Automatic Identification of Self-Admitted Technical Debt from Different Sources

Yikun Li, Mohamed Soliman, Paris Avgeriou
Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence
University of Groningen
Groningen, The Netherlands
{yikun.li, m.a.m.soliman, p.avgeriou}@rug.nl

***Index Terms*—deep learning, multitask learning, self-admitted technical debt, issue tracking systems, source code comments, commits, pull requests**

I. Introduction

Technical debt (TD) is a metaphor describing the situation that long-term benefits (e.g., maintainability and evolvability of software) are traded for short-term goals. When technical debt is admitted explicitly by developers in software artifacts (e.g., code comments or issue tracking systems), it is termed as *Self-Admitted Technical Debt* or *SATD* [1].

Technical debt could be admitted in different sources, such as source code comments, issue tracking systems, pull requests, and commit messages [2]. However, there is no approach proposed for identifying SATD from different sources. Thus, in this paper, we propose an approach for automatically identifying SATD from different sources (i.e., source code comments, issue trackers, commit messages, and pull requests).

The remainder of this paper is organized as follows. Section II presents the study design, and the results are shown in **??**. Finally, conclusions are drawn in Section **??**.

II. Study Design

The goal of this study, formulated according to the Goal-Question-Metric [3] template is to "***analyze*** *data from source code comments, commit messages, pull requests, and issue tracking systems* ***for the purpose of*** *automatically identifying self-admitted technical debt from different sources* ***with respect to*** *accuracy and explainability* ***from the point of view of*** *software engineers* ***in the context of*** *open-source software.*" This goal is refined into seven research questions (RQs):

- **(RQ1)** *How to accurately identify self-admitted technical debt from different sources?* In recent years, a great deal of research has been focused on identifying SATD from source code comments [4], [5]. However, SATD in other sources (i.e., commit messages, pull requests, and issues) is hardly explored and identified. Moreover, there is a lack of approaches to identify SATD from different sources. Being able to detect SATD from different sources can enable researchers to investigate SATD beyond source code comments, and thus to determine differences and relationships between SATD from different sources. Furthermore, this can also help practitioners to find and pay back the SATD documented in different sources.
- **(RQ2)** *How much self-admitted technical debt is documented in different sources?* Software engineers could admit technical debt in different sources, e.g. source code comments, commit messages, pull requests (including code reviews), and issue tracking systems. Quantifying the amount of different types of SATD can help to understand how SATD is distributed in different sources and the proportion of types of SATD in different sources. Subsequently, this aids in understanding the advantages and disadvantages of different sources for SATD management. For example, if most of the documentation debt is admitted in issue tracking systems, developers can focus on monitoring documentation debt in issues to keep it under control.

To investigate SATD in different sources in open-source software, we first need to know which sources potentially contain SATD statements. Thus, we look into common processes involved in contributing to open-source projects. According to the contribution guidelines [6]–[8], when developers find a problem or bug, they first always create an issue to report it. If changes are needed, developers discuss the issue and open pull requests to resolve it; otherwise, the issue is closed. If the pull requests pass review, changes are merged to the system. We can notice there are several chances for developers admitting technical debt in accordance with contribution guidelines, such as in issues, in pull requests, in commit messages, and in source code comments. Therefore, in this study, we focus on studying SATD in these four sources. This observation is consistent with the findings from the previous work [2], that the four most popular software artifacts to document technical debt are source code comments, commit messages, pull requests, and issue trackers.

A. Approach Overview

The overview of our approach is demonstrated in Fig. 1. In the first step, we collect issues, pull requests, commit messages, and source code comments from 103 selected open-source projects. Thereafter, we link the data from different sources for analysis. Following that, we cleanse and analyze

