DebtViz: A Tool for Identifying, Measuring, Visualizing, and Monitoring Self-Admitted Technical Debt

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

***Abstract*—Technical debt, specifically Self-Admitted Technical Debt (SATD), remains a significant challenge for software developers and managers due to its potential to adversely affect long-term software maintainability. Although various approaches exist to identify SATD, tools for its comprehensive management are notably lacking. This paper presents DebtViz, an innovative SATD tool designed to automatically detect, classify, visualize and monitor various types of SATD in source code comments and issue tracking systems. DebtViz employs a Convolutional Neural Network-based approach for detection and a deconvolution technique for keyword extraction. The tool is structured into a back-end service for data collection and pre-processing, a SATD classifier for data categorization, and a front-end module for user interaction. DebtViz not only makes the management of SATD more efficient but also provides in-depth insights into the state of SATD within software systems, fostering informed decision-making on managing it. The scalability and deployability of DebtViz also make it a practical tool for both developers and managers in diverse software development environments. The source code of DebtViz is available at https://github.com/yikun-li/visdom-satd-management-system and the demo of DebtViz is at https://youtu.be/QXH6Bj0HQew.**

***Index Terms*—self-admitted technical debt, technical debt management, technical debt visualization**

I. Introduction

Technical debt, a prevalent concept in software development, expresses the trade-offs often made between ideal development practices and short-term project needs [1]. These trade-offs may involve hasty decisions, shortcuts, or less-than-ideal solutions geared toward expediting feature implementation or reducing development time. While these quick-fixes may serve immediate needs, they can negatively impact long-term software maintenance. In certain situations, consciously incurring technical debt can offer short-term advantages, provided it is managed effectively [2]. However, un-controlled accumulation of such debt often evolves into formidable maintenance challenges over time. Thus, adept management of both intentional and unintentional technical debt is pivotal to maintain software quality and restrict the increasing cost of change [3].

Self-Admitted Technical Debt (SATD) is a specific variant of technical debt, where developers voluntarily document technical debt within various software artifacts like source code comments, commit messages, issue tracking systems, or pull requests [4], [5]. For example, a developer might mention pending tasks within a code comment like *"we need to remove the dead code"*, or acknowledge a method's complexity with *"this method is hard to understand and needs to be simplified"*. Systematically detecting SATD helps to make such items explicit to developers and managers, assists in formulating plans for their resolution and thereby contributes to enhancement of maintainability and evolvability.

Existing research in this field, primarily focuses on identifying SATD using source code comments [6], [7]. More recent studies also explore SATD detection from other software artifacts [8]–[10], such as issue tracking systems. Despite these advances, practical tools to assist developers in detecting and managing different types of SATD are markedly absent. While research studies present several machine learning models, there is only one Eclipse plugin proposed for identifying and presenting SATD in code comments [11]; even this tool is constrained to handling SATD in code comments and lacks a comprehensive dashboard for system-wide SATD monitoring. Thus, there is no comprehensive tool, capable of gathering, analyzing, visualizing and monitoring SATD from multiple software artifacts.

This paper introduces DebtViz, a SATD management tool capable of automatically detecting various types of SATD in source code comments and issue tracking systems using a Convolutional Neural Network-based approach. In addition to identifying SATD, DebtViz generates and visualizes statistics on SATD items within software repositories. Finally, the tool also serves as a real-time monitor for SATD: it constantly scans the software artifacts of a repository, updates the statistics, and visualizes the current state of SATD. The four aforementioned functionalities map to four of the technical debt management activities as defined by Li *et al.* [12]: TD *identification*, *measurement*, *representation* and *monitoring*.

The architecture of DebtViz is comprised of three main components: a back-end service, an SATD classifier, and a front-end module. The **back-end service** collects data from Git repositories and issue tracking systems, and subsequently pre-processes the data and populates it into a dedicated database. The SATD classifier, the core part of the tool, employs a pre-trained machine learning model [10] to classify the types of SATD (i.e., code/design, test, documentation, and requirement debt), following the classifications suggested in prior work [10]. Upon identification of SATD items, the classifier applies the deconvolution technique [7] to extract

Fig. 1. The dashboard for SATD in code comments

keywords that indicate the SATD classification. The **front-end module** of DebtViz serves as the interface between the user and the tool, communicating with the back-end service to offer comprehensive SATD dashboards to the tool users. It also provides a specialized browser for both issue trackers and source code. DebtViz is designed for ease of deployment, scalability, and user-friendly interaction. It supports the management of SATD, helping developers and managers gain a thorough understanding of SATD within their software systems.

