From Collaboration Characteristics to Software Quality

Anonymous Authors

***Abstract*—The goal of this research is to help software practitioners understand how ineffective team communication can negatively impact software quality, via an empirical study of open source repositories.**

I. Introduction

Software development is a social and collaborative activity wherein software practitioners collaborate to create a software artifact [1]. 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 team collaborates reflects in the quality of software artifact they produce [2], [3]. Poor collaboration may lead to the team delivering a poor quality software artifact, which in turn can have negative consequences such as loss of revenue. CPQS report [4] estimates the cost of poor quality software in the US for the year 2018 alone to be approximately $2.84 trillion of which 18.22% was from technical debt (i.e., bugs or issues that can be fixed by refactoring) and 37.46% from software failures among other reasons such as cancelled projects and other internal and external failures and deficiencies.

The quality of the collaboration can be characterized by a software team's characteristics which the team and its members exhibit during joint development of a software artifact [3], [5]. 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 [6]. Trust is another important characteristic of collaboration. Cheng et al. [7] 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 social and collaboration 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 understand how ineffective team communication can negatively impact software quality, via an empirical study of open source repositories.*

We analyze team structure, source code, commit messages, issues, pull requests and associated comments of [X] open source repositories hosted on GitHub. Specifically, we investigate the following research question:

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

To address RQ, we 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. We develop a tool, HAGRID, 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.

One way to measure the quality of a software artifact is via code smells. Code smells are characteristics in the code base of a software artifact which may indicate deeper problems [8].

We perform a case study on open source repositories in which we (1) compute the collaboration characteristics in open source teams; (2) measure the quality of the software artifacts they create; and (3) analyze the relation between the computed collaboration characteristics and software quality. Specifically, in our case study, we apply HAGRID on 77 open source projects hosted on GitHub. We measure software quality including code smells, vulnerabilities and bugs in these 77 projects using SonarQube, a static analysis tool, and analyze how collaboration characteristics influence quality measured by SonarQube.

Contributions: Our contributions include 1) a study on how collaboration characteristics influence the software quality; and 2) a tool to compute collaboration characteristics.

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

II. Related Works

Previous work from [2] 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.

In open source projects, contributors are motivated by both intrinsic and extrinsic factors, among which aspects related to

bonding social capital [9], such as contribution and commitment are highly important [10]. Prior work showed how emotional attachment, trust relationships [11], and shared goals and norms (all of which are more likely to develop in cohesive teams [12]) positively impact individual and team outcomes. It follows that bonding collaboration should positively impact the contributors' willingness to contribute in development teams.

This work uses community smells, as discussed in [2], 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.

While community contribution has been extensively studied in the social coding network literature, as argued by Dabbish et al. [13], these collaborative awareness tools are not suited for the operationalization of large software development teams in online production settings like GitHub where hundreds or thousands of collaborators can be involved on a project.

III. Method Overview, Dataset, and Metrics

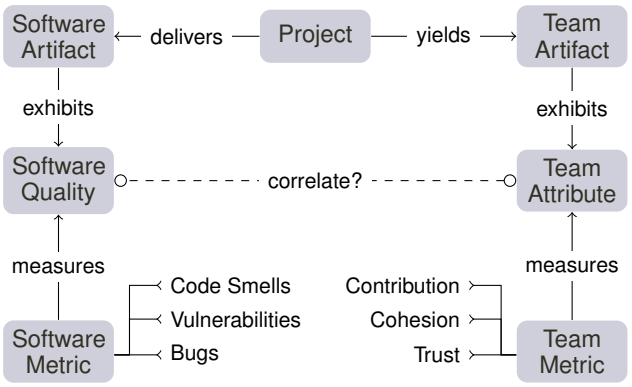

Fig. 1. Problem setting and claim.

In this section, we describe our methodology overview, dataset and metrics.

A. Methodology Overview

In our empirical study, we systematically investigate if collaboration characteristics of a software team correlate with the quality of software artifact the team produces.

To address RQ, we follow a three step method. First, we develop a set of metrics to measure the collaboration characteristics of a software development team via a literature review. We also identify metrics relevant to measure the quality of software artifact. Second, we conduct an exploratory analysis with 100 GitHub repositories to propose hypotheses about the relation between collaboration characteristics and software quality. Third, we conduct a case study on 300 GitHub repositories to validate the proposed hypotheses.

We now describe the dataset that we create to conduct our empirical study.

