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

**Authors:**

Roberta Coelho – (*corresponding Author*). Address -  Federal University of Rio Grande do Norte, CIVT/UFRN. Av. Senador Salgado Filho, 3000. Lagoa Nova, CEP: 59.078-970. Natal/RN. Brazil Tel.: +55-84-3342-2216  - E-mail: roberta@dimap.ufrn.br

Lucas Almeida -  E-mail: lucas.almeida@ppgsc.ufrn.br

Georgios Gousios - E-mail: georgios@cs.ru.nl

Arie van Deursen  - E-mail: arie.vandeursen@tudelft.nl

Christoph Treude  - E-mail: ctreude@dimap.ufrn.br

This is the response letter for the paper “*Exception Handling Bug Hazards in Android: Results from a Mining Study and an Exploratory Survey*” submitted to Empirical Software Engineering Journal (ESEM). This is an extended version of a paper submitted to MSR 2015, which was invited to submit to ESEM/MSR special issue.

Based on the reviews, we received the recommendation so submit a "Minor Revision" of the paper. We tried our best to substantively address all reviewer's comments. The tables bellow transcribe the Guest editors comments (which summarizes the main reviewers’ comments) as well as reviewer comment. The table also presents how each suggestion was addressed. In the table, “r.” stands for “reviewer number” (1, 2, or 3). Finally, we would like to thank all reviewers and editors for their time and comments for this extended version.

|  |  |  |
| --- | --- | --- |
| **r.** | **Guest editors’ comment** | **Authors’ comment** |
|  | adding the description of the procedure of an open coding methodology (Reviewer 1 and 3) | We added a detailed description about the coding process in section 4.2 (second paragraph). |
|  | explaining more motivation behind the questions on the survey (Reviewer 2). | In section “Developers’s Perspective we added the main motivation behind the survey questions. In summary:  The motivation behind the survey questions is twofold:  (i) questions aiming at learning from developers about the usage of the exception handling code in app development; and  (ii) questions focusing on getting developers percerptionds about the bug hazards detected in the first phase of this study. |
|  | clarifying the process the authors used to align two sources about the source code of the stack traces extracted from the issue tracker and the app on which the stack trace manifested. | We added such description and discussed the threats to internal validity associated to the process adopted. |
| r. | Reviewer’s comment | Authors’ comment |
| 1 | the authors should specify exactly how the authors coded and agreed. For example, the paper states that the two coders agreed, but was agreement done independently? How were conflicts handled? This information should be included in the paper. |  |
| 1 | How did the authors deal with the developers who claimed no java expertise? Maybe they could have excluded these developers in the final results (even if is only the 1.41%) |  |
| 1 | I would have liked to see an analysis related with the correlation of the developer's' experience with some of the results of the study. Alternatively, at least the authors could have shown the results divided in groups based on the developer's experience. This could have improved the paper. |  |
| 1 | Regarding question 3: "Are you aware of any best practices for developing the exception handling code in Java?". Here I would have preferred a Liket scale question, since the knowledge of best practices cannot be a completely binary decision. | We try to identify if the developer knew about any which would meant (at least one). A liketrt scale would give a more detailed view about their knowledge. Unfortunately, we could not repeat the survey but such suggestion will be very usefull in future works. |
| 1 | When discussing the answer of a developer that cited Crash-Fast, the authors could give more information about this approach. The paper also reports an incorrect link (The correct link at page 21 is: [http://www.slideshare.net/pyricau/crash-fast](http://www.slideshare.net/pyricau/crash-fast" \t "_blank)). | Indeed the previous ref was broken (<http://www.slideshare.net/pyricau/crash>). Many thanks for reporting it.The ref was corrected and we added a dicussion about the Crash-fast approach. |
| 1 | It would be beneficial for the authors to clearly define lessons from this survey and mining analysis. For example, what do these finding tell researchers in terms of support that developers need? | The discussion section now contains explicit information about tool support needed to help developers to deal with the bug hazards in the exception handling code. |
| 1 | The presentation of the survey results should be improved to pair the research questions to the survey questions better. For example, Figure 5's caption is vague stating "Summary of some survey question answers." Similarly, the survey question that led to the results is not always clear. | The list of survey questions presented in Figure 5 were related to RQ2.1 (first and second questions), RQ 2.2 (third) and R Q 2.5 (fourth question). We slited the list of yes no questions in two sets: (i) one Figure illustrating both questions related to R Q 2.1 and (ii) another one list the questions directly related to RQ 2.2 and 2.5. The lable is now reflecting explicitly the research questions related to each survey question. |
| 1 | The authors provide background and cite related papers to exceptions. However, they do no cite several surveys with Android developers. It would benefit the paper to explain how the proposed work differs and compares to these studies. Suggested papers are the following:  - P.S.Kochhar, F.Thung, N.Nagappan,T. Zimmermann,and D. Lo. Understanding the test automation culture of app developers. In 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), pages 1-10, 2015.  - M. E. Joorabchi, A. Mesbah, and P. Kruchten. Real challenges in mobile apps. In ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'13), pages 15-24, 2013.  - M. Linares-Vásquez, C. Vendome, Q. Luo, and D. Poshyvanyk. How developers detect and fix performance bottlenecks in Aandroid apps. In Proc. ICSME, pages 352-361. IEEE, 2015.  - Gabriele Bavota, Mario Linares Vásquez, Carlos Eduardo Bernal-Cárdenas, Massimiliano Di Penta, Rocco Oliveto, Denys Poshyvanyk: The Impact of API Change- and Fault-Proneness on the User Ratings of Android Apps. IEEE Trans. Software Eng. 41(4): 384-407 (2015. | TO DO |
| 2 | While not an actual contribution (since the details of how it works and an actual evaluation of it is not reported), the ExceptionMiner tool that automatically parses and extracts stack traces from bug reports seems like it would be useful in many other contexts as well.  I hope that the authors make this available to others. | The code of the ExceptionMiner was made available in the website: |
| 2 | The authors state that their tool automatically filters out directories that contain the source code of libraries.  While I commend the authors for doing this and I think it improves the validity of the findings, how exactly is this done?  Do they have their own tool or somebody else's.  Is there a citation that should be here? | We added a foot note to explain how the filtering was performed: \footnote{The filtering was performed in two steps:  firstly, we analyzed the bytecode of the JVM and identified all runtime exceptions defined by it (e.g.,  java.lang.NullPointerExceptoin,java.lang.ArrayIndexOutOfBounds), then  if the libraries/framework method was signaling one of such exceptions it was filtered out from the analysis}. |
| 2 | As one who has used the approach of wrapping exceptions, I found section RQ1.3 to be very interesting.  While the authors showed that some of these violate best practices, it didn't look like the authors really explained which types of wrappings led to which bug hazards.  From a developer standpoint, one of the things I'd like to learn from this paper is what I should be doing and right now many of the RQs, including RQ1.3, say what the authors say (counts, percentages, etc.) but the text is not very proscriptive. |  |
| 2 |  |  |
| 2 |  |  |
| 2 |  |  |
| 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**

