From Collaboration Characteristics to Code Smells

Nirav Ajmeri, Ruijie Xi, Amanul Haque, Munindar P. Singh, and Laurie Williams
NSA: Will finalize the author order later

***Abstract*—The goal of this research is to help software practitioners avoid ineffective collaboration characteristics of their software team which influence the code smells in the software the team produces, via an empirical study of open source repositories.**

I. Introduction

Software development is a social and collaborative activity wherein software practitioners come together to build a software artifact. Collaboration involves understanding of joint goals, discussing the issues, and engaging in joint development of software artifacts by writing software code. How well a software development team collaborates reflects in the quality of software artifact they produce—better the collaboration in a software development team, better the quality of software artifact they produce [1], [2]. The quality of a software artifact can be measured via code smells. Code smells are characteristics in the code base of a software artifact which indicate deeper problems [3]. These smells could result in critical flaws during execution. In this work, we focus on fault-prone code smells which need considerable reliability and maintainability effort [4]. Fault-prone smells occur not only because of lack of technical know-how but also because of presence of communication or coordination issues which occur in software development teams [5].

The quality of the collaboration is characterized by a software development team's characteristics which the team and its members exhibit during joint development of a software artifact [6], [2]. Previous works refer to a subset of these characteristics with negative connotation as community smells [1]. We understand *collaboration characteristics* as a umbrella concept which encompass community smells. Palomba and colleagues [1] identified community smells as an important factor in code refactoring decisions and the way developers act on code smells.

Existing studies in project management have identified communication between team members—a characteristic of the collaboration in a team—as a key aspect which affects team performance [7]. Trust is another important characteristic of collaboration. Cheng et al. [8] investigated the importance of trust in semi-virtual collaboration among students in multi-culture and uni-culture teams. Language, values (attitude and perception), and habitual behaviors of individuals are the factors that contribute to communication and trust difference between team members. How these factors affect performance of a software development team and how that reflects in software artifacts the team produces need to be studied.

Understanding these caveats, in this work, we study sociotechnical aspects of software development and software development teams, and understand the relation between the two.

The goal of this research is to help software practitioners avoid ineffective collaboration characteristics of their software team which influence the code smells in the software the team produces, via an empirical study of open source repositories. We analyze team structure, source code, commit messages, issues, pull requests and associated comments of 77 open source repositories hosted on GitHub.

Specifically, we investigate the following research question:

RQ. How does collaboration characteristics of a software team influence the code smells in the code base the team produces?

To address RQ, we first survey research works in social psychology and software engineering and analyze open source repositories on GitHub to identify collaboration characteristics of a software development team. Next, we develop a tool which leverages social network analysis and natural language techniques to automatically measure collaboration characteristics by analyzing team structure and interactions in the team occurring via commits, issues, and pull requests. Finally, we perform a case study on open source repositories in which we (1) compute collaboration characteristics in open source teams; (2) compute code smells in the software artifacts they produce; and (3) analyze the relation between the computed collaboration characteristics and code smells. Specifically, in our case study, we apply HAGRID on 77 open source projects hosted on GitHub. We identify code smells in these 77 projects using SonarQube, a static analysis tool, and analyze how collaboration characteristics influence code smells.

Contributions. Our contributions include 1) a list of collaboration characteristics with their definitions; 2) a tool to compute collaboration characteristics; 3) a study on how collaboration characteristics influence code smells.

Organization. NSA: Revisit Section II describe the relevant related works. Section III describes the dataset we analyze. Section IV details the metrics for collaboration and code smells and the method to extract these metrics. Section V describes our case study and its results on how collaboration characteristics influence code smells. Section VI concludes with discussion of future directions.

II. Related Works

Previous work from [1] aimed at empirically exploring the relation between social and technical debt, by investigating the connection between two noticeable symptoms behind community and code smells. Palomba et al.'s research method based on qualitative investigation features a survey of developers.

Their findings highlighted that the persistence of code smells not only depends on technical factors studied by past literature but also on additional aspects related to the social debt occurring in software communities.

This work uses community smells, as discussed in [1], but differs from previous work in terms of how these community smells are identified. In this work, we use text features and meta-data of communication among team members to automatically identify organizational smells as against previous studies which have relied on personal interviews and surveys.