B. Dataset

GitHub[1] metadata contains a wealth of information with which we could describe several phenomena surrounding a source code repository.

The GHTorrent project provides a queryable offline mirror of all Git and GitHub metadata [14]. 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 and approximately 16M users.

To identify the relevant repositories, we follow Munaiah et al.'s [15] 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" [16]. 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) the last update time after 2018
4) commits sustainability of the projects are longer than 1 year; and
5) number of commits are greater than 50.

We select Python and JavaScript as they are ranked as the one of top-most programming languages [17], [18]NSA: https://spectrum.ieee.org/at-work/tech-careers/top-programming-language-2020. Researchers have also identified JavaScript as a language prone to attacks [19].

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; and
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 2184 NSA: to verify repositories from the GHTorrent database. From this set of relevant repositories, we create two random and unequal datasets: (1) exploration dataset containing 100 repositories; and (2) case study dataset containing 300 repositories.

Next, for each repository in the exploration and case study dataset, 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 Munaiah et al.'s [15] five dimensions: 1) community as the number of members; 2) evolution as the duration; 3) project management as the number

[1]https://github.com

of commits; 4) documentation as the number of issues and pull request; and 5) repository size as the number of lines, 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

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 [15]. 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.

Measure	**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

C. Metrics: 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.

For every person and every project they contribute to, we compute yearly measures for different project-level measures such as team cohesion and trust. For every person's individual contribution to each project, we compute contribution in terms of number of commitment and measure the contribution deviation. To collectively measure the collaboration characteristics, we compute the team cohesion and trust among the whole development team by aggregating the response time and classifying the sentiment of their 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.

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

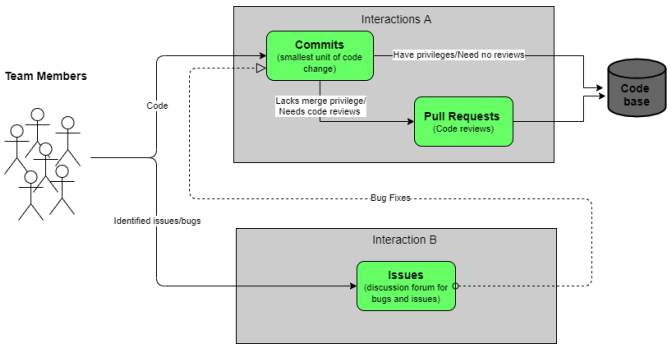

Fig. 2. Interactions on GitHub.

1) Contribution disparity: Project owners can make *commits*, i.e., changes to the code by directly modifying the contents of the code files. The amount of commits to a single project signaled that project as the performance of commitment and investment at the individual-level. Visible information about other developers' actions influenced perceptions of developers' commitment and general interests [13], i.e., unequal contribution could result in cohesion difference which in turn could result in poor performance of a team.

We quantify the developers' productivity from an abstract meaning into measurable metrics in order to comprehend the amount of contribution and commitment each developer exerts in a project. The amount of contribution for each developer is estimated by the number of participation in distinct projects where a commit is an update submitted to a project by a developer. Thus, we aggregate all the commits from each developers of distinct projects and compute the deviation for each developers of distinct projects.

Thus, we identify contribution disparity as the standard derivation value of number of commits from each contributors of a repository.

2) Team cohesion: Cohesion corresponds to the measure of willingness to contribute [11], [20]. Team cohesion is the willingness to take input from team members and a belief that a team is more important than individual members.

In GitHub communication mechanism, code-related communications between contributors occur when a contributor changes an artifact or interacts with another contributor *i.e. pull request and issue*. Thus, we compute team cohesion as time between comments on issues and pull requests in GitHub repositories. We assume when a team member j make a response to a contributor i, i can estimate how willing team member j wants to contribute toward the team. Thus, we compute team cohesion (co_{ij}) as follows, where art_{ij} represents

the average response time delay between team members i and j:

$$co_{\text{ij}} = \frac{1}{1 + art_{\text{ij}}} \tag{1}$$

3) Sentiment: Sentiment measures the polarity of a text based on the subjective information present in the communication text. As we have described on team cohesion section, prior research has proved the effective metrics for emotion, politeness and sentiment. Particularly, prior research has shown that sentiment affect task quality, productivity, creativity, group rapport and job satisfaction [21]. In addition, developers and managers need to be aware of the emotions of the projects they are involved to take corrective actions when necessary and to have a better understanding of the social factors affecting the project. We use VADER sentiment analyzer 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 [22].

