Cover Letter: “Exception Handling Bug Hazards in Android: Results from a Mining Study and an Exploratory Survey”

This paper “*Unveiling Exception Handling Bug Hazards in Android based on GitHub and Google Code Issues*” has been selected for sending an extended version to the MSR 2015 special issue published by the Springer journal Empirical Software Engineering (EMSE) .

**Additional Content:** This extended version entitled “Exception Handling Bug Hazards in Android: Results from a Mining Study and an Exploratory Survey” includes an exploratory survey which was specifically designed for this new paper version. The goal of this exploratory survey was to assess Android developer’s perceptions concerning the exception handling bug hazards found as a result of the repository mining study presented in MSR. This new study represents to more than 30% new work as can be observed on the extended paper version.

**Reviewer’s comments:** Moreover, we tried our best to substantively address the reviewer's comments. The tables in the next two sections transcribe the reviewers’ comments and our comments on how each suggestion was addressed. In the table, “r.” stands for “reviewer number” (1, 2, or 3). Finally, we would like to thank all MSR reviewers for their time and comments.

|  |  |  |
| --- | --- | --- |
| r. | Reviewer’s comment | Authors’ comment |
| 1 | An important finding for Android developers is that two thirds of all crashes come from the runtime exception mechanism, and most of these originate from the application layer. In the boxed out version of this finding, I recommend replacing the phrase "most of these", with something a little bit more precise. | We rephrased the boxed finding to be more precise:  Finding:Almost 65% of all crashes come from runtime  exceptions. Approximately 47\% of these originate from the application layer. |
| 1 | Figure number two could be improved by removing the grey background which serves no purpose to make some of the (very small) text rather hard to read, particularly for those with older eyes like this particular reviewer :-). A similar problem occurs in Fig. 1, where the text is very hard to read, though I didn't actually spend much time reading this since it's an example stack trace (!). | Both Figures were fixed already for MSR submission. |
| 1 | The word "battery" has two letter 't's in it. | Fixed - already for MSR submission. |
| 1 | Will the ExceptionMiner tool described in Section III.C be made publicly available? | We included the link on which the tool is available - already for MSR submission. |
| 2 | While the design of the study is well laid out, I had a hard time digesting the manner in which the authors presented the results. They frame the purpose of the study specifically in two ways: 1) In relation to finding bug hazards, and 2) in relation to a set of “Best Practices” regarding Java exception handling, and whether or not these adhere to Android applications based on the information they mine. Therefore, when I arrived at the results section I anticipated that the presentation of the results would be in relation to these purposes laid out by the author. However, this not the case. While there is some loose relation to the purpose, the authors seems to arbitrarily discuss certain results of the study and outline them as major findings. For instance, they present their results under two major generic headings entitled “Common Root Exceptions” and “Exception Types”. They loosely related some of the results presented in this this section back to bug hazards! and the programming guidelines, but they leave it up to the reader to infer any other connections. | We restructured the Way the results were presented. Firstly we presented the general research question that guided the exploratory study “RQ 1: Can the information available in exception stack traces reveal exception handling  bug hazards in both the Android applications and framework?”  Then to make the analysis easier, we further refine this question in subquestions, each one focusing on pieces of information  distilled from stack traces, more specifically: (i) the root exceptions (i.e., the exceptions that caused the stack traces);  (ii) the exception types (i.e, Checked, Runtime, Error, Throwable) and (iii) the exception wrappings.  Hence, the results section is centered around the following subquestions:  RQ 1.1: Can the root exceptions reveal bug harzards?; RQ 1.2 Can the exception types reveal bug hazards?; and RQ 1.3 Can the exception wrappings reveal bug hazards?.  Inside each section we stress the bughazards found and how they relate to the best practices. It is not up to the reader to make such commection anymore. |
| 2 | A more complete discussion of the results would have explicitly presented:   1. Examples of violations of the exception handling programming guidelines laid out at the inception of the study, 2. with actionable, justified information about how developers could fix these problems, or further research that needs to be done to facilitate aiding developers, | In the Mining Results section each piece information extracted from the stack traces is analyzed in detail to  check whether they can reveal bug harzards on the exception handling code -  related to examples of violations of the best practices.  Altough we do not present actionable, justified information about how developers could fix these problems (since we did not come up with them yet) we presente the need of tools and guidelines to aid developers when facing such bug hazards – such info was included in the Discussion section (already for MSR final version). |
| 2 | I think it would have helped the overall structure of the paper if the authors had outlined explicit research questions that they wished to answer as a result of performing the study. This way the contribution of the paper based on the results is concrete and immediately apparent to the user. | Done. The research questions that guided the mining study were made explicit in this extended version and was used to structure and present the results. |
| 2 | This is a minor point, but the filtering/mining technique of using a search based on the word “android” could be improved. Specifically, they could have checked whether the associated source code contained an Android Manifest File. | Indeed this step could have been done automatically, but after using a search based on the word “android” we manually inspected the projects website looking for the Android Manifest File and other characteristics that made it a real Android Project. In this step we also removed toy projects found. This filtering is descriped on the paper. |
| 2 | The authors offer a replication package on a public GitHub page they created, however, while the datasets that they mention in Section III E. are available, the source code for the ExceptionMiner is not. However, they may plan on making this available pending the acceptance of the paper. | The tool was made available on the projects website: https://github.com/souzacoelho/exceptionminer |
| 2 | The reviewer 2 listed a set of presentation Issues and Typos | All of them were fixed, except one:  Section VII: “Our work differs from Kechagia and Spinellis since it is \*\*not\*\* based on stack traces mined from issues reported by open source developers on GitHub and Googlecode.” I assume the authors meant to say: “Our work differs from Kechagia and Spinellis since it is  \*\*[ ]\*\* based on stack traces mined from issues reported by open source developers on GitHub and Google Code.”  Actually the Kechagia work was \*not\* based on traces \*mined from issues\*. The stack traces used in her work were stack traces reported by BugSense a Crash Report Tool. |
| 2 | Overall points for improvement:  - The results are not presented in terms of the purpose of the study, making interpretation of the results confusing;   - Source code missing from reproducibility package;   - Somewhat incremental improvement on the state of state of research. | Previous comments show how we addressed each point. Moreover, the exploratory survey described in this extended version represents another improvement on the state of state of research in this context. |
| 3 | 1) "Moreover, such bug hazards call for improvements on languages and tools to better support exception handling in Android and Java environments." - Why is this a language issue and not a programming issue, i.e., how are improvements in languages going to change anything here? | (e.g. to prevent null pointer dereferences) and tools to better support exception handling in Android and Java environments. |
| 3 | 2) I would strongly suggest that the authors annotate Fig 1. They could mark it with colours or just (a), (b), (c) etc and then refere to the codes in the paper, so that the reader knows exactly which line there are referring to. | Done |
| 3 | The reviewer pointed a set of typos. | Every typo was fixed. |
| 3 | 9) Page 6: "More than 50% of the uncaught exceptions..." - Since this was done on a sample, please mention the confidence interval and error margin based on the size of the sample with respect to the population. | In our study we mined the exception stack traces of all Android repositories in GitHub and Googlecode that could be identified by the term “android” and were not toy projects. Hence the whole population would be:   1. All Android apps available in GitHub and Googlecode. 2. All stack traces found on their issues.   In order to quantify the whole population we would need to firstly manually inspect all projects and issues from GitHub and Googlecode and secondly manually inspect each issue to check if it contains stack traces.  Those tasks are so time consuming that make them infeasible, and as a consequence makes it difficult to calculate what was asked. |