III. Dataset

GitHub metadata contains a wealth of information with which we could describe several phenomena surrounding a source code repository [1]. The GHTorrent project provides a queryable offline mirror of all Git and GitHub metadata [9]. GHTorrent monitors the GitHub public event time line. For each event, GHTorrent exhaustively retrieves the contents of the event and stores them in a relational database. We use MySQL dump from GHTorrent which was downloaded and restored on to a local server. The GHTorrent database dump used in this study was released on June 1, 2019. This database dump contains metadata for 16,331,225 GitHub repositories.

To identify the relevant repositories, we follow Munaiah et al.'s [10] suggestions and define the following inclusion and exclusion criteria as in context of the five dimensions aforementioned.

Inclusion criteria: The GitHub repositories were included in the research collection if they fulfill all the inclusion criteria, which were selected based on "criteria sampling" [11]. For each inclusion criteria we defined a threshold. An explanation of the criteria follows:

1) number of contributors are larger than 5
2) project languages are either Python or JavaScript
3) commits sustainability of the projects are longer than 1 year
4) number of commits are greater than 50

We select Python and JavaScript as they are ranked as the one of top-most programming languages [12], [13]. Researchers have also identified JavaScript as a language prone to attacks [14].

Exclusion criteria: We discard the repositories which meet the following exclusion criteria:

1) repositories that have been marked as deleted
2) repositories that have been marked as forked
3) repositories that have never been updated

Deleted repositories restrict the amount of data available for the analysis. Forked repositories can artificially inflate the results by introducing near duplicates into the sample.

Data collection: Based on the inclusion and exclusion criteria, we identified 77 repositories from the GHTorrent database.

After identifying the relevant GitHub repositories to include in the analysis, based on the exclusion and inclusion criteria, for each repository we extracted information related to project languages, project members, active commits lifetime of the projects.

For these repositories which meet our inclusion and exclusion criteria, we extract a list of attributes related to listed in Table I. These attributes are based on [10]'s five dimensions: 1) community; 2) evolution; 3) project management; 4) documentation; and 5) repository size, to perform analysis on open source software repositoriesNSA: to verify.

TABLE I
Attributes to describe each repository.

Metrics	Description
nMembers	Number of members from each repository
Duration	Length of the project in years
nCommits	Number of commits from each repository
nIssues	Number of issues from each repository
nPullreqs	Number of pull-requests from each repository
nLOC	Number of nonwhitespace lines not part of a comment
nComment	Number of lines with nonwhitespace comments

We identify two attributes of an open source repository—number of members and number of commits—to quantify the community established around the repository. The presence of continued change indicates that the software system is being modified to ensure its activity. We identify three attributes, duration, sustainability of commits, and the number of pull requests, to be the time and frequency by which a repository is undergoing change. Continued use of the GitHub issues is indicative of project management in an open source repository. On the importance scale, source code comments are second only to the actual source code [10]. We restrict ourselves to documentation in the form of source code comments. Number of lines of code is a measure for the repository size.

Table II provides a range for the attributes in Table I.

TABLE II
Basic statistics of the dataset.

Number of	Min	Max	Average	Median
nMembers	5	42	7	6
Duration (years)	1	4	3	3
nCommits	55	15,840	2,042	712
nPullreqs	1	5,172	310	39
nIssues	1	6,860	628	132
nLOC	206	489,269	65,416	16,762
nComment	0	123,303	15,962	1,895

IV. Method

We now define the metrics to characterize teams and software artifacts they produce, and how to compute those.

A. Collaboration Characteristics

We define collaboration characteristics as characteristics of a team and its members individually as well as collectively which are observed when the members collaborate.

We propose to measure these characteristics through contribution, team cohesion, and trust metrics:

[1]https://github.com

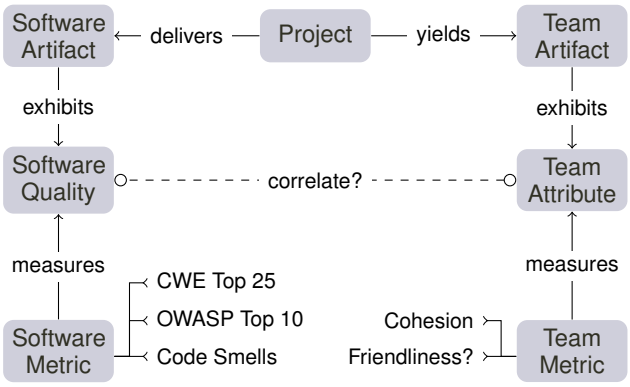

Fig. 1. Problem setting and claim.