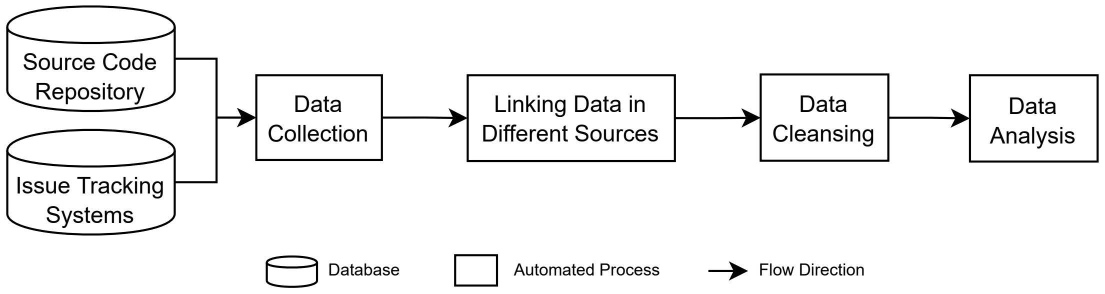


Fig. 1. The overview of our approach.

the data to answer research questions. We elaborate on each of the processes in the following subsections.

B. Data Collection

To identify and measure SATD in different sources, we first need to find appropriate projects and collect data. Thus, we look into Apache ecosystems, because these projects are of high quality, maintained by mature communities, and required to make all communications related to code and decision-making publicly accessible [9]. Since there are over 2,000 repositories in the Apache ecosystem on GitHub[1], we set the following criteria to select projects pertinent to our study goal:

1) The source code repository, commits, pull requests, and issues of the project are publicly available.
2) They have at least 500 issue tickets and 500 pull requests. This is to ensure sufficient complexity.

Based on the above criteria, we find 103 Apache projects on GitHub. The project information is obtained on March 2, 2021. The details of these projects are presented in Table I.

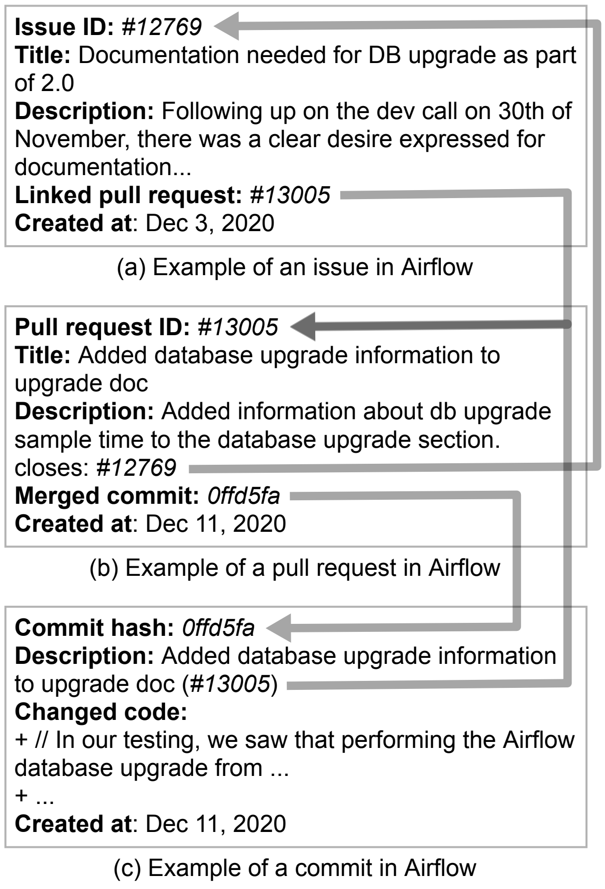
C. Linking Data in Different Sources

In order to study SATD in different sources, we have to build links between sources. More specifically, since pull request titles always contain the issue information, we can build connections between pull requests and issues. Besides, because commit messages contain the related pull request or issue information, we can link commits to pull requests or issues. Moreover, commits record changes to one or more files. Therefore, we can link code comment changes with commits. Examples of links between different sources are shown in Fig. 2.

D. Data Cleansing

In issues and pull requests, apart from comments left by developers, plenty of comments are automatically generated by bots. For example, when a new contributor opens a first pull request, a welcome bot could comment on a brief contribution guideline to help the new contributor. Since comments created by bots do not contain SATD, we filter out these comments.

