8-2020

# Prevalence, Contents and Automatic Detection of KL-SATD

Leevi RANTALA
*University of Oulu*

Mika MANTYLA
*University of Oulu*

David LO
*Singapore Management University*, davidlo@smu.edu.sg

# Prevalence, Contents and Automatic Detection of KL-SATD

Leevi Rantala, Mika Mäntylä
ITEE, M3S
University of Oulu
Oulu, Finland
Email: leevi.rantala@oulu.fi, mika.mantyla@oulu.fi

David Lo
Information Systems
Singapore Management University
Singapore, Singapore
Email: davidlo@smu.edu.sg

***Abstract*—When developers use different keywords such as TODO and FIXME in source code comments to describe self-admitted technical debt (SATD), we refer it as Keyword-Labeled SATD (KL-SATD). We study KL-SATD from 33 software repositories with 13,588 KL-SATD comments. We find that the median percentage of KL-SATD comments among all comments is only 1,52%. We find that KL-SATD comment contents include words expressing code changes and uncertainty, such as remove, fix, maybe and probably. This makes them different compared to other comments. KL-SATD comment contents are similar to manually labeled SATD comments of prior work. Our machine learning classifier using logistic Lasso regression has good performance in detecting KL-SATD comments (AUC-ROC 0.88). Finally, we demonstrate that using machine learning we can identify comments that are currently missing but which should have a SATD keyword in them. Automating SATD identification of comments that lack SATD keywords can save time and effort by replacing manual identification of comments. Using KL-SATD offers a potential to bootstrap a complete SATD detector.**

***Index Terms*—Natural language processing; self-admitted technical debt; data mining**

## I. Introduction

Technical debt is a term used to depict non-optimal choices made in the software development process. Several types of technical debt has been identified such as code debt, design and architectural debt, environmental debt, knowledge distribution and documentation debt, and testing debt [1]. Self-admitted technical debt (SATD) refers to a specific type of code debt, where the developer acknowledges admitting code debt into the system [2]. This admittance can be done in several ways, such as writing a comment into the code or explaining it in a commit message.

One specific way of marking SATD on a code level is to leave a comment with a specific keyword such as TODO. In this paper we focus on four different keywords, which are TODO, FIXME, HACK and XXX. These have all been referred in previous literature as indicators of self-admitted technical debt when present in code comments [3]–[5]. We call SATD messages labeled with one of these keywords as Keyword-Labeled SATD (KL-SATD). We note that not all SATD comments have these or any keywords.

In this paper we performed an empirical study on 33 repositories containing more than 500,000 comments to answer the following research questions about SATD comments:

- **RQ1:** *What is the prevalence of KL-SATD comments?* To answer this question, we detect comments for source code and analyze how many of them contain a SATD keyword.
- **RQ2:** *Do the contents of KL-SATD comments differ from other comments?* For this question, we perform a word distribution analysis for all comments, and examine if the comments marked with KL-SATD differ from other comments by their vocabulary.
- **RQ3:** *Can we automatically find comments that have omitted SATD keyword?* Here we use KL-SATD data to train a machine learning classifier but use it to detect SATD comments that have omitted the keyword. This can be beneficial as developers do not necessarily use keywords consistently in their code comments. This SATD detector can be seen as automated labeler that can be trained cheaply from existing SATD comments. Automating this step in SATD tooling is important, as it eliminates the manual labor, which is both time consuming and prone to errors.

The paper is structured as follows. Section II describes the methodology, starting from the used dataset, continuing with the processing of the comments, and ending with particularities pertaining the research questions . Section III shows the results, starting from KL-SATD prevalence, then looking into contents of KL-SATD comments, and ending with automatically finding comments that have omitted SATD keyword. Section IV talks about limitations pertaining to our work, and finally Section V presents the conclusions of the study.

## II. Methodology

The overview of the methodology is shown in Figure 1.

*a) Data.:* We utilize the Technical Debt Dataset created by Lenarduzzi et al. [6]. It includes 33 Java projects, which are all over three years old, have more than 500 commits, and include over 100 classes. The dataset has a total of over 140 thousand commits in it. We extract both multi- and single-line comments by identifying Java-style patterns with Regex. In total we extracted 862,342 comments.

