From Team Traits to Security 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 improve the security, correctness, and maintainability of software a team produces by identifying measurable team traits and their relationship to the security, correctness, and maintainability of the software produced 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].

The quality of the collaboration is characterized by a software development team's traits—traits which the team and its members exhibit during joint development of a software artifact. Previous works refer to the team traits with negative connotation as community smells [1]. We understand *team traits* 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. They discovered *community smells* through interviewing developers.

Effective communication is integral to efficient collaboration. Existing studies in project management have identified communication between team members as a key aspect which affects team performance [2], however, how communication affects performance of a software development team and how that reflects in software artifacts the team produces need to be studied.

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 social debt that occur in software development teams [1]. *Security smells* are a class of code smells which indicate security weaknesses in code base [5], that also need more reliability efforts and maintainability efforts from developers. In the remainder of this paper, *code smells* refer to fault-prone and security smells. NSA: Need to better scope the smells part

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 in understanding of team traits and how they relate to reliability and maintainability of the code base teams produce by an empirical study of open source teams, their interactions, and the produced code base. We analyze team structure, source code, commit messages, issues, pull requests and associated comments of NSA: m open source projects hosted on GitHub.

Specifically, we investigate the following research questions:

RQ 1 (traits): What are different traits of an open source team?

RQ 2 (detection): How can we automate the detection of the identified team traits?

RQ 3 (influence): How does the identified team traits influence the code smells in a code base?

To address RQ1, we survey research works in social psychology and software engineering and analyze NSA: m open source repositories on GitHub to identify team traits that characterize a software development team.

To address RQ2, we develop a tool, HAGRID, which leverage social network analysis and natural language techniques to automatically detecting team traits by analyzing team structure and interactions in the team occurring via commits, issues, and pull requests.

To address RQ3, we perform a case study on open source teams in which we (1) compute team traits from the collaboration in the team, (2) compute code smells in the software artifacts they produce, and (3) analyze the relation between the computed team traits and code smells. Specifically, in our case study, we apply HAGRID on NSA: n open source projects hosted on GitHub. We identify code smells in these NSA: n projects using SonarQube and Bandit, static analysis tools, and analyze how team traits influence code smells.

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

Organization. Section II describe the relevant related works. Section III describes the dataset we analyze. Section IV details the metrics of team traits and code base smells. Section V describes the method we use to extract metrics. Section VI describes our case study and its results on how team traits influence quality of code. Section VII concludes with discussion of future directions.

II. Related Works

MPS 1: See "Are SonarQube Rules Inducing Bugs?" from SANER 2020. http://www.taibi.it/node/168 It seems to question SonarQube so worth a reference but if we can get their dataset of open source projects, it may be worth seeing how our approach predicts the technical debt as they define it.

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.

Cheng et al. [6] investigated the importance of trust in semi-virtual collaboration among students in multi-culture and uni-culture teams. They identified three aspects, namely, language, values (e.g., attitude, perception) and habitual behaviors, that contributes to the trust difference between multicultural groups and uni-cultural groups.

III. Dataset

GitHub 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. It 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. In the remainder of the paper, whenever we use the term database we are referring to the GHTorrent database. The database dump used in this study was released on June 1, 2019. The database dump contained metadata for 16,331,225 GitHub repositories.

To identify the relevant repositories, we were encouraged by Munaiah et al. [7] 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" [8]. 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
5) number of pull requests are greater than NSA: p?
6) number of issues are greater than NSA: q?

We select Python because Python is ranked as the top-most programming language by IEEE Spectrum [9]. We select JavaScript because programming language researchers have identified JavaScript to be a language prone to attacks [**?**]. Its hard-to-analyze dynamic constructs such as *eval* and the lack of static typing can be exploited to launch browser-based security attacks [**?**].

Exclusion criteria.

We discard the repositories which meet the following exclusion criteria:

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

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

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

Data collection.

NSA: appx 2000 projects meet our filter criteria. Of these 2000, we create three random sets of 50 projects each.

We collect 150 repositories that we use to determine the team traits and security smells. We collect these 150 repositories from GHTorrent because GHTorrent monitors the GitHub public event time line. For each event, GHTorrent retrieves event's contents and their dependencies, exhaustively. For the repositories which meet the inclusion and exclusion criteria, we extract the attributes provided in Table I.

TABLE I
Basic statistics of the dataset. NSA: tbd: Include LOC

Number of	Min	Max	Mean	Median
Users	5	42	7	6
Commits	55	15,840	2,042	712
Pull requests	1	5,172	310	39
Issues	1	6,860	628	132
Duration (years)	1	4	3	3

In our study, a repository is represented using a set of five dimensions [7] which are represented on team traits and code base separately described in Section IV. These dimensions are:

1) Community, as evidence of collaboration.
2) History, as evidence of sustained evolution.
3) Issues, as evidence of project management.
4) Documentation, as evidence of project maintenance.
5) Repository Size, as a evidence of project scalability.

Collaboration in open source software development manifests itself as a community of developers. We propose two metrics, organization members and the number of commits, to quantify the community established around a source code repository.

