ICSE REVIEW

Thank you for the complete and in-depth reviews of our work.

Clone detection [R1-R2]:

We mistakenly reported that the similarity threshold was a standard recommendation for clone detection, indeed it is a NICAD recommendation, and we will change it for the final version. That said, to the best of our knowledge, it is the threshold most commonly used in the literature. We discuss the choice of the threshold in the threat section. As pointed out, NICAD only works with complete JAVA, C# and C files, but we spent a significant amount of time writing a txl grammar that accepts changesets helped by TXL creators and maintainers on http://www.txl.ca/forum/.

Comparison random/other approaches [R1-R2]:

To the best of our knowledge, this is the first approach to ever use commit code similarity instead of code or process metrics. It is an entirely new way of detecting risky changes. Consequently, comparing to other approaches will not be accurate. Also, few approaches are cross-project at all because they use code or process metrics and models are not easily adaptable. (see Jaechang et al. "Transfer defect learning." In ICSE 2013). The only reason we compared our approach with a random classifier is to have a baseline and show that we perform better than a simple baseline.

Clarification of the approach [R2-R3]:

We can certainly work on clarifying the approach. Indeed, the motivating example is confusing, and we will change it for the final version. We target any defect and not only libraries misuses. We cluster projects based on their library use because it allows to compute a measure of similarity between projects without actually needing access to the source code.

Threats [R1-R2-R3]:

Choice of the similarity threshold: As for the selection of the similarity measure, indeed, we verify this on a subset of the data, not all the data. We will also mention it in the threats. Testing all threshold for all the dataset is not computationally possible. It takes each run three months to run on 48 VPS running in parallel.

Commitguru accuracy: BIANCA indeed relies on commitguru accuracy. However, commitguru is built upon the SZZ algorithm. Consequently, BIANCA depends on of the accuracy of the SZZ algorithm. The SZZ algorithm has been widely used in the literature, and its accuracy has been repetitively proven.

Performances: We did not report the time required by BIANCA to compute a single commit. As per clarification rules, we cannot present new results. However, we can say that it is the magnitude of a minute.

High rate of defects introducing commits: R3 expresses some skepticism about the rate of defects introducing commits. The rates we report are in line with previous findings (Median = 20%) (Kamei et al. A large-scale empirical study of just-in-time quality assurance. IEEE TSE. https://doi.org/10.1109/TSE.2012.70)

Verifiability [R1-R2-R3]:

We do not have a straightforward reproduction package as our experiments involve virtual machines instrumentation and coordination. However, we are happy to share our data: https://github.com/MathieuNls/bianca-data

Review 1

Significance

This paper presents BIANCA (Bug Insertion ANticipation by Clone Analysis at commit time), a tool for helping developers to detect buggy commits before they reach the main code repository. The tool basically checks the similarity of the committed code with the code of previous commits that were blamed for having introduced a bug. In case they are similar, the developer is warned that the current commit might be buggy, and the tool shows the fix applied to the blamed commit. This is related to work on detecting buggy commits by analyzing committed code metrics and characteristics such as code size and location. The novelty here is in using similarity (across related projects, not just in a single project), instead of metrics, and in also suggesting fixes.

> True

Given the contribution is somewhat incremental, I would expect a rigorous comparison with related approaches.

> To the best of our knowledge, this is the first approach to ever use commit code similarity instead of code or process metrics. It is a completely new way of detecting risky changes. In addition, few approaches are cross-project at all because they use code or process metrics and models are not easily adaptable. (see Nam, Jaechang, Sinno Jialin Pan, and Sunghun Kim. "Transfer defect learning."

In Proceedings of the 2013 International Conference on Software Engineering, pp. 382-391. IEEE Press, 2013.)

The paper, however, only compares BIANCA to a random classifier. So does BIANCA lead to better precision and recall than other approaches? If not, does the advantage of suggesting fixes compensate for the loss?

> BIANCA leads to a better precision and a lower recall. We detect defect that are shared across projects rather than defects at large. Consequently, it is hard to do an apple to apple comparison. In addition, the fact that approaches do not provide fixes is one of the major factor that hinder the adoption of defect prediction tools by practitioners. (see Johnson, B., Song, Y., Murphy-Hill, E., & Bowdidge, R. (2013). Why don't software developers use static analysis tools to find bugs? In Proceedings of the 35th International Conference on Software Engineering (pp. 672–681). IEEE Press. Lewis, C., Lin, Z., Sadowski, C., Zhu, X., Ou, R., & Whitehead Jr., E. J. (2013). Does bug prediction support human developers? findings from a google case study. In Proceedings of the International Conference on Software Engineering (pp. 372–381). IEEE Press. Foss, S. L., & Murphy, G. C. (2015). Do developers respond to code stability warnings? In Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering (pp. 162–170). IBM Corp.)

