

# Research on Software Project Developer Behaviors with K-means Clustering Analysis

Xiaozhou Li

Tampere University  
Kalevantie 4 33100, Tampere, Finland  
xiaozhou.li@tuni.fi

**Abstract.** Research on technical debt and community smell have drawn increasing attention in the academia of software engineering in the latest decade. Furthermore, data mining methods have been widely applied in the very domain as well. However, limited studies have contribute to the understanding of software project community using data mining methods, especially regarding the analysis of developer behaviors. Using K-means clustering, this study provides a preliminary analysis on the classification of open source software project developers based on the statistics of their behaviors related to technical debts. The results show that developers can be categorized into three different behavior groups, including, *Veterans*, *Vulnerability Creators*, and *Fault Inducers/Commoners*.

**Keywords:** Developer Behaviors · Data Mining · K-means · Clustering · Technical Debt · Code Smell · Community Smell

## 1 Introduction

Technical debt (TD) refers to the technical compromises made postponing software maintenance activities that can yield short-term benefit but may sabotage the long-term health of a software system [7]. Many studies have contributed to the domain of TD management in terms of a number of perspectives [18]. Code smell, as a type of TD and also a TD identification technique, has been also studied regarding its relation to fault induction and impact on maintenance efforts [27, 37, 12, 22, 29]. However, technical factors, e.g., TD, are not always the key factor for software failures. Social and technical debt, as well as code and community smells are deeply connected [24, 33]. Thus, investigating the software projects as communities and research on developer behaviors shall contribute to the understanding of the social factors towards software success [21, 11]. On the other hand, towards the improvement of software productivity and quality, data mining methods have been increasingly applied in various sub-domains [35]. Regarding TD and code smell, many studies have also adopted data mining methods, towards various perspectives [10, 6, 14, 26]. However, limited studies have contributed to investigating the phenomena of community smell using data mining methods.

This study focus on the analysis of project developers based on their TD related behavior statistics, answering the research questions of *RQ1. What are the different kinds of project developer behaviors?* and *RQ2. How to classify the project developers based on their TD related behaviors?*. It aims to provide an approach to classify the software project developers using K-means clustering. The data used herein is adopted from the Technical Debt Dataset, which provides the measurement of the over 100k commits of 33 projects [15]. This study aims to contribute to the understanding of the open source project developer groups and the future studies on community smell.

The reminder of the article is organized as follows. Section 2 introduces the related work regarding TD, code and community smell, and using data mining methods in such domains. Section 3 describes the data used hereby. Section 4 introduces the method used to analyze the data while Section 5 presents the results. In addition, Section 6 provides further discussion on the limitation and future work of this study. Section 7 concludes the article.

## 2 Related Work

Regarding TD in the domain of software engineering, many studies have been conducted regarding its identification [37], measurement and monitoring [27, 20], and management [4, 17]. Specially regarding code smell, which is closely related to TD, has also been widely studied regarding the above mentioned perspectives [23, 29]. Studies on community smell emerge from the research on software communities evolution [8] and the contrast concept of social debt towards the TD counterpart [32, 31]. Focusing on the social factors toward community smells, recent empirical studies have contributed to the research on the effect of community diversity [5] as well as the ways of promoting community inclusiveness [9]. The general research regarding the social aspect of software community dates back to earlier [11]. Regarding categorizing developer behaviors, Nakakoji et al. propose an 'onion model' as well as a developer classification including peripheral developers, active developers, core members and project leaders [21]. Yu and Ramaswamy also present a classification of core and associate project member based on interaction frequency [36].

On the other hand, data mining methods have been applied in the research on TD as well. Many studies have used natural language process (NLP) techniques in self-admitted TD detection and identification [28, 14]. Agarwal et al. apply multiple machine learning methods in developing a contextual learning system tackling TD issues [1]. Baylor et al. propose a TensorFlow-based general-purpose machine learning platform to standardize the components, simplify the platform configuration and reduce the TD of platform creation and maintenance [3]. However, the studies regarding community smell analysis with data mining methods are still limited.