COMMENTS FOR THE AUTHOR:  
  
  
  
  
Dear Authors,  
  
thank you for extending the MSR paper for the EMSE special issue. All three reviewers appreciated the extension of this work, in particular the survey of 71 developers with the goal of capturing the findings of your MSR 2015 paper.  
  
In addition to the positive comments, the three reviewers also pointed out the following minor issues that should be addressed:  
- adding the description of the procedure of an open coding methodology (Reviewer 1 and 3)  
- explaining more motivation behind the questions on the survey (Reviewer 2).  
- clarifying the process the authors used to align two sources about the source code of the stack traces extracted from the issue tracker and the app on which the stack trace manifested.  
  
Based on the reviews, we recommended a "Minor Revision".  
  
We believe the included reviews are very constructive and will help you to revise this submission. We would like to see all comments addressed in your next revision. Please also add a response letter in which you clearly describe how and where you have addressed the comments of the reviewers.  
  
We look forward to your wonderful next revision.  
  
Romain, Yasu, and Martin  
Guest editors  
  
  
  
Reviewer #1: SUMMARY:  
  
The paper investigates exception handling bug hazards in Android apps. It considers two perspectives: i) a mining-based study and ii) developer survey. In this work, the developer survey constitutes the additional material beyond the conference paper.  
  
