended to support research in a broad range of software testing and maintenance activities.

II. ORIGINALITY OF THE DATASET

There are a number of datasets from prior work on automated bug report management. Most notable are the datasets from the papers introducing FUSION [5] (15 reports), EULER [2] (24 reports), YAKUSU [3] (48 reports and 12 trivial reports), and ReCDroid [4] (51 reports).

The ANDROR2 dataset is largely *complimentary* to these datasets (it includes only one bug report contained in these datasets) and exhibits several key differences that set it apart. First, the ANDROR2 dataset includes different types of bugs representing a more diverse population of faults (i.e.,

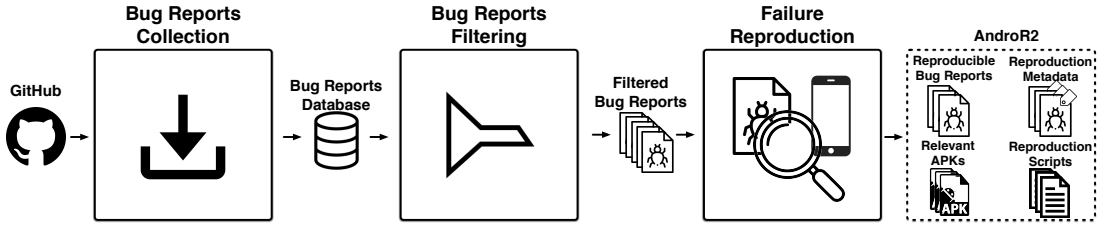

Figure 1: High-level overview on the methodology used to build ANDROR2.

faults leading to crashing and non-crashing failures), whereas the largest datasets from prior work focused *exclusively on crashes* [3], [4]. Second, ANDROR2 was built using a systematic methodology for mining, filtering, and reproducing the reports that focuses on reports written by non-contributors (e.g. likely originating from end-users). This diversity in reporters is important for future work on automatically analyzing bug reports, as prior work has shown that reporters of different levels of expertise construct bug reports differently [12]. Third, ANDROR2 contains important metadata related to the S2Rs of the collected bugs, which can support new and continuing lines of research as described in Section V. Finally, our dataset is both the largest to date and contains automated scripts for reproducing the reported bugs.

III. DATASET CREATION

In this section, we present the methodology used to build ANDROR2 and also discuss the key challenges faced during its construction. Figure 1 provides a high-level overview of our methodology workflow, which consisted of three main phases: the *bug reports collection* phase, the *bug reports filtering* phase, and the *failure reproduction* phase. The rest of this section describes the three phases in detail.

A. Bug Reports Collection

In the *bug reports collection* phase, we built a dataset of bug reports based on GitHub issues [13]. We selected GitHub as the source for building ANDROR2, both due to its popularity and integration of source code hosting and issue tracking, which allowed us to both mine relevant bug reports and build executable `.apk` files from source code. To identify relevant issues, we built a tool based on the GitHub REST API [14] that mines issues containing reported bugs.

First, we mined all GitHub issues posted between January 1st, 2016 and November 1st, 2020 that met the following criteria: (i) are part of repositories that use Java and (ii) have the label "bug". We performed this task by creating one query for each day during the 5 year date range, and the query was retrieving the issues created on the day (each individual query never returned more than 1,000 results, which is the search limit associated with the GitHub REST API [15]). We considered issues created in the last five years to avoid frequently encountered problems in building or finding the executables (i.e., `.apks` [16]) of app versions corresponding to older reports. We selected issues labelled as "bug" to effectively identify those likely related to bug reports and ignore issues discussing ideas, enhancements, or tasks.

Second, for each issue identified in the first step, we used our mining tool to determine whether the issue was part of a repository containing an Android app. This was accomplished by cloning the repository associated with the issue and checking whether it contained an `AndroidManifest.xml` file, as each Android app requires this file to properly compile [17]. Once we confirmed that an issue was part of a repository likely containing an Android app, we included the issue as a document in a MongoDB [18] database. At the end of this first phase, this *bug reports database* contained 82,455 issues.

