**Authors’ response**

Editor Comments  
  
Editor  
Comments to the Author:  
The reviewers all find novelty in the work of looking for technical debt in comments. However, there are several areas in which the manuscript requires improvement. In particular, the authors should:  
  
\* provide arguments or justification of why the studies focus on comments and do not validate the technical debt with the source code and whether it exhibits characteristics of technical debt. This point is related to the reviewers comments on choices not to compare to ways to assess technical debt based on source code.  
  
\* clarify how technical debt and requirements debt relate. The authors should carefully consider what information to fold in from the previous workshop paper to enable readers to have a self-contained means of understanding this manuscript.  
  
\* several reviewers commented on finding choices made for the random classifier (such as following the distribution of technical debt on the projects): these require clarification.  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
Reviewers' Comments  
  
Please note that some reviewers may have included additional comments in a separate file. If a review contains the note "see the attached file" under Section III A - Public Comments, you will need to log on to ScholarOne Manuscripts  to view the file. After logging in, select the Author Center, click on the "Manuscripts with Decisions" queue and then clicking on the "view decision letter" link for this manuscript. You must scroll down to the very bottom of the letter to see the file(s), if any.  This will open the file that the reviewer(s) or the Associate Editor included for you along with their review.

**Reviewer: 1**  
  
Public Comments (these will be made available to the author)  
The paper presents an approach to detect technical debt automatically from source code comments based on NLP. The approach is used on a number of open source systems to train the NLP classifier and provide validation. The authors also discuss the prominent words used to indicate design and requirements technical debt and argue that their approach only needs a small training set.  
The paper is easy to read and understand, written in a concise and clear manner. It tackles an important problem and reports on a well-design empirical study that provides sound evidence. However there are several points that need to be revised regarding the **motivation for the work**, the **design of the approach** and the **empirical study**. Detailed comments per section follow.

Introduction   
• (R1-1) Technical debt is not always incurred consciously. Martin Fowler was first to point this out.

Thank you for the comment. Indeed, we agree with the reviewer that technical debt can be incurred consciously and not consciously. Clearly, in this work we focus on the self-admitted technical debt, which would fall under the technical debt that is consciously (or deliberately as Fowler calls it) incurred. We clarified this point in the paper and added the following text to the manuscript in the Introduction, third paragraph:

“Technical debt can be deliberately or inadvertently incurred [5]. Inadvertent technical debt is technical debt that is taken on unknowingly. One example of inadvertent technical debt is architectural decay or architectural drift. To date, the majority of the technical debt work has focused on inadvertent technical debt [6]. On the other hand, deliberate technical debt, is debt that is incurred by the developer with knowledge that it is being taken on. One example of such deliberate technical debt, is self-admitted technical debt, which is the focus of our paper.”

As part of this change, we also added the following citations:

* [5]  “M. Fowler. Technical debt quadrant.”   
  http://martinfowler.com/bliki/TechnicalDebtQuadrant.html,   
  accessed: 2016-06-09.
* [6]  R. L. Nord, I. Ozkaya, P. Kruchten, and M. Gonzalez-Rojas, “In search of a metric for managing architectural technical debt,” in *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*, 2012, pp. 91–100.

• (R1-2) Static analysis does help in detecting debt and is the more popular approach but there are also other techniques proposed that you seem to completely ignore. I know you want to compare your approach with techniques that focus on source code but you cannot ignore other ways of detecting debt, like architecture reviews.

Thank you for the comment. We mainly focused on the source code-related technical debt since our approach is related to source code comments. However, as the reviewer points out it is a good idea to also mention other ways that technical debt has been measured in the past. Therefore, we modified the manuscript to add the following text in Introduction, paragraph 4:

“Some of the approaches analyze the source code to detect technical debt, whereas other approaches leveraged various techniques and artifacts, e.g., documentation and architectural reviews, to detect documentation debt, test debt or architecture debt (i.e., unexpected deviance from the initial architecture)[7][8] ”

As part of this change, we also added the following citations:

* [7]  N. Alves, T. Mendes, M. G. de Mendona, R. Spinola, F. Shull, and C. Seaman, “Identification and management of technical debt: A systematic mapping study,” *Information and Software Technology*, vol. 70, pp. 100–121, 2016.
* [8]  L. Xiao, Y. Cai, R. Kazman, R. Mo, and Q. Feng, “Identifying and quantifying architectural debt,” in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 488–498.

• (R1-3) You argue that detection based on source comments is better than source code, but you don’t justify it well. First, I don’t buy your third argument: why exactly is it more reliable to base on comments? Yes, developers write comments themselves but why does it make this approach more reliable? Also the first argument is not clear. So what if source code techniques require an AST? Such tools produce the end result in seconds so what is the problem exactly? In fact I am not at all convinced that detecting based on source comments is better at all – it is at best a complementary way.

Discuss after Nikos’ edits.

Indeed, we should clarify this and not claim that comments is better. We should just say that comments is something explicitly mentioned by the developers. We should focus on the fact that it is a complementary approach.

Approach  
• (R1-4) The approach looks in general sound but the manual classification is a major issue. First, spending 185 hours to perform the manual classification is major effort. Anyone that wants to reuse your approach will have to retrain the classifier and perform this manual classification from scratch for a different domain, language, technology stack. This is a major hindrance towards the applicability of your approach.