We note that *commits* usually have a neutral sentiment as they are meant for documentation (of any change to the code base) rather than communication. For example, *Remove jsdoc from release build as its causing errors.* and *Merge pull request 729 from eduardomb master.*. Hence, we do not take into account the sentiment of *commit* messages in our study, assuming they don't have much impact on software development process.

4) Trust: Following Kalia et al. [11], we consider trust as an affective process as indicators of team performance. We regard trust as the confidence of one team member toward another as confidence reflects the knowledge of a team member about a contributor's past performance. *If team members do not reciprocate with equal sentiment, politeness, and emotion, trust difference in the team increases.* As Kalia et al. has divided trust into two kinds: 1) *cognitive trust* [11], 2) *affective trust* [11]. We consider only cognitive trust since it is based on evidence ties between developers. For brevity, we refer to cognitive trust as trust. In our heuristic, trust is the emotional tie perceived between the team members. We compute trust based on sentiments associated with the interactions. We represent trust as event pairs $\langle r, s \rangle$, where r and s represent positive and negative evidence we extract from messages (*pull request and the comments accordingly*). Inspired by Kalia et al. [11], we include neutral evidence and equally increase positive and negative evidence as $\langle r + 0.5 * n, s + 0.5 * n \rangle$, where n is the representation of neutral evidence. Trust is computed as:

$$t = \frac{r + 0.5 * n}{r + s + n} \tag{2}$$

NSA: Adaptation of Yonghong's IJCAI paper.

D. Metrics: Software Quality

SonarQube [23] is one of the most popular Open Source static code analysis tools adopted both in academia and in industry [24]. 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. [8] but to a different set of rules, rules classified by SonarQube as "Code Smells" are termed maintenance issues.

We select two types of metrics *i.e., code quality (code smells, vulnerabilities and bugs) and technical debt* as shown in Table IV that have been extensively used for defect estimation [25] and can describe the projects. Note that the technical debt in SonarQube is different from Kruchten et al. in [26], the technical debt in SonarQube is the effort for solving maintenance issues based on the definition of code smells we have described above. 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 [24].

Lenarduzzi et al. [24] identified the most fault-prone SQ-Violations on Java projects, they applied the SZZ algorithm [27] to label the fault-inducing Jira commits on open-source projects and evaluated with SQ-Violations introduced on the same commits by machine learning models. They defined *importance* as a specific feature which was computed by the difference in cross-validated test accuracy between the newly trained machine learning model and the baseline machine learning model which was the one trained with full set of features as in *drop-column mechanism* [28]. We map these SonarQube rules with Python and JavaScript to select top 9 rules with highest *importance* score for Python and JavaScript as shown in Table III. We assume that Java and Python SQ-Violation rules have similarity on how it influence the maintainability of the projects. To improve the accuracy of the mapping process, we manually checked the specific cases of each rules that were selected based on Lenarduzzi et al.'s [24] 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. Specially, we only count the number of violations of the rules in Table III proved by Lenarduzzi et al. as the most important fault-prone smells by classifying them with machine learning method. We refer to different SonarQube violations with their id as "squID" in Table III.

IV. Exploratory Analysis: Select Hypotheses

In the exploratory analysis, we analyze 100 GitHub repositories.

TABLE III
METRICS TO ESTIMATE VIOLATIONS THAT CAUSE NSA: HIGHEST? MAINTENANCE EFFORT. NSA: REMOVE TYPE?

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

TABLE IV
METRICS FROM SONARQUBE.

Metrics	Description
nViolations	Number of fault-prone code smells
nSecurityHotSpots	Number of security issues
tMaintainability	Time taken to fix all fault-prone code smells
tFaultiness	Time taken to fix all bugs

A. Contribution

Contribution is measured via the number of commits made by individual team members.

a) Contribution disparity: We compute the contribution disparity for a project as the standard deviation of the number of commits made by each member of the team working that project.

Contribution disparity H1 and H2 in Figure 3 show the contribution disparity for projects separated by median of contribution disparity (which is 29.7). Q1–Q4 are the four quartiles of contribution disparity.

We observe that the number of warnings are not significantly different in Q1 and Q2, which indicates contribution disparity influences the code quality once it reaches a threshold.

Proposed hypothesis. Higher contribution disparity leads to higher warnings.