**COMPLETE MSR2015 REVIEWS**

----------------------- REVIEW 1 ---------------------  
PAPER: 50  
TITLE: Unveiling Exception Handling Bug Hazards in Android based on GitHub and Google Code Issues  
AUTHORS: Roberta Coelho, Lucas Almeida, Georgios Gousios and Arie van Deursen  
  
  
----------- REVIEW -----------  
This paper is concerned with the exceptions that can be raised by android code. The authors analyse 482 different projects mined from GitHub and 157 projects mind from Google code. Overall, this results in a little over 6,000 stack traces, which are mined to consider the ways in which exceptions are raised in these android applications. The authors consider unexpected cross type exception wrappings, undocumented runtime exceptions, and undocumented checked exceptions thrown by the platform.  
  
The goal of the mining activity is to be able to predict bug hazards (actually more accurately it explains and characterisers them, reporting on their prevalence, but does not seek to predict). The authors  use the stack traces to try and understand the kinds of mistakes that are likely to lead to these three kinds of bug.  
  
The authors find that programming logic on the one hand and  IO, memory and battery  resource issues on the other contribute approximately three quarters of the exceptions caused in the stack traces they mined. Security issues and concurrency, which I would've expected to cause more issues for Android, raised comparatively few of the exceptions, a little over 5%.  
  
Perhaps less surprising, null pointer exceptions turn out to be the most common of the program errors that lead to exceptions being raised.  
  