[1]https://github.com/apache



Issue ID: *#12769*
Title: Documentation needed for DB upgrade as part of 2.0
Description: Following up on the dev call on 30th of November, there was a clear desire expressed for documentation...
Linked pull request: *#13005*
Created at: Dec 3, 2020

(a) Example of an issue in Airflow

Pull request ID: *#13005*
Title: Added database upgrade information to upgrade doc
Description: Added information about db upgrade sample time to the database upgrade section.
closes: *#12769*
Merged commit: *0ffd5fa*
Created at: Dec 11, 2020

(b) Example of a pull request in Airflow

Commit hash: *0ffd5fa*
Description: Added database upgrade information to upgrade doc (*#13005*)
Changed code:
+ // In our testing, we saw that performing the Airflow database upgrade from ...
+ ...
Created at: Dec 11, 2020

(c) Example of a commit in Airflow

Fig. 2. Example of links between different sources in Airflow.

E. Data Analysis - SATD Identification from Multiple Sources

1) Machine Learning Models: In order to accurately identify SATD from different sources, we adopt several machine learning approaches and compare their performance in SATD identification. Because there is no approach designed to identify SATD from different sources, inspired by the work of Kim [10] and Liu *et al.* [11], we propose Multitask Text Convolutional Neural Network (MT-Text-CNN) to fill this gap. All utilized machine learning approaches are listed as below:

- **Traditional machine learning approaches (LR, SVM, RF)**: To illustrate the effectiveness of our approach, we

TABLE I
DETAILS OF COLLECTED DATA.

# Issues				**# Issue Comments**				**# Commits**			
Min	Max	Mean	Sum	Min	Max	Mean	Sum	Min	Max	Mean	Sum
526	24,938	3,658	581,584	856	303,608	19,408	3,085,986	112	70,865	8,325	1,323,682
# Pull Requests				**# Pull Comments**				**# Code Comments**			
Min	Max	Mean	Sum	Min	Max	Mean	Sum	Min	Max	Mean	Sum
507	32,072	3,034	312,591	113	408,824	13,156	1,355,158	326	2,894,056	229,717	23,660,869

TABLE II
DETAILS OF UNITS.

Contribution Flow	**Abbr.**	**#**	**%**	**#**	**%**
JIRA Issue $\rightarrow$ *Pull Request* $\rightarrow$ Commit $\rightarrow$ Code	IPC1	100,234	8.0		
GitHub Issue $\rightarrow$ *Pull Request* $\rightarrow$ Commit $\rightarrow$ Code	IPC2	13,681	1.1	114,073	9.1
JIRA Issue + GitHub Issue $\rightarrow$ *Pull Request* $\rightarrow$ Commit $\rightarrow$ Code	IPC3	158	0.0		
JIRA Issue $\rightarrow$ Commit $\rightarrow$ Code	IC1	386,314	30.9		
GitHub Issue $\rightarrow$ Commit $\rightarrow$ Code	IC2	12,897	1.0	399,326	32.0
JIRA Issue + GitHub Issue $\rightarrow$ Commit $\rightarrow$ Code	IC3	115	0.0		
Pull Request $\rightarrow$ Commit $\rightarrow$ Code	PC	141,910	11.4	141,910	11.4
Commit $\rightarrow$ Code	C	593,015	47.5	593,015	47.5

select and compare our approach with four prevalent traditional machine learning algorithms, namely Logistic Regression (LR) [12], Support Vector Machine (SVM) [13], and Random Forest (RF) [14]. We train these four traditional classifiers using the implementation in Sklearn[2] with default settings.

