csDetector: An Open Source Tool for Community Smells Detection

Nuri Almarimi
nuri.almarimi.1@ens.etsmtl.ca
ETS Montreal, University of Quebec
Montreal, QC, Canada

Ali Ouni
ali.ouni@etsmtl.ca
ETS Montreal, University of Quebec
Montreal, QC, Canada

Moataz Chouchen
moataz.chouchen.1@ens.etsmtl.ca
ETS Montreal, University of Quebec
Montreal, QC, Canada

Mohamed Wiem Mkaouer
mwm@se.rit.edu
Rochester Institute of Technology
Rochester, New York, USA

ABSTRACT

Community smells represent symptoms of sub-optimal organizational and social issues within software development communities that often lead to additional project costs and reduced software quality. Previous research identified a variety of community smells that are connected to sub-optimal patterns under different perspectives of organizational-social structures in the software development community. To detect community smells and understanding the characteristics of such organizational-social structures in a project, we propose csDetector, an open source tool that is able to automatically detect community smells within a project and provide relevant socio-technical metrics. csDetector uses a machine learning-based detection approach that learns from various existing bad community development practices to provide automated support in detecting related community smells. We evaluate the effectiveness of csDetector on a benchmark of 143 open source projects from GitHub. Our results show that the csDetector tool can detect ten commonly occurring community smells in open software projects with an average F1 score of 84%. csDetector is publicly available, with a demo video, at: https://github.com/Nuri22/csDetector

CCS CONCEPTS

• **Software and its engineering → Collaboration in software development**; • **Social and professional topics**;

KEYWORDS

Social debt, community smells, socio-technical factors

1 INTRODUCTION

Software engineering is by nature a social activity in which developers interact with each other to build a software product. The organizational social structure in a software development community (*e.g.*, the interactions and collaborations among developers), is an essential prerequisite for a successful software product. However, project community structure may face various communication and collaboration challenges, including different cultural backgrounds [12, 32], toxic communications, and differences in expertise levels or power distance [10, 11]. Such circumstances are known as *community smells* [34]. Community smells are tightly related to sub-optimal organizational and social issues that negatively impact the organizational health of the project. Prior research showed that community smells can be successfully detected using various heuristics such as social network connectivity characteristics, developers contributions, geographic dispersion, etc. [1, 2, 9, 38].

Community smells result into *social debt* [9, 33, 36] and often lead to software quality degradation [4, 28], an extra unforeseen cost to a software project by wasted resources (*e.g.*, time), and can even lead to project failures. Hence, detecting community smells is of crucial importance to keep a healthy development team and deliver quality software [1, 6, 36]. In our previous work [1, 2], we found that various social network connectivity metrics (*e.g.*, social network degree centrality, density and graph betweeness centrality, etc.) highly correlate with the presence of community smells. Therefore, these metrics could be used to effectively detect community smells. We believe that an early detection community smells can and help developers and project managers keep a healthy development community and re-organize the community structure.

In this paper, we present csDetector, a tool based on our previous work [1, 2]. The tool learns from a set of organizational-social symptoms that characterize the existence of a potential community smell using machine learning. csDetector first mines the change history of a software project to calculate various socio-technical metrics that will serve as features. Then, csDetector takes as input (1) the learnt community smells detection models, and (2) the list of mined socio-technical metrics. Then, it generates as output the list of calculated socio-technical metrics as well as the list of detected community smells. Currently, csDetector supports the detection

Table 1: Community Smells Types.

