# Mining Challenge 2012: The Android Platform

Emad Shihab*, Yasutaka Kamei+ and Pamela Bhattacharya#
*Queen's University, Kingston, ON, Canada**
*Kyushu University, Fukuoka, Japan+*
*University Of California, Riverside, CA, USA#*
*emads@cs.queensu.ca, kamei@ait.kyushu-u.ac.jp, pamelab@cs.ucr.edu*

***Abstract*—The MSR Challenge offers researchers and practitioners in the area of Mining Software Repositories a common data set and asks them to put their mining tools and approaches on a dare. This year, the challenge is on the Android platform. We provided the change and bug report data for the Android platform asked researchers to uncover interesting findings related to the Android platform. In this paper, we describe the role of the MSR Challenge, highlight the data provided and summarize the papers accepted for inclusion in this year's challenge.**

***Keywords*-Mining Software Repositories; MSR Challenge; Android Platform**

## I. Introduction

Software repositories such as source control, bug and, communication repositories are widely used in large software projects. These repositories were traditionally used to archive information about the development of large projects. However, the Mining Software Repositories (MSR) field has started to analyze the data stored in software repositories to uncover interesting and actionable information about software systems [5].

Mining software repositories has its challenges. The data in these repositories need to be extracted, preprocessed, and validated. The MSR challenge was put in place to enable the participation of researchers in the field of MSR without having to extract and preprocess data. Each year, data for one or more projects is extracted and posted on the MSR Challenge website. Researchers are encouraged to leverage this data to uncover interesting findings.

The focus of this year's challenge was the *Android platform*. Data from the source control and bug repositories were extracted, preprocessed and given in XML format. The choice to use the Android platform as a case study is driven by the fact that mobile software engineering seems to be a hot and upcoming topic. We believe that providing the data for the Android platform serves as a good starting point for researchers working in the area of mobile software engineering.

The rest of this paper is organized as follows. Section II provides a brief overview of the MSR Challenge. Section III describes the data provided in this year's MSR Challenge. The submissions accepted to this year's MSR challenge are briefly described in Section IV. Section V concludes the paper.

## II. The Challenge

Since its start in 2006, the MSR Challenge served as a venue where researchers and practitioner apply their mining techniques on a common data set, provided by the challenge organizers. The goal of the challenge is to encourage new comers to try their techniques without having the burden of preprocessing and extracting the data. In addition, it serves as a venue for current researchers to demonstrate their state of the art techniques on a public data set.

In the past, data for Eclipse (2007, 2008, 2011), Firefox (2008, 2011), the GNOME desktop suite (2009), Debian (2010), Chrome, and Netbeans (2011) was provided. This year, we provide data for the Android platform in order to encourage research in the emerging area of mobile software engineering. To simplify the data extraction process, the change and bug data for the Android platform was provided in XML format. We detail the data in Section III.

Similar to past MSR Challenges, researchers were asked to report their interesting findings in a 4 page challenge report. Each report was required to provide an introduction to the problem being addressed, an overview of the data used, a description of the approach and tools used, a description of the findings and their implications, and final conclusions. Each report was evaluated by at least three PC members.

## III. The Android Platform Data

This year, we provided change and bug data for the Android Platform. In this section, we describe the data extraction process and provide descriptive statistics about the provided data.

### A. Android Change Data

To obtain the change data for the Android platform, we mirrored the GIT version control repository using GIT's *clone* command. Since the Android platform is composed of a number of smaller sub-projects, wee mirrored the GIT repository for each sub-project individually[1]. We found 275 sub-projects (e.g., device/common and kernel/common).

[1]git clone git://android.git.kernel.org/ + subproject name

978-1-4673-1761-0/12/$31.00 © 

Table I
CHANGE DATA FIELDS

| GIT *log* Option [3] | XML Tag | Explanation |
|---|---|---|
| %H | <commit_hash> | Commit hash (i.e., Revision ID) |
| %T | <tree_hash> | Tree hash |
| %P | <parent_hashes> | Parent hashes |
| %an | <author_name> | Author name. The author is the person who originally wrote the patch. |
| %ae | <author_e-mail> | Author e-mail address |
| %ad | <author_date> | The commit date by the author |
| %cn | <committer_name> | Committer name. The committer is the person who applied the patch. |
| %ce | <committer_email> | Committer e-mail address |
| %cd | <committer_date> | The commit date by the committer |
| %s | <subject> | Subject of the commit |
| %b | <message> | The detail message of the commit |
| --name-only† | <target> | The list of changed files |

† Number of added/deleted lines is obtained using *--numstat* not *--name-only*.