An important finding for Android developers is that two thirds of all crashes come from the runtime exception mechanism, and most of these originate from the application layer. In the boxed out version of this finding, I recommend replacing the phrase "most of these", with something a little bit more precise. This will be one of the main actionable findings, since it tells developers that their applications are the cause of crashes, not the library nor  the platform itself.  
  
The paper is very well written, with sufficient material on the Android platform and Java exception model in order to make the paper self-contained. The study design is clearly set out, and the paper includes the detailed and authoritative related work section. The design explains how the authors arrived at their set of 482 apps from GitHub and 157 from Google Code. The approach adopted here seems to reflect best practice to me, in both mining and extracting and filtering data, and also in reporting this to support the replication.  
  
Overall, I think this paper is going to be very useful for practising Android developers, and will also provide interesting results for further research. I believe it is a good example of the kind of mining that commuting sites based on stack traces. I believe the paper should be accepted.  
  
  
Minor points the corrections:  
  
Figure number two could be improved by removing the grey background which serves no purpose to make some of the (very small) text rather hard to read, particularly for those with older eyes like this particular reviewer :-). A similar problem occurs in Fig. 1, where the text is very hard to read, though I didn't actually spend much time reading this since it's an example stack trace (!).  
  
The word "battery" has two letter 't's in it.  
  
Will the ExceptionMiner tool described in Section III.C be made publicly available?  
  
  
----------------------- REVIEW 2 ---------------------  
PAPER: 50  
TITLE: Unveiling Exception Handling Bug Hazards in Android based on GitHub and Google Code Issues  
AUTHORS: Roberta Coelho, Lucas Almeida, Georgios Gousios and Arie van Deursen  
  
  
----------- REVIEW -----------  
Summary:  
  
This paper presents a study based on mining 6,005 exception stack traces from  
482 and 157 Android projects hosted on Github and Google Code respectively. The goal of the studies as presented by the authors is to investigate whether this information can help reveal bug hazards, which the authors define as “a circumstance that increases the chance of a bug [being] present in software.” They state that a typical example of a bug hazard is some characteristic of exception handling code that might increase the likelihood of introducing uncaught exceptions.  
  
The study conducted by the authors takes the following form: After offering some preliminary background information on the Android platform and Java exception handling, the authors first explore related work that outlines general guidelines on how to use Java exceptions. They crystallize four major guidelines that they claim drives the analysis phase of their study.  
  
They develop a tool, which they call ExceptionMiner, to extract the exception stack traces embedded in issues on the Google Code and GitHub Issue pages, and combine this information with additional data extracted from source and bytecode analysis. The Github dataset they used for the study is one provided by the GHTorrent project, and to identify Android projects the authors did a case insensitive search for the term “android” in the repositories names and short descriptions. The authors then conducted a manual inspection of the apps inspecting the site of every Android project reporting at least one stack trace. The authors took a similar approach to mine Android projects from the Google Code repository, however, the obtained the initial dataset by implementing a web-crawler that extracts all of the issues and issue comments, storing them in a relational database. ExceptionMiner passes all of this information through a  
“distiller” component that utilizes a parser and some heuristics to identify exception names and stack traces inline with text. This component also extracts attributes from the stack traces themselves, such as the root exception and its signaler, as well as exception wrappers and their corresponding signalers. The authors also claim their tool can discover stack traces mixed with log file information. After the mining and distilling phases, ExceptionMiner combines the information extracted from the stack traces with relevant static and bytecode analysis, specifically to analyze the exception types and exception signalers. It is of note that the augmented analysis presented in this phase is tethered to a specific version of the Android platform (Version 4.4, API level 19). This automatic process was augmented by manual intervention on the authors’ part to 1) support identification of packages composing the Android platform, 2) identify the type of some exceptions not au!  
 tomatically identified by ExceptionMiner (generally because they were defined on previous versions of libraries, apps and Android Platform). The authors offer a replication package on their Github page: [https://github.com/](https://github.com/" \t "_blank)souzacoelho/exceptionminer  
  
The authors next present a section on the results of the study, followed by a discussion section offering comments on some of the more interesting results. The major findings from the study can be summarized as follows:  
  
1. 50% of uncaught exceptions are due to errors in programming logic, with the NullPointerException being the most common. 25% of the uncaught exceptions are related to resource constraints.  
  
2. About 2/3 of all crashes come from run-time exceptions, most of which originate from the application layer.  
  
3. A very small fraction of the runtime exceptions are programmatically thrown, and almost none of these were documented.  
  