### 3 Data Description

The dataset used in this study is the Technical Debt Dataset originally presented by Lenarduzzi et al. [15]. It selected 33 Java projects from the Apache Software Foundation (ASF) repository<sup>1</sup>, which are older than three years and developed in Java, contain more than 500 commits and more than 100 classes, and use Jira issue tracking systems with at least 100 issues. The data collection tools include PyDriller [30], Ptidej [16], Refactoring Miner [34] and SonarQube<sup>2</sup>. Specifically, to identify the fault-inducing commits from a project’s version history, the SZZ algorithm is used via the implementation of OpenSZZ [25].

The Technical Debt Dataset contains the following tables shown in Table 1.

**Table 1.** The Technical Debt Dataset Tables

Table Name	Description
COMMITTS	Commit information retrieved from the git log
COMMITTS CHANGES	Changes performed in each commit
FAULT INDUCING COMMITTS	Results from the execution of SZZ algorithm
JIRA ISSUES	Jira issues for the analyzed projects
PROJECTS	Links to the GitHub repository and Jira issue tracker
REFACTORING MINER	List of refactoring activities
SONAR ISSUES	SonarQube issues, anti-patterns and code smells
SONAR MEASURES	Measures SonarQube analyses from the commits
SONAR RULES	Rules monitored by SonarQube

As this study focus on the analysis of project developer behaviors, only the tables directly related to faults, issues and TD are selected, including *Commits*, *Fault inducing commits* and *Sonar issue*. The *Commits* table contains 128375 commits ranging from 2000-10-01 to 2018-11-01 committed by 915 unique developers (identified by committerIDs). The *Fault inducing commits* table contains the 27340 unique faults with the according inducing and fixing commits identified by the CommitHash, via which the developers who induce or fix the faults can be identified. The *Sonar issue* table contains the 1949899 issues’ information regarding the developers who create the issues via their commits, as well as the according severity, TD, and issue types (including code smell, bug, and vulnerability). Therein, 1869397 reported issues are code smell (95.9%) when bug and vulnerability cover respectively 57812 (3.0%) and 22690 (1.1%)

## 4 Method

### 4.1 Preprocessing

Focusing on identify each developer’s project committing behavior, the originally selected datasets are processed towards a collection of individual developers with

<sup>1</sup> <http://apache.org>

<sup>2</sup> <https://www.sonarqube.org/>

the according set of behavior data as features. Firstly, a set of unique developers are identified by concatenating the ‘CommitterId’ column of the *Commits* table and the ‘author’ column of the *Sonar issue* table. By doing so, 407 unique developers are selected.

Thereafter, for each selected developer, the calculated features regarding his/her committing behavior include *Commit number*, *Active days*, *Induced fault number*, *Fixed fault number*, *TD*, *Issue Number*, *average issue severity*, *Code smell issue rate*, *Bug issue rate*, and *Vulnerability issue rate*. Therein, *Commit number* is counted directly from the *Commit* table. *Active days* are the day counts from each developer’s first commit date to the last. *TD* is the sum of those of the developer related issues recorded in the *Sonar issue* table, from which issue number and the rate of each issue type for each developer can be also calculated. To calculate the average severity of issues for each developer, the severity levels (i.e., Blocker, Critical, Major, Minor, and Info) are quantified into ratings from 5 to 1 (5 being the most severe).

	id	commitsPD	faultsInducedPD	faultsFixedPD	debtPD	issuesPD	severity	csRate	bugRate	vulRate
0	a760104@aetna.com	0.041	0.000	0.000	0.075	0.233	0.636	0.524	0.0	0.476
1	aaron.dossett@target.com	0.154	0.000	0.000	0.001	0.002	0.603	1.000	0.0	0.000
2	abaranchuk@hortonworks.com	0.022	0.000	0.032	0.000	0.001	0.442	1.000	0.0	0.000
3	abaranchuk@hortonworks.com	0.040	0.009	0.032	0.000	0.000	0.500	1.000	0.0	0.000
4	adrian@apache.org	0.003	0.000	0.000	0.000	0.000	0.643	0.286	0.0	0.714

**Fig. 1.** Sample of the Developer Dataset

Furthermore, due to the difference in developers’ active periods, the *Commit number*, *Induced fault number*, *Fixed fault number*, *TD*, and *Issue Number* will be firstly normalized by the *Active days*. Then the above mentioned data, as well as the *average issue severity*, will be further normalized into values in  $[0, 1]$ . As the three issue type rates are originally between 0 and 1, the according values will remain. A sample of the preprocessed dataset is shown as Figure 1.