Thank you for the comment. Indeed, the manual effort put into the approach is significant, however, the main purpose of the manuscript is to put forward an approach that can be trained using our manually classified dataset so that such manual effort can be reduced or eliminated in the future. As we have shown in our RQ3 results, our NLP classifier performs well, even when tested on a project that it has never been trained on in the past. We also make our dataset publicly available so that others can avoid such manual effort in the future and easily build on our dataset.

As for the domain issue, we specifically chose our case studies to cover different domains. That said, we do not claim that our approach will generalize to all languages, domains, etc; achieving such generalizations is out of the scope of our paper and requires a different study specifically designed for such a purpose. To address this comment and clarify our intention here, we modified/added the following text to Sections 6 (Threats to Validity) and 7 (Conclusions and Future Work):

Section 6:

“To minimize the threat to external validity, we chose open source projects from different domains. That said, our results may not generalize to other open source or commercial projects, projects written in different languages, projects from different domains and/or technology stacks. In particular, our results may not generalize to projects that have a low number or no comments.”

Section 7:

“In addition, we plan to examine the applicability of our approach to more domains (than what we study in this paper) and software projects developed in different programming languages.”

------------------------------------------------------------------------------------------------------------------The manual classification is a significant effort, but the idea is that we have to do it to train the NLP classifier. Once we have enough data to train, we need not have so much manually generated data (this is what RQ3 shows). Now, for a different domain/language/domain, we don’t know if the manual data can apply across domains/languages/technologies or someone needs to regenerate the data (we leave this as future work).

------------------------------------------------------------------------------------------------------------------

• (R1-5) Second, inter-rate agreement measured according to Cohen’s Kappa is 0.81 which is good but not impressive. This should be added to your threats to validity

Thank you for the comment. We made sure to discuss the inter-rater agreement in the threats to validity of the original submission. However, we believe that the reviewer’s comment is related to the value achieved. After the reviewer’s comment, we investigated the characterization of different Cohen’s Kappa values. The literature shows that a Cohen’s Kappa value of < 0.4 as being poor, 0.40 - 0.75 as being fair to good, and > 0.75 as excellent. To address this comment, we modified our original text in Section 6, first paragraph:

“Like any human activity, our manual classification is subject to personal bias. To reduce this bias, we took a statistically significant sample of our classified comments and asked a masters student, who is not an author of the paper, to manually classify them. Then, we calculate the kappa's level of agreement between the classification given by the two different students. The level of agreement obtained was 0.81, which according to Fleiss [41] is characterized as excellent inter-rater agreement.”

As part of this change, we also added the following citation:

[41] J. Fleiss, “The measurement of interrater agreement,” *Statistical methods for rates and proportions.*, pp. 212–236, 1981.

• (R1-6) The selected OSS projects are at most a couple hundred thousands SLOC. Does this pose a threat to the external validity? Would your approach work on larger systems? Also please provide details on the application domains and explain in your threats to validity whether you can generalize to other application domains.

Thank you for the comment. As shown in Table 1, our projects range in size between ~81K - ~228K SLOC. This is comparable to other technical debt-related studies. For example, in [31], the authors study 2 projects that are 35K and 45K LOC, another study [32] used 12 subject systems that range in size between ~25K - ~81K. Therefore, we believe that our studied systems are in line with prior work. As for whether our approach would work on larger systems, we do not see any reason why the approach would not work on larger systems. The real question to consider however, is the quantity and quality of comments in these system, rather than their size. We mention that the quantity and quality of comments is a key component that may affect our approach. The following text in Section 6, paragraph 2:

“When performing our study, we used well-commented Java projects. Since our approach heavily depends on code comments, our results and performance measures may be impacted by the quantity and quality of comments in a software project.”

As for the last part of the comment, we added the domain description for each application in Section 2.1, first paragraph:

“Ant is a build tool written in Java, ArgoUML is an UML modeling tool that includes support for all standard UML 1.4 diagrams, Columba is an email client that has a graphical interface with wizards and internationalization support, EMF is a framework and code generation facility for building tools and other applications, Hibernate is a component providing Object Relational Mapping (ORM) support to applications and other components, JEdit is a text editor written in Java, JFreeChart is a chart library for the Java platform, Jmeter is a Java application designed to load test functional behaviour and measure performance, JRuby is a pure-Java implementation of the Ruby programming language and SQuirrel SQL is a graphical SQL client written in Java.”

Regarding the generalizability of our approach, we believe that this is related to comment R1-4 and has been addressed in that comment.

[31] N.Zazworka,M.A.Shaw,F.Shull,andC.Seaman,“Investigating the impact of design debt on software quality,” in *Proceedings of the 2nd International Workshop on Managing Technical Debt*, 2011, pp. 17–23.

[32] F. Fontana, V. Ferme, and S. Spinelli, “Investigating the impact of code smells debt on quality code evaluation,” in *Proceedings of the 3rd International Workshop on Managing Technical Debt*, 2012, pp. 15–22.

------------------------------------------------------------------------------------------------------------------

No, we cannot say that we apply to larger systems or to other domains. We should mention this as future work and in the threats.