B. Bug Reports Filtering

In this phase, we systemically collected a set of issues from the *bug reports database* with the objective of building a representative set of bug reports which could then be further analyzed manually. To this end, we first identified and selected issues that belong to repositories whose app is on the Google Play Store to help eliminate issues with trivial apps. This resulted in a set of 28,501 issues. Second, because ANDROR2 aims to foster research on bug reports and their S2Rs, we collected issues that contain the word 'steps' in any portion of the report. We used this measure to avoid manually processing a large number of issues without S2Rs during the failure reproduction phase. This step resulted in 6,365 issues. Third, we further refined the set of issues to only contain those created by a GitHub user that had *not* contributed to the repository, resulting in 3,842 issues. Fourth, we selected issues that were closed at the time the issues were mined (November 2020). We focused on closed issues so that we could more easily identify whether the issues were also originally reproduced by the developers and to provide a higher likelihood of manual bug reproduction in the next phase. This filtering resulted in 3,005 reports. Fifth, after analyzing the set of issues, we found that some repositories had a much larger number of issues compared to others. To avoid overfitting ANDROR2 to a specific app, we considered at most ten issues per repository. When a repository had more than ten issues, we randomly selected ten from this set (we did this operation for 24 repositories). The resulting set of issues consisted of 459 bug reports for 121 apps. Finally, to further facilitate the process of reproducing the issues, two authors of this paper read each of the 459 issues and filtered out those that were either not reproduced by the developers (170) by looking for this information in the discussion associated with the report or were trivial, i.e., occur by simply opening the app (15). This resulted in 274 issues for 88 apps, and we call this set the *filtered bug reports*.

C. Failure Reproduction Phase

In the last phase of our dataset creation process, we manually processed the filtered 274 bug reports to create ANDROR2. Specifically, we first analyzed each of the filtered bug reports to manually reproduced the failures. This process resulted in 90 successfully *reproducible bug reports*. Then, for each reproducible report, we derived *reproduction metadata* (detailed in Section IV) on the quality of the S2Rs in bug reports, and created executable *reproduction scripts*. The reproducible bug reports, the reproduction metadata, the reproduction scripts, and the *relevant APKs* (i.e., the relevant app executables) make up the content of ANDROR2 dataset.

Five of the authors worked to reproduce the failures described in the filtered bug reports. When reproducing a bug report, the authors first identified the version of the app associated with that report. If the bug report did not provide the version information, the authors used the latest version of the app that was released before the date the report was submitted. Second, the authors checked whether the report identified the version of the Android OS on which the user experienced the failure. If the report contained this information, the authors used that version to reproduce the failure. Otherwise, the authors extracted the `targetSdkVersion` value [19] from the app and used that value as the version of the Android OS on which to reproduce the failure. Finally, the authors attempted to reproduce the failure by interacting with a Pixel 2 emulator running the app and Android versions they identified.

To reproduce a failure, the authors followed the S2Rs contained in the bug report by mapping the steps to GUI actions in the app. If a report had missing S2Rs, the authors manually explored the functionality of the app to identify the minimal sequence of GUI actions that would account for those missing steps. (The authors used a trial-and-error approach in this situation.) For successfully reproduced failures, the authors repeated the reproduction steps at least one additional time as a sanity check and two authors tried to reproduced the same bug report to ensure that reproduced failure was the same as the one described in the report. The authors then encoded the GUI actions in a reproduction script using the UIAutomator framework.

To ensure the reliability of the manual reproduction phase, the work was divided into two stages. In the first stage, we divided the filtered bug reports among the five authors, and they attempted to reproduce the bug reports. In the second stage, we selected the bug reports that were not reproduced successfully and reassigned them so that another author made a second reproduction attempt. At the end of the second stage, we had 90 reproducible bug reports. The reasons we could not reproduce certain bug reports were as follows: we could not reproduce the failure even if we followed the S2Rs in the report (76 cases); could not build or find a suitable APK for reproducing the bug (56 cases); the bug report required the use of an additional device (24 cases); a personal account is needed for reproducing the report (16 cases); the bug report required the use of additional files we could not access (7 cases); the bug report required the use of a real device (5 cases). It is worth noting that the effort required to undertake this process was quite high: around six man-months. This phase was the one that introduced the main challenges in building ANDROR2 as it required carefully mapping S2Rs to GUI actions while accounting for missing or ambiguous S2Rs.

IV. DATASET CHARACTERISTICS

ANDROR2 is available as on Zenodo [10]. The dataset contains the reproducible bug reports, the reproduction metadata, the relevant `.apks`, and the reproduction scripts. The reproducible bug reports are located in the `reports` folder and are in their original HTML format. The reproduction metadata is in the `metadata` folder and encoded in JSON files. The relevant APKs are in the `apks` folder. The reproduction scripts are in the `scripts` folder and are encoded as UIAutomator tests. We used UIAutomator as it allows for interacting with both the Android OS and other apps, which is necessary for some reports. The dataset also contains a `README.md` that describes its content. Finally, the scripts and tools we developed to build ANDROR2 are in the `code` folder.

