A Preliminary Study on the Assignment of GitHub Issues to Issue Commenters and the Relationship with Social Smells

Haris Mumtaz
University of Auckland, New Zealand
hmum126@aucklanduni.ac.nz

Carlos Paradis
University of Hawaii, USA
cvas@hawaii.edu

Fabio Palomba
University of Salerno, Italy
fpalomba@unisa.it

Damian A. Tamburri
JADS, Netherlands
d.a.tamburri@tue.nl

Rick Kazman
University of Hawaii, USA
kazman@hawaii.edu

Kelly Blincoe
University of Auckland, New Zealand
k.blincoe@auckland.ac.nz

ABSTRACT

Background: GitHub is the world's largest software hosting platform. Its features affect millions of developers. Investigating the impact of GitHub features on software teams is essential to gain insights into features' usefulness.

Objective: As a preliminary step in this direction, this paper explores the relationship between the use of one GitHub feature and the social structure of the projects that adopt the feature. We explore whether the feature is used and whether the feature is associated with positive or negative changes in the team's social structure.

Method: In this paper, we report on a preliminary study of 13 projects that used the GitHub *"assign issues to issue commenters"* feature. We examine the social smells in the software teams before and after the introduction of this new feature using statistical and temporal analysis.

Results: Our results indicate that the usage of this feature varied across the analyzed projects. We also find that social smells that reflect low or missing communications (*Organizational Silo* and *Missing Links*) decrease in most of the projects that used the feature consistently.

Conclusion: The results suggest that the social structure of the teams has a positive relationship with the feature adoption. Still, future research should study the feature's impact (and its use cases) on other aspects and over longer time periods to learn its diverse and long-term benefits on the social structure of software projects.

CCS CONCEPTS

• **Software and its engineering** → **Programming teams**.

KEYWORDS

Socio-Technical Analysis; Social Smells; Community Smells; Open-Source Development; GitHub Features.

CHASE'22 , May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9342-3/22/05…$15.00
https://doi.org/10.1145/3528579.3529181

ACM Reference Format:
Haris Mumtaz, Carlos Paradis, Fabio Palomba, Damian A. Tamburri, Rick Kazman, and Kelly Blincoe. 2022. A Preliminary Study on the Assignment of GitHub Issues to Issue Commenters and the Relationship with Social Smells. In *15th International Conference on Cooperative and Human Aspects of Software Engineering (CHASE'22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3528579.3529181

1 INTRODUCTION

Code hosting platforms are an instinctive choice for a software community to enable communicative and collaborative development environment. GitHub is the world's largest and most popular software hosting platform. Its features affect millions of developers. However, how does GitHub go about choosing which features to offer? And how can a project assess whether they should adopt any given feature? This paper explores the relationship of one GitHub feature, *"assign issues to issue commenters*[1] *"*, with the social structure of the teams that use the feature. The rationale for choosing this GitHub feature is that it aims to bring new contributors to the project to assist the existing project team to develop a successful product. To understand the relationship between the feature and the teams social structure, we analyze the social smells in such teams. Social smells are sub-optimal patterns in the social and organizational structure of software teams [12]. Such sub-optimal patterns, if not removed, can lead to social debt resulting in software teams working inefficiently [11]. We measured social smells for one year before and after the introduction of the GitHub feature under analysis. This analytic lens allows us to take a socio-technical perspective, answering questions such as: is the feature used and is the use of the feature associated with positive or negative changes in the dynamics of a software team.

We structured our investigation around two research questions:

RQ1. *Did the software community adopt the GitHub feature after its initial usage?*

RQ2. *What is the relationship between the use of the GitHub feature and the socio-technical aspects of the software projects?*

Our results indicate that the level of adoption of this particular feature varied across the projects in our dataset, with some projects using it only rarely and others using it on many of their issue assignments. For those that used it consistently, we see that two of the three social smells that we analyzed (*Organizational Silo* and *Missing Links*) are lower after the adoption of the GitHub feature. Both of these smells indicate missing communication in

[1] https://github.blog/changelog/2019-06-25-assign-issues-to-issue-commenters/

the teams. This suggests that the feature might bring social benefits to the projects. However, we have examined only a small number of projects over a relatively short time period because the feature was only recently introduced.