IV. Metrics

We know define the metrics to characterize teams and software artifacts they produce.

A. Repository Context Metrics

The repository context metrics are based on [7]'s dimensions to perform analysis on open source software repositories. Table II lists these context metrics.

TABLE II
Context metrics to describe each repository.

Metrics	Description
nCommits	Number of commits from each repository
nMembers	Number of members from each organization
nIssues	Number of issues from each repository
nPullreqs	Number of pull-requests from each repository
nLOC	Number of lines that contain at least one non-whitespace character that is not part of a comment
nComment	Number of lines containing comments except where comment is just whitespace

NSA: revise the text below to be in sync with Table II. The presence of continued change indicates that the software system is being modified to ensure its activity. We propose two metrics, sustainability of commits and the number of pull-request, to be the time and frequency by which a repository is undergoing change.

In our study, we assume the continued use of the GitHub Issues feature to be indicative of management in a source code repository. We propose a metric, the number of issue, to quantify the continued use of GitHub Issues in a repository.

Source code comments were found to be most important, second only to the source code itself [7]. We restrict ourselves to documentation in the form of source code comments. We propose a metric, comment ratio, to quantify a repository's extent of source code documentation.

As [7] mentioned, repository size is statistically significantly associated with the binary-valued dimensions with at least medium effect size. Moreover, the utility of the number of lines as a metric is as a covariate of other metrics.

B. Team Traits Metrics

MPS 2: Does Herbsleb have relevant work here?

Team traits refer to characteristics of a team and its members individually as well as collectively which are observed when the members collaborate. The traits include:

- Communication quotient: is each member's contribution to information sharing within the team.
- Team cohesion: is the willingness to take input from team members and a belief that team is more important than individual members [10]. NSA: look at [11]
- Friendliness: is the emotional tie perceived between the team members. NSA: Use trust formula to compute [12]
- Situational awareness: NSA: whether team members show solidarity to others; do they give opinions and suggestions when asked for? MPS 3: see Arwen H. DeCostanza definition NSA: tbd: revise based on [13]

C. Code Quality Metrics

Prior research has demonstrated the existence of insecure coding practices in open source team [5]. Rahman et al. [14] found 13 types of security smells occur in Python Gists and found these smells do not often differentiate between famous and infamous authors. Falessi et al. [15] provides a listing of 221 rules of JavaScript code smells that vulnerabilities exist in coding practices propagating across the projects. These aforementioned sources raise the fact that insecure coding practices in open source available online can be a major source of introducing and spreading software security weakness throughout commercial projects. The code smells includes:

- Code Smell: Violations on a repository which are fault-prone [16]. NSA: We use SonarQube.
- Security Smell: Python smells of each repository that refer to security weakness. NSA: We use Bandit.

TABLE III
Code quality to describe each repository.

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

V. Method: Detecting Traits and Computing Metrics

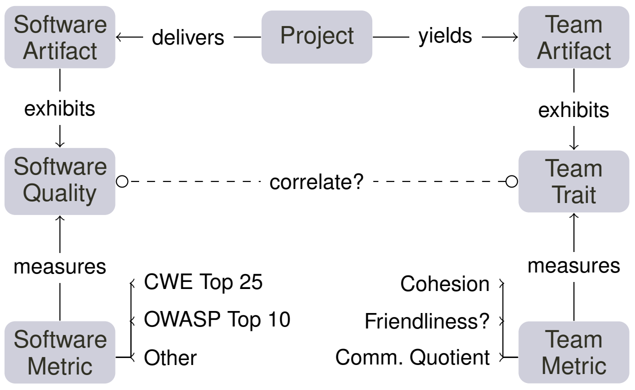

Fig. 1. Problem setting and claim. MPS 4: needs work;

We now describe our method to extract team traits and to compute code quality metrics.

A. Extracting Team Traits

NSA: The text below is less tooling and more metrics related.

We extract team traits 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

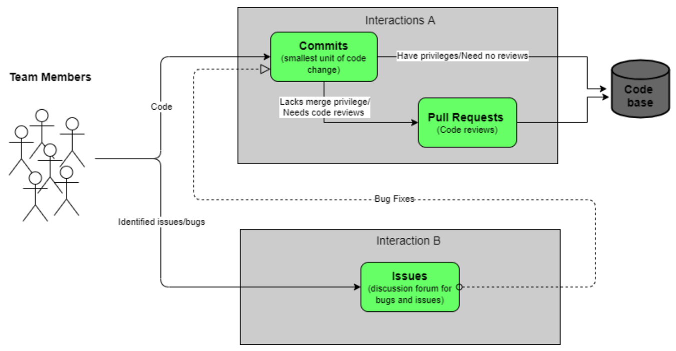
Team Members
Interactions A
Commits
(smallest unit of code change)
Have privileges/Need no reviews
Lacks merge privilege/ Needs code reviews
Pull Requests
(Code reviews)
Code base
Code
Bug Fixes
Identified issues/bugs
Interaction B
Issues
(discussion forum for bugs and issues)

Fig. 2. Interactions on GitHub. NSA: make bigger

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.