Then, we use the *log* command[2] to extract the revision data for each file in the sub-projects. Table I shows the extracted fields and provides a brief explanation of each field. Finally, the data was preprocessed and stored in XML format.

The data was grouped by change-list. For each change-list, we extracted:

- The name of the sub-project the change-list belongs to
- The author name, e-mail and author commit date
- The committer name, e-mail and commit date of the change-list
- The commit time of the change-list
- The subject of the change-list
- The message describing the change-list
- The list of files modified by the change-list

### B. Descriptive Statistics of the Android Change Data

Android is composed of five main projects (i.e., device, kernel, platform, toolchain and tools) and 275 sub-projects (e.g., device/common). Figure 1 shows the distribution of the number of commits in the provided change data. The x-axis shows the number of commits for each sub-project and the y-axis shows the number of sub-projects. We observe that nine sub-projects (e.g., *kernel/linux-2.6* and *platform/frameworks/base*) are very active, having more than 10,000 commits. The large number of commits suggests that these sub-projects are the core systems of the Android platform. On the other hand, there are 183 sub-projects (e.g., *platform/external/protobuf* and *platform/vendor/renesas/ms7724*) that have less than 100 commits.

[2]git log --pretty=format:"%H,%T,%P,%an,%ae,%ad,%cn,%ce,%cd,%s,%b" --name-only

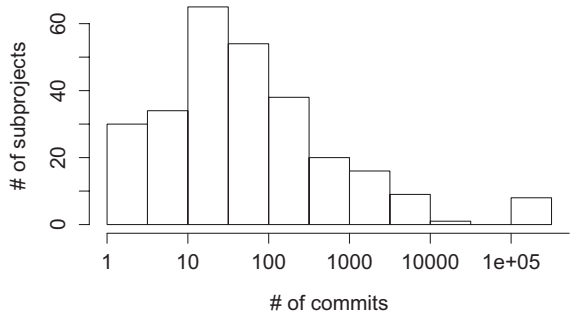


Figure 1. Distribution of the Number of Commits in the Android Sub-projects

Table II
STATISTICS OF THE TOP 10 MOST ACTIVE ANDROID SUB-PROJECTS

| Sub-project | # Commits | # Authors | # Committers | # Modified Files |
|---|---|---|---|---|
| kernel/linux-2.6 | 254,073 | 8,212 | 316 | 57,557 |
| kernel/omap | 234,235 | 7,749 | 297 | 55,381 |
| kernel/tegra | 213,250 | 7,237 | 276 | 52,444 |
| kernel/common | 211,941 | 7,172 | 273 | 52,270 |
| kernel/qemu | 211,915 | 7,169 | 271 | 52,269 |
| kernel/samsung | 202,857 | 6,976 | 277 | 51,291 |
| kernel/msm | 202,773 | 6,958 | 270 | 51,280 |
| kernel/experimental | 110,251 | 4,473 | 157 | 35,186 |
| platform/frameworks/base | 30,775 | 524 | 354 | 16,940 |
| platform/external/bluetooth/bluez | 7,249 | 85 | 36 | 758 |

Table II shows the number of commits, authors, committers and modified files in the top 10 most active sub-projects. We see that eight of the 10 sub-projects belong to the *kernel* project, and two sub-projects belong to the *platform* project. One interesting observation we made is that in all 10 sub-projects, we found that the number of authors is much more than the number of committers. This result indicates that the Android platform is supported by a lot of contributors, however, only a few committers can actually commit patches to the GIT repository.