The paper's structure is as follows: Section II discusses related work. Section III offers an expansive overview of our DebtViz tool, while Section IV outlines its architecture. Section V shows the results of a preliminary evaluation. Conclusions and future directions are presented in Section VI.

II. Related work

The first work exploring SATD was conducted by Potdar and Shihab [4] in source code comments. They analyzed four open-source projects and found that SATD comments were present in 2.4% to 31% of source files. Interestingly, they found that only a portion of SATD comments, ranging from 26.3% to 63.5%, were resolved after being documented. Maldonado and Shihab [13] extended this foundational work by refining the classification of SATD into five distinctive categories, namely design, requirement, defect, documentation, and test debt. This classification was achieved by meticulously examining 33,000 code comments from five open-source projects. Their results highlighted design debt as the most pervasive form of SATD, contributing to 42% to 84% of the categorized cases.

Subsequent to these initial explorations in the sphere of SATD, a considerable body of research has pivoted towards devising methods for automating SATD detection. Several machine learning methodologies [7], [9], [10], [14] have been utilized to detect diverse types of SATD instances from various sources. Ren *et al.* [7] proposed a Convolutional Neural Network-based method to improve SATD detection's accuracy and explainability, particularly enhancing cross-project prediction. Similarly, Li *et al.* [9] generated a dataset of 4,200 issues from seven open-source projects and proposed a machine learning approach to detect SATD in issue tracking systems, outperforming baseline methods, benefiting from knowledge transfer, and extracting intuitive SATD keywords. Li *et al.* [10] also proposed an automated SATD identification approach that leveraged a multitask learning technique to analyze multiple sources, including source code comments, commit messages, pull requests, and issue tracking systems. Finally, Guo *et al.* [15] introduced a straightforward heuristic approach for SATD identification, proving it to perform similarly or even better than existing methods.

Further research efforts have focused on creating tools to facilitate SATD management. Liu *et al.* [11] introduced a tool called SATD detector, capable of automatically detecting SATD comments using text mining techniques, and highlighting, listing, and managing detected comments in an integrated development environment. This tool was designed with a back-end Java library and an Eclipse plug-in as its front-end. Further

expanding the toolbox for SATD management, Phaithoon *et al.* [16] presented a GitHub bot specifically designed to handle issue-related *On-hold SATD*, a situation in which developers delay proper implementation due to issues in the project issue tracker that are pending. This bot leverages machine learning techniques to automatically detect On-hold SATD comments in source code and identify the referenced issues. Upon resolution of the referenced issues, the bot notifies the developers accordingly. In contrast to these earlier works, our research proposes a tool that: 1) focuses on identifying SATD from multiple sources, 2) offers a web-based dashboard to monitor SATD status in software systems, and 3) presents new visualizations of SATD data, such as heatmaps and SATD line charts.

III. An Overview of the DebtViz Tool

This section explains the main functionalities of the DebtViz tool through UI screenshots. Utilizing the pre-trained SATD detection model [10], DebtViz can automatically determine the types of SATD within code comments and issues in issue trackers. Upon completion of data collection and analysis from a specific software repository, the tool generates two distinct dashboards: one presents details of SATD within code comments (subfigure A in Fig. 1), while the other elaborates on SATD occurrences within issue tickets (subfigure B in Fig. 1).

The dashboard depicted in subfigure A in Fig. 1 displays the variety of SATD types (i.e., code/design, test, requirement, and documentation debt) in code comments in the form of a pie chart, providing developers with a broad perspective of SATD categories prevalent in the software system. A secondary pie chart illustrates the distribution of SATD instances across different types of code comments, namely, inline, multi-line, block, and documentation block comments. Additionally, a line chart tracks the evolution of SATD in the system by representing the count of SATD items over time.

Similarly, the dashboard depicted in subfigure B in Fig. 1 serves to portray the variety of SATD categories prevalent within issue tracking systems. As can be seen, the first two pie charts show the number of different types of SATD (i.e., code/design, test, requirement, and documentation debt) in the issue summary and issue description. Moreover, it enumerates the instances of SATD occurring in different types of issues, such as tasks or bugs. This detailed view allows developers to discern patterns of SATD accumulation within specific issue types, thereby aiding in strategic response planning. Furthermore, the dashboard visualizes SATD items based on their status, for instance, distinguishing between open and closed issues.