Following is a list of raw metrics we compute based on interactions among team members on GitHub.

Activity captures an individuals participation in these interactions using non-textual features. It is based on promptness in response and willingness to initiate conversations or information sharing. We compute activity for each team member based on the average response time of the member on online discussion threads, distribution of interactions across team members and timestamps, how often does a team member start a conversation vs participates in one.

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 [17] to compute politeness using features from online conversational texts.

Sentiment measures the polarity of a text based on the subjective information present in the comments and captures the social sentiments the text. We use VADER sentiment analyzer [18] 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 [18].

MPS 5: Can we consider one of Zhe's approaches for sentiment? NSA: Possibly. Need to think

Emotion captures the emotion expressed by words in a text based on associations of words with emotions. We use emotion lexicons proposed in [19] that associates 6400 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 [20]. We also include other lexical cues like emoticons, cuss words and ALL CAPS words in computing overall emotion metric.

We categorize the raw metrics to the following metrics:

Communication quotient is a measure of each individuals contribution in information sharing via various communication channels on GitHub, namely, issues discussion forum, commit messages, and pull request messages that are frequently used to share information and update team members of any code changes. NSA: Communication is computed from based on activity.

Team cohesion is the willingness to take input from team members and a belief that team is more important than individual members. We present team cohesion as encouraged by [10] which is the idea of willingness to contribute. We calculate the the average response time delay in seconds between messages sent and responses to such messages between team members.

Situation awareness

Friendliness is the emotional tie perceived between the team members. We compute friendliness based on sentiments associated with the interactions.

B. Extracting Code Quality Metrics

We first select sources that already have an existing insecure coding practices for multiple programming languages with examples of coding patterns. The list of these sources are 1) CWE; 2) Openstack Security Guidelines; 3) Bandit; 4) Sonar Source Security Hotspots; 5) OWASP.

Next, we map each of those identified smells to a potential security weakness indexed in CWE [21] and OWASP [22]. We have chosen CWE and OWASP mapping of security smells as they provide a list of common software security weaknesses which are through community-led open source software projects [22] and maintained by software security experts [21]. As aforementioned, the programming languages we select to detect security smells are Python and JavaScript, we identify security smells using Bandit [23] for Python which those security weakness are indexed in CWE [21], and violations such as reliability, maintainability and security using SonarQube [24] for both languages.MPS 6: explain the measures, including Sqale SonarQube 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 which 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 only referred to maintenance issues. We selected 20 change metrics that have been extensively used for defect estimation [15] and can describe the projects. Furthermore, we collected 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 [16].

Valentina et al.'s work [4] has verified the most fault-prone SQ-Violations on Java projects, we mapped those 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 Valentina 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" as in Table IV.

TABLE IV
METRICS TO ESTIMATE VIOLATIONS PER REPOSITORY.

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

C. Metric Collection

D. Social Network

We construct a directed social network for each discussion thread using the raw metrics described V-B. The social network has participants (or team members) as nodes and the edges are labelled with aggregate of the metrics for each pair of individuals involved in the discussion. We also construct an overall directed social network for the the entire project to see how individuals relate to each other in the social network and how it compares with individual threads for each individual. Outliers in these social networks are what we are looking for.

VI. CASE STUDY: INFLUENCE OF TEAM TRAITS ON SECURITY SMELLS

A. RQ 1. Traits

B. RQ 2. Detecting traits

C. RQ 3. Influence

D. Limitations and Threats to Validity

Internal validity.
External validity.
Construct validity.

VII. CONCLUSIONS AND FUTURE DIRECTIONS

MPS 7: Fix your bib files about the warnings—I fixed some

MPS 8: use full first names for each author or editor in your bib files

MPS 9: cite better sources

MPS 10: I fixed some bib warnings but Team.bib in unacceptably messed up.

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] 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 - INT J PROJ MANAG*, vol. 30, 02 2012.

[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] A. Rahman, C. Parnin, and L. Williams, "The seven sins: Security smells in infrastructure as code scripts," in *Proceedings of the 41st IEEE/ACM International Conference on Software Engineering (ICSE)*. Montreal: IEEE, 2019, pp. 164–175.

[6] 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, no. C, pp. 267—-276, Sep. 2016. [Online]. Available: https://doi.org/10.1016/j.chb.2016.03.093

[7] 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.

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] M. R. Rahman, A. Rahman, and L. Williams, "Share, but be aware: Security smells in python gists," in *Proceedings of the 35th IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Cleveland: IEEE, 1999, pp. 536–540.

[15] 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.

[16] 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

[17] 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.

[18] 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.

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

[20] 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

[21] CWE, "Cwe top 25 most dangerous software errors," 2019, https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html.

[22] OWASP. (2020) Owasp. [Online]. Available: https://owasp.org/

[23] Bandit. (2020) Bandit. [Online]. Available: https://github.com/PyCQA/bandit

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