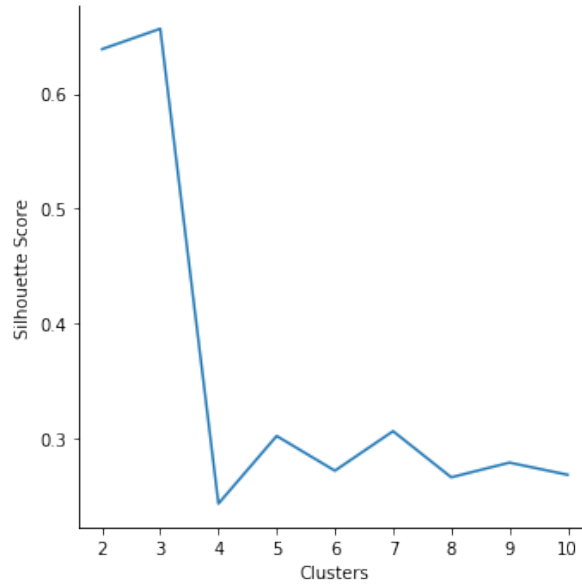
## 4.2 K-means Clustering

In order to classify the unlabelled data shown above, in this study K-means clustering is used. K-means clustering was used to extract clusters from the dataset. K-means, as a partitional clustering method, has been widely adopted in various data mining related domains due to its ease of implementation and efficiency in application [13]. K-means clustering takes in an  $m * n$  data matrix (i.e.,  $m$  data points,  $n$  features) to create  $k$  clusters. The cluster number  $k$  must be pre-defined when each data point will be assigned to one of the  $k$  clusters. The iterative process starts by assigning  $k$  random data points each to one cluster as the cluster center. Each of the remaining  $m - k$  data points will be assigned to the cluster whose center is closest to it. Thereafter for each iteration, the

center of the obtained  $k$  clusters will be relocated when each data point will be adjusted to the cluster of the closest newly defined center. The  $k$  clusters are finalized when no data point will change clusters. For this study, the number of clusters  $k$  is determined using the Silhouette Analysis [19] when the initial seeding is optimized using k-means++ algorithm [2].

## 5 Results

The optimized number of clusters  $k$  is determined by the Silhouette Analysis. The average Silhouette score is calculated for each  $k$  from 2 to 10, with the result shown in Figure 2.

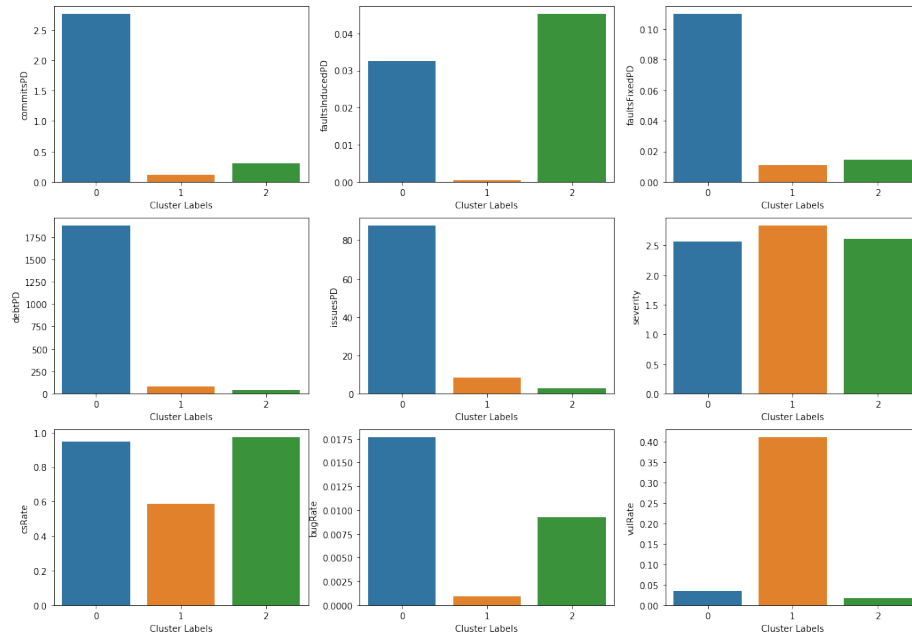


**Fig. 2.** The Average Silhouette Scores of Different Cluster Number

Accordingly, the highest score obtained from 2 to 10 clusters is 0.6567 at  $k = 3$ , which is selected as the number of clusters. With the selected  $k$ , the K-means clustering is performed on the preprocessed data, obtaining three clusters containing respectively 20, 12, and 375 data points. Differences between the clusters of developers regarding each feature can be seen in Figure 3.