------------------------------------------------------------------------------------------------------------------

Case Study results  
• (R1-7) When formulating RQ1, do you consider that identifying technical debt is the same as predicting it? Also what exactly does effectiveness mean in this case?

Thank you for the comment. Indeed, we agree with the reviewer that identifying and predicting technical debt are distinct things, and should therefore be addressed as such. While predicting technical debt is related to time (using older debt to predict future debt), identifying technical debt is related to the disclosure of the same. Thus, our approach is about identifying self-admitted technical debt. To address the reviewer comment we modified the RQ1 in the manuscript as follow:

“RQ1. Is it possible to effectively identify self-admitted technical debt using NLP techniques?”

As for what we mean by effectiveness in this case is that source code comments are written on natural language and therefore very difficult to be identified by an automated process. As effectiveness defined as the capability of producing a desired result, we expect that our approach will be capable of identifying self-admitted technical debt better than the current approach (comment patterns) does it. To address this comment we added the following clarification on section 3 paragraph 4

“During this study, we built the necessary means to measure and compare the performance of both mentioned approaches. More than just determining if the NLP Classifier can successfully identify self-admitted technical debt we want to determine if our approach can do it effectively. As effectiveness defined as the capability of producing a desired result, we expect that our approach will generate better results than the previous approach to identify self-admitted technical debt.”

------------------------------------------------------------------------------------------------------------------

I am not sure about the predicting it vs. identifying TD. We should add a clear definition of what we mean by effectiveness.

------------------------------------------------------------------------------------------------------------------

• (R1-8) I find the improvement over the comment patterns approach convincing (for design debt), but the comparison with the random approach (for requirements debt) is really not impressive. The F1-measure looks fine without making this comparison. This 18x improvement sounds like an oversell considering you only compare to the random classifier.

Thank you for the comment. Due the fact that self-admitted technical debt entries are very few in the studied dataset the random classifier F1-measure serves us as lower boundary baseline. Our intention when comparing our approach baseline to the random classifier baseline is to show how difficult it is to find one true instance of self-admitted technical debt by chance, and how our approach performance is better than chance. However, as the reviewer points out, the F1-measure looks fine without making this comparison. We modified the manuscript removing the emphasis on the 18x improvement in the conclusion box for RQ1 as follow:

“We find that our NLP-based approach, is more effective in identifying self-admitted technical debt comments compared to the the current state-of-art approaches. We achieved an average F1-measure of 0.620 when identifying design debt (an average improvement of 4.5x and 7.1x compared to other approaches) and an average F1-measure of 0.403 when identifying requirement debt.”

------------------------------------------------------------------------------------------------------------------We should tone down the 18X improvement and mention both, the improvement over random and over comment patterns. We should also compare to the requirement debt, although I am not 100% sure what exactly he means.

------------------------------------------------------------------------------------------------------------------

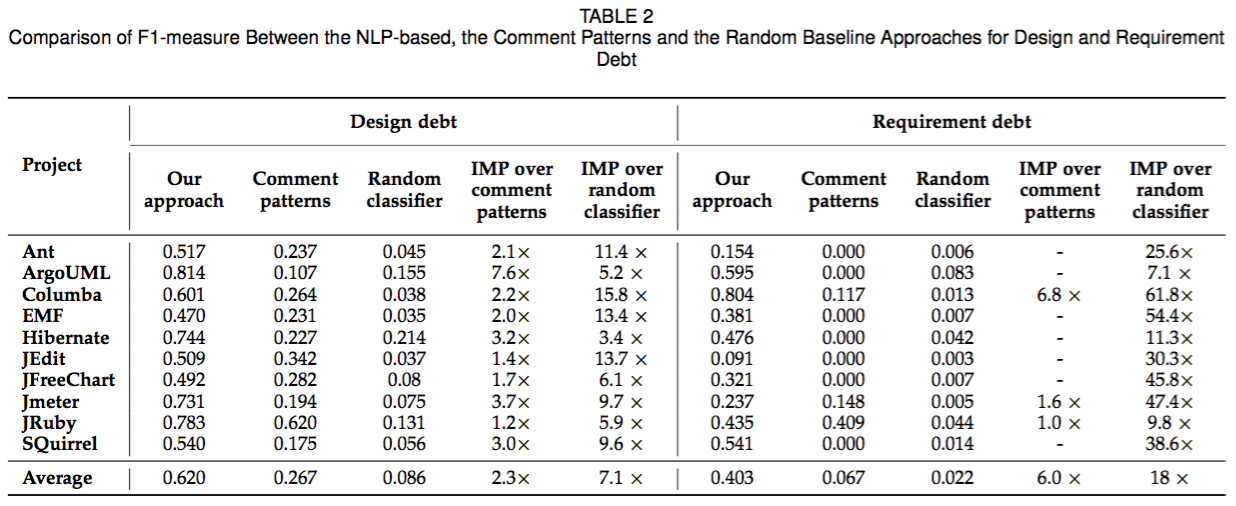
• (R1-9) Also, why wasn’t the comment patterns approach not able to detect any requirements debt?

