View Letter

Close

Date: Jul 11, 2014
To: "Daniel Krutz" dan7800@yahoo.com
From: "Empirical Software Engineering" allan.nebres@springer.com
Subject: Decision on your Manuscript #EMSE-D-14-00063

Dear Dr. Daniel Krutz:

We have received the reports from our advisors on your manuscript, "Examining the Effectiveness of Using Concolic Analysis to Detect Code Clones".

With regret, I must inform you that, based on the advice received, the Editor-in-Chief has decided that your manuscript cannot be accepted for publication in Empirical Software Engineering.

Attached, please find the reviewer comments for your perusal.

I would like to thank you very much for forwarding your manuscript to us for consideration and wish you every success in finding an alternative place of publication.

Best regards,

The Editorial Office
Empirical Software Engineering

Comments for the Author:

****

Associate Editor's Comment:

The paper proposes an interesting approach to detect various types of code clone. However, the paper contains a lot of issues and concerns as reviewer pointed out, so we cannot accept this paper for publication in Empirical Software Engineering. I hope the authors will find the reviewers' comments useful, and that it will help them in their future work.

Reviewer #1: Summary:
This paper proposes a new clone detection technique. The proposed technique detects all the four types of clones by using concolic analysis. The authors have implemented two prototypes based on the proposed technique. One is for C language, and the other is for Java. Two experiments were performed to evaluate the effectiveness of the proposed technique. The first one was conducted for checking what types of clones the proposed technique is effective at detecting. Two datasets provided by Krawitz and Roy et al. were used in this evaluation. The second one was conducted for comparing the proposed technique to some existing techniques. In this evaluation, precision, recall, f-score, and accuracy were computed and compared.

Evaluation:
I understand that the authors had a significant effort to develop two prototypes and build a new dataset for the second experiment. However,

there are serious problems in this paper, so that I cannot recommend this paper for publication.

Please see the following comments and consider revising the paper based on them.

+ A new clone detection technique is proposed.
- Comparing tools that detect clones at different levels.
- The number of correct clones used in the first experiment is too small.
- The dataset used in the second experiment seems problematic.

I think the comparison between your tools and the existing tools is problematic in the experiments. Your tool detects clones at method or function level. But, some of the compared tools detect clones at different levels. For example, Simian detects line-based code clones and Nicad detects clones at block level. Why did you compare tools that detect clones at different levels?

The authors used two datasets of clone references provided by Krawitz and Roy et al. But, there are only 20 clones in total. I think the number 20 is too small to compare and discuss the capabilities of clone detection tools. Please consider using Bellon's dataset for comparison. It does not include Type-4 clones, but it includes hundreds of Type-1, Type-2, and Type-3 clones.

The authors created a new clone dataset by manual investigation on tool's detection result. The 45,109 clones were the base of the manual investigation. I wonder how the clones were detected. Were all the clones detected by your tool? If so, comparison between your tool and the existing tools is not fair because all the correct clones came from the your tool. Please write how you created the new dataset clearly.

There are also relatively small issues.

- Figure 1 is hard to understand. The first rectangle labeled "Source Code" means a product, but the fourth rectangle labeled "Clone Candidate Identification" means a process. There are two different meanings represented by rectangles.

- If my understanding is not wrong, this paper revealed that clone detections with concolic analysis is infeasible for Java software.

- In Subsection 3.2, there is a description that the authors made two logistic regression models, but I was not able to understand how they had been used. There seems to be no sentence that explains how the models were used.

- In P.9, I was not able to catch what you mean in the last sentence (Based on the anemic results...further analysis.) in a paragraph.

Small comments:
- In p.2, there is a grammatical error at line 35.
- In my humble opinion, I do not agree that type-4 clone is a kind of code clone. Type-4 clones have the same functionality but they have different syntax. They are not duplicated code.
- In p.14, the last line "scores of 0-40 with 5 points increments" should be "with 10 points"? Figures 2 and 3 have the labels with 10 points increment.

Reviewer #2: Concolic testing automates test input generation by combining the concrete and symbolic (concolic) execution of the code under test. In this paper, the authors examine their tool CCCD that uses concolic analysis to detect clones of the first four clone types. This is a new application towards clone detection, and the authors should be commended for their efforts toward improved detection of type 4 clones. The authors evaluate their tool against existing tools to demonstrate how it improves the state of clone detection. However, there are significant issues and anomalies in their tool evaluation. Their results also reveal problems in their technique that limit its usefulness to clone detection. The major problems are as follows: suspect quality of their two benchmarks, the selection and configuration of their tools, and a non-standard recall and precision analysis. This review begins with comments on these trouble areas, and then overviews problems in other areas of the paper per section. I hope that the authors find these reviews helpful for improving the paper.

