Authors' response for Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt R1

Editor Comments

Editor
Comments to the Author:
The authors are congratulated for addressing many of the reviewers comments. Reviewer #1 and #3 ask for some points of clarification in the final manuscript. Reviewer #2 raises some more substantial concerns. The authors are requested, in particular, to address Reviewer #2's concerns of:

1) how automatic detection through comments can complement other detection approaches, and

2) guidance on how the definitions for requirement and design debt might be better distinguished.

Response:
Thank you. We revised the manuscript to address all of the reviewer's comments, including the two aforementioned points, which were specifically addressed in comments R1-1 and R2-6/R1-2 below. We hope that the revised manuscripts meets the standard for publication in TSE. If there are any further comments or revisions, we would be happy to perform them.

****

Reviewer: 2

(R2-1)2-2: Section 4.4 is a useful addition. On its own it does not convincingly answer the question (is there a link with self-admitted debt and code smells), but it does a good job of pointing the way to further research. Statistically I'm a little dubious about merely associating files with smells against files with debt comments, but this is a good introduction to where to go. For example, it would be better to compare a baseline (% of files with long methods) to the %SATD with long methods. We really want to know the amount above the baseline. For instance, in ArgoUML, you have 419 files with SATD. Of that, 255 have long method smells. What is the % of "all files" with long method, and are the SATD files greatly above that ratio? If so, I would be inclined to think the developers and the smell information are agreeing on something. (Aside: I think in future you can just point me to the section rather than doing that AND copy/pasting it into the response).

Response:
Thank you for the comment. Indeed, this is a very interesting idea. We added Table 7, which contains detailed information about the total number of files and the total number of code smell detected to contrast with the self-admitted technical debt shown in Table 8. As can be observed, the overlap between SATD and the code smells is higher than the ratio of files that have code smells, hence they do agree. We also modified section 4.4 to discuss the revised results.

(R2-2)2:6 I see how you do this now. I guess I had thought of it as a classifier that randomly labeled a comment as SATD or not, which produces a result you then evaluate for P/R against the known (manual) labels. Instead you are saying the lower bound is this worst-case.

I would reword the sentence on page 8, col 1, line 5 as "The probability of randomly labeling a comment as

SATD". I also think "Performance Lower Bound" makes me think of labeling speed. Maybe reword this as "F1 Lower Bound". THat more accurately captures that this measure is about the worst a classifier could possibly do.

Response:
Thank you for the comment to modify some of our wording. We made the suggested modifications in Section 3 of the paper.

(R2-3)2:12 I think this is much clearer, but it would be useful to make mention of the cross-project implications. The example I am thinking of is Gelman's radon example from http://www.stat.columbia.edu/~gelman/research/published/multi2.pdf. The idea is that houses give off radon (analog to "SATD"), but the data is spotty and distributed differently throughout counties (analog to "project"). Fitting a MLM allows Gelman to weight each county by the number of data points it has, while retaining the broader characteristics of country wide radon information (the prior). SOmehting similar would be useful here, so that your population wide indicators could be used as priors for the project specific labeling, without biasing it to the entire set of 10. Anyway, future work.

Response:
Thank you for sharing this idea. It is interesting to explore this in the future.

(R2-4)2:14 Thanks for the discussion on comment relevance. It does seem like they track reasonably well with code changes. Could you squeeze this last reference to Potdar in? I think it would be nice for readers to see. The construct validity discussion is much improved.

Response:
Thank you. The revised manuscript contains references to the Potdar and Flurri's study in the Threats to Validity section.

(R2-5)p10 line 40 calls it "none TD" which should be "non TD"

Response:
Thanks. Fixed.

(R2-6)I would still like it if you could provide the 1 hr tutorial provided to the raters on how to separate design from requirements on the Github site, as well as (ideally!) the filtering criteria used (e.g. regular expressions). This would help replication, as well as understand what might be omitted.

Response:
Thank you for your comment. We added the tutorial and the regular expression used in our heuristic in our public Github repository created to host supplementary information about our work. The files can be located on https://github.com/maldonado/tse_satd_data/ under the folder replication and understanding. We added a link to the online tutorial in section 2.4 of the paper (ref [30] in the paper).