### C. Android Bug Data

We collected the Android bug reports from Android's public bug tracker [1]. After the extraction of the Android bug reports, we extracted several fields from the bug reports, preprocessed them and stored the data in an XML file. The schema and the tags used in the XML file are shown in Table III. Next, we provide descriptive statistics of the Android bug data and highlight some interesting facts about the collected data.

Table III
XML SCHEMA FOR THE ANDROID BUG DATA

| Tag | What | Values |
|---|---|---|
| bugid | Bug report ID | 1—18162 |
| Title | Title of the bug report | Text |
| Status | Status of the bug | New, Assigned, Fixed, Duplicate, Spam, Invalid, Wont fix, Closed, Verified |
| Owner | Developer who owns the bug | Email ID |
| Type | Kind of bug | Defect, Enhancement |
| Priority | Priority of the bug | Critical, High, Medium, Low |
| Component | Sub-project/module the bug belongs to | Text |
| ClosedOn | Date the bug was closed | mm-yyyy (if closed) "Open Issue" (otherwise) |
| Stars | How many people starred the bug | Integer ($\geq 0$) |
| ReportedBy | User who reported the bug | Email ID |
| OpenedDate | Date the bug was reported | mm-dd-yyyy |
| Description | Description of the bug by the bug-reporter | Text |
| Comment | Details of each comment | |
| Author | User who commented on the bug report | Email ID |
| When | Date the comment was posted | mm-dd-yyyy |
| What | Description of the comment posted | Text |

Table IV
USER CONTRIBUTION DISTRIBUTION

| Contribution count | Number of Users | |
|---|---|---|
| | Reported bugs | Commented on bug reports |
| 1 | 8,803 | 34,514 |
| 2–10 | 2,710 | 12,378 |
| 11–50 | 133 | 545 |
| 51–100 | 8 | 32 |
| 101–200 | 0 | 14 |
| 201–500 | 1 | 7 |
| 501–1000 | 0 | 5 |
| 1001 or more | 0 | 5 |

- We found that the average description length of a bug by the user consists of an average of 189 words. However, only 16% of the bug reports contain steps to reproduce details.
- The public Android bug tracker we analyzed contains 65% newly reported bugs. Only 5% of the bugs reported so far has been fixed while 12% of the bugs were declined because either they were invalid or will not be fixed for miscellaneous reasons.
- 99% of all bugs in the Android bug repository that we analyzed are of medium priority.

Table V
BUG COMPONENT DISTRIBUTION

| Component | Percentage |
|---|---|
| Market, Docs, Build | |
| User, Web, System | 2 |
| GfxMedia | 1.7 |
| Device | 1.8 |
| Browser | 2 |
| Google | 2.1 |
| Dalvik | 2.2 |
| Framework | 3.3 |
| Tools | 3.8 |
| Applications | 4 |
| No component | 77.9 |

- The average time it takes to fix a bug is 2.43 months.
- In Table V, we present the component distribution of the bug reports. We find that most bug reports are not assigned to any component.
- We found that total of 11,655 individual users submitted bug reports while 47,500 individuals commented on bug reports and participated in the bug-fix process. In Table IV, we show the distribution of user participation in reporting bugs and commenting on the bug reports. We found that majority of the users (attributing to 72%–75% approximately) submit or comment on a bug report only once during their lifetime association with the project.

## IV. SUBMISSIONS

We received a record number of submissions to the MSR Challenge this year. A total of 17 challenge reports were submitted, of which we accepted 6 (35% acceptance rate). In this section, we summarize the accepted papers.

Asaduzzaman *et al.* [2] map bug reports and changes to identify bug-introducing changes in Android. The authors main goal was to better understand the circumstances in which these bug-introducing changes occur. In order to draw attention to the most bug-introducing parts of the code, the authors highlight the top five most buggy files in the Android platform. They also find that in contrast to prior work [10], the highest percentage of the bug-introducing changes occur on Saturdays for the Android platform. Finally, the authors find that bug-introducing changes are larger than the changes that do not introduce bugs.

Martie *et al.* [7] use LDA to identify 100 high level topics discussed in the Android bug repository. They use the timestamps in the bug reports to study how topics change over time and their relation to the release dates. Analyzing and modelling the evolution of topics provides practitioners and researchers with valuable insight about the state of the project.