Benchmark #1 - Preliminary Study:
The benchmark used in the preliminary study is not convincing. It contains 20-30 clones per language, and those clones are based upon examples found in clone literature. These examples are used to demonstrate the difference between clone types, and are very concise. To benchmark you need to use clones that are convincingly real. A small sample of completely contrived clones are not sufficient to evaluate tools, at least now a days. Yet you eliminated two tools from your next study based on this preliminary study. The tools were executed for their defaults, which may prefer larger clones, which may be why some of the tools perform very poor for these short example clones from literature.

Benchmark #2 - Full Evaluation:
For your full evaluation, you use a clone oracle you developed and published in MSR 2014. This was a very interesting work that suggests using more judges for clone oracling. However, reviewing your previous work, I see some limitations that show that it might not be enough to show the effectiveness of your new tool. Also there are some contradictions in your current paper with respect to your previous publication as well.

To create your oracle, you selected 3-6 classes from three C systems and oracle a sample of the function pairs using 2 judges. In total you identified 66 clone pairs, including 43 type 2 clones, 13 type 3, and 9 type 4, but no type 1. These are perhaps too few clones to base a benchmark on, and there is no type 1 representation. Also, the clones between such a small set of classes likely has very little variety. In comparison to previous work, Bellon's benchmark contains 4319 clones (499 type 1, 2946 type 2, 874 type 3), and there is still concern in the community that Bellon oracle contains too few clones. Also there seem to be difference between your published benchmark and the one you used in this paper?

The challenges of creating a clone benchmark is to come up with a context of "what is a clone". There is much disagreement in this regard, but to ensure consistent oracling and to provide some interpretation of the benchmark you need a clear definition. However, your paper does not seem to address this obstacle. Consider, after identifier normalization, two fragments can be syntactically identical. But their original fragments may be completely unrelated from a functional, refactoring, bug detection context. Is this still a clone? Also you don't seem to address the major challenge of type 4 clones. When is a clone type 4 instead of type 3? You define type 4 in the data paper as "the same output for the same input". Does the syntactic types take precedence over type 4 classification? (Type 1 clones have the same input/output). If yes, at what point are two code fragments no longer syntactically similar (type 3)? By not addressing these major obstacles in clone oracling, your results perhaps do not have high confidence.

Another issue, and a contradiction between the papers, is in the data paper you state "results from various tools were used by the researchers to assist with the decision making process". This is a major threat to your data. The tools use metrics to guess if something is a clone, they are not an authority on what a true clone is. Yet in this paper you state "the oracle was created before any clone detection efforts to reduce bias during manual analysis", this is clearly a contradiction.

In the introduction you state that MeCC is capable of reliably detection type 4 clones. Yet this contradicts your findings that MeCC fails to detect any of the type 4 clones in your benchmark. This makes it look like there is a problem in your benchmark. Or there is a problem in MeCC. You need to investigate this further. Similarly for NiCad, you noted that NiCad was not able to detect one of the example (Table 3) type 3 clones. Did you investigate why it didn't? I used NiCad for a couple of times for my own study and I think I know the problem. Looks like you didn't use NiCad properly. NiCad uses lots of normalizations and source transformation of the source before comparison. Have you really applied that? At least the blind renaming? Also by default NiCad's min clone size is set to 10 pretty-printed lines. A function below 10 lines in pretty-printed format, by default NiCad will ignore that from comparison. Depending on the size of the clone pairs of your benchmark, NiCad could miss all of the clones if you don't use the features of NiCad properly.

Non-Standard Recall and Precision Measurement:
You measure recall and precision with respect to the detection of "defect prone" and "not defect prone" files. This is not a standard definition of recall and precision in clone detection. Recall is the ratio of the clone

pairs in a system that a tool is able to detect, and precision is the ratio of the clone pairs a tool reports that are true clones not false positives. You cite Zibran et al. in this section, but follow a completely different measure.

The process you use sounds complex, involving "multivariate logistic regression models", "all the metrics", "smaller set of statistically and minimally collinear metrics". You have a complex measure but you do not describe what you did well. It sounds like your measurement comes from a different domain of study. You either need to use the standard definitions, or defend why your new approach is a better measure. Clone detectors report clones. Clones are bad smells, and can be or can lead to defects. Clone detectors do not identify "defect prone" and "not defect prone" files, and should not be evaluated as such.

Tool Selection:
You motivate your new detection algorithm as being able to detect all four clone types. You demonstrate this by comparing it to other tools. However, you evaluate very few tools. Rattan et al. found that there are at least 70 tools in publication. Quite a few are easy to find and execute, including: CCFinderX, ConQat, CPD, CtCompare, Deckard, Duplo, iClones, Scorpio, Simian, SeByte. Your evaluation needs to consider more tools to be convincing.