Dataset Details. To detail the data stored in ANDROR2, we use a bug report (*BR#*160) contained in the dataset as a running example. Figure 2 shows the relevant portion of the bug report, Figure 3 presents the metadata associated with the bug report, and Figure 4 illustrates part of the reports' reproduction script.

The bug report contains three S2Rs (under the header "*Reproduction Steps*"). The reproduction metadata provides information on the failure reproduction task and how it connects to the bug report's information. The metadata also contains a detail natural language description of all the S2Rs necessary to reproduce the failure of the report. (We provide a full description of the metadata in [10].) In short, `id` is the identifier for the bug report in ANDROR2, `github_issue` provides the link to the GitHub issue, and `failure_type` describes the type of failure associated with the report. We classified failures under three categories: *crash* identifies a crash in the app, `output` represents an error in the output of the app, and `gui` identifies an error in the properties of the app's GUI. For *BR#*160, the failure is of type `gui` because an app icon is not displayed correctly.

Further, `s2rs` provides the number of S2Rs in the report. We counted S2Rs as follows. If the report provided a bulleted (numbered) list for the S2Rs, we counted how many bulleted (numbered) items are listed. If the S2Rs are described through text paragraphs, we counted the number of sentences in the paragraphs. `gui_actions` is the number of GUI actions exercised by authors to reproduce the failure. `setup_gui_actions` is the number of GUI actions that were necessary before the author could perform the GUI action(s) associated with the first S2R. For *BR#*160, there are eight such actions. Two actions are necessary after a fresh install of the app, and six actions are required to create a flashcard required by the S2Rs of the report. `rs_gui_actions` provides the number of report-specific GUI actions, that is, the number of actions associated with the sequence of S2Rs in the report. ANDROR2 also includes scripts for reproducing the failures of the reports.

Title: Problem with the eraser tab

Reproduction Steps
1. Click on whiteboard tab while reviewing flashcards and use the feature to write/draw anything
2. Click on the erase tab till you have undone the drawing and the erase tab grays out
3. Redraw anything by clicking on the whiteboard icon

Expected Result
Upon redrawing, the eraser tab should revert its color from gray to white

Actual Result
On redrawing, the eraser tab still remains grayed out.

Debug info
...AnkiDroid Version = 2.10beta3
Android Version = 10...

Figure 2: Bug report #160.


```
{"id": 160,
 "github_issue": "https://...",
 "commit_id": "abd1db2...",
 "android_os":"11",
 "failure_type":"gui",
 "s2rs":3,
 "gui_actions":14,
 "setup_gui_actions":8,
 "rs_gui_actions":6,
 "rs_missing_gui_actions":0,
 "multiple_gui_action_s2rs:":1,
 "gui_actions_in_mgas2rs":4,
 "sgas_in_android_os":1,
 "sgas_outside_app":0,
 "rsgas_in_android_os":0,
 "rsgas_outside_app":0,
 "sga_nl_description": "...",
 "rsga_nl_description": "..."}
```


Figure 3: Metadata.

```
1 public class Script160 {
2  ...
3  @Test
4  public void reproduce() {
5   ...
6   UiObject2 Default = mDevice.wait(
7    Until.findObject(By.text( "Default")),2000);
8   Default.click();
9
10  UiObject2 More = mDevice.wait(
11   Until.findObject(By.desc( "More options")),2000);
12  More.click();
13
14  UiObject2 Whiteboard = mDevice.wait(
15   Until.findObject(By.text( "Enable whiteboard")),2000);
16  Whiteboard.click();
17
18  mDevice.drag(300,300, 600, 600, 1);
19  ... }}
```

Figure 4: Reproduction script.

Figure 4 shows part of the script for reproducing *BR#*160 and includes four GUI actions (lines 6, 10, 14, and 18).

Dataset Summary Statistics. AndroR2 contains 90 reproducible bug reports. Among the bug reports, 23 describe crashes, 34 detail output errors, 33 report GUI errors, 77 require setup GUI actions, 33 have report-specific GUI actions not mentioned by the S2Rs, and 57 contain at least one S2R that leads to multiple GUI actions. AndroR2 also contains 90 executable scripts reproducing the failure of the bug reports.

V. Applications, Limitations, and Extensions

The primary use of our dataset is to facilitate research related to automated analysis, understanding, and reproduction of bug reports. For example, recent approaches that utilize program analysis and natural language processing to automatically reproduce crashes in Android apps [20], [3], [21] could use our dataset to measure the fraction of successfully reproduced reports. Such approaches can also utilize our manually-produced "ground truth" execution scripts, comparing the number of steps in the automated and manually produced versions to ensure that an automated reproduction does not produce excessively long and cumbersome scripts. AndroR2 could be also used be used to perform bug report prioritization based on the quality of S2Rs. Finally, the dataset could be used by techniques that aim to map S2Rs into GUI actions [22].