- **Text Convolutional Neural Network (Text-CNN)**: Text-CNN is a state-of-the-art text classification algorithm proposed by Kim [10], which has been used in several SATD identification studies [4], [15]. The details of this approach are given, as they are background knowledge for understanding the differences between Text-CNN and MT-Text-CNN. The architecture of Text-CNN is demonstrated in Fig. 3. As can be seen, Text-CNN consists of five layers, namely embedding layer, convolutional layer, max-pooling layer, concatenation layer, and output layer.
    - **Embedding layer**: It is the first layer that converts the tokenized input sentence (the length of the sentence is n) into a matrix of size $n \times k$ using an k-dimensional word embedding (see Section II-E3). For example in Fig. 3, the input sentence is *document should be updated to reflect this*, which is transformed into a 7×5 matrix as the input sentence contains 7 words and the word embedding dimensionality equals to 5.
    - **Convolutional layer**: It is the fundamental layer of CNN that performs convolution operation to extract the high-level features from the sentence matrix. A convolution operation associates a filter, which is a matrix that has the same width as the sentence matrix (i.e., k) and the height of it varies. The height of the filter is denoted by region size. The filter with a region size of h can be applied to a window of h words to generate a new feature. Thus, by sliding a filter with a region size of h over the whole sentence matrix, a feature map of size $n-h+1$ is produced. For instance in Fig. 3, when the model has filters whose region sizes are 1, 2, and 3, the sizes of produced feature maps are 7, 6, and 5 respectively.
    - **Max-pooling layer**: It is a layer that calculates the maximum value of each feature map to reduce the spatial size of the representation.
    - **Concatenation layer**: It is a layer that concatenates the scalar features to form the penultimate layer.
    - **Output layer**: It is the last layer that computes the probability of input text to be a SATD text. Because Text-CNN is for single task, it performs a linear transformation of the features from the previous layer by $Y = \mathbf{W} \cdot X + B$, where **W** and B denotes weight and bias. The length of Y equals to the number of classes. Then the Softmax function is applied to Y to calculate the probability of input text belonging to each class. For example, there are two classes: SATD text or non-SATD text in Fig. 3.

- **Multitask Text Convolutional Neural Network (MT-Text-CNN)**: Although SATD in different sources has substantial similarities, there still are significant differences between them [15]. This could lower the accuracy of Text-CNN when detecting SATD from multiple sources,