To augment the understanding of SATD distribution across diverse modules, DebtViz generates comprehensive heatmaps, as exemplified in subfigure C in Fig. 1. For example, we observe that the */src/test/* directory contains 15 SATD items out of a total of 59 detected within the entire system. This functionality extends to various types of SATD, thereby offering developers and managers a visual representation of SATD dispersion throughout their software systems.

An additional key feature of DebtViz is a file browser functionality (subfigure D in Fig. 1). For example, we discern that *LoadTestCase.java* contains SATD within a total of 20 comments, with code/design debt slightly outweighing test debt. Conversely, there are no SATD items found within *logTestCase.java*. This enables a thorough enumeration of distinct SATD types across individual files and folders, thereby visualizing the distribution of varied SATD types across the software system.

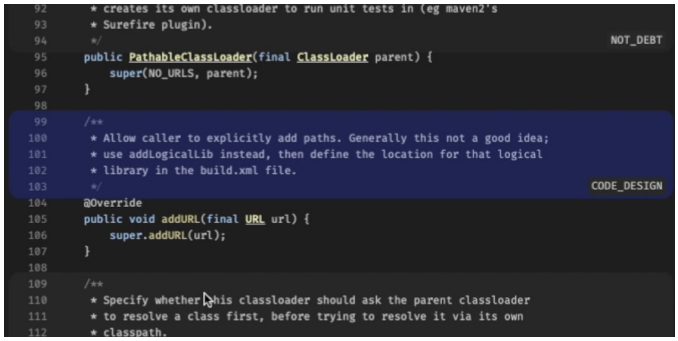

Fig. 2. The code viewer presenting the classification of different types of SATD or non-SATD for each code comment

Upon selection of a specific source code file, DebtViz reveals the source code accompanied by a classification of SATD types for each code comment, as depicted in Fig. 2. The tool distinctly highlights code comments classified as SATD, with the associated classifications such as code/design debt clearly marked. This feature facilitates developers in pinpointing the exact location of SATD within the source code.

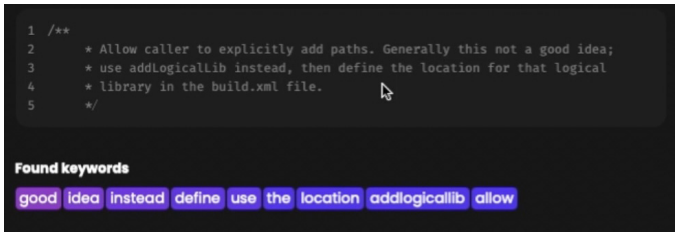

Fig. 3. Upon clicking the identified SATD code comments, the corresponding keywords highlighting the presence of SATD are displayed

Finally, DebtViz enables developers to dig deeper into the SATD code comments: when they select a code comment classified as SATD, DebtViz underscores the corresponding keywords that signal the presence of SATD, as demonstrated in Fig. 3. By drawing attention to these keywords, developers are given insight into the potential root causes of the technical debt item.

IV. System Architecture

The DebtViz tool architecture, as depicted in Fig. 4, comprises three primary components that will be elaborated in the following sub-sections: the back-end, the front-end, and the classifier.

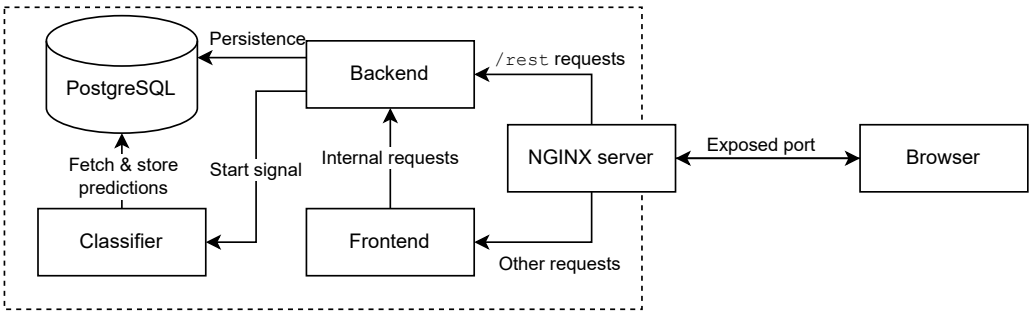

Fig. 4. The overview architecture of DebtViz

A. Back-end service

This is the backbone of the DebtViz tool, and was developed utilizing the Spring framework. This module is entrusted with scanning Git repositories for code comments and JIRA boards for issue tickets. Upon extraction, the data is systematically stored in a PostgreSQL database.