Definition	References
Organizational Silo Effect (OSE): This refers to the presence of isolated subgroups, and lack of communication and collaboration between community developers. As a result, this smell cause an extra unforeseen cost to a project by wasted resources (e.g., time), as well as duplication of code.	[33, 36, 38]
Black-cloud Effect (BCE): This reflects an information overload due to lack of structured communications due to limited knowledge sharing opportunities (e.g., collaborations, discussions, daily stand-ups, etc.), as well as a lack of expert members in the project that are able to cover the experience or knowledge gap of a community.	[36, 38]
Prima-donnas Effect (PDE): This smell appears when a team of people is unwilling to respect external changes from other team members due to inefficiently structured collaboration within a community.	[33, 36, 38]
Sharing Villainy (SV): This smell is caused by a lack of high-quality information exchange activities (*e.g.*, face-to-face meetings). The main side effect of this smell limitation is that community members share essential knowledge such as outdated, wrong and unconfirmed information.	[36]
Organizational Skirmish (OS): The OS is caused by a misalignment between different expertise levels and communication channels among development units or individuals involved in the project. The existence of this smell leads often to dropped productivity and affect the project's timeline and cost.	[36]
Solution Defiance (SD): The solution defiance smell occurs when the development community presents different levels of cultural and experience background, and these variances lead to the division of the community into similar subgroups with completely conflicting opinions concerning technical or socio-technical decisions to be taken. The existence of the SD often leads to unexpected project delays and uncooperative behaviors among the developers.	[36]
Radio Silence (RS): The ratio silence smell occurs when a high formality of regular procedures takes place due to the inefficient structural organization of a community. The RS community smell typically causes changes to be retarded, as well as a valuable time to be lost due to complex and rigid formal procedures. The main effect of this smell is an unexpected massive delay in the decision-making process due to the required formal actions needed.	[36]
Truck Factor Smell(TFS): The truck factor smell occurs when most of the project information and knowledge are concentrated in one or few developers. The presence of this smell eventually leads to a significant knowledge loss due to the turnover of developers.	[3]
Unhealthy Interaction (UI): This smell occurs when discussions between developers are slow, light, brief and/or contains poor conversations. It manifests with low developers participation in the project discussions (*e.g.*, pull requests, issues, etc.) having long delays between messages communications.	[30, 40]
Toxic Communication (TC): This smell occurs when communications between developers are subject to toxic conversations and negative sentiments containing unpleasant, anger or even conflicting opinions towards various issues that people discuss. Developers may have negative interpersonal interactions with their peers, which can lead to frustration and stress. These negative interactions may ultimately result in developers abandoning projects.	[17, 30, 40]

of ten common types community smells (cf. Table 1). csDetector has been designed to be easy to extend, *i.e.*, developers can easily calibrate the predefined detection models. Moreover, although csDetector currently detects 10 common community smell types, it is designed with a high level of flexibility to incorporate new community smell types easily, and also permits the customization of the existing smell detection models if needed as shown later in Section 3, where we discuss the tool's architecture.

To evaluate the performance of csDetector, we performed an experimental study on a set of 143 open source projects containing various types of community smells. The results indicate that csDetector can correctly detect community smells with an average F1 score of 84%.

Open source tool and documentation. csDetector is publicly available as an open source in our tool repository [21] including the (1) tool source code, along with (2) a demonstration video and (3) the documentation on how to use it.

2 BACKGROUND AND RELATED WORK

2.1 Community Smells

Community smells are defined as a set of sub-optimal organizational and social circumstances in the development team of a software project that influence the production, operation, and evolution of software [28, 34, 36]. Table 1 provides the definitions of ten common community smells that csDetector supports in its detection. We primarily focus on the mentioned smells since they are frequently encountered in the software industry and have a negative impact on the social structure of the project and shown to lead to software quality degradation [1, 3, 15, 20, 28, 28, 33, 36, 36, 40].

2.2 Related Work

Recent works on organizational social aspects in software engineering have identified various aspects around community smells, and proposed mechanisms to ease their detection.

Tamburri et al. proposed a tool called YOSHI to automate to detect organizational structure patterns across open source communities [37]. The proposed tool maps open-source projects onto community patterns, and introduces measurable attributes to identify organizational and social structure types and characteristics through formal detection rules. CodeFace is another tool developed as a Siemens product [13] and designed to identify developers communities based on building developer networks. Another tool called *Codeface4Smell* has been proposed later as an extension of CodeFace to identify community smells based on community metrics and statistical values to measure the quality and health characteristics of software development communities [38]. *Codeface4Smell* uses the development history and mailing lists to assess the social network among developers. Avelino et al. introduced a tool called *Truck Factor* [3] to measure information concentration within community members and support the software development community to deal with turnover of developers. The proposed approach estimates the truck factor values of Github projects based on historical information about developers contributions and collaborations from the commit log.