*b) Preprocessing.:* The parsed code comments contain lots of noise, and we reduce it processing the comments using

978-1-7281-9532-2/20/$31.00 ©2020 IEEE
DOI 10.1109/SEAA51224.2020.00069


Fig. 1. Overview of the methodology

following steps: First, we remove comments that are code using NLoN tool [7]. Second, similar to earlier works [1], [8], we deleted all comments that contained license or copyright information, and Javadoc comments containing tags. It has been shown earlier that these comments are not likely to contain SATD [5]. Third, we remove comments that appear in renamed files by excluding all the comments that were deemed as renamed ones according to the Git log in the Technical Debt Dataset. Fouth, all messages were tokenized; next, we transformed all the words to lower case, removed non-alphanumeric characters, and deleted possible email and website addresses. Then we filtered out words that were less than three characters long. Fifth, we filtered out stop words. Sixth, even with stop word removal, some noise was still left to the dataset such as misspelled words, meaningless words, and even some very rare terms like "abstractauthenticationtoken". We solved this by choosing that a word has to appear at least in 5 repositories before it is included in the final vocabulary. This removes even further specific words relating only to couple of projects, and gets rid of many of the misspelled words. Important benefit is that it makes the vocabulary and our results more generalizable. Preprocessing reduces the total number of comments to 507,254.

*c) RQ1:* asks about prevalence of KL-SATD comments. To answer this, we calculated on how many commits new comments appeared, and how many of these comments contained SATD keyword. This enables us both to compare projects with each other, and form a general conception of KL-SATD prevalence.

*d) RQ2:* looks into how the contents of KL-SATD comments differ from other comments. For this paper, we examined the differences visually by creating comparison word clouds based on word frequencies between KL-SATD comments and other comments. This allows us to see whether they differ in their contents.

*e) RQ3:* uses a machine learning classifier to detect SATD comments. We selected logistic regression with lasso penalty from Glmnet package[1] as our machine learning approach. It has been shown to have fast performance, and to work with large and sparse matrices [9] which are typical in NLP tasks. It has been applied successfully to different NLP tasks [10]. Logistic regression performs automatically feature selection, and prevents overfitting with penalty term lambda. We perform stratified 10-fold cross-validation and report the results using lambda, which gives the maximum mean for area under the receiver operating characteristics curve (AUC-ROC). We chose AUC-ROC as it performs well with unbalanced datasets [11]. To further alleviate class imbalance, we assign weights to the training data so that KL-SATD comments weigh as much as the non-KL-SATD comments. This equal weighting has improved performance when predicting defects from imbalanced datasets [12]. We performed term frequency–inverse document frequency (tf-idf) transformation for the comments before entering them to the classifier. This common procedure ensures that common terms are given less value than rarer terms.

[1]https://cran.r-project.org/web/packages/glmnet/index.html

TABLE I
SUMMARY OF ALL COMMENTS AND KL-SATD COMMENTS PER REPOSITORY

| | All Comments | KL-SATD Comments | KL-SATD Percentage |
|---|---|---|---|
| Min. | 565 | 2 | 0.29 |
| 1st Qu. | 2,849 | 25 | 0.90 |
| Median | 4,567 | 68 | 1.52 |
| Mean | 15,371 | 411.8 | 2.29 |
| 3rd Qu. | 10,931 | 324 | 3.05 |
| Max. | 112,780 | 2,943 | 8.78 |
| Total | 507,254 | 13,588 | |

## III. RESULTS

### A. RQ1. What is the prevalence of KL-SATD comments?

Total number of comments from all projects after preprocessing was 507,254, from which 13,588 were KL-SATD comments. Descriptive statistics are in Table I. The mean of KL-SATD comments per repository was 2.29%, while median was 1.52%. The largest percentage of KL-SATD comments were found on commons-bcel project, on which 8.78% (n=954) from all 10,867 comments present on that project were KL-SATD comments. The least amount in terms of percentage was found on commons-fileupload, which had 2 KL-SATD comments out of 690 comments in total (0.29%).

We further explored why a repository starts getting a higher KL-SATD comment percentage using a Spearman's rank correlation test. We find a low positive correlation between KL-SATD percentage and the following project variables: number of commits (r=0.18), number of developers (r=0.38), and lines of code (r=0.21). We looked the project duration, and found a low negative correlation (r=-0.25) between development time in months and KL-SATD percentage. As a conclusion, projects with larger size, more developers and which have shorter

development history are weakly correlated with a higher KL-SATD comment ratio. However, as the correlations were quite weak, we can't define a strong reason why this happens.

Our study shows that KL-SATD comment percentages between projects vary greatly, and the median amount for projects being 1.52%. Previous study [13] analyzed manually comments of five different projects and found SATD percentage between 2.10 - 3.95% per project (median 3.41%). Our study's lower percentage means that simply catching KL-SATD comments with keywords is not enough to catch all the SATD related comments. Manually checking and labeling of the comments that have omitted a SATD keyword is time consuming, which shows the need to detect these comments automatically.