[2]https://scikit-learn.org

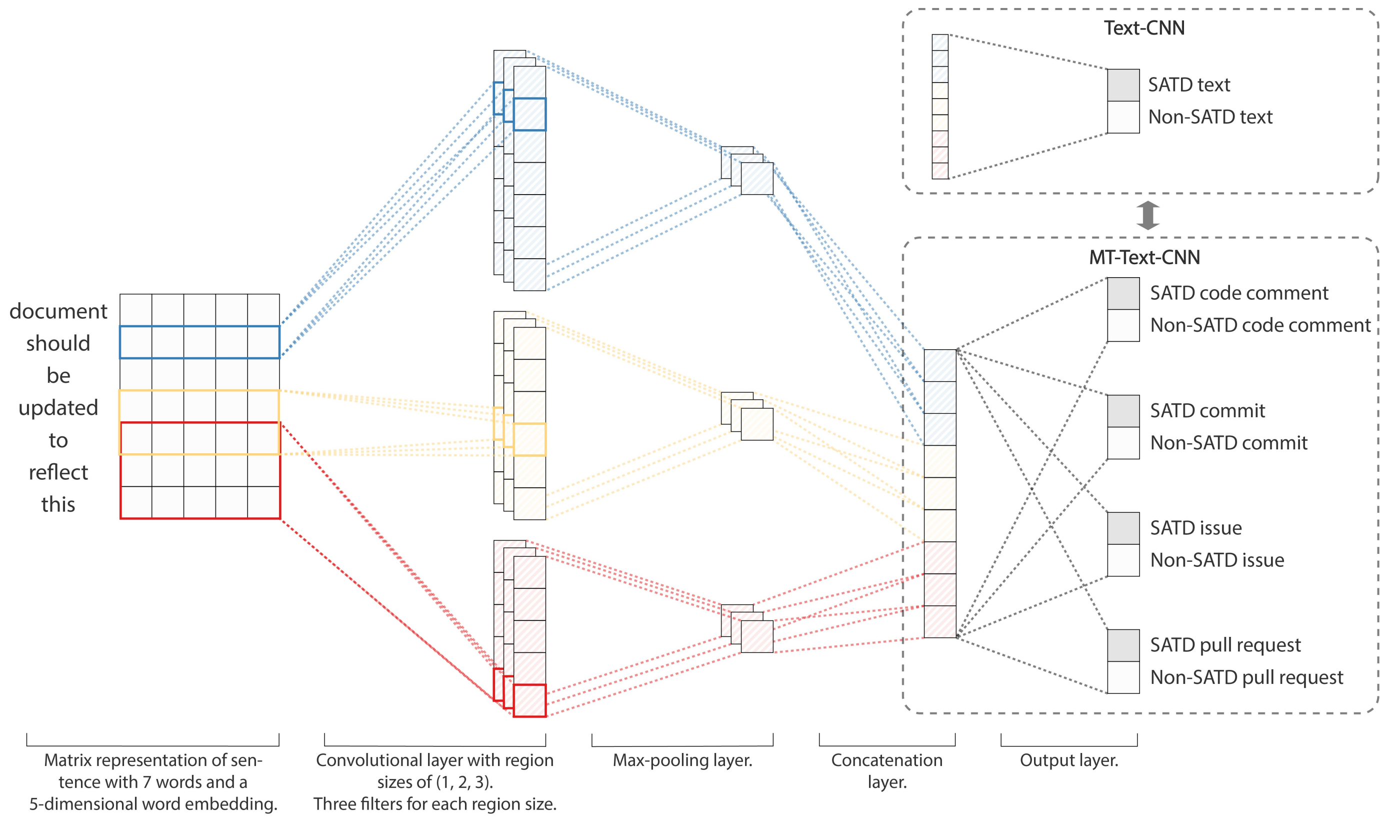


Fig. 3. Architectures of Text-CNN and MT-Text-CNN.

as the standards for SATD identification are slightly different for different sources. Thus, we propose MT-Text-CNN to accurately identify SATD from different sources. The architecture of MT-Text-CNN is illustrated in Fig. 3. As we can see, apart from the output layer, the rest of layers are identical with Text-CNN. Inspired by the work of Liu *et al.* [11], for each task, we create a task-specific output layer, which also performs a linear transformation of the features from the previous layer by $Y^{(t)} = \mathbf{W}^{(t)} \cdot X + B^{(t)}$, where t denotes different tasks (i.e., identifying SATD from different sources). Then the Softmax function is applied to $Y^{(t)}$ to calculate the probability of input text belonging to each class for task t.

In this study, we implement machine learning approaches using Pytorch library[3]. Machine learning models are trained on NVIDIA Tesla V100 GPUs.

2) Baseline Approaches: We implement two baseline approaches to compare the results with machine learning approaches.

- **Baseline 1 (Random)**: It classifies text as SATD randomly according to the probability of random text being SATD text. For instance, if the database contains 1,000 pieces of SATD text out of 10,000 pieces of text, this approach assumes the probability of new text to be SATD text is $1000/10000 = 10\%$. Then this approach randomly classifies any text as SATD text corresponding to the calculated probability (10%).
- **Baseline 2 (Keyword)**: In the previous work [1], 64 SATD keywords were summarized manually by authors, such as *todo*, *fixme*, and *temporary solution*. Then these keywords were used for SATD identification in several studies [16], [17]. This keyword-based approach classifies a text as a SATD text when the text contains one or more of these SATD keywords.

3) Word Embedding: Word embedding is a type of word representation that words are represented similarly if they have high similarities. Typically, words are represented in the form of real number vectors. Training word embedding using data in the same context of target task has been proven to outperform randomly initialized or pre-trained word embeddings by a large margin for SATD identification task [15]. In this study, we train word embedding on our collected data (i.e., source code comments, commit messages, pull requests, and issues) using fastText technique [18] while setting word embedding dimension to 300.

4) Training Procedure: We follow the guideline proposed by Collobert and Weston [19] to perform joint training on multiple tasks. Training is done in a stochastic manner with

[3]https://pytorch.org

the following steps:

- Randomly pick up a task.
- Get a random training sample for this task.
- Train the machine learning model using the sample.
- Go to the first step.