In particular, csDetector builds on top of our previous work [1, 2] where used various machine learning and rule-based based detection models to build detection rules for each community smell type [14, 22–27, 31]. Hence, our goal is build a comprehensive tool that can detect a variety of smells to help characterizing and analyze organizational-social structures in software development communities.

3 ARCHITECTURE

A high-level overview of the architecture of csDetector is depicted in Figure 1. csDetector is implemented as an open-source command line and it has two main modules (1) metrics extraction module and (2) smells detection module. First, csDetector starts by retrieving and analysing developers relationships of a given software development community using different software

development artefacts mined from the Version Control Systems. Initially, our tool extracts developers alias (A). Thereafter, these aliases are used to build social graph of the developers (B), and calculate sentiments-related metrics (C). Then, the social network graph is used to to calculate the socio-technical metrics quantifying the collaboration between the project developers. Finally, in the smells detection module, the extracted features are used by our pre-trained models (E) to detect any existing community smells. In the following subsections, we provide details for each module.

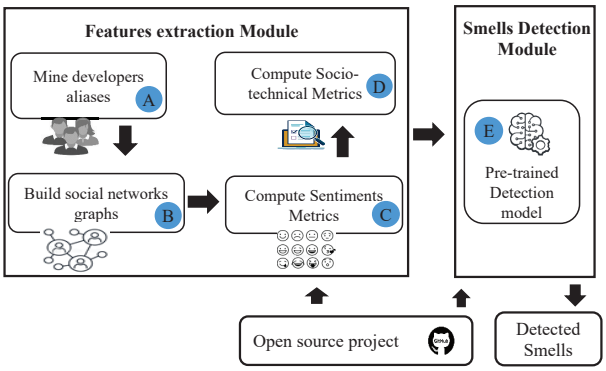

Figure 1: High-level architecture of csDetector

3.1 Features Extraction Module

Our metrics framework is built based on well-known organizational social metrics that are related to community smells and heavily exploited in previous studies [3, 7, 18, 19, 29, 30, 38, 40]. We also extended the organizational social metrics with a new set of metrics to capture more community-related proprieties that can be mined from the project's history. Such metrics analyze different aspects in software development communities including organizational dimensional, social network characteristics, developers collaborations, truck numbers, and sentiments aspects in developers communications. We calculate the set of features from a given project by analyzing its repository through commit information history and the discussions that are available in a version control system (*e.g.*, pull request and issues communications, commit messages, etc.). The following paragraphs elaborate the details of our tool modules.

Ⓐ **Mine developers aliases:** The author alias mining and consolidation consists of the following sub-steps [3]: (*i*) retrieval all unique dev-emails, where for each git commit has a dev-email associated with it, (*ii*) Retrieval of GitHub logins related to developers, (*iii*) similarity matching of emails and logins: by applying Levenshtein distance [3], all aliases are compared and, with a certain degree of threshold value at most one, consolidated, and (*iv*) replacement of author emails by their respective aliases [3]. The final transformation goes through all the commits once again and replacing original authors by their main alias. As a result, if there is a developer associated with commits with different names, we consider them as a single developer, and the output will be presented in a new aliases list. For example, "Bob.Rob" and "Bob Rob" are different names for a single developer associated to commits. Thus, we consider them as the same developer in a new aliases list as a single identical substitution.

Ⓑ **Build a social network graph:** Social networks analysis (SNA) have been used for studying and analyzing the collaboration and organization of developers who are working in teams within software development projects [16]. Our developers network model is based on socio-technical connections during software development. We devised different social network analysis metrics to describe the community structure and quantify various quality factors in a software project. csDetector builds a developer network from the version control system by tracking the change logs at three levels: (1) commits, (2) issues, and (3) pull requests. csDetector build the developers networks using the `networkx` Python library[1] in the form of graph where the nodes represent developers and edges are the connections between two developers, when they have participated in the same issue or in the same pull request discussions. Such social networks allows then to calculate the different social network metrics including the degree centrality, closeness centrality, network density, etc.