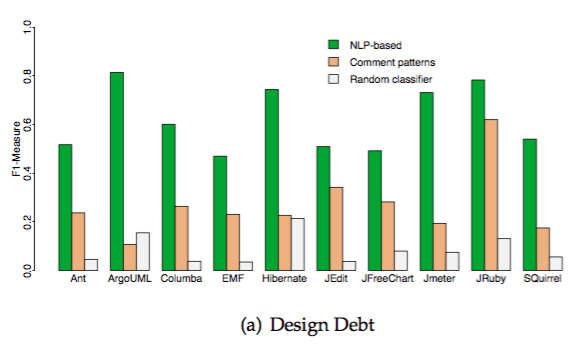
Thank you for the comment. To give a proper answer to this comment we revisited the experiment that was made to measure the performance of the comment patterns approach while identifying requirement technical debt. We found that we should have controlled for putting all comments in lowercase. As this control was left out of our experiment, comment patterns that should identify self-admitted technical debt were failing to do so. We corrected the experiment end analysed the results obtained. We find that the comment patterns approach was able to identify requirement debt on 3 out of 10 analysed projects.

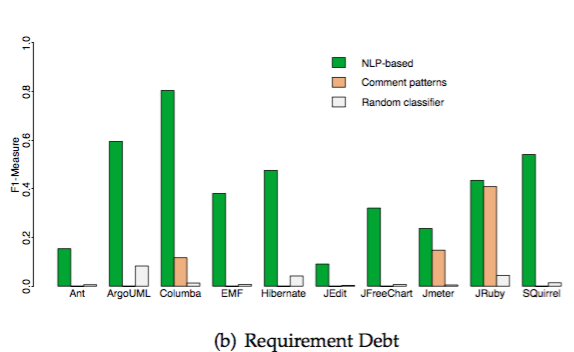
This result is expected as during the development of the comment patterns approach the authors did not differentiate between different kinds of debt, and moreover, they did not devise specific patterns to identify requirement debt. This can be one of the possible reasons that explains why the comment patterns approach is not able to identify requirement debt. To address this comment, we modified our original text in Section 3, paragraph 13:

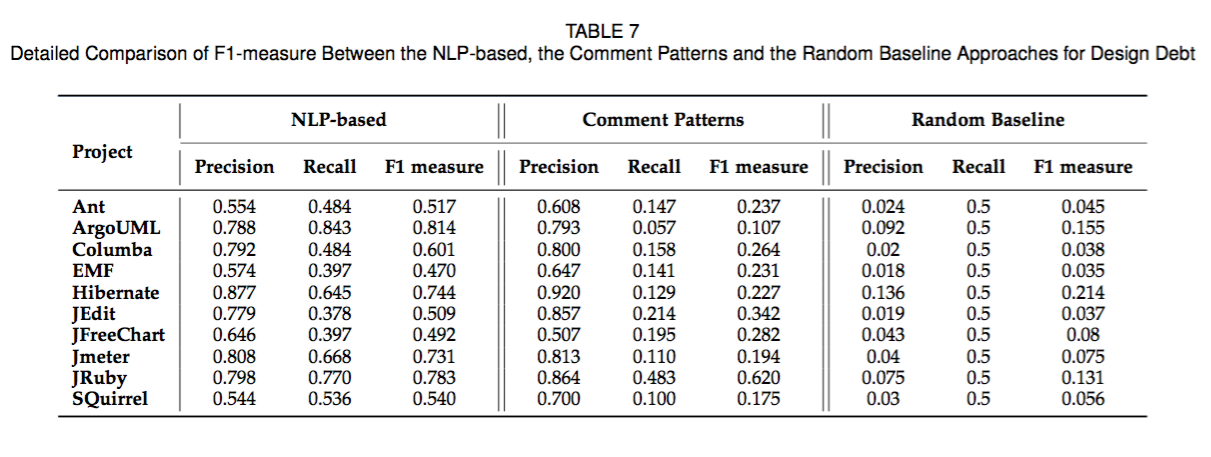
“The comment patterns approach was able to identify requirement self-admitted technical debt only on 3 out of 10 projects that we analyzed. One possible reason for this, is that originally the authors of the comment patterns approach did not differentiate between types of self-admitted technical debt, and therefore, there are not patterns (i.e., words and phrases) specifically depicted for the identification of requirement self-admitted technical debt.”

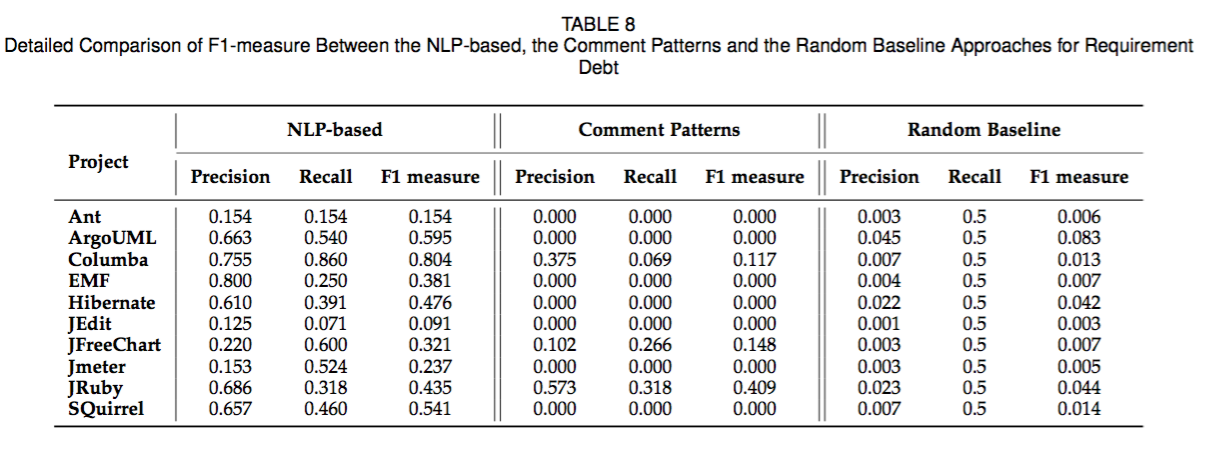
As for the the necessary changes to correct the experiment results presented in our original text we updated the following items: section 3 Table 2, section 3 Figure 2a, section 3 Figure 2b, appendix Tables 7 and 8.