### *B. RQ2: Do the contents of KL-SATD comments differ from other comments?*

For analyzing the differences between KL-SATD comments and other comments, we looked at the distribution of words. We created a comparison wordcloud based on the frequencies of words appearing in either KL-SATD comments or non-KL-SATD comments using R-package wordcloud[2]. Figure 2 shows how different words are used in KL-SATD comments (after keyword removal) when compared to other comments.

Words in KL-SATD comments relate to modifying, verifying or adding code functionalities: "remove", "add", "need", "fix" and "check". KL-SATD comments also include words that show uncertainty such as "maybe", "probably" and "consider". Meanwhile, words in comments that are not KL-SATD are more neutral in their tone, and include terms such as "value", "default", "code" and "element". Therefore, they seem more like descriptions of code functionality, perhaps related to documenting code.

Looking at the manually labeled SATD comments from previous work [2], we can see that they share similar vocabulary as our KL-SATD comments. Both contain words such as "use", "need", "remove", "implement", and "consider". Even with differences, we can conclude that generic SATD comments and our KL-SATD comments share similarities.

### *C. RQ3. Can we automatically find comments that have omitted a SATD keyword?*

Here we first trained a machine learning classifier using logistic regression to identify comments with KL-SATD. Glmnet logistic regression with stratified 10-fold cross validation gave us an AUC of 0.88 for predicting KL-SATD comments, meaning that our classifier was able to reliably discover which comments had KL-SATD in them with out seeing the keywords.

Examining the classifier's predictor words (Table II) with the most positive and negative coefficients, we see a somewhat different picture than in the word clouds of Section III-B. This is because we performed tf-idf transformation for the classifier, and did not rely on basic appearance counts. The most positive predictor words include words that are not very visible in the word clouds but ones that you would expect, e.g. refactor, perhaps, consider, and maybe. The most negative predictors include words like "determined", "certain", "returns" and "sequence". The full list of predictors can be downloaded from Figshare[3].

Having shown that the classifier is able to detect reliably KL-SATD comments, we then examine the comments that according to the classifier should have KL-SATD in them, but had omitted it. We chose a conservative cutoff point for the predictions, and considered only predictions that were both: a) Not labeled with KL-SATD, and b) Predicted with over 70% confidence, that they should contain KL-SATD. This gave us 14,690 unique comments that are potentially missing an SATD keyword. To examine the results, we took a random sample of 100 comments, which are availabe from Figshare[4]. Two of the authors went over the comments independently, and labeled them with following scale: 0 - Does not contain SATD; 1 - Might contain SATD; 2 - Contains SATD; Empty - Can't tell


Fig. 2. Comparison wordcloud of SATD and non-SATD comments

TABLE II
10 PREDICTOR WORDS WITH THE MOST POSITIVE AND NEGATIVE COEFFICIENTS PREDICTING KL-SATDFREQUENCIES

| | Positive | Negative |
|---|---|---|
| 1. | visit | certain |
| 2. | consider | determined |
| 3. | detailed | concrete |
| 4. | perhaps | unlike |
| 5. | refactor | raised |
| 6. | say | edit |
| 7. | think | returns |
| 8. | would | hidden |
| 9. | implement | casts |
| 10. | fact | sequence |

[2]https://cran.r-project.org/web/packages/wordcloud/

[3]DOI: 10.6084/m9.figshare.11907216

[4]DOI: 10.6084/m9.figshare.12039945

TABLE III
LABELING RESULTS OF 100 RANDOMLY SELECTED COMMENTS

| Label | Labeler 1 | Labeler 2 |
|---|---|---|
| 0 (Does not contain SATD) | 41 | 60 |
| 1 (Might contain SATD) | 23 | 12 |
| 2 (Contains SATD) | 20 | 18 |
| Empty (Can't tell if SATD or not) | 16 | 10 |

if SATD or not. The labeling results are presented in Table III.

After labeling, we had 20 different comments with empty label with one or two of the authors. This left us with 80 comments. There were in total 10 comments, which both authors had labeled as 2. These can be considered to be very accurate predictions. In total, there were 23 different comments, which were labeled either as 1 or 2 by both authors, and 24 cases where one reviewer gave label of 0, and then other either 1 or 2. These are more uncertain cases, where the context of the comment (e.g. the code around the comment) might help to make more accurate estimation. And finally there were 33 cases, where both authors said that the comment does not include SATD.

In many cases, it was hard to determine whether SATD was present in a particular comment. For example, it is hard to differentiate whether a comment is a description of what the code currently does or what it should do.

Overall, both labelers agreed that on 1/3 of the comments there were no SATD present. This means, that up to 2/3 ($\approx 10,000$ comments) of our classifier sample may contain technical debt. However, in many cases more information of the project and source code would be needed to determine the true share of technical debt.