- **Contribution deviation** is the standard derivation value of number of commits from each contributors of a repository. Lower is better. NSA: Unequal contribution could result in cohesion difference which in turn could result in poor performance of a team.
- **Team cohesion** is the willingness to take input from team members and a belief that team is more important than individual members. Inspired by [15], [16] where cohesion corresponds to the measure of willingness to contribute, we compute team cohesion as the average response time between messages. Specifically, we compute cohesion as time between comments on issues and pull requests in GitHub repositories.

NSA: If team members do not reciprocate with equal sentiment, politeness, and emotion ... trust difference in the team increases.

- **Sentiment** measures the polarity of a text based on the subjective information present in the communication text. We use VADER sentiment analyzer [17] to compute sentiment scores from text features. VADER is specially suited for our text as it has been shown to perform well for computing sentiment in microblog-like contexts [17].
- **Emotion** captures the emotion expressed by words in a text based on associations of words with emotions. We use emotion lexicons proposed in [18] that associates 6,400 lexicons in 40 languages with one of the eight human emotions namely, anger, fear, anticipation, trust, surprise, sadness, joy, and disgust. We further improve the emotion metric by using the Word Affect Intensity lexicon proposed by [19]. We also include other lexical cues like emoticons, cuss words, and ALL CAPS words in computing the overall emotion metric.
- **Politeness** in formal interactions is usually viewed as essential to showing respect, perhaps more so in online interactions where cues like body language and facial expressions are missing. We use Stanford Politeness library [20] to compute politeness using features from online conversational texts. MPS 1: look at Stephen Levinson's work; politeness is a tricky concept
- **Trust** is the emotional tie perceived between the team members MPS 2: averaged over pairs?NSA: no. Following [21], we compute trust based on sentiments associated with the interactions.
- **Situational awareness** measures the extent to which the team members give opinions and suggestions when asked for [22]. MPS 3: see Arwen H. DeCostanza definition

GitHub team members interact via commits, issues and pull requests. Figure 2 outlines the interactions.

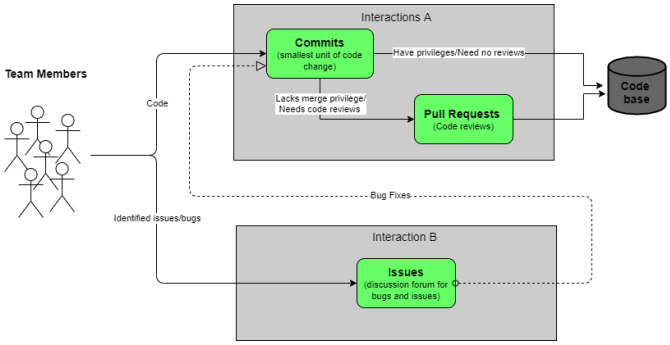

Fig. 2. Interactions on GitHub.

We extract collaboration characteristics using the interactions among team members on GitHub via issues and comments, pull requests and comments, and commit messages. We use text features and metadata including timestamp and author association with the project from these interactions to compute scores that define an individual's participation in these interactions. We further use these scores to construct a directed social network with team members as nodes and their weighted edge computed based on these metrics define their relationship with other members in the team.

TABLE III
COLLABORATION CHARACTERISTICS.

Number of	Min	Max	Mean	Median
Pull request cohesion	0.000	1.000	0.200	0.030
Issue cohesion	0.000	1.000	0.097	0.002
Pull request trust	0.000	1.000	0.069	0.750
Issue trust	0.000	1.000	0.596	0.636
Contribution	2.54	622.80	76.73	45.80

B. Code Smells

SonarQube [23] is one of the most common Open Source static code analysis tools adopted both in academia and in industry [4]. SonarQube proposes a set of coding rules that represent something wrong that will be reflected as fault or cause maintenance effort as *Violations*, such as *Code smells*. It calculates several metrics such as the number of lines of code and the code complexity, and verifies the code's compliance against a specific set of "coding rules" defined for most common development languages. In case the analyzed source code violates a coding rule or if a metric is outside a predefined threshold, SonarQube generates an *issue*. It is important to note that the term *code smells* adopted in SonarQube does not refer

to the commonly known code smells defined by Fowler et al. [3] but to a different set of rules, rules classified by SonarQube as "Code Smells" are termed maintenance issues.