------------------------------------------------------------------------------------------------------------------

We should add the text from the paper that explains this in the response letter.

------------------------------------------------------------------------------------------------------------------

• (R1-10) One thing you do not explain is what you consider as design debt vs. requirement debt. This is a threat to your construct validity.

Thank you for the comment. We explained what we consider design and requirement debt in our previous work, and this information was provided in the original text through a citation [15]. However, we agree with the reviewer that layering the background on self-admitted technical debt, its different types and the criteria that we used when classifying them are of total importance to the reader comprehension. To address this comment, we added the pertinent text from our previous work in section 2.4 paragraphs 3 – 6.

“Below, we define design and requirement self-admitted technical debt that we were able to detect and provide example comments to help the reader grasp the different types of self-admitted technical debt comments.

Self-admitted design debt: These comments indicate that there is a problem with the design of the code. They can be comments about misplaced code, lack of abstraction, long methods, poor implementation, workarounds or a temporary solution. Lets consider the following comments:

“TODO: - This method is too complex, lets break it up'' - [from ArgoUml]

“/\* TODO: really should be a separate class \*/'' - [from ArgoUml]

These comments are clear examples of what we consider as self-admitted design debt. In the above comments, the developers state what needs to be done in order to improve the current design of the code. Although the above comments are easy to understand, during our study we came across more challenging comments that expressed design problems in an indirect way. For example:

“// I hate this so much even before I start writing it. // Re-initialising a global in a place where no-one will see it just // feels wrong. Oh well, here goes.''   
- [from ArgoUml]

“//quick & dirty, to make nested mapped p-sets work:” - [from Apache Ant]

In the above example comments the authors are certain to be implementing code that does not represent the best solution. Intuitively, we know that kind of implementation will degrade the design of the code and should be avoided.

“// probably not the best choice, but it solves the problem of // relative paths in CLASSPATH” - [from Apache Ant]

“//I can't get my head around this; is encoding treatment needed here?”   
- [from Apache Ant]

The above comments expressed doubt and uncertainty when implementing the code and were considered as self-admitted design debt as well.

Self-admitted requirement debt: Requirement debt comments express incompleteness of the method, class or program as observed in the following comments:

“/TODO no methods yet for getClassname''} - [from Apache Ant]

“//TODO no method for newInstance using a reverse-classloader''}   
- [from Apache Ant]

“TODO: The copy function is not yet \* completely implemented - so we will \*   
have some exceptions here and there.\*/” - [from ArgoUml]

The citation mentioned and that was used to add this text is the following:

[15] E. d. S. Maldonado and E. Shihab, “Detecting and quantifying different types of self-admitted technical debt,” in *Proceedings of the 7th International Workshop on Managing Technical Debt*, 2015, pp. 9–15.

------------------------------------------------------------------------------------------------------------------

We should add 1-2 examples of what is design and what is requirement debt (although, I thought we do this already).

------------------------------------------------------------------------------------------------------------------

• (R1-11) If I look at the keywords from RQ2 I am really not sure how you classified requirements debt. The fact that you have the same keyword appearing in both design and requirements debt indicates you may not have a clear distinction. To make matters worse, what you seem to hint is that requirements debt concerns requirements not yet implemented in code. This is in contrast to the orthodox perception on technical debt (see P. Kruchten et al. Technical Debt: From Metaphor to Theory and Practice)

We should contrast our work with the mentioned paper and also clarify that we do this from the code comments perspective, which is different than what the mentioned paper does. It would be even better to find another paper to support our definition of requirement debt.

Discussion  
•       The similarity of terms in requirements and design debt cannot be intuitively confirmed by looking at the top-ten terms from RQ2. There it looks like some terms (e.g. convention, configurable, apparently, fudging) are less similar than those for design debt. Can you explain that?

I am not sure what he is asking here. Perhaps our examples of what is design and what is requirement debt can help provide some intuition for the terms. However, we should make it clear in the paper (RQ2 and threats) that these terms may change for different projects and are there mainly because of the projects we chose and the comments we manually classified into the different categories.

Threats to validity  
•       It seems you have confused internal validity. It concerns causality which you do not study in your paper. The threats you mention, like the bias during manual classification is a threat to construct validity.

Thank you, we fix it.  
•       Please extend your discussion with threats to construct validity and reliability

Yes, we will do that, especially with the things he mentioned above.