## IV. LIMITATIONS

Choice of machine learning algorithm can be seen as a limitation. However, it has been shown that at least in the context of detecting code smells the choice of the algorithm does not give meaningful difference in performance [14].

The choice to repositories used in can be seen also as a limitation. We used 33 repositories gathered in a previous study [6], which is a much larger than used in prior works of automated SATD detection, e.g. 10 in [15] and 7 in [1].

## V. CONCLUSION AND FUTURE WORK

In this paper, we studied the prevalence of KL-SATD comments, their contents and automatically finding comments that should include a SATD keyword. All the data and the R-code used to perform the study are available online [5]. The results show that developers use SATD keywords sparingly (median 1.52% of comments). We showed that KL-SATD comment contents are different from comments without it, and typically express code changes and uncertainty. KL-SATD comment words are similar to the SATD lexicon shown in prior works. We built a KL-SATD detector that achieved relatively high AUC-ROC of 0.88. Our KL-SATD detector can also find comments where developers have omitted a SATD keyword even though it clearly would need one. Thus, the amount of KL-SATD comments does not accurately reflect the number of comments that should have SATD keyword in them. This points to a problem with the state-of-the-practice detectors such as SonarQube, which rely on keyword detection as one part of their technical debt detectors.

We manually labeled only 100 randomly chosen samples, and this does not necessarily reflect the true performance of the classifier. We fully acknowledge this problem in our short paper. To fix shortcomings of our model, we plan to test our classifier with an industry partner (Softagram), and we will also retrain our model with Active Learning methods, which will further enhance its performance.

[5]https://github.com/M3SOulu/KLSATD_SEAA_2020

## REFERENCES


[1] M. Yan, X. Xia, E. Shihab, D. Lo, J. Yin, and X. Yang, "Automating change-level self-admitted technical debt determination," *IEEE Transactions on Software Engineering*, 2018.

[2] A. Potdar and E. Shihab, "An exploratory study on self-admitted technical debt," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on software maintenance and evolution (ICSME)*. IEEE, 2014, pp. 91–100.

[3] M. A. de Freitas Farias, M. G. de Mendonça Neto, A. B. da Silva, and R. O. Spínola, "A contextualized vocabulary model for identifying technical debt on code comments," in *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*. IEEE, 2015, pp. 25–32.

[4] M.-A. Storey, J. Ryall, R. I. Bull, D. Myers, and J. Singer, "Todo or to bug," in *2008 ACM/IEEE 30th International Conference on Software Engineering*. IEEE, 2008, pp. 251–260.

[5] E. d. S. Maldonado and E. Shihab, "Detecting and quantifying different types of self-admitted technical debt," in *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*. IEEE, 2015, pp. 9–15.

[6] V. Lenarduzzi, N. Saarimäki, and D. Taibi, "The technical debt dataset," in *Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering*, 2019, pp. 2–11.

[7] M. Mäntylä, F. Calefato, and M. Claes, "Natural language or not (nlon)-a package for software engineering text analysis pipeline," in *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. IEEE, 2018, pp. 387–391.

[8] E. da Silva Maldonado, E. Shihab, and N. Tsantalis, "Using natural language processing to automatically detect self-admitted technical debt," *IEEE Transactions on Software Engineering*, vol. 43, no. 11, pp. 1044–1062, 2017.

[9] J. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *Journal of statistical software*, vol. 33, no. 1, p. 1, 2010.

[10] A. Genkin, D. D. Lewis, and D. Madigan, "Large-scale bayesian logistic regression for text categorization," *Technometrics*, vol. 49, no. 3, pp. 291–304, 2007.

[11] R. C. Prati, G. Batista, and M. C. Monard, "A survey on graphical methods for classification predictive performance evaluation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 11, pp. 1601–1618, 2011.

[12] S. Herbold, "Training data selection for cross-project defect prediction," in *Proceedings of the 9th international conference on predictive models in software engineering*, 2013, pp. 1–10.

[13] S. Wehaibi, E. Shihab, and L. Guerrouj, "Examining the impact of self-admitted technical debt on software quality," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 179–188.

[14] F. A. Fontana, M. V. Mäntylä, M. Zanoni, and A. Marino, "Comparing and experimenting machine learning techniques for code smell detection," *Empirical Software Engineering*, vol. 21, no. 3, pp. 1143–1191, 2016.

[15] X. Ren, Z. Xing, X. Xia, D. Lo, X. Wang, and J. Grundy, "Neural network based detection of self-admitted technical debt: From performance to explainability," *ACM transactions on software engineering and methodology*, vol. 28, no. 3, pp. 1–45, 2019.