The back-end service effectively leverages the JGit project, a Java implementation of the Git version control system, to store and manage files along with their associated revisions. For the extraction of comments from source code files, we employed ANTLR (ANother Tool for Language Recognition). ANTLR's grammar parsing capabilities facilitate the creation of simple grammars customized to the diverse types of comments we encounter in the code.

The process of scanning a JIRA project requires interfacing with the JIRA server's REST API, a well-documented API with numerous open-source clients available. In this project, we opted for the Jira REST Java Client (JRJC).

B. Front-end service

The front-end module of DebtViz is assigned with presenting an interactive and responsive user interface. This module communicates with the back-end service, primarily by transmitting HTTP requests. To create an engaging, high-performance UI, the front-end module was constructed using the versatile React library along with the Next.JS framework. The functionalities and UI of this module were discussed and presented more extensively in Section III.

C. SATD classifier

The SATD classifier module scans the database for unclassified issues and comments, retrieving each entry for classification. Using a pre-trained machine learning model [10], it predicts the type of SATD (i.e., code/design, test, requirement, and documentation debt) for each unclassified entry. Specifically, the pre-trained machine learning model leverages the multitask learning technique [17] in combination with Text-CNN [18]. This model is capable of detecting SATD from multiple sources, namely code comments, commit messages, issue trackers, and pull requests. We note that DebtViz currently uses only code comments and issues, as these are the two most popular sources; commit messages and pull requests will be covered in the tool's next version. In cases where the data is classified as SATD, the deconvolution technique [7] is employed to extract the likely causative keywords. For instance, given a comment such as "todo: we need to remove the dead code", the extracted keywords could be "todo" and "dead code". Once the extraction is complete, the prediction is saved back into the database. The SATD classifier module incorporates a straightforward Flask server, for communication with the back-end service. This setup ensures real-time data processing and classification, keeping SATD data up to date.

V. Preliminary Evaluation

A. Evaluation setup

In order to obtain some preliminary evidence on the usefulness of the DebtViz tool, we designed a small-scale study involving six developers. Specifically, we acquired their feedback on the use of DebtViz via a targeted survey (see the replication package [1]). The survey revolved around three main focal points: *accuracy* pertains to the correctness of SATD detection by the tool; *awareness* focuses on how well the tool helps users understand the state and distribution of SATD in the system; and *effectiveness* assesses how much the tool assists users in managing SATD. The questions were designed to not only provide individual feedback on each source (code comments or issue trackers) but also on using both sources.

We analyzed the results by first studying each focal point and then comparing the individual and combined scores of the two data sources (code comments and issue trackers). This comparative analysis served to identify the efficacy of these data sources both independently and in tandem, offering insight into their individual strengths and the synergistic effects when used together.

B. Results

In Fig. 5, we provide a graphical representation of the average scores for each focal point. Regarding *accuracy*, we found that the visualization of SATD in source code comments achieved a high score of 4.25 out of 5. Participants reported only a few instances of inaccurate classification, primarily attributed to misleading keywords in comments. Meanwhile, the visualization of JIRA issues garnered a good average score of 4 out of 5. Participants reported occasional discrepancies

[1] https://github.com/yikun-li/visdom-satd-management-system

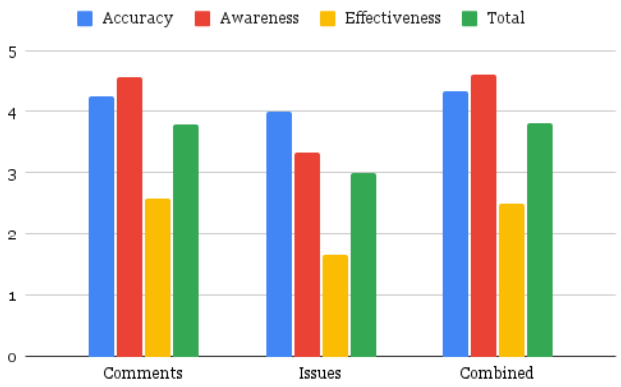

Fig. 5. Evaluation results

between the classification of the summary and description, once again tracing the errors back to certain keywords. The combined accuracy, however, was rated even higher at 4.33 out of 5, reflecting the tool's robust capacity to detect SATD in software projects.