4. For native methods, checked exceptions can be thrown without being documented on the exception interface.  
  
5. Cross-Type exception wrappings are very common, and can lead to lengthy exception chains.  
  
Comments:  
  
The major intellectual merit of this paper stems from the analysis that the authors perform on 6,005 stack traces in relation to bug hazards. The authors present a tool, ExceptionMiner that can automatically perform analysis on stack traces and corresponding byte and source code and present detailed information about the types of root and wrapper exceptions included in a stack trace. A tool that can perform this level of analysis on Android Stack traces has not been created to date. There have been previous studies done in relation to Android Stack Traces but it was presented in a short paper by Kechagia and Spinella, and their focus was specifically on Android API methods with undocumented exceptions that are part of application crashes. By looking at stack traces from the point of view of bug hazards the authors offer an incremental advancement regarding the state of research.  
  
While the design of the study is well laid out, I had a hard time digesting the manner in which the authors presented the results. They frame the purpose of the study specifically in two ways: 1) In relation to finding bug hazards, and 2) in relation to a set of “Best Practices” regarding Java exception handling, and whether or not these adhere to Android applications based on the information they mine. Therefore, when I arrived at the results section I anticipated that the presentation of the results would be in relation to these purposes laid out by the author. However, this not the case. While there is some loose relation to the purpose, the authors seems to arbitrarily discuss certain results of the study and outline them as major findings. For instance, they present their results under two major generic headings entitled “Common Root Exceptions” and “Exception Types”. They loosely related some of the results presented in this this section back to bug hazards!, and the programming guidelines, but they leave it up to the reader to infer any other connections.

A more complete discussion of the results would have explicitly presented 1) Examples of violations of the exception handling programming guidelines laid out at the inception of the study, with actionable, justified information about how developers could fix these problems, or further research that needs to be done to facilitate aiding developers, and 2) Present definite bug hazards inferred from the data analyzed by ExceptionMiner, and again, actionable information on how developers could mitigate these hazards or further research that could be done to facilitate this. This being said, I believe the results reported in the current version of the paper are of value, however, they are poorly laid out and it is difficult to extract the real research value from the current presentation.  
  
I think it would have helped the overall structure of the paper if the authors had outlined explicit research questions that they wished to answer as a result of performing the study. This way the contribution of the paper based on the results is concrete and immediately apparent to the user.  
  
This is a minor point, but the filtering/mining technique of using a search based on the word “android” could be improved. Specifically, they could have checked whether the associated source code contained an Android Manifest File.  
  
The authors offer a replication package on a public GitHub page they created, however, while the datasets that they mention in Section III E. are available, the source code for the ExceptionMiner is not. However, they may plan on making this available pending the acceptance of the paper.  
  
One related work by Kechagia and Spinellis examined the stack traces embedded on crash reports sent by 1,800 Android apps from a crash report management service. This work was a short paper that appeared in MSR’14. One limiting factor of this work is the amount of intellectual merit that it provides in addition to the results presented in the previous short paper. The major finding of this previous work is that many crashes could have been avoided if undocumented exceptions had the corresponding documentation added to them. However, the authors do differentiate themselves from this prior work in two major ways 1) Their study mapped the origin of each exception and took into account “best programming practices in their analysis”; 2) They also identified the type of each exception mined and analyzed exception wrappings.  
  
Also, the type and the characteristics of the exception, analyzed using the bytecode, should have been done considering the source code of the system at that particular time (time of the issue). In fact, one of the problems in the design was that the exception could not be found neither automatically nor manually because the authors used always the latest version of the code). However, I think this is a minor issues since: 1) I do not expect that Exceptions could be changed in their nature (error, catched, unmanaged) after their definition 2) The cases of exceptions not found are very few.  
  
Presentation Issues:  
  
Page 1:  
  
Abstract: “Such undocumented exceptions make difficult” => “Such undocumented exceptions make [it] difficult”  
  
Section I: “A bug hazard [14] is a circumstance that increases the chance of a bug to be present in the software.” => “A bug hazard [14] is a circumstance that increases the chance of a bug [being] present in the software.”  
  
Page 3:  
  
Section III: “open-source apps have also been the target of other research [33], [41] addressing the reuse and API stability” => “open-source apps have also been the target of other research [33], [41] addressing [ ] reuse and API stability”  
  
Page 4:  
  
Section III C: “component is based on a combination between a parser (based on regular expressions) and heuristics able to identify and filter exception names and stack traces inline with text.” → “component combines a parser (based on regular expressions) and heuristics able to identify and filter exception names and stack traces inline with text.”  
  