Ⓒ **Compute Sentiment Metrics:** We computed sentiment metrics using the state-of-the-art SentiStrength[2] tool which allows estimating the degree of positive and negative sentiments in short texts, even for informal language. SentiStrength assigns fixed scores for sentiment two polarizations:

- Negative: -1 (slightly negative) to -5 (extremely negative).
- Positive: 1 (slightly positive) to 5 (extremely positive).

SentiStrength is a lexical sentiment extraction tool based on a list of words. We used SentiStrenght to measure the sentiment of developers in commit comments, issue comments, and pull request comments (which often are short). Moreover, we used the Perspective API[3] to identify whether a comment is perceived as "toxic" in a given communication context. We also applied Stanford's politeness detector [8] to analyse politeness sentiments in the developers communications. For space limitations, we provide the list of sentiment metrics and their definitions in our tool website [21].

Ⓓ **Compute Socio-technical Metrics:** In this step, we use the collected artifacts and social network graph built in previous steps to compute a variety of socio-technical metrics [21]. These metrics analyze different aspects in software development communities including developer contributions, social network analysis, community metrics, geographic dispersion, formality metrics, truck numbers, communication metrics, and sentiment analysis metrics. All metrics are available online for future extension and replication on our tool website [21].

3.2 Smells Detection Module

Once the necessary socio-technical metrics are collected, the smells detection module uses pre-defined models to detect community smells for a given project's link in Github.

Ⓔ **Pre-trained Model:** We built decision tree-based models that were trained from real world instances of community smells in our prior work [2]. In particular, for each community smell type, a pre-trained model is used by csDetector to check if a given project is affected by that smell based on its collected features (*i.e.*, by the Features Extraction Module). The different models are loaded

[1] https://networkx.org/
[2] http://sentistrength.wlv.ac.uk/
[3] https://www.perspectiveapi.com

in a specific folder in our tool repository [21], namely, *Models*. This makes our tool flexible and extensible in such a way that the models can be updated or customized from the developers or new community smells can be considered by simply adding their corresponding pre-trained models in the models folder.

4 CSDETECTOR USAGE SCENARIO

csDetector takes as input the target project, *i.e.*, repository url in Github, then it automatically extracts the socio-technical metrics. Next, the extracted metrics are provided to the pre-trained model to detect existing smells. The detection strategy of each smell type is self-contained within its own pre-trained detection model. Finally, for each smell that is detected by its corresponding model, csDetector shows its name to the user through the command line interface, *e.g.*, detected smells with the last commit data of the target project.

csDetector Usage. As a command line tool, csDetector can be executed as executable file under windows by command line *devNetwork.exe* with the specific parameters. All necessary information to run our tool are provided in the csDetector website [1]. The command line is as follows:

```
devNetwork.exe <parameters>
```

where the different parameters are used in the following order:

```
-p <GitHub PAT (personal access token) used for querying the GitHub API>
-g <Google Cloud API Key used for authentication with Perspective API>
-r <GitHub repository URL to be analyzed>
-s <local directory path to the SentiStregth tool>
-o <Local directory path for analysis output>
-sd <The desired date to start analyzing a project YYYY-MM-DD>
```

After the detection process is complete, the detected smells will show up on the command line interface, and CSV files containing all the metrics values will be created on the local directory path for analysis output, as well as the social network graphs.