By observing such results, the different developer behaviors (i.e., each cluster) can be summarized as follows.

*Veteran (Cluster 1)* Veteran developers contribute the highest number of commits per active day (avg. 2.75 compared to 0.12 of Cluster 2 and 0.3 of Cluster 3).



**Fig. 3.** Feature Differences for Clusters

With a relevantly high fault inducing rate (avg. 0.03), Veterans have the highest fault fixing rate (nearly 10 times the others). Importantly, they create the highest TD (avg. 1877.96 compared to 82.83 and 42.48) and the number of issues (avg. 87.58 compared to 8.44 and 2.88) of all developer groups. Despite only 20 out of 407 developers belong to this group, they inject the most issues (923949), including 883549 code smell issues, 10449 bugs, and 29951 vulnerability issues. Nonetheless, the average severity of their created issues is the lowest, but cannot be significantly distinguished. 94 percent of the issues created are code smells. Surprisingly, Veterans have the lowest active day duration (avg. 277.5 days).

*Vulnerability Creator (Cluster 2)* Vulnerability Creators have the highest vulnerability issue rates of all groups (avg. 0.41). Accordingly, they have also the lowest code smell rate and bug rate. Vulnerability Creators have also the lowest commits per active day (avg. 0.12), lowest fault inducing (avg. 0.0004) and fixing rate (0.01). However, the average issue severity level of them is the highest among the groups. They have, on average, nearly 3 times the active period length compared to Debt Creators (avg. 725.0 days). 12 out of 407 developers belong to this group. However, compare to the other behavior groups, they inject the lowest number of issues in total (6475).

*Fault Inducer/Commoner (Cluster 3)* Fault Inducer/Commoner is statistically the most common developer behavior group, as 375 out of the 407 developers

belong to this behavior group. Compared to the other groups, they have the highest fault inducing rate (avg. 0.05 compared to 0.03 and 0.0004) while the similarly low fault fixing rate compared to Vulnerability Creators. On the other hand, they have the lowest TD and number of issues created but the highest code smell percentage. They also have the longest active period length of all groups (avg. 1061 days). Despite being the majority based on behaviors, the total number of issues (620241) injected by this behavior group is less than that of the 20 veteran developers, as well in each issue type.

To summarize, project developers can be classified into three different developer behavior groups, including *Veterans*, *Vulnerability Creators*, and *Fault Inducers/Commoners*. The most common developers are the Fault Inducers/Commoners, who most likely induce faults despite of the low commit rates. But the Commoners seldom create TD and issues. A small group of developers (i.e., Veterans) create the most TD and issues (over 40 times those of the Commoners). They also the highest commit rates, fault fixing rates and the lowest active duration. The third group of developers (i.e., Vulnerability Creators) have the lowest commit rate and fault inducing rate, but twice the TD created compared to the Commoners. They also have the highest vulnerability issue creation rate, despite of the lowest issue injection number.

## 6 Discussion

This study provides a preliminary analysis on software developer classification based on their committing behaviors using K-means clustering. It contributes to the larger domain of data-driven software engineering [35]. By classifying the different project developers and identifying their similarities regarding committing behaviors, proactive mechanisms can be deployed for different developers accordingly towards tackling issues like code smell and community smell effectively.

One of the limitations of this study is that only limited perspectives of the developer behavior are taken into account. More details of the sonar issues contained in the *Sonar measures* table can be mapped to each commit in the *Commits* table, which leads to adding more features to the developer data points. Furthermore, despite that K-means being the most commonly adopted clustering method, other clustering methods, e.g., Agglomerative Hierarchical Clustering and Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [13] can be used in the future for comparison. Evaluation of the obtained clustering results is not conducted in this study. It can be done using purity metrics [38] when ground truth of the developer labeling is obtained. In addition, a worth noticing phenomena in the original dataset is that, when the CommitterID in *Commits* table and that of *Commit Changes* table are mapped via CommitHashes, all the mappings are not unique (meaning one developer email can be mapped to more than one developer names, and vice versa). It indicates the the results can be to some extent inaccurate by considering each CommitterID/email representing only one individual developer.