Is BIANCA more expensive than other approaches?

> What does "expensive" refers to ? Execution time ?

The finding that only 8.6% of the risky commits detected by BIANCA match other commits from the same project is important evidence that this (similarity based) approach only works with access to a number of related projects.

> Yes, contrary to other approaches, we leverage decades of open source systems that have public issues, commits and fixes. Not only will developers be warned if they commited a similar defect in their own history but if someone, in a similar project, commited a similar defect.

I wonder how related the current paper is to Interactive Code Review for Systematic Changes, Tianyi Zhang, Myoungkyu Song, Joseph Pinedo, Miryung Kim, ICSE' 15: Proceedings of 37th IEEE/ACM International Conference on Software Engineering, pages 111-122

> It is a advanced search & replace refactoring tool. It used AST to specify a template against which location of the source code are matched. The goal being to identify places that should have change. Why we maybe could use AST to achieve cross-project defect detection, this paper have a different purpose.

Soundness

The paper compares BIANCA to a random classifier, showing that the used dataset is unbalanced; buggy (blamed) commits are a small part of the analyzed commits. This is used to justify the low recall and F1-measure values, which should have been compared to the obtained by alternative approaches.

> We can compare to single-project detection approach. Comparison does not take into account the fact that BIANCA provides fixes.

I like that the paper mentions assumptions like the existence of a link between commits and bug tracker information, for example indicating commits that fix bugs. Besides that, it would be important to discuss possible limitations caused by developers that commit tangled changes (like changes that correspond to a bug fix and a refactoring). This might lead to false positives. To blame commits for a bug, the paper uses an existing tool, Commitguru. The paper reports that this tool is accurate, but does not make it clear whether the paper that reports its accuracy uses the same sample, nor why this accuracy would also aplpy to the current sample. It is also not clear how much of the lack of accuracy of BIANCA is introduced by Commitguru. This should be discussed as a threat. What if the fix was performed in a series of commits, and just the last one was marked as a bug fix? Is this rare? In case a bug was introduced to a line, which was later refactored (say via extract method), and then fixed, wouldn't Commitguru detect the refactoring as having caused the bug?

> These are all valid points. Commit Guru is, however, based on the SZZ algorithm that is widely used and it assesses these concerns.

I like the chosen sample, and the approach used for selecting the similarity threshold. However, I don't precisely understand how BIANCA compares commits, nor what precisely is the threshold. In lexical approaches for clone detection, this is often the minimum duplicated token sequence size that should be considered a clone. But here I'm not sure what that means. I have the same issue with the comparison of actual fixes and the proposed fixes. How is this done? What if they do not contain the same number of blocks? And if the first block in one is similar to the last in the other?

> Blocks are pretty-printed as shown in table I. A change can contains many blocks. Pretty-Printed blocks are compared to each block known to have introduce a defect in the past. When compared blocks are uneven, they are "aligned" according to their first matching line, then, we count the number of matching lines and compute a matching percentage.

To avoid missing conflicts, the paper does not consider the most recent six months in project history, and assumes that defects not reported within six months are not relevant. This is a threat that should be discussed. The results and conclusions should explicitly refer to "reported defect commits", and warn readers that other

defect commits might not have been reported in the issue system (they might have been noticed early and immediately fixed before someone else noticing). Moreover, other defects might not have been detected.

> This is true. This is a common issue for defect prediction approaches.

The paper concludes that "The relatively low recall is to be expected, since BIANCA considers risky commits that are also in other projects", but that doesn't seem right. It seems that not considering other projects would further reduce recall.

> Yes, it would. However we rely on the common parts between projects and while project sharing dependencies are exposed to common flaws, they still are largely unique. Consequently, it exists a glass ceiling that approaches such as BIANCA will not be able to breach.