Also, you try to show the unique need of your tool as a type 4 detector. But you miss other type 4 detectors to evaluate against, such as SeByte (Java-bytecode) and Scorpio (Java). These could have participated in at least your preliminary study. Also they should be mentioned in related work, as well as any other type 4 capable tools.

You initially begin with 6 tools, but eliminate 2 after the preliminary study for poor performance.   Given the limits of the preliminary study, I believe it was premature to eliminate them. As a reader of the paper, it sounds more like you were trying to avoid efforts. The tools you kept can be executed at the function granularity (same as the benchmark). The tools you dropped required you to handle the mismatch in granularity.

Tool Configuration:
The versions of the clone detectors used and their configurations are not clearly stated in the paper. In the preliminary study you say you use their default settings. It has been shown that benchmarking clone detectors using their default settings is not appropriate. See "Searching for better configurations" by Wang et al. Default settings are often optimized for demonstration use or for a first pass of clone detection. Namely, fast detection time, small reports. Clone detectors need to be configured for their target, including enabling normalizations.

In your full evaluation, you mention that you optimized the tools clone size and similarity. The results of this is only mentioned for NiCad. You mention you cannot find clone size configuration in NiCad, but you set similarity to 50%. The clone size is in the same configuration file, and a similarity of 50% is likely to hurt NiCad's precision?   Did you enable NiCad's transformations? This is likely why you needed such a high similarity. This incorrect configuration of NiCad and no acknowledgement of the configuration of the other tools, suggests that you possibly did not thoroughly explore their operation and configurations. As I noted I used NiCad and some other tools you listed here before and my experience is totally different. I really suggest you to look at the tools' configurations carefully and play with them. For example, NiCad's precision could have been much better if you would have used those normalizations and higher similarity values. Similarly, for the size parameters NiCad is missing lots of clones. It is by default 10 LOC. I guess you should set this to 6 LOC or so depending on the size of the clones of your benchmark. Did you really explore which clones these tools are missing and why? At least why didn't you feel the importance of exploring why MeCC couldn't detect a single clone of your benchmark? What message are you trying to convey with this? Similarly, you noted NiCad was able to detect one of the Type 4 clones? Did you explore why? I think it was because, you set 50% threshold values where many of the fragments might look similar. This will drastically harm NiCad's precision.

Introduction:
The definitions of clone types are not quite standard. Type 2 clones often allow differences in literals as well, see papers by Bellon et al. and Roy et al. It is perhaps more clear to state that Type 3 clones can have statements added or removed with respect to one another (although this is identical depending on the direction of edit considered). Type 4 clones are usually more vaguely defined. Including fragments that are syntactically dissimilar, but implement similar logic/semantics. Here your definition is very specific: produce identical results when executed. This is perhaps a subtype of type 4 (a sort of type 1 of semantics).

Related Work:
This section is very minimal. You discuss MeCC but not the other participating tools. CCFinder is historically

a very popular clone detection tool. You state that you do not evaluate t because it "no longer appears to be freely available". CCFinder is freely available and open-source (http://www.ccfinder.net/index.html). It is surprising that you missed this. You criticize Tempero for only considering Java-based systems, but your primary benchmark only considers C systems. Why is Tempero's work lower confidence compared to your work?

You missed many related works with respect to clone detector benchmarking. Including Bellon et al.'s benchmark, which has been the de-factor benchmark in clone detection, as well as Higo et al.'s extension of this benchmark, and Baker's comments on the benchmark. Also Svajlenko et al.'s Mutation Framework, Burd and Biley's benchmark, etc. How can readers be confident in your benchmark if you do not demonstrate a thorough understanding of the previous efforts?

Execution Times:
Here you list the execution times of the tools for various systems. You compare control (a single file) against the tools execution times for "much larger" applications (Postgresql, etc). Did you only execute these only for the files in your benchmark? These execution times are far too short to be for the full systems. It is very misleading how you have presented this data. For example, again, I am confused about the timing for NiCad for example. Even the authors noted that NiCad takes a significant amount of time in detecting clones, as it does the preprocessing and then detection. How could it take just a second for example?

CCCD Limitations:
It seems the major limitation of CCCD is its execution time. However, you only briefly discuss this, and suggest it may be possible to reduce. Clone detection is moving towards large system, inter-project, and big data clone detection. From this paper, it seems CCCD cannot scale past a single file for Java systems, and has long execution times for a handful of C files. If you propose this concolic technique to improve type 4 clone detection, you need to defend that the tool is also practical to be used in real scenarios.

Close