Arjona and Robles [8] analyze the Android source control repository to understand how localization and translation are managed. In particular, the authors focus on who participates

in localization and translation tasks and how Android benefits from external contributions. They find that Android lacks specialized groups that are dedicated only to localization tasks. They also find that contributors to localization tasks need to follow the same procedures as those contributing code. The authors argue that Android should simplify the contribution process if they would like to draw more non-technical people to help with the translation efforts.

Guana *et al.* [4] study the architectural layers of Android using data mined from the Android bug repository. The authors set out to determine the affect of architecture on bug density, lifetime, and author and community following. They find that the framework layer had the highest bug concentrations, that bug lifetimes are similar across architectural layers (although the framework layer has long lasting bugs), and that central developers pay close attention to the framework layer, whereas, the community closely looks at the kernel components.

Hu *et al.* [6] compare Android's concrete architecture, based on build dependencies, and compare it to the conceptual architecture. The authors find that Android's concrete architecture conforms to its conceptual architecture. The authors argue that the findings of this research are valuable for understanding and impact analysis of software changes.

Sinha *et al.* [9] mine the change history of Android and perform an exploratory study. The authors profile the various sub-projects and the developer community and industrial contributions to Android. They find that there is a large amount churn in the Android developer community, that Google and Android are the main contributors to the project, however, other companies such as Intel, RedHat, IBM, Oracle, TI and SGI also contribute to the Android project.

## V. Conclusion

The MSR Challenge serves as a venue for new comers and established researchers, both in academia and industry, in the MSR community to work on a common data set and communicate their interesting findings about it. This year, the focus of the challenge was the Android platform, in hopes of facilitating future research in the area of mobile software engineering. We accepted six challenge reports that cover a wide range of topics and unveil very interesting avenues for future work in the area.

## Acknowledgment

This challenge would not have been possible without the MSR Challenge committee. We would like to thank Alberto Bacchelli, Olga Baysal, Pamela Bhattacharya, Oscar Callau, Julius Davies, Emanuel Giger, Lucia, Yasutaka Kamei, Donghoon Kim, Francisco Servant, Stephen Thomas, and Annie T.T. Ying for diligently reviewing the MSR Challenge submissions and participating in the decision discussions. Also, thanks to Pavel Senin for his help in validating the Android data. Thanks to Max Di Penta, Tao Xie and Michele Lanza for their feedback and support.

## References

[1] Android bug tracker. http://code.google.com/p/android/issues/list.

[2] Muhammad Asaduzzaman, Michael Bullock, Chanchal K. Roy, and Kevin Schneider. Bug introducing changes: A study with android. In *The 9th Working Conference on Mining Software Repositories*, page to appear, 2012.

[3] Scott Chacon. *Pro Git*. Apress, 2009.

[4] Victor Guana, Fabio Rocha, Abram Hindle, and Eleni Stroulia. Do the stars align? multidimensional analysis of android's layered architecture. In *The 9th Working Conference on Mining Software Repositories*, page to appear, 2012.

[5] A.E. Hassan. The road ahead for mining software repositories. In *Frontiers of Software Maintenance (FoSM'08)*, pages 48–57, 2008.

[6] Wei Hu, Dan Han, Abram Hindle, and Kenny Wong. The build dependency perspective of android's concrete architecture. In *The 9th Working Conference on Mining Software Repositories*, page to appear, 2012.

[7] Lee Martie, Vijay Krishna Palepu, Hitesh Sajnani, and Cristina Lopes. Trendy bugs: Topic trends in the android bug reports. In *The 9th Working Conference on Mining Software Repositories*, page to appear, 2012.

[8] Laura Arjona Reina and Gregorio Robles. Mining for localization in android. In *The 9th Working Conference on Mining Software Repositories*, page to appear, 2012.

[9] Vibha Sinha, Senthil Mani, and Monika Gupta. Mince: Mining change history of android project. In *The 9th Working Conference on Mining Software Repositories*, page to appear, 2012.

[10] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? In *MSR '05: Proceedings of the 2005 international workshop on Mining software repositories*, pages 1–5, 2005.