Reviewer: 1

Public Comments (these will be made available to the author)
The authors have done a diligent work revising the manuscript. They have addressed my comments to a large extent. However there are still a few points that need attention, especially the notion of requirements debt and its relation to design debt.

(R1-1)The proposition that self-admitted TD is useful to prioritize the pay-back because of its relevance to the developers does not really make sense. There may be several reasons why self-admitted TD should not receive higher priority than other types, e.g. developers may prefer to comment on lightweight types of TD and leave out the more serious cases, the inadvertent TD may be more critical to pay back, or the relevance between different developers may differ substantially. I would urge the authors to re-think how this automatic detection through comments can complement other detection approaches. Why would a practitioner want to use your approach when she is accustomed to e.g. Sonarcube analysis?

Response:
Thank you. We understand the concerns of the reviewer, but we would like to support our argument with further evidence.
"*In a recent survey [24] with 152 developers of a large financial organization (ING Netherlands), 88% of the participants responded that they annotate poor implementation choices (i.e., design technical debt) with comments in the source code (i.e., self-admitted technical debt), and when time allows, they act on them by trying to refactor such smells using some automated tool support (71%), or manually (29%).*"

C. Vassallo, F. Zampetti, D. Romano, M. Beller, A. Panichella, M. D. Penta, and A. Zaidman, "Continuous delivery practices in a large financial organization," in Proceedings of the 32nd International Conference on Software Maintenance and Evolution, ser. ICSME'16, 2016.

The fact that the developers act on their self-admitted technical debt, when time allows, is an indication that they really care about paying back their technical debt, and their comments serve as a reminder to do so. Therefore, prioritizing the pay back of technical debt based on self-admitted technical debt makes a lot of sense, and seems to be a very practical approach. To address the reviewer's comment, we added the above paragraph to the introduction section of the paper.

(R1-2)The definitions you give for requirement and design debt do not provide an unambiguous way of distinguishing between the two. The architecture/requirements community has long struggled with the dichotomy between the two concepts and what makes something a requirement instead of a design decision (see for example de Boer, R. C., & van Vliet, H. (2009). Controversy Corner: On the similarity between requirements and architecture. J. Syst. Softw., 82(3), 544–550. doi:http://dx.doi.org/10.1016/j.jss.2008.11.185). Even in the examples you give, it is not clear how to classify some of the comments. For example "no method for newInstance using a reverse- classloader" could be thought of as design debt, while "is encoding treatment needed here?" could be considered as requirement debt (according to your definitions). I think you need a much crisper way of distinguishing between the two, otherwise this is a serious threat to construct validity.

Response:
Thank you for the comment. It is difficult to be 100% sure that something is unambiguously requirement or design debt. As we showed, even within our two raters, there was some disagreement, although the agreement was excellent overall. That said, we do understand the need to allow for the replication of the study, hence, in addition to providing our data, we also share the tutorial used by the raters as requested by R2. We hope that the tutorial and the publicly available data is enough to allow others to replicate our study.

(R1-3) I appreciate the fact that you changed the definition of requirements debt towards partially implemented requirements. But how do you ensure that the incidents you find with keywords from Table 3 are indeed partially implemented requirements instead of requirements not implemented at all? The first two keywords "to do" and "needed" can certainly apply in both cases. I think you need to manually check this, as it constitutes

another threat to construct validity.

Response:
Thank you for the comment. Indeed, the comments used to come up with these words were manually labeled. Hence, our comments were manually checked by default. We hope this clarifies the concern.

(R1-4) You have included construct validity threats and reliability threats in the same category even though they are completely different categories.

Response:
Thank you. We revisited the threats to validity section and separated the construct and reliability threats.

Reviewer: 3

(R3-1)I have some very minor suggestions. In response to R3-7, the authors are still not clear about whether punctuation is kept with the word tokens ore separated out. I assume from Table 6 that they are not separated, but it may be worth explicitly stating this in the paragraph that talks about normalization (i.e., "hack" "Hack" "HACK") from a replication stand point.

Response:
Thank you for your comment. We agree with the reviewer that this point could be more clear. We added the punctuations that are being removed explicitly in the above mentioned paragraph in section 2.5, fourth paragraph.

(R3-2)Typos:
- 4.2 "can also be interesting in the case THAT those fine-grained classes of debt are not considered necessary..."
- Table 7: fFles -> Files

Response:
Thank you. Fixed.