We also notice that different projects adopt the feature in different ways, and thus, will experience different types of benefits. We encourage future research to more systematically study a wide range of GitHub features, and over longer time periods. For the software community to make the best use of the features (and to know which features to avoid), a better understanding is needed on the benefits that are derived from the ways each feature can be adopted. Both technical and social benefits should be examined, and investigated over the long term.

2 DATA DESCRIPTION AND ANALYSIS

This section describes our dataset, data collection process, and the methods employed to address our **RQs**.

2.1 Data Description

Dataset: We randomly select 13 Apache projects inspired by Palomba et al. [6], which emphasized selecting projects with sufficient contributors and development activity. In this study, all the projects use the feature under study at least once, and they have considerable developer involvement, commit activity, and longevity, making them suitable for studying their social structures. The project demographic data is presented in Table 1.

Social smells: To examine the relationships between the feature adoption and the socio-technical aspects of the software projects, we need measures that explain the socio-technical behavior in software teams. Since social smells measure sub-optimal socio-technical patterns [12], we investigate the three social smells in relation to the GitHub feature under analysis. The descriptions of social smells employed in this study are as follows:

(1) *Organizational Silo:* This smell occurs when there are highly decoupled development tasks involving isolated sub-groups of developers, causing a lack of communication in the community [11].

(2) *Missing Links:* This smell occurs when contributors make changes in the source code in isolation without any communication between their peers [11].

(3) *Radio Silence:* This smell occurs when all the interactions between the sub-communities happen through one or two team members [11].

The rationale for choosing these social smells is because they capture the sub-optimal patterns in the communication of the developers and the GitHub feature under investigation aims to introduce new contributors to the projects who might aid in lessening these sub-optimal communication patterns.

Data collection: The data collection process was divided into two phases. First, we collected the issue events that were assigned to the commenters of the issues for the analyzed projects. The second phase collected communication data to compute the projects' social smells. The data collection steps for identifying the issues assigned to issue commenters (GitHub feature) are:

- Collect project's issues using GitHub's *issue events API*[2];
- Fetch project's commits using GitHub's *list commits API*[3];
- Identify the contributors with no merge access by filtering issue assignees who are non-committers, indicating that the issues are assigned using the feature;
- Validate the obtained list of contributors (non-committers) using the committer information provided on the project's official webpage and the Apache Committers Directory[4].

The steps for computing the social smells are:

- Manually select the branches of the analyzed projects that overlap with the period of feature usage (i.e., pre-feature and post-feature);
- Collect communication data from Apache mailing lists, Jira, and GitHub using *Kaiaulu*[5] [8];
- Compute social smells for the selected branches of the projects using *Kaiaulu*.

The computed social smells and configuration files (for replication purpose) are provided in the supplementary material[6].

2.2 Analysis Methods

Method for RQ1 (feature adoption): We calculate the ratio of issue assignments using the feature to total issue assignments as a metric to show the extent to which projects have adopted the feature. The issue assignments made through the feature are computed for each quarter of the post-feature time period (i.e., one year) to reveal the changes over time in the frequency of feature usage. If issue assignments made using the feature occur in each quarter of the post-feature period, it is an indication that the project community finds the feature useful and has adopted it.

Methods for RQ2 (socio-technical relationships): We perform t-tests and temporal analysis to examine the relationship between the use of the GitHub feature and social smells. The t-test shows whether a statistically significant difference is observed in the social smells, pre-feature versus post-feature. The temporal analysis shows the behavior of social smells over time and explains whether the project team's dynamics change (improve or worsen) after feature adoption. The temporal analysis is performed by creating line plots for the pre- and post-feature time periods. A decline in the trend lines of the plots indicate that the social smells are decreasing (i.e., the social structure of the software teams has improved).

In our analysis, we only include those projects that used the feature consistently (at least three of the four quarters in the post-feature time period, based on the results of **RQ1**). Table 2 annotates the projects used for answering **RQ2**.