I like the authors effort in manually analyzing the results of a few projects, but I couldn't understand "fact that it provides specific features makes it, at first sight, unlikely to share defects with other projects". What if these features use APIs other projects use too? People might make the same mistake in both projects when invoking API methods. Is that the whole motivation behind BIANCA? The comment "After manual analysis of the 169 defect-commits, we were not able to draw any conclusion on why we were not able to achieve better performance. We can only assume that Eclipse's developers are particularly careful about how they use their dependencies and the quality of their code in general. Only 2% (169/7,818) of their commits introduce new defects." also din't make much sense to me. Isn't this a consequence of bugs being of different nature?

> Or that, indeed.

Verifiability

The paper details the used sample and tools used, but I could not find a reference to a replication package.

> True

Presentation quality

The paper is, in large part, well written and easy to read, but a number of parts are quite confusing, as detailed below. Sections III.B and III.C, in particular, are confusing and not much informative; algorithm 1 and the discussions about TXL syntax and Commitguru components and metrics add little. I like the clear figures explaining the overall approach.

Here are typos and points for further clarification.

Page 3 "We perform project dependency analysis to identify groups of highly-coupled projects.": So not only that depend on the same libraries but that depend on themselves?

> No, that depend on the same libraries. That are "transitively" coupled to each other via their libraries.

Page 3 "n our case, a node corresponds to a project that is connected to other projects on which it depends.": Not a directed graph?

> Yes, it is a directed graph.

Page 3 "Once the project dependency graph is extracted, we use a clustering algorithm to partition the graph.": Is this done for a single project or for a group of projects in a repository?

> For all the projects in our dataset.

Page 4 "More precisely, before each commit, we only extract the blocks belonging to the modified parts of the source code.": What does this actually mean?

> We only treat the code that changed inside a file, not the whole file.

Page 5 "The pre-commit hook runs before the developer specifies a commit message. It is used to inspect the modifications that are about to be committed.": Doesn't make much sense. In git, for example, the message can be specified at commit time. What would happen to those commits.

> In git, the message can be entered with the `-m` argument (i.e., `git commit -m "message"`). However, the way it works, `git commit` is executed first and pre-commit hook are executed on added files. If the pre-commit hooks fails, the commit as a whole is aborted. Otherwise, the commit is tagged with a SHA1 ID and a message.

Page 5 "This is because lexical and syntactic analysis approaches (alternatives to text-based comparisons) would require a complete program to work, i.e., a program that compiles.": This is imprecise. Both require less that full compilation, since neither demands type checking, for example.

> Indeed, but it has to be syntactically correct. Commits changes are not.

Page 5 "2 clones in terms of modification of literals, identifiers, types, indentation, whitespaces, and comments but also contain added or deleted code statements": I thought considering identifiers would be important for identifying common errors in API usage. Isn't that important here?

> Are you referring to Type2 or Type3 definition ?

Page 6 "Figure 5 shows the project dependency graph.": What does that actually represent? The dependencies inferred from the most recent maven files in all projects?

Exactly.

Page 7 "Because our approach relies on commit pre-hooks to detect risky commit": Because our approach relies on commit pre-hooks to detect risky commits

Fixed

Page 7 "Each time BIANCA classifies a commit as risky, we can check if the risky commit is in the database of defect-introducing commits.": This is confusing. When does Bianca classifies a commit as risky? Isn't it exactly when there is a similar one in the database?

yes.

Page 7 "The difference between this golden database and the database described in Section III-B is that, with this one, we unwind the whole history instead of building the history as it happens.": So one is a subset of the other? I was also confused here.

Sentence removed.

Page 7 "we had to find a way to replay past commits. To do so, we cloned our test subjects, and then created a new branch called BIANCA. When created, this branch is reinitialized at the initial state of the project (the first commit) and each commit can be replayed as they have originally been.": I couldn't see why this is needed

> Say you have commit A, B, C and, D which happens the one after the other. Commits A and D introduce similar defects. When evaluating our approach on commit A, we cannot match it to commit D as it would not exist yet. When evaluating commit D, however, we can match it to A. In addition, the implementation of the approach is a commit pre-hook, if commit are not replayed, then, pre hooks are not executed.

Page 7 "To address this, we did not include the last six months of history. Following similar studies [41], [63–65], if a defect is not reported within six months then it is not considered.": Threat

Agreed.

Page 8 "It is important to note that we do not claim that 37.15% of issues in open-source systems are caused by project dependencies.": Where does that come from?

> Our precision is 37.15%. Since we are analyzing cross-project defects, one could jump to the conclusion that in any system, 37.15% are directly cause by misuses of dependencies.

Page 8 "common appraoch to ground detection results is to compare it to a simple baseline.": common approach to ground detection results is to compare it to a simple baseline.