5) Strategies for Handling Imbalanced Data: According to the previous study [4], [15], only a very small proportion of source code comments or issue comments are SATD comments, so the dataset is seriously imbalanced. It has been shown that using weighted loss could effectively improve the SATD identification accuracy [4], [15], which penalizes harder the wrongly classified items from minority classes (i.e. false negative and false positive errors) during training. Thus, we use weighted loss function in this work.

6) Evaluation Metrics: We use the following statistics: true positive (TP) represents the number of items correctly classified as SATD items; true negative (TN) represents the number of items correctly classified as non-SATD items; false positive (TN) represents the number of items that are wrongly classified as SATD items; false negative (FN) represents the number of items that are wrongly classified as non-SATD items. Sequentially, we calculate **precision** ($\frac{TP}{TP+FP}$), **recall** ($\frac{TP}{TP+FN}$), and **F1-score** ($2 \times \frac{precision \times recall}{precision + recall}$) to evaluate the performance of different approaches. High evaluation metrics indicate good performance. We use F1-score to evaluate the performance of approaches because it incorporates the trade-off between precision and recall. It is noted that when identifying different types of SATD, we first calculate the F1-score for each type of SATD, and then average the F1-score to obtain the macro F1-score.

References

[1] A. Potdar and E. Shihab, “An exploratory study on self-admitted technical debt,” in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 91–100.

[2] F. Zampetti, G. Fucci, A. Serebrenik, and M. Di Penta, “Self-admitted technical debt practices: a comparison between industry and open-source,” *Empirical Software Engineering*, vol. 26, no. 6, pp. 1–32, 2021.

[3] R. van Solingen, V. Basili, G. Caldiera, and H. D. Rombach, “Goal Question Metric (GQM) approach,” in *Encyclopedia of Software Eng.* Hoboken, NJ, USA: John Wiley & Sons, Inc., jan 2002, pp. 528–532.

[4] 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 (TOSEM)*, vol. 28, no. 3, pp. 1–45, 2019.

[5] 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.

[6] Aaron Stannard, “How to use github professionally: Best practices for working with github in team settings.” [Online]. Available: https://petabridge.com/blog/use-github-professionally/

[7] Richard Monson-Haefel, “It’s easy! your first tomee pull-request: Using jira.” [Online]. Available: https://www.tomitribe.com/blog/its-easy-requesting-and-discovering-jira-tickets-in-tomee/

[8] Akira Ajisaka, “Hadoop contributor guide - how to contribute.” [Online]. Available: https://cwiki.apache.org/confluence/display/HADOOP/How+To+Contribute

[9] Apache Software Foundation. Briefing: The apache way. [Online]. Available: http://www.apache.org/theapacheway/index.html

[10] Y. Kim, “Convolutional neural networks for sentence classification,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1746–1751.

[11] X. Liu, J. Gao, X. He, L. Deng, K. Duh, and Y.-y. Wang, “Representation learning using multi-task deep neural networks for semantic classification and information retrieval,” in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, May–Jun. 2015, pp. 912–921.

[12] 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.

[13] A. Sun, E.-P. Lim, and Y. Liu, “On strategies for imbalanced text classification using svm: A comparative study,” *Decision Support Systems*, vol. 48, no. 1, pp. 191–201, 2009.

[14] L. Breiman, “Random forests,” *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[15] Y. Li, M. Soliman, and P. Avgeriou, “Identifying self-admitted technical debt in issue tracking systems using machine learning,” 2021, unpublished.

[16] G. Bavota and B. Russo, “A large-scale empirical study on self-admitted technical debt,” in *Proceedings of the 13th International Conference on Mining Software Repositories*, 2016, pp. 315–326.

[17] Y. Kamei, E. d. S. Maldonado, E. Shihab, and N. Ubayashi, “Using analytics to quantify interest of self-admitted technical debt.” in *QuA-SoQ/TDA@ APSEC*, 2016, pp. 68–71.

[18] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, “Advances in pre-training distributed word representations,” in *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[19] R. Collobert and J. Weston, “A unified architecture for natural language processing: Deep neural networks with multitask learning,” in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 160–167.