Small Details  
•       “this is a dirty hack it’s better do to something” -> to do  
•       “”conjucture”

Fix

**Reviewer: 2**  
Public Comments (these will be made available to the author)  
This paper presents a NLP approach to detecting 'self-admitted technical debt', that is, comments in the code that confess to (presumably) the following code being debt-laden. The authors previously approached this with a word-bag model which identified TD using common terms (like "hack"). In this paper they extend the approach using several NLP techniques to create a classifier from their large gold-standard of manually labeled TD items. They show significant improvement over the previous approach and a random approach to classification. They also present some results showing sensitivity of the classifier to training set size, which is useful for industry applications.  
  
The paper is well written, aside from frequent reference to their earlier work [10].

We should reduce the frequency of citations to [10].

The idea of detecting TD from code comments using NLP tools is novel and empirically demonstrated in the paper; a tool that is publicly available (for eg. in SonarQube) would be interesting to test with developers. However, there is no connection made between the authors' construct of self-admitted TD, with other notions of TD. Therefore, the study is really just a labeling exercise using categories defined in another paper, then the use of an off-the-shelf NLP tool.

We should clarify that the journal paper adds new datasets and also performs an empirical study to show the effectiveness of NLP in detecting SATD. We also share our data, which enables future studies on SATD.

To demonstrate a useful contribution, the work should try to validate the labeling (and subsequent classifier) against either other static analysis tools, or with working developers. That way readers will know whether the classifier is actually detecting TD or not.

Thank you for the suggestion, comparing to a tool such as SonarQube is a good idea (and we should add that). We expected (and as we will see) that these two approaches are complementary.

As for the user study, we do not believe it is necessary since the comments we use comments that developers use. For a user study, we would need to detect something that is not written by the developers so they can verify it (e.g., use metrics to detect TD), but that is not the case here. We should add a small part in the paper to address this.  
  
== Major comments to authors  
  
It would be nice to have something more concrete on the relationship with [10]. For instance the process overview is nearly identical except for the label NLP classification. For historical record, it would be useful to have both studies amalgamated here (which I think is permissible given copyright). The most glaring omission in my view is the criteria by which some comment is classified as requirements vs design debt. I think readers would be curious to know how you distinguish between these two types, and from the other 3 types. (e.g. P25 of this submission). There is certainly some room for debate in how you are classifying them. Perhaps another option is to link to the training manual you provided coder 2 for his/her task.

Reviewer 1 had a similar issue, so we can add the examples and also provide more details on how we classified the two things (provide 2-3 point form points for each type of TD).

Finally, you repeatedly refer to study [10] as 'state-of-the-art' which is true, inasmuch as it represents the only other study to my knowledge approaching the issue of identifying TD through code comments. But the paper would be improved by merging the two papers, in my view, and I'm curious why you chose to divide them.

Sure, we can try to merge some parts of the two.

We initially decided to divide them since one was about the dataset and determining what types of TD are most common, whereas this paper is more about using NLP to detect SATD.  
  
F-measure is the harmonic mean, which implies both P and R are valued equally. HOwever, there are good reasons for thinking this is not the right model for software problems (see e.g. Berry et al REFSQ 2012 "The case for dumb RE tools") and that instead recall should be the target. In this case, one use for the tool is to find code with technical debt. WOuld a dev rather see all the code with TD, at the expense of some more noise, or greatly reduce the noise and miss some actual TD? My instinct tells me the latter.

Can we try to find some related work (and there is tons that interviews developers) to say what they prefer, precision or recall.

I would like to see your view on the subject. In any case a naive 50/50 split like F1 seems incorrect. From Appendix table 7 it seems like your approach (in this paper and the previous one) favor high precision vs high recall. Can you explain why this is desirable? Perhaps the case made in Sadowski's ICSE2014 paper on industrial static analysis tools (now called Shipshape), namely, devs hate the noise.

Yes, we can say this and cite the ICSE2014 work, or see if the Brazilian guys also have some motivation for higher precision.

The random classifier puzzles me. It sounds like you have it randomly bucketing something as TD based on the underlying model you derive from the manual labeling. E.g. if the source dataset had 6% TD, 6% of the time (randomly) this classifier assigns the TD label. But why should the random classifier have to know the underlying distribution? What would you get if you set it to 50%?

No, it would be 50% if the dataset was balanced – but in this case it clearly is not balanced. This is why we actually use the distribution in the data to know the accuracy of the random classifier. We should give an example here, e.g., red vs. blue balls in a bucket. We can also cite other work (preferably not ours) that did this.

What I am saying is that the random classifier in your approach has this prior that in reality a naive classifier wouldn't get. And at 50% I suspect the recall would be much closer to the NLP approach. To be honest the discussion on page 6 was very unclear on how you ran this. I don't see why it needs some particular calculation for precision or recall, since you will simulate it just like a regular classifier, then measure the P/R based on the classification results.

No, it would be 50% if the dataset was balanced – but in this case it clearly is not balanced. This is why we actually use the distribution in the data to know the accuracy of the random classifier. We should give an example here, e.g., red vs. blue balls in a bucket. We can also cite other work (preferably not ours) that did this.

Finally, I expected to see in the results something that looked at TD vs non-TD (that is, no categories). The reasoning for this is that assuming the categories are invalid, even knowing there is TD of some kind would be helpful; it would therefore be interesting to know if performance changes.