Turning our attention to the *awareness* generated by the tool, participants noted that they discovered forgotten areas of interest within source code comments. The SATD visualization in comments scored an impressive average of 4.55 out of 5, with participants recognizing the potential value of tracking SATD over a longer period. Visualization of SATD in issues received a lower score of 3.33, as participants felt that issues were already well-sorted and organized within JIRA. Overall, the tool scored highest in enhancing awareness, reaching 4.61 out of 5.

The final focus point, *effectiveness*, received a lower rating from participants, with an average score of 2.58 out of 5 for comments and 1.66 out of 5 for issues, culminating in a combined average of 2.5. The participants argued that the visualized SATD had already existed in their work for a longer time, thus it did not have a high priority for refactoring. They also stated that they already prioritized SATD items using their issue tracker, so using our tool would not effectively assist in this task.

Finally, when evaluating the combined total rating of both sources, the average score rose to 3.81, indicating a slight advantage when utilizing both sources concurrently, compared to the separate scores of 3.80 and 3.00 for the comments and issues respectively.

VI. Conclusion

This paper presented DebtViz, a tool designed specifically to detect, classify, monitor and visualize various types of SATD in source code comments and issue tracking systems. The system comprises a back-end service, a SATD classifier, and a front-end module for interactive visual exploration. DebtViz aids in the management of SATD by providing an overview of the current state of SATD within software systems, which in turn supports informed decision-making regarding technical debt management. Potential areas for future enhancement include expanding the range of software artifacts (e.g., pull requests and commit messages) considered in SATD detection and refining the machine learning models for improved classification accuracy. As a contribution to the field of SATD management, DebtViz offers an integrated, multifaceted approach to identifying and visualizing SATD across multiple sources.

References

[1] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162)," *Dagstuhl Reports*, vol. 6, no. 4, pp. 110–138, 2016.

[2] E. Allman, "Managing technical debt," *Communications of the ACM*, vol. 55, no. 5, pp. 50–55, 2012.

[3] E. Lim, N. Taksande, and C. Seaman, "A balancing act: What software practitioners have to say about technical debt," *IEEE software*, vol. 29, no. 6, pp. 22–27, 2012.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] K. Dai and P. Kruchten, "Detecting technical debt through issue trackers." in *QuASoQ@ APSEC*, 2017, pp. 59–65.

[9] Y. Li, M. Soliman, and P. Avgeriou, "Identifying self-admitted technical debt in issue tracking systems using machine learning," *Empirical Software Engineering*, vol. 27, no. 131, Jul 2022.

[10] ——, "Automatic identification of self-admitted technical debt from four different sources," *Empirical Software Engineering*, vol. 28, no. 65, Apr 2023.

[11] Z. Liu, Q. Huang, X. Xia, E. Shihab, D. Lo, and S. Li, "Satd detector: a text-mining-based self-admitted technical debt detection tool," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, 2018, pp. 9–12.

[12] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, vol. 101, pp. 193–220, 2015.

[13] 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.

[14] Z. Guo, S. Liu, J. Liu, Y. Li, L. Chen, H. Lu, and Y. Zhou, "How far have we progressed in identifying self-admitted technical debts? a comprehensive empirical study," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 4, pp. 1–56, 2021.

[15] Z. Guo, S. Liu, J. Liu, Y. Li, L. Chen, H. Lu, Y. Zhou, and B. Xu, "Mat: A simple yet strong baseline for identifying self-admitted technical debt," *arXiv preprint arXiv:1910.13238*, 2019.

[16] S. Phaithoon, S. Wongnil, P. Pussawong, M. Choetkiertikul, C. Ragkhitwetsagul, T. Sunetnanta, R. Maipradit, H. Hata, and K. Matsumoto, "Fixme: A github bot for detecting and monitoring on-hold self-admitted technical debt," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 1257–1261.

[17] 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.

[18] 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.