3 RESULTS ANALYSIS

3.1 Adoption of feature (RQ1)

We found that level of feature adoption varied widely across projects. 6 out of the 13 analyzed projects used it very sparsely, while the

[2]https://docs.github.com/en/rest/reference/issues#events
[3]https://docs.github.com/en/rest/reference/repos#commits
[4]https://home.apache.org/phonebook.html
[5]https://github.com/sailuh/kaiaulu/; For reproducibility, we used commit hash version: 44cebf8409d1403158a6f7e11f76c16d7662b1c3.
[6]https://figshare.com/s/a2e2aeec9f83e83dbaab

Table 1: Characteristics of the Apache projects analyzed in our paper.

					Communication channels			
Project	**#Commits**	**#Devs**	**#Branches**	**#Included branches**	**Mailing list**	**Jira**	**GitHub**	**Feature start**
Fineract	6,275	158	5	3	✓	✓	✓	16–07–2020
Camel-quarkus	3,189	57	31	14	✓	✓	✓	26–02–2020
Geode	10,583	133	14	10	✓	✓	—	06–12–2018
HBase	18,767	360	78	16	✓	✓	✓	27–04–2020
Ratis	1,191	51	11	3	✓	✓	✓	20–10–2020
ShardingSphere	30,248	287	5	4	✓	—	✓	15–05–2020
SkyWalking	6,762	380	15	2	✓	—	✓	18–06–2019
Ambari	24,588	134	63	8	✓	✓	✓	31–01–2018
Bookkeeper	2,517	120	19	11	✓	✓	✓	28–06–2017
CloudStack	34,572	342	195	3	✓	✓	✓	21–10–2020
Drill	4,047	170	30	3	✓	✓	✓	05–01–2019
Helix	4,110	38	41	7	✓	✓	✓	17–01–2020
Hive	15,742	288	42	2	✓	✓	✓	24–08–2020

Table 2: Issue assignments after feature adoption

	Issue assignments using feature				Total assignments				Percentage			
Project	**Q1**	**Q2**	**Q3**	**Q4**	**Q1**	**Q2**	**Q3**	**Q4**	**Q1**	**Q2**	**Q3**	**Q4**
Fineract	1	—	—	—	23	12	1	—	4.34	—	—	—
Camel-quarkus†	7	4	13	6	337	94	113	60	2.07	4.25	11.5	10
Geode	1	—	—	—	11	8	4	2	9.09	—	—	—
HBase†	11	4	—	14	136	71	37	67	8.08	5.63	—	20.89
Ratis	1	—	—	—	16	—	—	—	6.25	—	—	—
ShardingSphere†	24	60	36	57	250	229	176	134	9.6	26.2	20.45	42.53
SkyWalking†	10	22	28	24	103	127	109	76	9.7	17.32	25.68	31.57
Ambari†	44	47	53	2	790	646	524	190	5.56	7.27	10.11	1.05
Bookkeeper†	9	37	44	14	191	303	298	200	4.7	12.21	14.76	7
CloudStack†	21	42	22	1	89	181	120	1	23.59	23.2	18.33	100
Drill	1	1	—	—	22	6	3	3	4.54	16.66	—	—
Helix	5	—	3	—	49	6	29	2	10.2	—	10.34	—
Hive	3	—	1	—	42	17	24	13	7.14	—	4.16	—

† Project used for addressing RQ2

remaining 7 projects used it more regularly. Table 2 shows the number and percentage of issue assignments made using the feature (i.e., where someone is assigned who does not have merge access) for each of the four quarters after feature adoption. The ShardingSphere project had the highest ratio of issues being assigned to someone without merge access (over 40% of issues in the fourth quarter). Similarly, the SkyWalking project showed an escalating trend. Conversely, projects like Fineract and Geode did not use the feature after its initial usage.