Sure, we can add an experiment that does TD vs. non-TD. The reason we initially did the categories since we thought it is the most informative option, but we do agree with the reviewers very good point that it is worthwhile knowing TD vs. non-TD.  
  
I don't understand why in S 2.4, for the coding agreement portion, you selected a random dataset that nearly matched the breakdown of the real world datasets. This results in a very low number of actual true positives (e.g., only 1 comment of the 659 was doc debt). So when we calculate Kappa, my concern is that kappa here is really measuring a comment is "debt or non-debt", and not the specific categories. In other words it does not help us assess whether what coder 1 is calling "requirements debt" is also what coder 2 calls requirements debt. A mitigation here is to include category specific kappa scores, or to use a non-representative sample (I don't get why this is important to the task).

We kept the fraction of the selected data to be the same proportion as the real world dataset since we did not want to bias the outcome of the classification. The reviewer has a good point about the fact that we in essence are comparing whether there is TD or not, and not focusing on the type per se. What we can do is either give the category specific kappa value if we have it or re-do this with a non-representative sample.  
  
Your text frequently says "NLP classifier" (and you mean max entropy classifier). A decision tree would be interesting here, given your RQ2 and the problem of what words are important. I would think the decision tree could present this quite nicely. There are other classifiers to attempt too (and something like Weka would give you access to all of them and let you see which is most suitable). Other SE papers on NLP, e.g. Andi Marcus or Abram Hindle in topic modeling, discuss these extensively. The paper would benefit from presenting other ways of doing the classification. The state of the art in the SE/NLP world is moving to a deeper consideration of these questions - you would benefit from having discussions with the linguistics folks at your institution. (I see later you do discuss 2 others in Section 4.2. But now I see you discounting Naive Bayes although it does better on recall - see above for why this might well be fine). The related work covers some of the applicable work, but I'm not sure just because the subject matter is different - e.g. traceability vs self-admitted debt - that the underlying NLP approach is not still relevant for comparison. Some more discussion of your classifier and why it makes sense in the context of the domain is merited.

Let’s try to add a discussion here of the different underlying classifiers, and more importantly, why they work. I think that using decision trees would be crazy since we will have many branches, but I am not sure. Perhaps we can try that.

We should also re-word the naïve bayes so that it is not “discounted” and highlight its strengths.  
  
Re: the number of comments needed to classify, you say something like 1400 for design and 300 for requirements. Fine. But I notice that per-project the totals you have are much lower for overall TD comments. So are you implying that with only (say) Columba, I would not get a satisfactory F1 because the number of comments is too low? Perhaps you can suggest some mitigation to that, e.g. with transfer learning. Bigger question: what leads you to think a comment that is requirement debt in project A is also requirement debt in project B? My experience is that these projects are very context specific, particularly in requirements debt.

Yes, indeed we believe that some comments are specific and some are more general. The beauty of a NLP-based approach is that it can not only learn words, but also learn from the structure of the comments, hence it can (and as we show does) transfer. We should discuss this in the paper.  
  
The RQ3 is really about active learning. There are in fact tools and approaches in this sub-field entirely concerned with reducing the amount of labeling needed. See <https://en.wikipedia.org/wiki/Active_learning_(machine_learning)>

Should mention this in the related work section at the least.

It seems like you cannot decide whether the dataset should be one giant bag of all 10 projects, or 1 project at a time (which suggests some sort of multilevel regression, but anyway). In evaluating amount of training data needed, I don't understand why you approach this on a per-project basis. Wouldn't it make more sense to either look at the incremental improvement overall, or only on one project? There seems to be a huge assumption that these projects have similar (identical) feature distributions. That seems dangerous; perhaps not in this case, where the projects are all open source, Java projects, but certainly when we broaden the developers involved, I would expect to see a lot of drift (for example, would a German project use words like "hack"?)

Yes, that is a good point and we should mention this in the threats (if we do not do that already). As for the point about per-project, we try to make the scenario as realisitic as possible, therefore, we envision that people will not give one comment but rather classify an entire project and train on it since we need positive and negative instances.

Furthermore, could you be explicit (or rephrase, if I missed it) whether the analysis changes the ordering of the projects? For example, since Ant has a lot of examples, using that first I would expect would be more useful than Columba. This is another argument for merging all the labeled features into one giant bag, and pulling them out comment by comment (or 10/100 whatever).

We do not do that, we use the project with the most comments first and go down the line. We should do an experiment where we put all comments in one line and add 10 or 50 comments per time and see what is the optimal number of comments to train on.  
  
You don't really discuss contruct validity, which I think is key to your work. Namely, does the construct of 'self-admitted TD' match with what is actually TD? In other words, does labeling a comment TD imply that the code following is actually TD (since ultimately managers etc care about the code). You could determine this by either inspecting the code history (to show the comments are still relevant), or by running some of the code smell work against the sections of code you identified to see how much overlap there is (ideally, 100% of the fragments you identify are found by a TD code smell detector, but that won't be the case).

We should mention this in the threats to construct validity, however, we do not expect an overlap since this is self admitted TD. We can do a small experiment to validate this and/or cite the ICSME paper which says that code and comments co-change most of the time.  
  