5 EVALUATION

The performance of the csDetector tool strongly depends on the detection approach it implements. As explained above, the introduced tool is based on our recent work [2]. In particular, we conduct an evaluation of csDetector on a dataset that consists of 143 GitHub large projects selected based on their community size, number of commits, and availability of their pull request and issue tracking systems in Github [21].

csDetector has successfully computed a variety of 52 socio-technical metrics including social network metrics, sentiments metrics, and communication metrics. We retrained the models in our experimental data as the final detection results depend on the performance of these pre-trained models. To generate the models, we used out-of-sample bootstrapping validation process repeated 100 times since it was showed to be stable for similar software engineering problems [39]. This process starts by randomly sampling with replacement N elements from the data set (in our case $N = 143$) and use it as a training set and we use the elements that does not occur in the training set to test the tool. Since the sampling process is done with replacement, on average 32.6% of elements would not figure in the training set. This process is repeated 100 times and we use the average of the obtained results to asses the performance of the tool. Note that, the models do not need to be trained for each dataset as we provide the default pre-trained models in our csDetector repository.

Table 2: The results for each smell type.

Smell Type	Accuracy	Precision	Recall	F1	AUC
OSE	0.91	0.91	0.87	0.89	0.96
BCE	0.87	0.88	0.82	0.84	0.94
OS	0.84	0.85	0.81	0.83	0.91
PDE	0.80	0.83	0.73	0.77	0.89
RS	0.90	0.91	0.86	0.89	0.96
SD	0.84	0.86	0.81	0.83	0.92
SV	0.83	0.85	0.81	0.82	0.91
TF	0.82	0.83	0.71	0.76	0.90
UI	0.89	0.90	0.88	0.89	0.94
TC	0.89	0.87	0.90	0.88	0.94
Average	**0.86**	**0.87**	**0.82**	**0.84**	**0.93**

To measure the performance of our tool, we used different classification metrics namely, accuracy, precision, recall, $F1$, and AUC [5]. Table 2 reports of our results for the ten considered community smell types (cf. Table 1) in terms of accuracy, precision, recall, F1, and AUC. The table presents the detection results for each smell type. As shown in the table, our tool achieves a high level of correctness with AUC ranging from 0.89 to 0.96 and average of 0.93. Moreover, we observe csDetector does not have a bias towards the detection of any specific smell type. As shown in the figure, csDetector achieved good performance and low variability in terms of the performance metrics. For instance, the accuracy is ranging from 0.80 to 0,91 and F1 is ranging from 0.77 to 0.89 across the 10 considered smell types. The highest F1 was obtained for the organisational silo effect (OSE) with 0.89 which heavily relies rely on the notion of developers social network and sub-groups. This higher performance is reasonable since the existing guidelines [28, 35–38] rely heavily on the notion of social network. But for smells such as the Prima-donnas Effect (PDE), the notion of social network is less important and this makes this type of smells harder to detect using such information.

6 CONCLUSION & FUTURE WORK

In this paper, we introduced csDetector, a tool to automatically detect community smells in a given software project, to help developers detecting and understanding social debt in their software projects. The csDetector tool is based on pre-trained community smells detectors using historical projects data (social network and communications among developers, developers contributions, sentiments analysis, etc.). We also described the architecture of csDetector, the ease of integrating new smell types into the tool. To evaluate our tool, we conducted a set of experiments on the performance of csDetector. The results show that our tool achieves a high performance in terms of F1 with an average of 84% on all the considered community smells types from 143 open source projects.

As future work, we plan to enhance the visualization part of our tool with a comprehensive dashboard that help developers track the evolution of the health of their development community and consider other aspects of social debt.

ACKNOWLEDGMENT

This work is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) discovery grant RGPIN-2018-05960.

REFERENCES

[1] Nuri Almarimi, Ali Ouni, Moataz Chouchen, Islem Saidani, and Mohamed Wiem Mkaouer. 2020. On the detection of community smells using genetic programming-based ensemble classifier chain. In *Proceedings of the 15th International Conference on Global Software Engineering*. 43–54.

[2] Nuri Almarimi, Ali Ouni, and Mohamed Wiem Mkaouer. 2020. Learning to detect community smells in open source software projects. *Knowledge-Based Systems* 204 (2020), 106201.

[3] Guilherme Avelino, Leonardo Passos, Andre Hora, and Marco Tulio Valente. 2016. A novel approach for estimating truck factors. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*. IEEE, 1–10.

[4] Nicolas Bettenburg and Ahmed E Hassan. 2010. Studying the impact of social structures on software quality. In *2010 IEEE 18th International Conference on Program Comprehension*. IEEE, 124–133.