Additional Usages. Beyond the bug report management scenarios, AndroR2 could be used to benefit research in:

1) Targeted app exploration approaches, e.g., [23], [24], [25], [26], rely on static and dynamic program analysis to force execution towards a particular line of code or app screen. Such approaches could set faults in our dataset as exploration targets, verifying the ability of an approach to successfully reach the target (without relying on the bug report for that purpose) and, when successful, comparing the number of steps in the produced execution to those in our execution scripts.

2) AndroR2 can be used as a benchmark to assess the efficiency and scalability of dynamic application slicing techniques [27], [28]. Such techniques are typically used for debugging purposes, with a fault set as the slicing criteria.

3) Similarly, fault localization techniques [29], especially for Android applications [30], could benefit from our collection of apps with "verified" faulty behaviors.

4) Finally, automated approaches for analyzing video recordings of Android app usages into replayable scenarios, e.g., [31], could use execution scripts in our dataset as a testbed for evaluating whether the tools can accurately analyze and replay scenarios from screen recordings.

Dataset Extensions. The most obvious way of extending our dataset is to analyze and reproduce more bug reports. To facilitate such extension, we include with AndroR2 a tool for mining, extracting, and storing issues from GitHub. In addition, we provide a database of current 82,455 collected issues.

AndroR2 can be further extended through the addition of assertion statements to the reproduction scripts of bugs without an explicit oracle (i.e., crashing bugs) in order to create a set of functional tests. With such assertions added, the dataset could help to foster research related to automated bug repair techniques, to validate that the proposed repairs are successful and lead to a passing test. Furthermore, augmenting the execution scripts with assertions can provide the needed reference model to evaluate automated assertion generation techniques, e.g., [32].

Another way to extend AndroR2 is by including multiple different variants of bug reproduction scripts, i.e., those that perform different sets of actions that lead to the same bug, as opposed to the minimal action sequences currently provided. Such an extension could facilitate research which examines test case selection and prioritization techniques [33] as well as research on detecting duplicate video-based bug reports [34].

VI. Conclusions

This paper presented the AndroR2 dataset. We believe that this dataset fosters open, reproducible future research on automated bug report management, software testing, and software maintenance more broadly.

Acknowledgment

This work was partially supported by a gift from Facebook and the NSF CCF-1955853 grant.

REFERENCES

[1] G. Tassey, "The economic impacts of inadequate infrastructure for software testing," National Institute of Standards and Technology, Tech. Rep., 2002.

[2] O. Chaparro, C. Bernal-Cárdenas, J. Lu, K. Moran, A. Marcus, M. Di Penta, D. Poshyvanyk, and V. Ng, "Assessing the Quality of the Steps to Reproduce in Bug Reports," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Association for Computing Machinery, 2019, p. 86–96.

[3] M. Fazzini, M. Prammer, M. d'Amorim, and A. Orso, "Automatically Translating Bug Reports into Test Cases for Mobile Apps," in *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis*. Association for Computing Machinery, 2018, pp. 141–152.

[4] Y. Zhao, T. Yu, T. Su, Y. Liu, W. Zheng, J. Zhang, and W. G. J. Halfond, "ReCDroid: Automatically Reproducing Android Application Crashes from Bug Reports," in *Proceedings of the 41st International Conference on Software Engineering*. IEEE Press, 2019, pp. 128—-139.

[5] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, and D. Poshyvanyk, "Auto-Completing Bug Reports for Android Applications," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. Association for Computing Machinery, 2015, pp. 673––686.

[6] (2021, Jan.) Google play. [Online]. Available: https://play.google.com/store

[7] D. Han, C. Zhang, X. Fan, A. Hindle, K. Wong, and E. Stroulia, "Understanding android fragmentation with topic analysis of vendor-specific bugs," in *Proceedings of the 2012 19th Working Conference on Reverse Engineering*, ser. WCRE '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 83–92. [Online]. Available: http://dx.doi.org/10.1109/WCRE.2012.18

[8] "Android fragmentation statistics http://opensignal.com/reports/2014/android-fragmentation/," 2014.

[9] N. Jones, "Seven best practices for optimizing mobile testing efforts," Gartner, Technical Report G00248240.