Fixed

Page 8 "It is worth mentioning is that the random classifier": It is worth mentioning that the random classifier

Fixed

Page 8 "Meaning that the proposed fixed must be at least 35% similar to the actual fix. On average, the proposed fixes are above the alpha=35% threshold.": An example, and a counter example, would help

Type 3 clone detecting via NICAD is performed on the proposed fix. If the fix is at least 35% similar to the actual fix, we consider it to be helpful.

Page 10 "Eclipse Che is part of the Eclipse IDE ttha provides": ?

Fixed

Review 2

The paper presents to classify fresh commits as being risky at commit time. The approach works on building a history over past and current commits from which the buggy commits are extracted and used as a database. Fresh commits are then compared against the database by finding similar commits via clone detection. Moreover, the projects are being clustered based on project dependences. The approach is evaluated on 42 projects and shows high precision but low recall.The overall idea is interesting and builds upon previous work to ensure software quality at commit time.

True

However, despite being an easy read, the paper is badly presented. The authors present their approach in an implementation-driven way in which they describe what they have done step-by-step. They don't discuss what the overall aims are, what they want to achieve, why they are doing it, and, most importantly, how everything fits together on a high level. Instead, they give a lot of technical detail that is not needed, for example, explaining how NiCad and TXL works. Why this is a problem can easily be seen by the last phase, the recommendation of a fix. Only in the later pages, the authors claim the benefit that their approach recommends fixes. However, this is not discussed at all in the paper when the overall approach is presented, except of appearance in a figure.

The approach presentation could be more theoretical and less implementation-oriented, I agree. The advantage of providing a fix could be better stressed early on, I agree.

The authors also don't describe how their approach will be used in practice and how the feedback will be provided. They only present their experiment that

analyses a repository by replaying the comment history. Such an experiment can only be seen as an evaluation if the approach may be successful in a tool, but the authors seem to not have the actual implementation in a tool.

> This contradicts previous comment. The way the paper is written is to show that it would be successful in a tool. Yet, this is a research paper, not a tool demonstration.

The choice of NiCad for a clone detector raises the question of how it has been used in their experiment. The paper states that the commits are only expanded to the block containing the changes. However, as far as the reviewer knows, NiCad can only work with complete Java files.

> We have written a changeset syntax and Algorithm 1 explain how we make NICAD work with uncomplete Java file.

The paper states that a similarity threshold of 30% is considered an adequate threshold of similarity of clone detection. This is not true, as the 30% is actually only a recommended configuration of NiCad.

> True.

Except for the threshold setting, the authors don't explain the configuration they used. Any approach using similarity is largely affected by the used tool and its configuration, which needs to be discussed [A].

> We used constant renaming, normalized conditions and, pretty-printing in addition of the 35% threshold.

The way the threshold is chosen is causing a huge threat to validity. The threshold is chosen after-the-fact, i.e. the authors analyse (a subset of) the projects to establish the optimal setting. This is of course not a valid approach. The author need to establish the threshold on a completely different set of projects. Setting the threshold is a known problem [B].

> Testing all threshold for all the dataset is not computationally possible. It take each run three months to run on 48 Amazon VPS (Virtual Private Server) running in parallel (4e8 comparisons). Then, where do you stop testing (i.e., 1.0%, 1.01%, 1.01%, 1.001%?) ?

The evaluation does not show how long it takes for one commit to be classified. The authors only state the overall time of the evaluation. However, one needs to know how long a developer has to wait for feedback from the system.

> Indeed, it takes, in average 57 seconds for each commit. This time is heavily impacted by the machines running the comparison (SSD, core frequency).

The discussion of the manual analysis is insufficient. For example, for Otto, the authors explicitly highlight that 5 of the fixes are from other projects, but they don't discuss the 5 fixes.

No space to do so.

[A] T. Wang, M. Harman, Y. Jia, J. Krinke: Searching for Better Configurations: A Rigorous Approach to Clone Evaluation. ESEC/FSE, Saint Petersburg, Russia, August 2013.

[B] C. Ragkhitwetsagul, J. Krinke, D. Clark: Similarity of Source Code in the Presence of Pervasive Modifications. 16th International Working Conference on Source Code Analysis and Manipulation (SCAM), Raleigh, NC, USA, October 2016.

Soundness

The approach suffers from choosing the similarity threshold after-the-fact, i.e. it choses the threshold that produces the best results.