[5] Gürol Canbek, Seref Sagiroglu, Tugba Taskaya Temizel, and Nazife Baykal. 2017. Binary classification performance measures/metrics: A comprehensive visualized roadmap to gain new insights. In *2017 International Conference on Computer Science and Engineering (UBMK)*. IEEE, 821–826.

[6] Gemma Catolino, Fabio Palomba, Damian A Tamburri, Alexander Serebrenik, and Filomena Ferrucci. 2020. Refactoring community smells in the wild: the practitioner's field manual. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Society*. 25–34.

[7] Stefano Invernizzi Elisabetta Di Nitto Damian A. Tamburri, Simone Gatti. 2016. Re-Architecting Software Forges into Communities: An Experience Report. In *JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS*. 1–26.

[8] Cristian Danescu-Niculescu-Mizil, Moritz Sudhof, Dan Jurafsky, Jure Leskovec, and Christopher Potts. 2013. A computational approach to politeness with application to social factors. *arXiv preprint arXiv:1306.6078* (2013).

[9] Manuel De Stefano, Fabiano Pecorelli, Damian A Tamburri, Fabio Palomba, and Andrea De Lucia. 2020. Splicing Community Patterns and Smells: A Preliminary Study. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. 703–710.

[10] Daniel J Greenhoe. 2016. Properties of distance spaces with power triangle inequalities. *arXiv preprint arXiv:1610.07594* (2016).

[11] Gert Jan Hofstede, Catholijn M Jonker, and Tim Verwaart. 2008. Modeling power distance in trade. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*. Springer, 1–16.

[12] Hannu Jaakkola. 2012. Culture sensitive aspects in software engineering. In *Conceptual Modelling and Its Theoretical Foundations*. Springer, 291–315.

[13] M. Joblin, W. Mauerer, S. Apel, J. Siegmund, and D. Riehle. 2015. From Developer Networks to Verified Communities: A Fine-Grained Approach. In *37th IEEE International Conference on Software Engineering (ICSE)*, Vol. 1. 563–573.

[14] M. Kessentini and A. Ouni. 2017. Detecting Android Smells Using Multi-Objective Genetic Programming. In *IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. 122–132.

[15] Mika Mäntylä, Bram Adams, Giuseppe Destefanis, Daniel Graziotin, and Marco Ortu. 2016. Mining valence, arousal, and dominance: possibilities for detecting burnout and productivity?. In *Proceedings of the 13th International Conference on Mining Software Repositories*. 247–258.

[16] Andrew Meneely and Laurie A. Williams. 2011. Socio-technical developer networks: should we trust our measurements? *2011 33rd International Conference on Software Engineering (ICSE)* (2011), 281–290.

[17] Alessandro Murgia, Marco Ortu, Parastou Tourani, Bram Adams, and Serge Demeyer. 2018. An exploratory qualitative and quantitative analysis of emotions in issue report comments of open source systems. *Empirical Software Engineering* 23, 1 (2018), 521–564.

[18] Nachiappan Nagappan, Brendan Murphy, and Victor Basili. 2008. The Influence of Organizational Structure on Software Quality: An Empirical Case Study. In *Proceedings of the 30th International Conference on Software Engineering* (Leipzig, Germany). 521–530.

[19] Martin Nordio, H Christian Estler, Bertrand Meyer, Julian Tschannen, Carlo Ghezzi, and Elisabetta Di Nitto. 2011. How do distribution and time zones affect software development? a case study on communication. In *2011 IEEE Sixth International Conference on Global Software Engineering*. IEEE, 176–184.

[20] Nicole Novielli, Fabio Calefato, and Filippo Lanubile. 2015. The challenges of sentiment detection in the social programmer ecosystem. In *Proceedings of the 7th International Workshop on Social Software Engineering*. 33–40.

[21] Moataz Chouchen Mohamed Wiem Mkaouer Nuri Almarimi, Ali Ouni. 2021. csDetector. https://github.com/Nuri22/csDetector.