The authors extracted issues from GitHub and Google Code. For GitHub, the authors relied on GHTorrent's database of extracted information. For Google Code, the authors developed a web crawler. The authors identified 482 projects from GitHub with 31,592 issues that amounted to 4,042 exception stack traces and 157 projects form Google Code with 127,456 issues that amounted to 1,963 exception stack traces. From this dataset, the authors built a custom parser using regular expressions and employed heuristics to classify the exception and stack traces. The parser also extracts relevant information such as class and package. Subsequently, the exception type was investigated by analyzing byte code to reach the base Java Exception type (when unavailable, source code was utilized). Additionally, the signaler was classified by considering the package names. Finally, "undefined" exceptions were manually evaluated (31 exceptions that spanned 60 stack traces).  
The authors found that programming mistakes were the most common reason for exception and of these NullPointerExceptions accounted for 27.71% of the exceptions and NPE were present in 51.96% of projects. By preforming concern analysis, the authors found that programming logic and resource limitations were responsible for 75% of the exceptions. The authors also found that almost half of the exceptions originated in the application layer while 17.7% were in libraries and 34.3% were in the Android Platform. Importantly, the authors found that only 0.4% of the runtime exceptions were documented.  
  
The authors conducted a survey with GitHub developers from the identified Android Applications using multiple choice (10 questions) or Likert scale (5) section and an open response section (13 questions). The authors utilized Grounded Theory with two author agreement. This survey with developers appears to be the main contribution (extension) over the conference version of the paper.  
  
In terms of exception handling, the developers indicated that they handle exceptions most of the time (64.8%), but only throw exceptions some of the time or seldom (80.3%). NPE are a pervasive problem as they have impacted 96% of the respondents at least once. The application life cycle and framework were the most common reasons (79%). Predominantly, exception wrapping was considered a bad practice because it impairs proper handing and may crash the application if it remains uncaught. Interestingly, it was only known to 4% of developers that a method could throw a checked exception without declaring it. Finally, the authors indicated that the developers found exception handling improved robustness.  
  
COMMENTS:  
  
The authors performed a good qualitative analysis aimed at understanding developers' perspective regarding the main results of the mining study. The survey is well defined and the usage of code fragments guided the developers to more informed and meaningful answers. I appreciated the usage of the developer ID that enables: (i) traceability; and (ii) understanding of the variety of the developers that actually answered the open questions. All in all, the paper appears to meet the requirements for an extension with the addition of the survey. Hence, I recommend a minor revision.  
  
Soundness:  
  
The mining-based study appears to be sound. However, the authors should expand on the heuristics and regular expressions that were used by the parser. The manual inspection of "undefined" exceptions also benefits the evaluation of the mining-based approach.  
  
The survey appears to be sound in its construction. While Grounded Theory helps the soundness of the open response questions, the authors should specify exactly how the authors coded and agreed. For example, the paper states that the two coders agreed, but was agreement done independently? How were conflicts handled? This information should be included in the paper.  
  
The inspection of the projects to verify the quality and that they are Android improves the quality of the survey, since the results would be more representative.  
  
Also, the following points need to be addressed:  
  
How did the authors deal with the developers who claimed no java expertise? Maybe they could have excluded these developers in the final results (even if is only the 1.41%)  
I would have liked to see an analysis related with the correlation of the developer's' experience with some of the results of the study. Alternatively, at least the authors could have shown the results divided in groups based on the developer's experience. This could have improved the paper.  
  
Regarding question 3: "Are you aware of any best practices for developing the exception handling code in Java?". Here I would have preferred a Liket scale question, since the knowledge of best practices cannot be a completely binary decision.  
  
When discussing the answer of a developer that cited Crash-Fast, the authors could give more information about this approach. The paper also reports an incorrect link (The correct link at page 21 is: [http://www.slideshare.net/pyricau/crash-fast](http://www.slideshare.net/pyricau/crash-fast" \t "_blank)).  
  
Originality and Importance:  
  
The work highlights an important problem of handing exceptions. The mining analysis demonstrated the types of exceptions that are most prevalent in Android applications. The findings demonstrate that programming logic and resources are predominantly the reason for exceptions and that a lack of documentation may also impact the problem.  
  
The developer survey strengthens the paper since it corroborates the mining analysis and demonstrates the perspective of developers with respect to exception handing. It would be beneficial for the authors to clearly define lessons from this survey and mining analysis. For example, what do these finding tell researchers in terms of support that developers need?  
  
Presentation:  
  
The presentation of the survey results should be improved to pair the research questions to the survey questions better. For example, Figure 5's caption is vague stating "Summary of some survey question answers." Similarly, the survey question that led to the results is not always clear.  
  
Related Work:  
  
The authors provide background and cite related papers to exceptions. However, they do no cite several surveys with Android developers. It would benefit the paper to explain how the proposed work differs and compares to these studies. Suggested papers are the following:  
  
- P.S.Kochhar, F.Thung, N.Nagappan,T. Zimmermann,and D. Lo. Understanding the test automation culture of app developers. In 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), pages 1-10, 2015.  
  