Page 5:  
  
Section III C: “An specific implementation” => “A specific implementation”  
  
“To discover the packages for the first two origins we can use to the Android specification.” => “To discover the packages for the first two origins we can use [ ] the Android specification.”  
  
Section III C: “previous versions of libraries, apps and Android Plaform)” => “previous versions of libraries, apps and Android Pla[t]form)”  
  
Page 7:  
  
Section IV C: “not expected by the programmer, when such method throws” => “not expected by the programmer, when such [a] method throws”  
  
Page 9:  
  
Section VI: “bug hazards observed in this study are due to characteristics of Java exception model” => “bug hazards observed in this study are due to characteristics of [the] Java exception model ”  
  
Page 10:  
  
Section VII: “Our work differs from Kechagia and Spinellis since it is not based on stack traces mined from issues reported by open source developers on GitHub and Googlecode.” I assume the authors meant to say: “Our work differs from Kechagia and Spinellis since it is [ ] based on stack traces mined from issues reported by open source developers on GitHub and Google Code.”  
  
Section VII: “Such analysis revealed intrigging” => “Such analysis revealed [intriguing]”  
  
“The main limitation of approaches based static analysis based approaches are the number of false positives they can generate, and the problems the faced when dealing” => “The main limitation of approaches based [on] static analysis [ ] are the number of false positives they can generate, and the problems [ ] faced when dealing”  
  
“Vsquez et al. [33] analyzed approximately 7K free Android apps” => “V[a]squez et al. [33] analyzed approximately 7K free Android apps”  
  
Strengths:  
  
 - Large number of stack traces examined;  
 - Engineering behind the study is sound;  
 - Dataset is available to aid in reproducibility of the study  
  
Points for improvement:  
  
- The results are not presented in terms of the purpose of the study, making interpretation of the results confusing;  
 - Source code missing from reproducibility package;  
 - Somewhat incremental improvement on the state of state of research.  
  
  
----------------------- REVIEW 3 ---------------------  
PAPER: 50  
TITLE: Unveiling Exception Handling Bug Hazards in Android based on GitHub and Google Code Issues  
AUTHORS: Roberta Coelho, Lucas Almeida, Georgios Gousios and Arie van Deursen  
  
  
----------- REVIEW -----------  
Summary: The authors propose to study the the extent to which exception handling in android apps can be a cause for bugs (bug hazards). They examine 160K issues from 600+ projects to extract stack traces. From the stack traces they are able to understand how exceptions are handled in the apps.  
  
Strengths: A strong case study with very good motivation (supported by the background section). Section 2.C is especially very useful to place the work in context. The authors are also careful in their conclusions. Overall liked it very much.  
  
Improvements: Overall I  liked it very much. There are some small corrections would make it a great paper:  
  
1) "Moreover, such bug hazards call for improvements on languages and tools to better support exception handling in Android and Java environments." - Why is this a language issue and not a programming issue, i.e., how are improvements in languages going to change anything here?  
  
2) I would strongly suggest that the authors annotate Fig 1. They could mark it with colours or just (a), (b), (c) etc and then refere to the codes in the paper, so that the reader knows exactly which line there are referring to.  
  
3) Page 3 - Then the stack traces embedded each -> Then the stack traces embedded in each  
  
4) In Fig 2, the authors say xMiner. Do they mean ExceptionMiner?  
  
5) Page 4: 482 + 101 apps =  588 apps. How did the authors gets 589 projects?  
  
6) Table 1: How come there are no stacktrace labels for github data?  
  
7) Page 5: "were not be automatically identified" -> "were not automatically identified"  
  
8) Page 6: "Batery" -> "Battery"  
  
9) Page 6: "More than 50% of the uncaught exceptions..." - Since this was done on a sample, please mention the confidence interval and error margin based on the size of the sample with respect to the population.  
  
  
-------------------------  METAREVIEW  ------------------------  
PAPER: 50  
TITLE: Unveiling Exception Handling Bug Hazards in Android based on GitHub and Google Code Issues  
  
- strengths of the paper  
1) Well written paper with strong related work section and Section 2.C  
2) Extensive case study  
  
- weaknesses of the paper  
1) Could the authors rewrite the results sections to tie it back closely to Section 2.C, ie the best practices. Currently the results are organized in a different hierarchy. Instead they could organize it based on the best practices.  
2) Making ExceptionMiner open source.  
  
- points of agreement among reviewers  
Well thought out and executed paper. Would be useful for the developers and researchers.  
  
- points of disagreement among reviewers  
None.