I think the internal validity is otherwise ok, I appreciated you used another rater. I think my comments on other NLP approaches above also touch on improving internal validity (e.g., recall over precision).  
  
== Minor  
- could you report totals/avg overall in Table 1? Also percentage of TD comments might be helpful in addition to absolute #  
- in S2.4, does the previous study use the same projects as this study?  
- Section 3 is called "Case Study Results". This isn't a case study but rather an experiment.  
- You cite the Stanford NLP tools [15], but you aren't using them (AFAIK). You are using instead the max entropy classifier at <http://nlp.stanford.edu/software/classifier.html>, right? That should be cited instead.

Sure, we should address all these.

**Reviewer: 3**  
Public Comments (these will be made available to the author)  
In this paper, the authors propose an approach to determine self-admitted technical debt (i.e., code improvements) based on comments in the source code using natural language processing techniques and machine learning. They apply their approach to 10 open source projects.  
  
The work is very interesting, and I enjoyed reading about the methodology for identifying technical debt comments. However, I have concerns about how this work will be used and some possible threats to validity:  
  
(1) I readily agree that technical debt is an important topic, but it’s not clear how important detecting it is — what actions can be taken that could improve software quality or a software’s lifecycle in some way? I would like to see the authors provide further motivation on how their approach could be used.

We should mention that once the TD is identified, then it can be managed or removed. We should cite some of the current work that says TD is unavoidable, and needs to be managed.

(2) Did the authors do any vetting on their development set of whether or not the debt is actually present in the code when it is in the comment? In essence, are they just finding comments that \*admit\* technical debt, or are they actually finding locations in the code that actually \*have\* technical debt? For instance, did the authors consider the impact of obsolete comments? If a developer eliminated the technical debt, would the comment indicating the debt remain?

We should cite the ICSME paper or do our own analysis here of comment and code co-change.  
  
This idea is related to my first point; I can envision some scenarios where detecting admitted technical debt might be useful (such as quick statistics for software quality), and where detecting actual technical debt might be valuable (e.g., to target maintenance efforts).  
  
There is related work to support that 97% of comment changes are consistent, which would go a long way to sidestep the obsolete comment issue. However, it is possible that the remaining 3% could in theory represent a significant sample of TD comments, given how few comments include technical debt. I would recommend the authors manually evaluate a small, representative random sample to see if this is an issue.

Sure, we can do this and also point to the ICSME paper.

I believe the work in its current form identifies technical debt comments. I think the paper would benefit from a deeper discussion of the potential benefits of this information independent of knowing whether there actually \*is\* technical debt present.

Is this the same as the first point? I don’t understand what he/she is asking here.

(3) My final concern is with RQ 3 in determining the necessary amount of training data. Ideally, this RQ should be answered with random samples of different sizes from all projects. Did the authors at least try every possible ordering of the projects to identify how many projects were needed to reach within 10% of the highest F1 measure? It’s unclear in the text exactly what methodology was used, but the graphs in the appendix on p. 16 seem to indicate that they tried many different combinations (I just can’t determine if the order of projects was random).

The order of the projects was not random, we put in the largest project first. Our analysis for reviewer 2 should address this comment as well.  
  
My issue with the conclusions the authors are making is all about the number of \*comments\* are needed, where in fact the only variable they are changing is the number of \*projects\*. Thus, I would tone down the claims in RQ3 to discuss number of projects, rather than number of comments, because it’s possible that a different ordering of projects would lead to a different conclusion in terms of comments. For example, maybe training can take place on 1 large project or 2 different projects with fewer comments and reach the same results with different numbers of comments. Ideally, I would have preferred to see more data points in this section before drawing conclusions about the number of comments needed. In fact, I would suggest the authors remove the data for Comment patterns & the random classifier and instead report line graphs for the NLP-based approach only, condensed into fewer graphs (one for each kind of debt).  
  
Sure, we can tone down and discuss per project. I don’t think it is a good idea to remove the random and comment comparison.

Specific comments:  
  
- Table 2: I recognize that the data set is small, but I think it would be worth reporting some statistical analysis. The relationships seem quite pronounced and might still be statistically significant.

Sure  
  
- RQ 2: I notice that some textual features include punctuation, while others don’t. Should punctuation be separated out? Is it the presence of the question mark alone in “needed?” or the word \*with\* the question mark that indicates technical debt? A brief justification of the author’s handling of punctuation would help. Did the authors try just “?” alone in predicting technical debt? I notice a number of features include a question mark. Did the authors try technical features with & without the punctuation, or with the punctuation separated out to see the impact it would have?

Is this a phenomena of the NLP tool?  
  
- section 7: I think there needs to be a new paragraph introduced on line 44 at “Then,”

Sure

- I don’t think the appendix on p. 16 is needed — I think this data can & should be integrated into two figures (one for each kind of debt), by removing the training data impact for comment patterns & the random classifier. The authors’ NLP-based approach is clearly superior, so I don’t think it’s useful to learn about the impact of training a random classifier. To achieve just 2 graphs, the y-axis would need to be changed to using the % of the max F measure of each iteration rather than the raw F measure (or something similar that is appropriate for all the graphs on the following pages). I think this data is critically important to support RQ3, and should not be relegated to an appendix.

Sure

Typos/grammar: please see attached pdf. The paper could use a careful reading for typos & grammar.

Easy fixes