We select 20 change metrics that have been extensively used for defect estimation [24] and can describe the projects. We collect two types of technical debt defined by SonarQube: Maintainability remediation effort (also known as *Squale Index*) and reliability remediation effort. We did not considered security remediation effort, since SonarQube does not provide software metrics clearly useful to predict it [25].

Lenarduzzi et al.'s work [4] has verified the most fault-prone SQ-Violations on Java projects, we map these SonarQube rules with Python and JavaScript to select top 17 rules for Python and top 21 rules for JavaScript as following table. We report the SQ-Violations with an importance higher or equal than 0.00% [4].

In this work, we focus on all types of selected SQ-Violations, as we are interested in the fault-inducing smells [4]. To increase the accuracy of the mapping process, we manually checked the specific cases of each rules that were selected based on Lenarduzzi et al.'s [4] work. Some of the rules suitable for Java may not be accurate for Python. For example, the *read* as a loop counter in *for read in paired_reads* is regarded as *Unused local variables*, which needs to be removed from the selected fault-prone smells. We refer to different SonarQube violations with their id as "SquID" in Table IV.

TABLE IV
METRICS TO ESTIMATE VIOLATIONS THAT CAUSE MAINTENANCE EFFORT.

SquID	Severity	Type	Language
S1192	Critical	Code Smell	Python
S1128	Minor	Code Smell	JavaScript
S1481	Minor	Code Smell	JavaScript, Python
S1117	Major	Code Smell	JavaScript
S125	Major	Code Smell	Python
S1126	Minor	Code Smell	JavaScript
S108	Minor	Code Smell	JavaScript
S1066	Major	Code Smell	Python
S1854	Major	Code Smell	Python

We adopt the following metrics from SonarQube for our evaluation of code smells.

TABLE V
METRICS FROM SONARQUBE.

Metrics	Description
Comment Density	$\frac{\text{NComment}}{\text{NComment+Ncloc}}$
nSmells	Number of code smells
tMaintainability	Time taken to fix all code smells
tFaultiness	Time taken to fix all bugs

V. INFLUENCE OF COLLABORATION CHARACTERISTICS ON CODE SMELLS

A. Hypotheses.

We propose three null hypotheses that state collaboration characteristics do not influence code smells. The alternative hypotheses indicate there is an influence.

H1: Team cohesion does not influence the code smells.

H2: Trust within team members do not influence the code smells.

H3: Difference in contribution by team members do not influence the code smells.

To test the statistical significance, we apply the following statistical analysis.

We perform a one-way ANOVA to test the null hypotheses H1, H2, and H3. If a null hypothesis is rejected—indicating there is an influence, in the post hoc analysis, we perform a series of pairwise t-tests to isolate the differences.

Two-tailed t-test. We perform a two-tailed t-test to test the significance of hypotheses H1, H2, H3 and isolate the differences. We adopt 0.05 as the significance cutoff for p-values, below which each null hypothesis is rejected [26]. To avoid a type-1 error, we apply the Bonferroni correction.

Hedge's g. To measure the effect size for the difference between means, we compute Hedges' g, which is well-suited to measuring effect for small (and unequal) sample sizes [27], [28].

Pearson's r. To measure the strength and direction of correlation (linear relationship) between two variables [29].

NSA: Analysis needs to be redone

B. Influence of Team Cohesion

We evaluate H1 by computing the correlation between pull request cohesion and issue cohesion metrics and code smells for each repositories. The mean of pull request cohesion (0.24) for repositories with less commits was lower than the mean of pull request cohesion (0.35) for those of more commits. And the mean of issue cohesion (0.11) for repositories with less commits was lower than the mean of pull request cohesion (0.38) for those of more commits. The correlation between pull request cohesion and code quality is 0.378 for less commits group and 0.117 for the other group. In addition, the correlation between issue cohesion and code quality is 0.009 for less commits group and 0.843 for the other group.

C. Influence of Trust

We evaluate H2 by computing the correlation between pull request trust with code quality. The mean of violations (3254) for repositories with low trust was much higher than the mean of trust (709) for those of with high trust teams. The correlation between trust and code quality is -0.0434 for less trusted groups and 0.159 for the other group with higher trust.

D. Influence of Contribution

We evaluate H3 by computing the contribution of each contributors of each repositories in the groups. The mean of contribution (38.25) for repositories with less commits was

higher than the mean of contribution (107.044) for those of more commits. The correlation between contribution and code quality is 0.201 for less commits group and 0.328 for the other group.