The future work of this study include the continuous exploratory research on software developer analysis using larger volume of data. Other clustering methods shall be used for comparison. Furthermore, based on the current clustering results, further investigation of each unique type of developer behavior can be conducted with the text mining on e.g., the messages in *Commits* table and *Sonar issue* table via NLP techniques.

## 7 Conclusion

This study presents a preliminary analysis on the classification of software project developers based on their committing behaviors using K-means clustering. Three unique developer behavior groups are obtained, including *Veterans*, *Vulnerability Creators*, and *Fault Inducers/ Commoners*. The different developer behaviors are distinguishable in terms of their differences in commit rates, fault inducing and fixing rates, technique debts, issues creating, active periods and so on. This study contributes to the larger picture of data-driven software engineering and also specifically the research on TD and community smell.

## References

1. Agarwal, A., Bird, S., Cozowicz, M., Hoang, L., Langford, J., Lee, S., Li, J., Melamed, D., Oshri, G., Ribas, O., et al.: Making contextual decisions with low technical debt. arXiv preprint arXiv:1606.03966 (2016)
2. Arthur, D., Vassilvitskii, S.: k-means++: The advantages of careful seeding. In: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. pp. 1027–1035. Society for Industrial and Applied Mathematics (2007)
3. Baylor, D., Breck, E., Cheng, H.T., Fiedel, N., Foo, C.Y., Haque, Z., Haykal, S., Ispir, M., Jain, V., Koc, L., et al.: Tfx: A tensorflow-based production-scale machine learning platform. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1387–1395. ACM (2017)
4. Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., MacCormack, A., Nord, R., Ozkaya, I., et al.: Managing technical debt in software-reliant systems. In: Proceedings of the FSE/SDP workshop on Future of software engineering research. pp. 47–52. ACM (2010)
5. Catolino, G., Palomba, F., Tamburri, D.A., Serebrenik, A., Ferrucci, F.: Gender diversity and women in software teams: How do they affect community smells? In: Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Society. pp. 11–20. IEEE Press (2019)
6. Codabux, Z., Williams, B.J.: Technical debt prioritization using predictive analytics. In: Proceedings of the 38th International Conference on Software Engineering Companion. pp. 704–706. ACM (2016)
7. Cunningham, W.: The wycash portfolio management system. ACM SIGPLAN OOPS Messenger 4(2), 29–30 (1993)
8. Datta, S., Sindhgatta, R., Sengupta, B.: Evolution of developer collaboration on the jazz platform: a study of a large scale agile project. In: Proceedings of the 4th India Software Engineering Conference. pp. 21–30. ACM (2011)