Table 3: Statistical *t*-test

Social smell	***t*-test**
Organizational Silo	4.08^{***}
Missing Links	3.28^{**}
Radio Silence	2.31^{*}

* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

3.2 Social-Technical Relationships (RQ2)

The *t*-tests, listed in Table 3, show a significant difference in the social smells between the pre-feature and post-feature time periods. In many projects, the temporal analysis (line plots) shows a decline in the instances of *Organizational Silo* and *Missing Links* in the post-feature period. However, the changes in radio silence are arbitrary. One potential cause for this could be due to the absence of author names and e-mails for commenters who never made code changes. However, this limitation does not affect *Organizational Silo* and *Missing Links* as the metrics require, as a pre-condition, the existence of code changes without corresponding communications.

In the rest of the section, our results are described in terms of their characteristic patterns. Due to space limitations, we are only

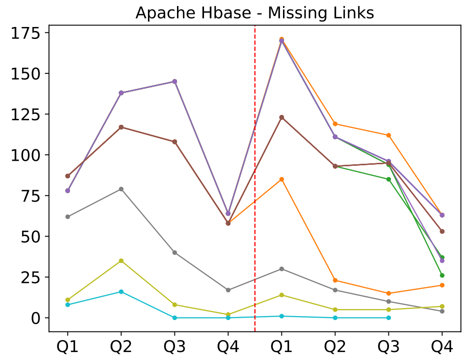

Figure 1: Immediate decline pattern of *Missing Links*. Each line represents a branch in the project.

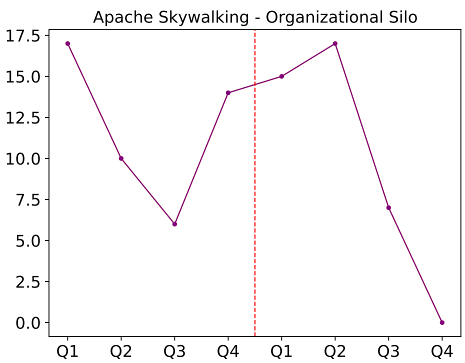

Figure 2: Dilatory decline pattern of *Organizational Silo*.

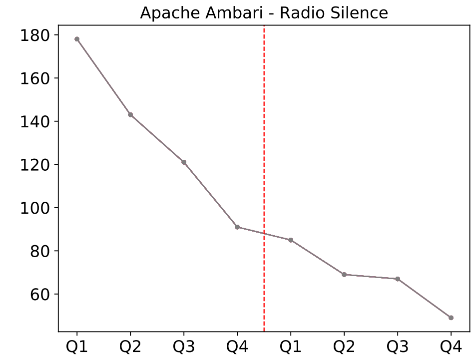

Figure 3: Continuous decline pattern of *Radio Silence*.

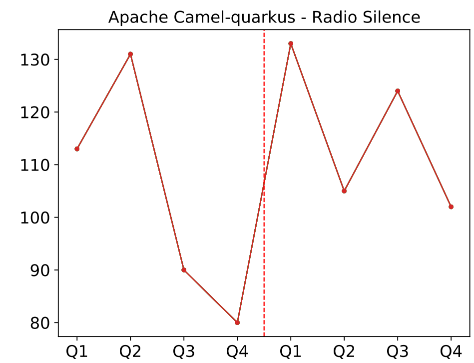

Figure 4: Arbitrary pattern of *Radio Silence*.

describing the frequently occurring patterns. Though, given the small sample size, these patterns need further validation.

Immediate decline pattern: The social smell immediately decreases after the feature is adopted. For instance, the line plot of *Missing Links* in Figure 1 shows the decline in the smell instantly after the feature usage[7]. We find that the immediate decline pattern occurs frequently in *Organizational Silo* and *Missing Links*.

Dilatory decline pattern: The social smell decreases after feature adoption, but not immediately, e.g., a smell declines after 90 days. Figure 2 is an example of this pattern[8], where *Organizational Silo* instances did not go down immediately but after the first quarter (90 days), a downward slope is evident. This pattern is observed in *Organizational Silo* and *Missing Links*.

