

# Enhancing Android Application Bug Reporting

Kevin Moran  
College of William & Mary  
Department of Computer Science  
P.O. Box 8795  
Williamsburg, VA 23187-8795, USA  
kpmoran@cs.wm.edu

## ABSTRACT

The modern software development landscape has seen a shift in focus toward mobile applications as smartphones and tablets near ubiquitous adoption. Due to this trend, the complexity of these “apps” has been increasing, making development and maintenance challenging. Current bug tracking systems do not effectively facilitate the creation of bug reports with useful information that will directly lead to a bug’s resolution. To address the need for an improved reporting system, we introduce a novel solution, called FUSION, that helps reporters auto-complete reproduction steps in bug reports for mobile apps by taking advantage of their GUI-centric nature. FUSION links information, that reporters provide, to program artifacts extracted through static and dynamic analysis performed beforehand. This allows our system to facilitate the reporting process for developers and testers, while generating more reproducible bug reports with immediately actionable information.

## Categories and Subject Descriptors

D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement

## General Terms

Experimentation, Design

## Keywords

Bug reports, android, reproduction steps, auto-completion

## 1. MOTIVATION & RESEARCH PROBLEM

Software maintenance activities are known to be generally expensive and challenging [22] and one of the most important maintenance tasks, especially in hyper-competitive mobile marketplaces, is bug report resolution. However, current bug tracking systems such as Bugzilla [1], Mantis [6], the Google Code Issue Tracker (GCIT) [4], the GitHub Issue Tracker [3], and commercial solutions such as JIRA [5] rely mostly on unstructured natural language bug descriptions.

Permission to make digital or hard copies of part or all 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 third-party components of this work must be honored. For all other uses, contact the Owner/Author.

Copyright is held by the owner/author(s).

ESEC/FSE’15, August 30 – September 4, 2015, Bergamo, Italy  
ACM. 978-1-4503-3675-8/15/08  
<http://dx.doi.org/10.1145/2786805.2807557>

The problem facing many of these current reporting systems is that typical natural language reports capture a coarse grained level of detail that makes developer reasoning about defects difficult. This highlights the underlying *task* that issue tracking systems must accomplish: *bridging the lexical knowledge gap between typical reporters of a bug and the developers that must resolve the bugs*. Previous studies on bug report quality and developer information needs highlight several factors that can impact the quality of reports [11, 14, 10]: **1)** Other than “Interbug dependencies” (i.e., a situation where a bug was fixed in a previous patch), *insufficient information* in bug reports is one of the leading causes of non-reproducible bug reports [14]; **2)** Developers consider (i) *steps to reproduce*, (ii) *stack traces*, and (iii) *test cases/scenarios* as the most helpful sources of information in a bug report [10]; **3)** Information needs are greatest early in a bug’s life cycle, therefore, a way to easily add the above features is important during bug report creation [11].

Using these issues as motivation, we developed an augmented bug reporting solution, FUSION, that gleans contextual information from the GUI of a mobile app to accomplish the following goals: **(i)** *provide bug reports to developers with immediately actionable knowledge (reliable reproduction steps)* and **(ii)** *facilitate reporting by providing this information through an auto-completion mechanism*.

## 2. BACKGROUND & RELATED WORK

**In-Field Failure Reproduction:** A body of work known as in-field failure reproduction [9, 16, 23, 13, 15, 8, 17, 12] shares similar goals with our approach. These techniques collect run-time information (e.g., execution traces) from instrumented programs that provide developers with a better understanding of the causes of a failure, which subsequently helps expedite the fixing of the corresponding faults. However, there are three key points that set our work apart and illustrate how FUSION improves on the state of research.

*First*, in-field failure reproduction techniques rely on potentially expensive program instrumentation. FUSION is completely automatic, our static and dynamic analysis techniques only need to be applied once for the version of the program that is released for testing. *Second*, current in-field failure reproduction techniques require an oracle to signify when a failure has occurred (e.g., a crash). FUSION is not an approach for crash or failure detection, it is designed to support testers during the bug reporting process. *Third*, these techniques have not been applied to mobile apps and would likely need to be optimized further to be applicable for the corresponding resource-constrained environment.