[10] (2021, Mar.) AndroR2 Dataset. [Online]. Available: https://doi.org/10.5281/zenodo.4646313

[11] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss, "What makes a good bug report?" *IEEE Transactions on Software Engineering*, vol. 36, no. 5, pp. 618–643, 2010.

[12] P. Rodeghero, D. Huo, T. Ding, C. McMillan, and M. Gethers, "An empirical study on how expert knowledge affects bug reports," p. 542–564, 2016.

[13] (2021, Jan.) About issues. [Online]. Available: https://docs.github.com/en/github/managing-your-work-on-github/about-issues

[14] (2021, Jan.) Github rest api. [Online]. Available: https://docs.github.com/en/rest

[15] (2021, Jan.) Search. [Online]. Available: https://docs.github.com/en/rest/reference/search

[16] (2021, Jan.) Build and run your app. [Online]. Available: https://developer.android.com/studio/run

[17] (2021, Jan.) App manifest overview. [Online]. Available: https://developer.android.com/guide/topics/manifest/manifest-intro

[18] (2021, Jan.) The database for modern applications. [Online]. Available: https://www.mongodb.com

[19] (2021, Jan.) <uses-sdk>. [Online]. Available: https://developer.android.com/guide/topics/manifest/uses-sdk-element

[20] K. Moran, L.-V. Mario, C. Bernal-Cárdenas, C. Vendome, and D. Poshyvanyk, "Automatically Discovering, Reporting and Reproducing Android Application Crashes," in *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation*, 2016, pp. 33–44.

[21] Y. Zhao, T. Yu, T. Su, Y. Liu, W. Zheng, J. Zhang, and W. G. J. Halfond, "ReCDroid: Automatically Reproducing Android Application Crashes from Bug Reports," in *Proceedings of the ACM/IEEE International Conference on Software Engineering*, 2019, pp. 128–139.

[22] H. Liu, M. Shen, J. Jin, and Y. Jiang, "Automated classification of actions in bug reports of mobile apps," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. Association for Computing Machinery, 2020, p. 128–140.

[23] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, and W. Zou, "SmartDroid: An Automatic System for Revealing UI-based Trigger Conditions in Android Applications," in *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2012, pp. 93–104.

[24] C. S. Jensen, M. R. Prasad, and A. Møller, "Automated Testing with Targeted Event Sequence Generation," in *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2013, pp. 67–77.

[25] J. Schütte, R. Fedler, and D. Titze, "ConDroid: Targeted Dynamic Analysis of Android Applications," in *Proceedings of IEEE International Conference on Advanced Information Networking and Applications*, 2015, pp. 571–578.

[26] D. Lai and J. Rubin, "Goal-Driven Exploration for Android Applications," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, 2019, pp. 115–127.

[27] T. Azim, A. Alavi, I. Neamtiu, and R. Gupta, "Dynamic Slicing for Android," in *Proceedings of the ACM/IEEE International Conference on Software Engineering*, 2019, pp. 1154–1164.

[28] K. Ahmed, M. Lis, and J. Rubin, "Mandoline: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis," in *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation*, 2021.

[29] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A Survey on Software Fault Localization," *IEEE Trans. Software Eng.*, vol. 42, no. 8, pp. 707–740, 2016.

[30] H. Mirzaei and A. Heydarnoori, "Exception Fault Localization in Android Applications," in *Proceedings of the ACM International Conference on Mobile Software Engineering and Systems*, 2015, pp. 156–157.

[31] C. Bernal-Cárdenas, N. Cooper, K. Moran, O. Chaparro, A. Marcus, and D. Poshyvanyk, "Translating Video Recordings of Mobile App Usages into Replayable Scenarios," in *Proceedings of the ACM/IEEE International Conference on Software Engineering*, 2020, pp. 309—-321.

[32] C. Watson, M. Tufano, K. Moran, G. Bavota, and D. Poshyvanyk, "On Learning Meaningful Assert Statements for Unit Test Cases," in *Proceedings of International Conference on Software Engineering*, 2020, pp. 1398–1409.

[33] R. Kazmi, D. N. A. Jawawi, R. Mohamad, and I. Ghani, "Effective Regression Test Case Selection: A Systematic Literature Review," *ACM Comput. Surv.*, vol. 50, no. 2, 2017.

[34] N. Cooper, C. Bernal-Cárdenas, O. Chaparro, K. Moran, and D. Poshyvanyk, "It Takes Two to Tango: Combining Visual and Textual Information for Detecting Duplicate Video-Based Bug Reports," in *Proceedings of the ACM/IEEE International Conference on Software Engineering*, 2021.