Continuous decline pattern: The social smell is already declining before feature adoption and continues to decline in the post-feature time period. An instance of this pattern is shown in Figure 3, where it can be seen that *Radio Silence* smells are decreasing before the feature introduction and continue to decline after feature adoption. The continuous decline before the feature introduction could be because of the short release cadence of the project, pushing the contributors to stay active in their collaboration.

Arbitrary pattern: The social smell increases and decreases arbitrarily, not following a consistent trend. Figure 4 shows an instance of this pattern. The arbitrary pattern is mostly observed in *Radio Silence*.

4 RELATED WORK

Several studies have examined the effects of social decay [12] and architectural decay on project success, such as [10], which looked at the decline of SourceForge. At the same time, a number of studies of open-source forges—sometimes even specific case-studies thereof [2, 5]—and community portals such as GitHub have revealed the role of technical features over specific project characteristics (e.g., release readiness [1] or issue lifetimes [4]) but never really narrowing down the potential effects around the adoption of specific technical features on the long-lived organizational characteristics of software communities.

Conversely, organizational research has already investigated complex communities of practice from several perspectives, including software—e.g., escalation of commitment [3]—and put forth models and theories to mediate the nasty effects taking place across them [7] - this collective knowledge requires further research and empirical validation from a software engineering perspective.

5 DISCUSSION

This section describes the implications and limitations of this study and discusses the possible new directions for future work.

[7]In this and subsequent figures, the vertical *red* line splits pre-feature and post-feature time periods. Q1, Q2, Q3, and Q4 represent the four quarters exactly before and after the feature adoption date

[8]In this and subsequent figures, the difference in the branches is negligible, resulting in the lines overlapping each other.

Implication 1: *Long-term analysis of feature adoption is needed.* This study investigated a short time period after feature adoption because most of the analyzed projects started using the feature only recently. Despite this short time-frame, we already observed trends that indicate that a different level of usage of the feature might be associated with different socio-technical aspects in the software communities. For instance, in some projects, we found that the decrease in smells is associated with the frequency of the feature usage. For example, in Apache SkyWalking, *Organizational Silo* instances started declining as feature usage increased (see Figure 2 and Table 2). This suggests that the social structure of the project is getting better as more contributors are attached with the project. While our results need to be further corroborated, this seems promising. Indeed, our findings suggest possible long-term effects of feature usage on the social aspects of software projects and on project outcome and success measures, e.g., bug rates, bug resolution times, feature velocity, community size). Such long-term analysis can reveal whether and to what extent any feature is truly useful in the long run.

Implication 2: *Features are adopted in different ways and adoption patterns need further study.* Through some initial manual analysis of issues that employed the feature, we noted that the way the feature is being used appears to vary across projects. In some projects, we saw project contributors eliciting volunteers and assigning the issue to the person who volunteered. In this case, the feature could add accountability and also increase awareness of who is working on the issue. In other cases, we saw the issue assignment was made after the work had already been completed by someone without merge access. On these projects, this usage could provide incentive for contributors to receive credit for their work through this post-hoc assignment. The project may also utilize the feature to better track who has contributed to the project. Future research can more deeply study the different ways the feature is being used and evaluate and quantify the benefits and impacts that GitHub features bring across different usage scenarios. Such analysis can help projects decide not only which feature to adopt, but also *how* to adopt those features.

Implication 3: *Towards a broader study of human aspects of feature adoption.* This preliminary study of one GitHub feature can lay the foundation for future studies of a wider range of such features. In addition to examining social smells, future studies can examine a variety of human and social aspects related to the adoption of new features. Future studies could examine if and how new features attract new contributors. We could investigate how many new assigned contributors actually contributed to the project and how many of those new contributors become long-term members of the team. The types of issues that are assigned to the new contributors (e.g., documentation, bug fixes, non-trivial features, or system capabilities) could also be studied. Studying the diversity of the new contributors could also shed light onto which features contribute to inclusion on software teams.