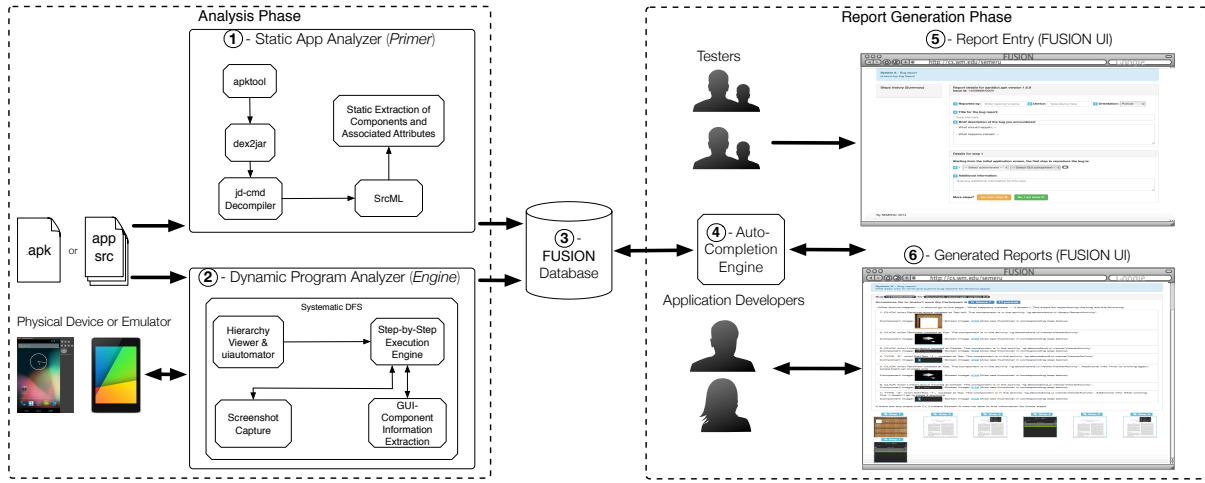


Figure 1: Overview of FUSION Workflow

### 3. APPROACH

In this section we present our approach for Android bug reporting that utilizes an *Analyze*  $\rightarrow$  *Generate* workflow corresponding to two major phases. In the **Analysis Phase** FUSION collects app and GUI information and in the **Report Generation Phase**, FUSION operates on this information to aid users in constructing bug reports. The overall design of FUSION can be seen in Figure 1. Please refer to the following materials [19, 20] for a complete description and dataset.

#### 3.1 Analysis Phase

The *Analysis Phase* has two major components: 1) static analysis (*Primer*), and 2) dynamic program analysis (*Engine*) of a target app. Both components store their data in the FUSION database (Fig. 1 - ③).

**Static Analysis (Primer)** The goal of the *Primer* (Fig. 1 - ①) is to extract all of the GUI components and associated information from the app source code. For each GUI component, the *Primer* extracts: (i) possible actions on that component, (ii) type of the component (e.g., Button, Spinner), (iii) activities the component is contained within, and (iv) class files where the component is instantiated. Thus, this phase results in a universe of possible GUI components for the application, and establishes traceability links connecting these components to specific code-level artifacts.

**Dynamic Analysis (Engine)** The *Engine* (Fig. 1 - ②) is used to glean dynamic contextual information, such as the location of the GUI component on the screen, and enhance the database with both run-time GUI and application event flow information. The goal of the *Engine* is to explore an app in a systematic manner ripping and extracting run-time information related to the GUI components during execution including: (i) text associated with a component, (ii) contextual screenshots of the component, (iii) the activity or screen the component is located within, and (iv) the location of the component on a given app screen. During the ripping, before each step is executed on the GUI, the *Engine* calls *UIAutomator* subroutines to extract the contextual information outlined above regarding each GUI component on the screen. To effectively explore and model the application we took inspiration from our previous work [18] and implemented a systematic depth-first search (DFS) algorithm that performs click/tap events on the components in the GUI hierarchy for a given app.

#### 3.2 Report Generation Phase

The design of the *Report Generation Phase* component of FUSION has two goals: 1) Allow for traditional natural language input in order to give a high-level overview of a bug. 2) Auto-complete the reproduction steps of a bug through suggestions derived by tracking the position of the reporter’s step entry in the app event flow. During the *Report Generation Phase* FUSION’s Auto-completion engine populates a decision tree based on the extracted app model and aids the reporter in constructing the steps needed to recreate a bug by making suggestions based upon the “potential” GUI state reached by the declared steps. This means for each step  $s$ , FUSION infers – online – the GUI state  $GUI_s$  in which the target app should be by taking into account the step history. For each step, FUSION verifies that the suggestion made to the reporter is correct by asking the reporter to select a full contextual screenshot corresponding to the {action, gui-component} tuple they intended to record. The end result is a detailed bug report including natural-language general information and contextualized reproduction steps with screen-shots for each step and traceability links from steps to the app source code.