9. Ford, D., Milewicz, R., Serebrenik, A.: How remote work can foster a more inclusive environment for transgender developers. In: Proceedings of the 2nd International Workshop on Gender Equality in Software Engineering. pp. 9–12. IEEE Press (2019)
10. Fu, S., Shen, B.: Code bad smell detection through evolutionary data mining. In: 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). pp. 1–9. IEEE (2015)
11. Goeminne, M., Mens, T.: Analyzing ecosystems for open source software developer communities. *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry* pp. 247–275 (2013)
12. Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S.: A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering* **38**(6), 1276–1304 (2011)
13. Han, J., Pei, J., Kamber, M.: *Data mining: concepts and techniques*. Elsevier (2011)
14. Huang, Q., Shihab, E., Xia, X., Lo, D., Li, S.: Identifying self-admitted technical debt in open source projects using text mining. *Empirical Software Engineering* **23**(1), 418–451 (2018)
15. Lenarduzzi, V., Saarimäki, N., Taibi, D.: The technical debt dataset. In: The Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE’19) (Sept 2019). <https://doi.org/10.1145/3345629.3345630>
16. Lenarduzzi, V., Sillitti, A., Taibi, D.: A survey on code analysis tools for software maintenance prediction. In: International Conference in Software Engineering for Defence Applications. pp. 165–175. Springer (2018)
17. Letouzey, J.L., Ilkiewicz, M.: Managing technical debt with the sqale method. *IEEE software* **29**(6), 44–51 (2012)
18. Li, Z., Avgeriou, P., Liang, P.: A systematic mapping study on technical debt and its management. *Journal of Systems and Software* **101**, 193–220 (2015)
19. Lleti, R., Ortiz, M.C., Sarabia, L.A., Sánchez, M.S.: Selecting variables for k-means cluster analysis by using a genetic algorithm that optimises the silhouettes. *Analytica Chimica Acta* **515**(1), 87–100 (2004)
20. Marinescu, R.: Assessing technical debt by identifying design flaws in software systems. *IBM Journal of Research and Development* **56**(5), 9–1 (2012)
21. Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., Ye, Y.: Evolution patterns of open-source software systems and communities. In: Proceedings of the international workshop on Principles of software evolution. pp. 76–85. ACM (2002)
22. Olbrich, S., Cruzes, D.S., Basili, V., Zazworka, N.: The evolution and impact of code smells: A case study of two open source systems. In: 2009 3rd international symposium on empirical software engineering and measurement. pp. 390–400. IEEE (2009)
23. Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., De Lucia, A., Poshyvanyk, D.: Detecting bad smells in source code using change history information. In: Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering. pp. 268–278. IEEE Press (2013)
24. Palomba, F., Tamburri, D.A., Serebrenik, A., Zaidman, A., Fontana, F.A., Oliveto, R.: How do community smells influence code smells? In: 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion). pp. 240–241. IEEE (2018)
25. Pellegrini, L., Lenarduzzi, V., Taibi, D.: Openszz: A free, open-source, web-accessible implementation of the szz algorithm (2019). <https://doi.org/10.5281/zenodo.3337791>, <https://zenodo.org/record/3337791>

26. Rubin, J., Henniche, A.N., Moha, N., Bouguessa, M., Bousbia, N.: Sniffing android code smells: an association rules mining-based approach. In: Proceedings of the 6th International Conference on Mobile Software Engineering and Systems. pp. 123–127. IEEE Press (2019)
27. Seaman, C., Guo, Y.: Measuring and monitoring technical debt. In: Advances in Computers, vol. 82, pp. 25–46. Elsevier (2011)
28. da Silva Maldonado, E., Shihab, E., Tsantalis, N.: Using natural language processing to automatically detect self-admitted technical debt. *IEEE Transactions on Software Engineering* **43**(11), 1044–1062 (2017)
29. Sjøberg, D.I., Yamashita, A., Anda, B.C., Mockus, A., Dybå, T.: Quantifying the effect of code smells on maintenance effort. *IEEE Transactions on Software Engineering* **39**(8), 1144–1156 (2012)
30. Spadini, D., Aniche, M., Bacchelli, A.: Pydriller: Python framework for mining software repositories. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 908–911. ACM (2018)
31. Tamburri, D.A., Kazman, R., Fahimi, H.: The architect’s role in community shepherding. *IEEE Software* **33**(6), 70–79 (2016)
32. Tamburri, D.A., Kruchten, P., Lago, P., Van Vliet, H.: Social debt in software engineering: insights from industry. *Journal of Internet Services and Applications* **6**(1), 10 (2015)
33. Tamburri, D.A., Palomba, F., Serebrenik, A., Zaidman, A.: Discovering community patterns in open-source: A systematic approach and its evaluation. *Empirical Software Engineering* **24**(3), 1369–1417 (2019)
34. Tsantalis, N., Mansouri, M., Eshkevari, L.M., Mazinanian, D., Dig, D.: Accurate and efficient refactoring detection in commit history. In: Proceedings of the 40th International Conference on Software Engineering. pp. 483–494. ACM (2018)
35. Xie, T., Thummalapenta, S., Lo, D., Liu, C.: Data mining for software engineering. *Computer* **42**(8), 55–62 (2009)
36. Yu, L., Ramaswamy, S.: Mining cvs repositories to understand open-source project developer roles. In: Fourth International Workshop on Mining Software Repositories (MSR’07: ICSE Workshops 2007). pp. 8–8. IEEE (2007)
37. Zazworka, N., Izurieta, C., Wong, S., Cai, Y., Seaman, C., Shull, F., et al.: Comparing four approaches for technical debt identification. *Software Quality Journal* **22**(3), 403–426 (2014)
38. Zhao, Y., Karypis, G.: Criterion functions for document clustering: Experiments and analysis (2001)