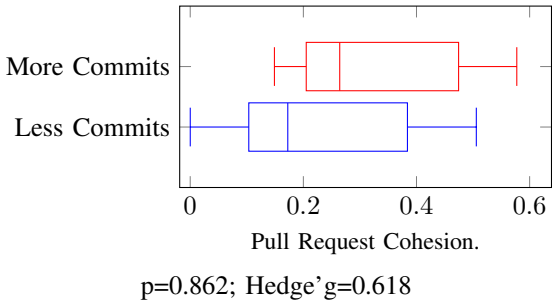

Fig. 3. Pull Request Cohesion of repositories with small and large number of commits.

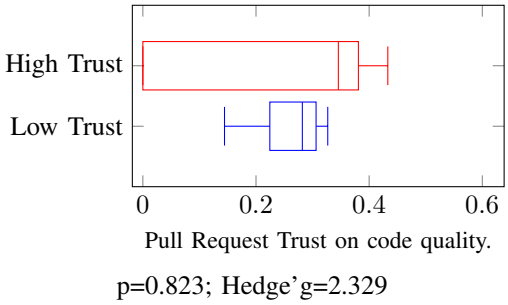

Fig. 4. Pull Request trust of repositories with low and high trust.

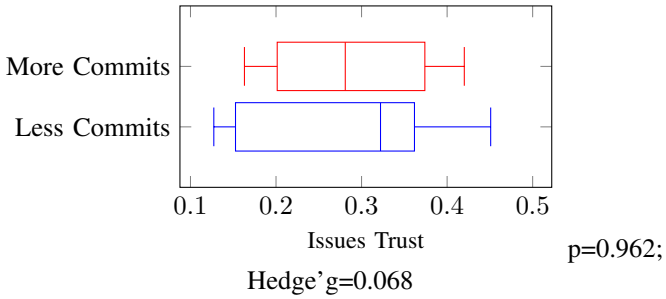

Fig. 5. Issue Trust of repositories with small and large number of commits.

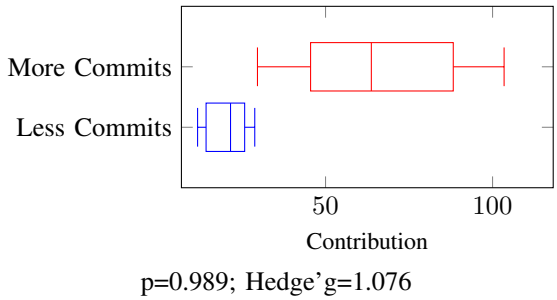

Fig. 6. Contribution with small and large number of commits.

E. Limitations and Threats to Validity

Internal validity.

External validity.

Construct validity.

VI. Conclusions and Future Directions

REFERENCES

[1] F. Palomba, D. A. A. Tamburri, F. A. Fontana, R. Oliveto, A. Zaidman, and A. Serebrenik, "Beyond technical aspects: How do community smells influence the intensity of code smells?" *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–22, 2018, early access.

[2] C. Raibulet and F. A. Fontana, "Collaborative and teamwork software development in an undergraduate software engineering course," *Journal of Systems and Software (JSS)*, vol. 144, pp. 409–422, 2018.

[3] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Boston: Addison-Wesley Professional, 2018.

[4] V. Lenarduzzi, F. Lomio, H. Huttunen, and D. Taibi, "Are sonarqube rules inducing bugs?" in *27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020, London, ON, Canada, February 18-21, 2020*, K. Kontogiannis, F. Khomh, A. Chatzigeorgiou, M. Fokaefs, and M. Zhou, Eds. London, ON, Canada: IEEE, 2020, pp. 501–511. [Online]. Available: https://doi.org/10.1109/SANER48275.2020.9054821

[5] D. A. Tamburri, P. Kruchten, P. Lago, and H. van Vliet, "What is social debt in software engineering?" in *Proceedings of the 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 2013, pp. 93–96.

[6] Y. Lindsjørn, D. I. Sjøberg, T. Dingsøyr, G. R. Bergersen, and T. Dybå, "Teamwork quality and project success in software development: A survey of agile development teams," *Journal of Systems and Software (JSS)*, vol. 122, pp. 274–286, 2016.

[7] T. Daim, A. Ha, S. Reutiman, B. Hughes, U. Pathak, W. Bynum, and A. Bhatla, "Exploring the communication breakdown in global virtual teams," *International Journal of Project Management*, vol. 30, no. 2, pp. 199–212, 2012.