### 4. RESULTS AND CONTRIBUTIONS

We have evaluated the proposed framework with two empirical studies using 15 real bugs in 14 open-source Android applications extracted from the F-Droid marketplace [2] comparing FUSION against the GCIT [4] and the original bug reports in two maintenance activities involving reporting (*Study 1*) and reproducing bugs (*Study 2*). *Study 1* involved eight graduate students from the College of William & Mary (W&M) who created reports from videos of the bugs for the two systems, for a total of 120 (60 FUSION, 60 GCIT) bug reports. In *Study 2*, 20 new participants, also W&M students, attempted to reproduce the 15 bugs from the bug reports created in the first study, along with the original issue tracker reports. Our results show that, overall FUSION produces more reproducible bug reports than current issue trackers (13/120 non-reproducible with FUSION, versus 23/120 non-reproducible with GCIT). The major contribution of this work is the following conclusion: *By leveraging program analysis and the GUI-centric nature of mobile apps, mobile issue trackers can be augmented to facilitate users creating more detailed and reproducible bug reports.*

## 5. REFERENCES

- [1] Bugzilla issue tracker <https://bugzilla.mozilla.org>.
- [2] F-droid. <https://f-droid.org/>.
- [3] Github issue tracker <https://github.com/features>.
- [4] Google code issue tracker <https://code.google.com/p/support/wiki/IssueTracker>.
- [5] Jira bug reporting system <https://www.atlassian.com/software/jira>.
- [6] Mantis bug reporting system <https://www.mantisbt.org>.
- [7] Usersnap bug reporting tool <https://usersnap.com/features/feedback-widget-for-screenshot-bug-reporting>.
- [8] S. Artzi, S. Kim, and M. Ernst. Recrash: Making software failures reproducible by preserving object states. In J. Vitek, editor, *ECOOP 2008 – Object-Oriented Programming*, volume 5142 of *Lecture Notes in Computer Science*, pages 542–565. Springer Berlin Heidelberg, 2008.
- [9] J. Bell, N. Sarda, and G. Kaiser. Chronicler: Lightweight recording to reproduce field failures. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 362–371, Piscataway, NJ, USA, 2013. IEEE Press.
- [10] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '08/FSE-16, pages 308–318, New York, NY, USA, 2008. ACM.
- [11] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann. Information needs in bug reports: Improving cooperation between developers and users. In *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work*, CSCW '10, pages 301–310, New York, NY, USA, 2010. ACM.
- [12] Y. Cao, H. Zhang, and S. Ding. Symcrash: Selective recording for reproducing crashes. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pages 791–802, New York, NY, USA, 2014. ACM.
- [13] J. Clause and A. Orso. A technique for enabling and supporting debugging of field failures. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 261–270, Washington, DC, USA, 2007. IEEE Computer Society.
- [14] M. Erfani Joorabchi, M. Mirzaaghaei, and A. Mesbah. Works for me! characterizing non-reproducible bug reports. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 62–71, New York, NY, USA, 2014. ACM.
- [15] W. Jin and A. Orso. Bugredux: Reproducing field failures for in-house debugging. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 474–484, Piscataway, NJ, USA, 2012. IEEE Press.
- [16] W. Jin and A. Orso. F3: Fault localization for field failures. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, ISSTA 2013, pages 213–223, New York, NY, USA, 2013. ACM.
- [17] F. Kifetew, W. Jin, R. Tiella, A. Orso, and P. Tonella. Reproducing field failures for programs with complex grammar-based input. In *Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference on*, pages 163–172, March 2014.
- [18] M. Linares-Vásquez, M. White, C. Bernal-Cárdenas, K. Moran, and D. Poshyvanyk. Mining android app usages for generating actionable gui-based execution scenarios. In *12th Working Conference on Mining Software Repositories (MSR'15)*, to appear, 2015.
- [19] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, and D. Poshyvanyk. Auto-completing bug reports for android applications. In *10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'15)*, to appear, 2015.
- [20] K. Moran, M. L. Vasquez, C. B. Cardenas, and D. Poshyvanyk. Fusion online appendix <http://www.fusion-android.com>.
- [21] B. Solutions. Bugdigger. <http://bugdigger.com>, December 2014.
- [22] G. Tassey. The economic impacts of inadequate infrastructure for software testing. Technical report, National Institute of Standards and Technology, 2002.
- [23] J. Zhou, H. Zhang, and D. Lo. Where should the bugs be fixed? - more accurate information retrieval-based bug localization based on bug reports. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 14–24, Piscataway, NJ, USA, 2012. IEEE Press.