Limitations: In addition to the relatively short time-frames that we considered, a limitation of this study is that we have only examined a small set of Apache projects. Thus, we see broadening our scope to additional projects and from other organizations (with different cultural norms) an important dimension of future work. Furthermore, there could be other confounding factors (e.g., COVID-19 pandemic, milestone releases, team membership, or other project-specific variables) that may explain the causality in the change of social smells during the studied time-frames. Future work could apply methods, such as Causal Impact Analysis or statistical methods (e.g., average smell value change over releases) to assess whether the change in the social health of the projects is truly due to the Github feature introduction. To further corroborate our findings, we plan to conduct a survey with the developers of the analyzed projects to confirm if they think the improvement in the social dynamics resulted from the Github feature adoption. The survey can also help obtain insights on the usefulness and quality aspects of the studied feature (and other features) from developers' point of view.

6 CONCLUSION

This is our first study investigating the relationship between the usage of a GitHub feature and a team's social structure and health. We see this research as a prototype for a much broader research program that could examine the consequences of any feature (or policy) that affects the social dynamics of a complex team.

Software projects are developed by teams of people and people are, de-facto, social animals. Most software engineering research has focused on the technical aspects of a project [9]. The work presented herein is complementary and, we believe, crucial for project success. That is, for a large-scale project to be successful over the long term attention needs to be paid to both technical and non-technical dimensions.

REFERENCES

[1] S. M. Didar Al Alam, S. M. Shahnewaz, Dietmar Pfahl, and Guenther Ruhe. 2014. Monitoring Bottlenecks in Achieving Release Readiness: A Retrospective Case Study Across Ten OSS Projects. In *8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement.* ACM, 60:1–60:4.

[2] D. Germán. 2003. The GNOME Project: A Case Study of Open Source, Global Software Development. *Softw. Process. Improv. Pract.* 8, 4 (2003), 201–215.

[3] Mark Keil, Bernard C. Y. Tan, Kwok Kee Wei, Timo Saarinen, Virpi Kristiina Tuunainen, and Arjen Wassenaar. 2000. A Cross-Cultural Study on Escalation of Commitment Behavior in Software Projects. *MIS Quarterly* 24, 2 (2000), 299–325.

[4] R. Kikas, M. Dumas, and D. Pfahl. 2016. Using Dynamic and Contextual Features to Predict Issue Lifetime in GitHub Projects.. In *13th Working Conference on Mining Software Repositories.* ACM, 291–302.

[5] S. Koch and G. Schneider. 2002. Effort, Co-operation and Co-ordination in an Open Source Software Project: GNOME. *Information Systems Journal* 12, 1 (2002), 27–42.

[6] F. Palomba, D. Tamburri, F. Arcelli Fontana, R. Oliveto, A. Zaidman, and A. Serebrenik. 2021. Beyond Technical Aspects: How Do Community Smells Influence the Intensity of Code Smells? *IEEE Transactions on Software Engineering* 47, 1 (2021), 108–129.

[7] G. Pan, S. Pan, M. Newman, and D. Flynn. 2004. Unfreezing-changing-refreezing of Actors' Commitment: The Transition from Escalation to De-escalation of Commitment to Information Technology Projects.. In *European Conference on Information Systems.* 1476–1487.

[8] C. Paradis and R. Kazman. 2021. Design Choices in Building an MSR Tool: The Case of Kaiaulu. In *1st International Workshop on Mining Software Repositories for Software Architecture.*

[9] M-A Storey, N. Ernst, C. Williams, and E. Kalliamvakou. 2020. The Who, What, How of Software Engineering Research: A Socio-technical Framework. *Empirical Software Engineering* 25, 5 (2020), 4097–4129.

[10] D. Tamburri, K. Blincoe, F. Palomba, and R. Kazman. 2020. "The Canary in the Coal Mine...": A Cautionary Tale from the Decline of SourceForge. *Software: Practice and Experience* (2020).

[11] D. Tamburri, P. Kruchten, P. Lago, and H. Van Vliet. 2015. Social Debt in Software Engineering: Insights from Industry. *Journal of Internet Services and Applications* 6, 1 (2015), 1–17.

[12] D. Tamburri, F. Palomba, and R. Kazman. 2019. Exploring Community Smells in Open-Source: An Automated Approach. *IEEE Transactions on Software Engineering* (02 2019), 1–1.