- M. E. Joorabchi, A. Mesbah, and P. Kruchten. Real challenges in mobile apps. In  
ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'13), pages 15-24, 2013.  
  
- M. Linares-Vásquez, C. Vendome, Q. Luo, and D. Poshyvanyk. How developers detect and fix performance bottlenecks in Aandroid apps. In Proc. ICSME, pages 352-361. IEEE, 2015.  
  
- Gabriele Bavota, Mario Linares Vásquez, Carlos Eduardo Bernal-Cárdenas, Massimiliano Di Penta, Rocco Oliveto, Denys Poshyvanyk: The Impact of API Change- and Fault-Proneness on the User Ratings of Android Apps. IEEE Trans. Software Eng. 41(4): 384-407 (2015.  
  
  
  
Reviewer #2: This paper explores how exception handling is managed in Android mobile apps.  
  
I liked this study.  
  
The methodology seemed correct to me.  Definitely one of the challenges in doing mobile app studies is finding the source code to such apps, so I think using GitHub to find mobile apps is just fine.  I liked the way that the authors did their filtering and that they got their stack traces from bug reports on GitHub.  
  
One strength from a methodological standpoint is that the authors did a fair of manual analysis to try to understand trends, verify results, and to filter out noise.  
  
While not an actual contribution (since the details of how it works and an actual evaluation of it is not reported), the ExceptionMiner tool that automatically parses and extracts stack traces from bug reports seems like it would be useful in many other contexts as well.  I hope that the authors make this available to others.  
  
The authors state that their tool automatically filters out directories that contain the source code of libraries.  While I commend the authors for doing this and I think it improves the validity of the findings, how exactly is this done?  Do they have their own tool or somebody else's.  Is there a citation that should be here?  
  
The findings are interesting.  Some are not too surprising (prevalence of NullPointerException).  However, the findings related to non-checked exceptions being thrown from native code was surprising and apparently I wasn't the only one, as most developers were unaware of this as well.  This is one actionable and insightful takeaway from this paper that I really like.  
  
As one who has used the approach of wrapping exceptions, I found section RQ1.3 to be very interesting.  While the authors showed that some of these violate best practices, it didn't look like the authors really explained which types of wrappings led to which bug hazards.  From a developer standpoint, one of the things I'd like to learn from this paper is what I should be doing and right now many of the RQs, including RQ1.3, say what the authors say (counts, percentages, etc.) but the text is not very proscriptive.  
  
Surveying developers and contrasting findings with their quantitative mining study was a strong part of this paper.  I believe that actually interacting with developers can yield insight and that was the case in this paper.  I'd like to see more motivation behind the questions on the survey, however.  For instance, how is it useful to know "How often do developers throw exceptions"?  That said, I did find the results interesting.  For instance, understanding the practices developers use when handling exceptions gives an idea of what best practices are actually used and which are ignored (for instance, in table 9) as well as what the differences are between Android specific development and general java development.  
  
The paper as it stands feels very much like two papers put together (and that's understandable given the earlier MSR paper).  The survey section does not often refer to the earlier study, but I believe they're complementary and should be more integrated in the text.  The initial study simply gives numbers and tells us what the authors found, that is, what goes on in the source code.  But the  survey results give more insight into how the code got that way and why developers handle exceptions in certain ways.  It would be quite valuable if the authors were able to integrate some of their earlier results into the discussion of the later RQs (RQ2.1 - RQ2.5).  
  
One large takeaway that I get from this paper is that NullPointerExceptions are an issue, but that they really are a very large bucket that essentially "mask" many different types of errors in applications.  This is supported both by the initial MSR-type study and also by the survey of developers.  While I'm not asking for it in this paper, it would be useful to explore these exceptions in more depth, as it feels like this is kind of a catch-all and perhaps there is some help that could be provided to developers if we understood which causes of this exception were more prevalent than others.  
  
I think this paper could be accepted as it is now, as there is nothing wrong with it in my view.  I also believe that the authors could make this a stronger and more cohesive paper by integrating or discussing the results of both studies together.  I'm not suggesting that sections 3 and 4 be merged, but rather that when mining and survey results or questions are highly related, they are discussed together to give a more comprehensive picture.  
  
Minor Issues:  
  
As a minor nit, many times in percentages, commas are used instead of decimal points.  I realize this is often country-specific, but in broadly disseminated scientific writing, a decimal point should be used.  
  