B. Response Cohesion

We measure cohesion via timeliness of pull request and issues comments, and number of comments made.

a) Timeliness: We compute the timeliness by measuring the average response time of commits for each project. *H1*, *H2* in Figure 4are the timeliness separated by median of timeliness. *Q1* - *Q4* are the quartiles of timeliness. The median value of lower timeliness (*H1*) is 0.0000023 compared to the median value of higher timeliness (*H2*) 0.000036 which indicates the higher timeliness leads to higher warnings. However, the median values for *Q1* - *Q4* are not regular as there is no pattern for them. We assume there is implicit influence from timeliness on code quality.

Proposed hypothesis. Higher timeliness leads to lower warnings.

b) Message contribution (Kurtosis): We compute the message contribution by measuring the kurtosis of messages

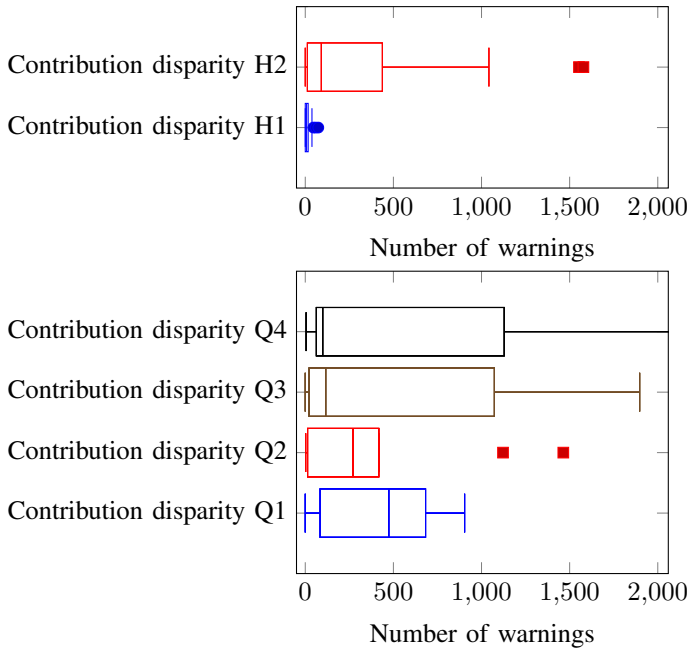

Fig. 3. Code quality for teams with high and low contribution variation. H1-H2: g=11.01, p=0.06; Q1-Q4 g=3.70, p=0.12

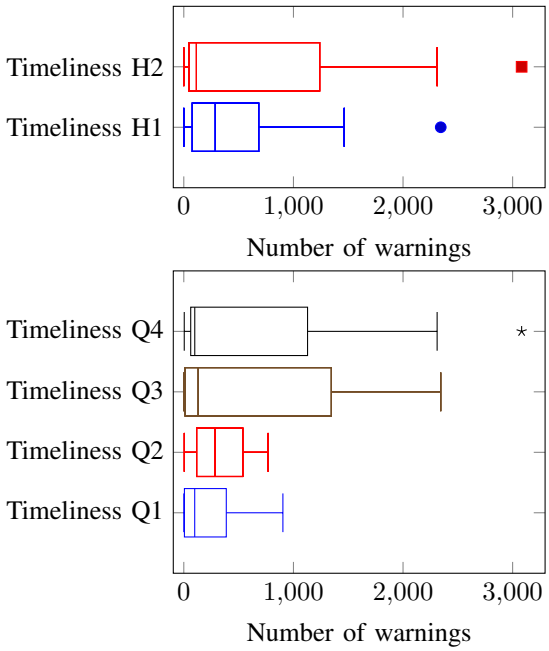

Fig. 4. Code quality for teams with high and low timeliness. r=-0.20; H1-H2 p=0.02 g=0.3; Q1-Q4: p=0.5, g=6.83

(*i.e. pull request, issue*) for each project. *H1*, *H2* in Figure 4 are the message contribution separated by median of message contribution. *Q1* – *Q4* are the quartiles of message contribution.

Proposed hypothesis. Higher message contribution leads to lower warnings.