Discussed earlier

Significance

The approach can be interesting to identify risky commits, however, the experiment is only a first step to an actual tool.

Discussed earlier

Verifiability

The paper does not mention the availability of a tool or the data set they used.

It doesn't

Presentation

- The paper describes the implementation of the approach without giving motivation, aims, and reason.

Review 3

This paper presents BIANCA, an approach to detect potentially problematic commits before they are uploaded to the version control system. Furthermore, BIANCA can also suggest potential fixes for the problematic issues. The technique uses code clone detection techniques to identify code snippets that are similar to the commits and related to bug fixes. These snippets originate from projects with

similar characteristics. BIANCA was evaluated on a set of 42 github projects, each with at least 2K followers. Results indicate that BIANCA has high precision (~90.7%) and relatively low recall (~37.1%), being effective for pinpointing some (not all) problematic issues at commit time.

> True

This paper presents a simple idea to prevent the introduction of recurrent issues within and across projects. The strong part of the paper is the idea of relating commit information across projects to prevent problematic fixes. Although the idea is not completely new (reuse of code across project has been previously investigated in automated program repair, for example), I like the simplicity of the technique and I think the general approach is promising. My main objections are described in the following.

> Ok

Soundness

The paper lacks evidence on why the approach should work. At the beginning I expected that library misuses were the (main/only?) source of problematic commits the paper would focus. Not only a motivating example was given about that (see reference to JUNG in the first page of the paper) but the technique clusters projects based on library usage. I expected the paper would provide such explanation. During evaluation, Section V.C, the role of libraries became even more unclear. I did not know, for example, if similar buggy snippets originate from (a) projects in the same organization, (b) projects that offer similar features, or (c) projects that contain similar dependencies.

> a. No
> b. Assumed because of their dependencies
> c. yes

Again, I expected that the paper would focus on case (c) for the reasons mentioned above but I saw references to cases (a) and (b) and no references to (c) in the discussion from Section V.C. If library misuse is not the focus then why clustering is done per library? Is it just a proxy to group somewhat similar projects? If so, what is the impact of not using clustering? To sum, it is important to understand why the technique works . Answering questions above can help; providing concrete examples (/characterizing complexity of bug fixes can also help) can also help.

> The clustering is indeed a proxy. Not using clustering decrease the recall while not improving the precision significantly. Also, without clustering, there is even more commit to analyze.

Evaluation

- I strongly suggest to make results public (or explain why it is not), including the list of pairs of blaming commits (most of which relating different projects).

> The clone detection output weigh hundreds of gigabytes. I could provide the list of pairs.

- Please explain the rate "Bug Introducing Commit"/"#Commits". I could not find an explanation for these column, but I understand that "Bug Introducing Commit" is the number of commits found to be buggy (to introduce a bug), which is found using the process described in the paper to recover the link between the issue tracker and the version control system. My concern is that this rate seems way too high in my understanding. For example, according to this interpretation, ~26% of the commits are buggy in PrestoDb, ~21% in Square.dagger, ~19% in Google.auto. These number raised some skepticism.

> The interpretation is correct. These are the numbers provided by the SZZ algorithm.

Minor

I encourage authors to clarify why code clone detection was used as opposed to standard text similarity/information retrieval techniques (e.g., tfidf + cosine similarity). It is not clear, particularly, how the technique captures clones of type 3.

> tfidf + cosine similarity are pure text, they do not account for the structured aspect of code. Type 3 detection do (i.e. literals, identifiers, types and so on).

Significance

This paper is of practical relevance provided that results are correct, the bugs prevented are relevant, and experiments are generalizable. Unfortunately, given the material provided, I am a bit skeptical on these counts.

> ...

Verifiability

I could not find public material.

> true

Presentation Quality:

Section 3 needs clarification as to explain design decisions. For example, why BIANCA uses clustering? what would be the result if it did not use. why to use clone detection as opposed to typical IR approaches? BTW., if authors agree that these are valid questions, I encourage to evaluate these choices (i.e., run experiments w/wo them).

Discussed earlier

I was confused with Figures 4 and Figure 5. In Figure 4 nodes are projects and links indicate dependency as per co-occurrence of libraries these projects use. Figure 5, however, shows a number of nodes much higher compared to the number of projects analyzed in the experiment; it should be 42 (projects) instead of 592 (number of nodes in the figure).

It's 42 projects + their dependencies.

It is a bit awkward to present a single research question (beginning of section V) and to not show exactly where it is answered.

Agreed