[8] X. Cheng, S. Fu, J. Sun, Y. Han, J. Shen, and A. Zarifis, "Investigating individual trust in semi-virtual collaboration of multicultural and unicultural teams," *Computers in Human Behavior*, vol. 62, pp. 267—-276, Sep. 2016.

[9] G. Gousios, "The ghtorrent dataset and tool suite," in *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*. San Francisco: IEEE, 2013, pp. 233–236.

[10] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Curating github for engineered software projects," *Empirical Software Engineering*, vol. 22, no. 6, pp. 3219–3253, 2017.

[11] F. D. Michael Patton, *Qualitative Evaluation and Research Methods.* Sage, Newbury Park: Cambridge University Press, 2002.

[12] S. Cass. (2018, Jul.) The 2018 top programming languages. [Online]. Available: https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages

[13] StackOverflow, "2020 developer survey," Feb. 2020. [Online]. Available: https://insights.stackoverflow.com/survey/2020

[14] S. Guarnieri, M. Pistoia, O. Tripp, J. Dolby, S. Teilhet, and R. Berg, "Saving the world wide web from vulnerable javascript," in *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, 2011, pp. 177–187.

[15] A. K. Kalia, N. Buchler, A. DeCostanza, and M. P. Singh, "Computing team process measures from the structure and content of broadcast collaborative communications," *IEEE Transactions on Computational Social Systems*, vol. 4, no. 2, pp. 26–39, 2017.

[16] D. J. Beal, R. R. Cohen, M. J. Burke, and C. L. McLendon, "Cohesion and performance in groups: A meta-analytic clarification of construct relations," *Journal of Applied Psychology*, vol. 88, no. 6, p. 989, 2003.

[17] C. J. Hutto and E. Gilbert, "Vader: A parsimonious rule-based model for sentiment analysis of social media text," in *Proceedings of the International AAAI Conference on Weblogs and Social Media (ICWSM)*. Ann Arbor: AAAI, 2014, pp. 216–225.

[18] S. Mohammad and P. Turney, "Crowdsourcing a word-emotion association lexicon," *Computational Intelligence*, vol. 29, 08 2013.

[19] S. Mohammad, "Word affect intensities," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki: European Language Resources Association (ELRA), May 2018. [Online]. Available: https://www.aclweb.org/anthology/L18-1027

[20] C. Danescu-Niculescu-Mizil, M. Sudhof, D. Jurafsky, J. Leskovec, and C. Potts, "A computational approach to politeness with application to social factors," in *Proceedings of the 2013 Annual Conference of the Association for Computational Linguistics (ACL)*. Sofia: Association for Computational Linguistics, Aug. 2013, pp. 250–259.

[21] A. K. Kalia, N. Buchler, A. H. DeCostanza, and M. P. Singh, "Computing team process measures from the structure and content of broadcast collaborative communications," *IEEE Transactions on Computational Social Systems (TCSS)*, vol. 4, no. 2, pp. 26–39, Jun. 2017.

[22] E. Salas, C. Prince, D. P. Baker, and L. Shrestha, "Situation awareness in team performance: Implications for measurement and training," *Human Factors*, vol. 37, no. 1, pp. 123–136, 1995.

[23] Sonar. (2020) Javascript sonar source. [Online]. Available: https://rules.sonarsource.com/javascript

[24] D. Falessi, B. Russo, and K. Mullen, "What if i had no smells?" in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. Toronto, ON, Canada: IEEE, 2017, pp. 78–84.

[25] V. Lenarduzzi, A. Martini, D. Taibi, and D. A. Tamburri, "Towards surgically-precise technical debt estimation: Early results and research roadmap," *CoRR*, vol. abs/1908.00737, pp. 37–42, 2019. [Online]. Available: http://arxiv.org/abs/1908.00737

[26] M. Hollander and D. A. Wolfe, *Nonparametric Statistical Methods*. New York: Wiley, 1999.

[27] R. J. Grissom and J. J. Kim, *Effect Sizes for Research: Univariate and Multivariate Applications*. Abingdon-on-Thames: Routledge, 2012.

[28] L. V. Hedges and I. Olkin, *Statistical Methods for Meta-Analysis*. Orlando: Academic Press, Inc., 2014.

[29] D. Freedman, R. Pisani, and R. Purves, *Statistics (4th Edition.* New York: W. W. Norton & Company, 2007.