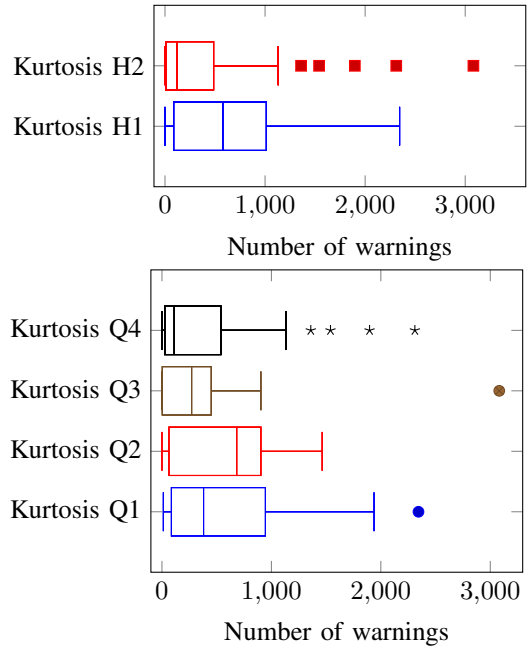

Fig. 5. Code quality for teams with high and low message contribution (Kurtosis).

C. Sentiment

a) Mean sentiment: We measure the mean sentiment for each project by computing the mean sentiment score of all messages in pull requests and issues. A positive attitude could improve the performance toward teamwork.

Proposed hypothesis. Higher sentiment leads to lower violations.

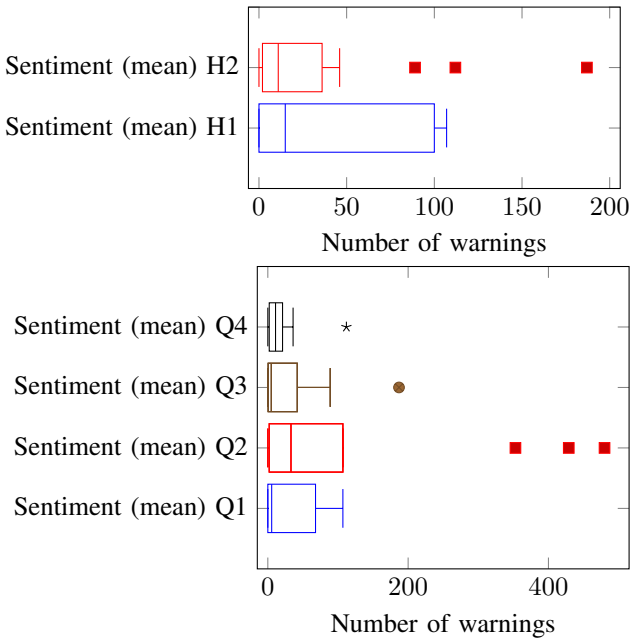

Fig. 6. Code quality for teams with high and low mean sentiment.

b) Sentiment disparity: We compute sentiment disparity by computing the standard deviation of sentiment score for each project. Figure 7 shows projects in group *Q3* has the worst performance. We assume in each development team, the balanced sentiment of each contributor could improve the code quality.

Proposed hypothesis. Higher sentiment disparity leads to higher warnings.

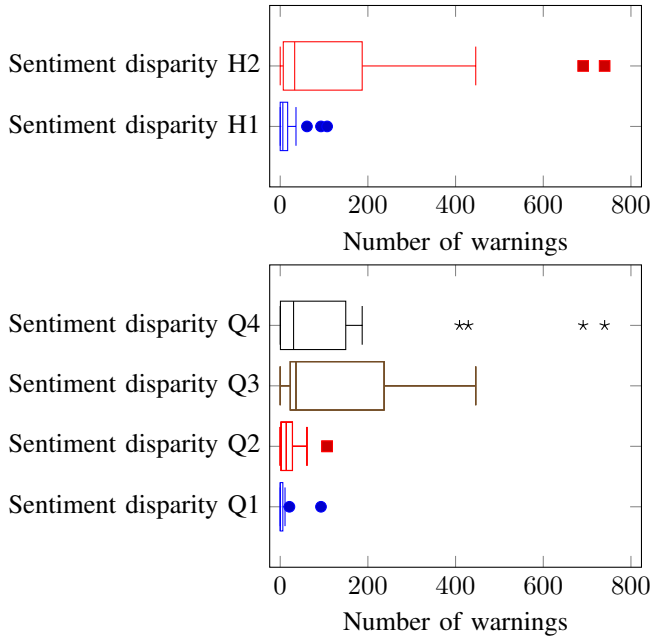

Fig. 7. Code quality for teams with high and low sentiment deviation.

D. Trust

We measure trust based on the sentiment of the messages exchanged in pull request and issue discussions.

a) Mean trust: We compute trust by calculating the proportion of sentiment for each project. The median value of *Q1* and *Q2* does not have a significant difference. Figure 8 shows *Q2* has the largest number of warnings which indicates positive attitude could improve the performance toward teamwork.