Section 4.3 refers to previous work [16], but isn't that work the work presented at the beginning of this paper?  I realize this journal is an invited paper, but it seems odd to refer to a paper that is a proper subset of this paper.  
  
  
  
  
Reviewer #3: The paper focuses on the exception handling in Android applications. The authors present two studies. The first, already present in their MSR 2015 paper, is a mining study analysing 482 apps mined from GitHub and 157 apps mined from Google code. In particular, the authors mined the issue trackers of such apps to identify those reporting stack traces. This process resulted in the collection of 6,005 stack traces, that have been analysed in conjunction with source code and bytecode to identify bug hazards (i.e., ''circumstances that increase the chance of a bug being present in software''). Findings of this study include, but are not limited to: (i) programming logic and resources (IO, Memory, Battery) management represent 75% of the exceptions that caused the stack trace, (ii) 65% of crashes come from runtime exception, mostly generated in the application layer.  
  
In the second study, that represents the main point of extension over the MSR 2015 paper, the authors surveyed 71 developers with the goal of capturing their perspective about the bug hazards identified in the mining study. The survey qualitatively support and nicely complement the quantitative findings of the mining study. Probably, the most interesting finding from the second study is that just 4% of the surveyed developers are aware of the threats caused by JNI undocumented checked exceptions.  
  
The paper includes a background section explaining how exception handling works in Java, an excellent related work section, and it provides a publicly available replication package.  
  
  
I really liked this work. Both studies are solid and interesting and the paper is very well written and easy to follow. As in any paper there are things that can be improved here and there (see my following comments), but perform these fixes should not require more than a minor revision. The comments that I consider more important are tagged with [IMPORTANT].  
  
1. On the heuristic exploited to identify Android apps. The authors identify Android apps in GitHub and Google Code by looking for repositories containing in the title and/or in the description the word ''android''. This heuristic clearly leads to a quite high number of false positives. The authors solved this issue by manually inspecting the results, so there should be no problems in the collected data. However, in future studies the authors might consider a more solid (and yet very simple) heuristic: check if the repository contains the Android Manifest file.  
  
2. The first study is almost copied/pasted from the MSR paper. In this extension it would be beneficial to add and discuss some real examples you identified in the study, so that the reader can get a better view of possible issues one could encounter while handling exceptions in Android. This would be beneficial especially for practitioners.  
  
3. [IMPORTANT] In the mining study, when describing the exception type analysis, the authors explain that they inspect the source code/byte code of the stack traces extracted from the issue tracker. However, it is not clear if they inspect exactly the same version of the app on which the stack trace manifested. In other words, there might be a misalignment between the data extracted from the issue tracker and the source code/byte code analysed to extract the data needed for the study. The authors should explain the process they used to align these two sources. I guess it should be enough to checkout the version of the system closer to (and older than) the date in which the issue reporting the stack trace has been opened. If the authors did not adopt such a process but they just analysed the last version of the system available in the repository, this is something that \*must\* be discussed in the threats to validity section.  
  
4. [IMPORTANT] The response rate of the survey is quite low (3%). This is typical for surveys performed with open source developers. Still, this is a point that pops up in the reader mind while reading the paper and thus, the authors should explain that 3% (and especially, 71 developers involved) is quite inline with similar previous surveys. See e.g.:  
  
Ko et al.: Information needs in collocated software development teams ICSE 2007.  
Hindle et al.: Do topics make sense to managers and developers? Empirical Software Engineering 2014.  
Bavota et al.: The Impact of API Change- and Fault-Proneness on the User Ratings of Android Apps. IEEE Transactions on Software Engineering 2015.  
  
The authors could discuss this point in the threats to validity section.  
  
5. The surveyed developers are contributors of at least one of the GitHub apps the authors analysed in the mining study. I found strange that the authors did not exploit this opportunity for asking their opinion about the specific issues you identified in \*their\* apps. IMHO, this would provide much more qualitative feedback (while of course requiring the customisation of the survey for each specific developer). Can you please justify this choice in the paper?  
  
6. [IMPORTANT] The authors adopt an open coding methodology to classify the open answers provided by the surveyed developers. The description of such a procedure is not detailed enough. Who are the two coders involved in the procedure? Did they analyse each answer independently? If yes, what was their level of agreement?  
  
7. Minor issues:  
- Table 5 is referenced too far in Page 11  
- The font-size in Figure 2 is too small