[22] Ali Ouni. 2020. Search-based Software Engineering: Challenges, Opportunities and Recent Applications. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*. 1114–1146.

[23] Ali Ouni, Raula Gaikovina Kula, Marouane Kessentini, and Katsuro Inoue. 2015. Web service antipatterns detection using genetic programming. In *Annual Conference on Genetic and Evolutionary Computation (GECCO)*. 1351–1358.

[24] A. Ouni, M. Kessentini, K. Inoue, and M. Ó. Cinnéide. 2017. Search-Based Web Service Antipatterns Detection. *IEEE Transactions on Services Computing* 10, 4 (July 2017), 603–617.

[25] Ali Ouni, Marouane Kessentini, Houari Sahraoui, and Mounir Boukadoum. 2013. Maintainability defects detection and correction: a multi-objective approach. *Automated Software Engineering* 20, 1 (2013), 47–79.

[26] Ali Ouni, Raula Gaikovina Kula, and Katsuro Inoue. 2016. Search-based peer reviewers recommendation in modern code review. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 367–377.

[27] Ali Ouni, Raula Gaikovina Kula, Marouane Kessentini, Takashi Ishio, Daniel M German, and Katsuro Inoue. 2017. Search-based software library recommendation using multi-objective optimization. *Information and Software Technology* 83 (2017), 55–75.

[28] F. Palomba, D. A. Tamburri, F. Arcelli Fontana, R. Oliveto, A. Zaidman, and A. Serebrenik. 2019. Beyond Technical Aspects: How Do Community Smells Influence the Intensity of Code Smells? *IEEE Transactions on Software Engineering (TSE)* (2019), 1.

[29] Martin Pinzger, Nachiappan Nagappan, and Brendan Murphy. 2008. Can Developer-module Networks Predict Failures?. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 2–12.

[30] Naveen Raman, Minxuan Cao, Yulia Tsvetkov, Christian Kästner, and Bogdan Vasilescu. 2020. Stress and Burnout in Open Source: Toward Finding, Understanding, and Mitigating Unhealthy Interactions. In *International Conference on Software Engineering, New Ideas and Emerging Results (ICSE-NIER)*.

[31] Islem Saidani, Ali Ouni, Moataz Chouchen, and Mohamed Wiem Mkaouer. 2020. Predicting continuous integration build failures using evolutionary search. *Information and Software Technology* 128 (2020), 106392.

[32] Helen Sharp, Hugh Robinson, and Mark Woodman. 2000. Software engineering: community and culture. *IEEE Software* 17, 1 (2000), 40–47.

[33] D. A. Tamburri, R. Kazman, and H. Fahimi. 2016. The Architect's Role in Community Shepherding. *IEEE Software* 33, 6 (2016), 70–79.

[34] D. A. Tamburri, P. Kruchten, P. Lago, and H. van Vliet. 2013. What is social debt in software engineering?. In *International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. 93–96.

[35] Damian A Tamburri, Philippe Kruchten, Patricia Lago, and Hans van Vliet. 2013. What is social debt in software engineering?. In *International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 93–96.

[36] Damian A. Tamburri, Philippe Kruchten, Patricia Lago, and Hans van Vliet. 2015. Social debt in software engineering: insights from industry. *Journal of Internet Services and Applications* 6, 1 (04 May 2015), 10.

[37] Damian A. Tamburri, Fabio Palomba, Alexander Serebrenik, and Andy Zaidman. 2018. Discovering community patterns in open-source: a systematic approach and its evaluation. *Empirical Software Engineering* (2018).

[38] D. A. A. Tamburri, F. Palomba, and R. Kazman. 2019. Exploring Community Smells in Open-Source: An Automated Approach. *IEEE Transactions on Software Engineering* (2019), 1–1.

[39] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. 2016. An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering* 43, 1 (2016), 1–18.

[40] Parastou Tourani, Yujuan Jiang, and Bram Adams. 2014. Monitoring sentiment in open source mailing lists: exploratory study on the apache ecosystem.. In *CASCON*, Vol. 14. 34–44.