Proposed hypothesis. Higher trust leads to lower violations.

V. Findings: Evaluate Hypotheses

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

H2: Team cohesion does not influence the code smells.

H3: Trust within team members do not influence the code smells.

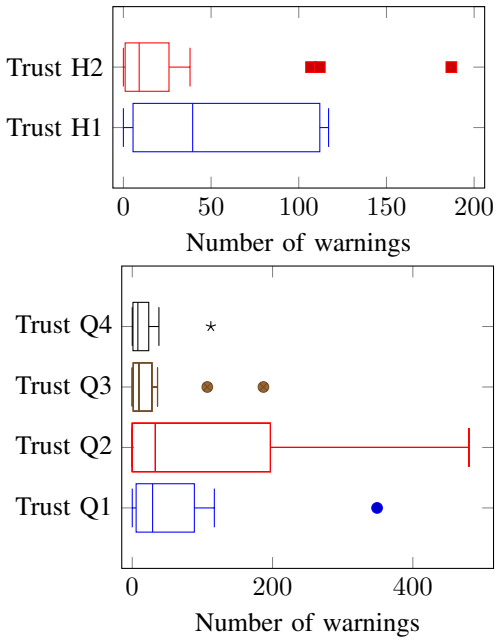

Fig. 8. Code quality for teams with high and low trust.

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 [29]. 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 [30], [31].

Pearson's r. To measure the strength and direction of correlation (linear relationship) between two variables [32].

NSA: Analysis needs to be redone

NSA: Outlier removal:
https://statisticsbyjim.com/basics/remove-outliers/
https://statisticsbyjim.com/basics/outliers/
https://answers.microsoft.com/en-us/msoffice/forum/all/filter-outliers-when-using-standard-deviation/402b1462-4e3f-4a1c-86a0-3520d2b4776a
https://www.howtogeek.com/400211/how-and-why-to-use-the-outliers-function-in-excel/

B. H1: Influence of Contribution

C. H2: Influence of Team Cohesion

D. H3: Influence of Trust

E. Limitations and Threats to Validity

Internal validity.

External validity.

Construct validity.

VII. Conclusions and Future Directions

REFERENCES

[1] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development productivity," in *Proceedings of the 2nd ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. Kaiserslautern: ACM, Oct. 2008, pp. 2–11.

[2] F. Palomba, D. 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.

[3] 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.

[4] H. Krasner, "The cost of poor quality software in the us: A 2018 report," *Consortium for IT Software Quality, Tech. Rep*, vol. 10, 2018.

[5] 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.

[6] 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.

[7] 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.

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

[9] R. Burt, *Structural Holes versus Network Closure as Social Capital. In Social Capital: Theory and Research.* Elsevier Science Inc., 01 2001.

[10] M. Y. Allaho and W.-C. Lee, "Analyzing the social ties and structure of contributors in open source software community," in *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. Niagara, Canada: ACM, 2013, pp. 56–60. [Online]. Available: https://doi.org/10.1145/2492517.2492627

[11] 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.

[12] B. Xu and D. Jones, "Volunteers' participation in open source software development: A study from the social-relational perspective," *ACM SIGMIS Database*, vol. 41, no. 3, pp. 69–84, Sep. 2010.

[13] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in GitHub: Transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW)*. ACM, 2012, pp. 1277–1286.

[14] 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.

[15] 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.

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

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

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

[19] 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 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. Toronto: ACM, 2011, pp. 177–187.

[20] 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.

[21] E. Guzman, D. Azócar, and Y. Li, "Sentiment analysis of commit comments in github: An empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 352–355.

[22] 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.

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

[24] V. Lenarduzzi, F. Lomio, H. Huttunen, and D. Taibi, "Are SonarQube rules inducing bugs?" in *Proceedings of the 27th IEEE International Conference on Software Analysis, Evolution and Reengineering SANER*. London, Canada: IEEE, Feb. 2020, pp. 501–511.

[25] 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.

[26] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *IEEE Software*, vol. 29, no. 6, pp. 18–21, 2012.

[27] J. J.Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, p. 1–5, May 2005. [Online]. Available: https://doi.org/10.1145/1082983.1083147

[28] T. Parr, K. Turgutlu, C. Csiszar, and J. Howard. (2020) Beware default random forest importances. [Online]. Available: https://explained.ai/rf-importance/index.html

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

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

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

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