Exploring the Use of Labels to Categorize Issues in Open-Source Software Projects

Jordi Cabot, Javier Luis Cánovas Izquierdo, Valerio Cosentino, Belén Rolandi

Exploring the Use of Labels to Categorize Issues in Open-Source Software Projects

Jordi Cabot, Javier Luis Cánovas Izquierdo, Valerio Cosentino, Belén Rolandi
AtlanMod team (Inria, Mines Nantes, LINA), Nantes, France
{jordi.cabot,javier.canovas,valerio.cosentino,maria.rolandi}@inria.fr

***Abstract*—Reporting bugs, asking for new features and in general giving any kind of feedback is a common way to contribute to an Open-Source Software (OSS) project. This feedback is generally reported in the form of new issues for the project, managed by the so-called issue-trackers. One of the features provided by most issue-trackers is the possibility to define a set of labels/tags to classify the issues and, at least in theory, facilitate their management. Nevertheless, there is little empirical evidence to confirm that taking the time to categorize new issues has indeed a beneficial impact on the project evolution. In this paper we analyze a population of more than three million of GitHub projects and give some insights on how labels are used in them. Our preliminary results reveal that, even if the label mechanism is scarcely used, using labels favors the resolution of issues. Our analysis also suggests that not all projects use labels in the same way (e.g., for some labels are only a way to prioritize the project while others use them to signal their temporal evolution as they move along in the development workflow). Further research is needed to precisely characterize these label "families" and learn more the ideal application scenarios for each of them.**

I. Introduction

One of the main advantages of Open-Source Software (OSS) is that it enables anyone to be part of the development process at large [1]. Among the different ways to contribute (e.g., testing, coding, etc.), maybe the most common one is giving feedback by reporting bugs and requesting either improvements or new features. This feedback is reported in the form of new issues for the project, managed by the so-called issue-trackers.

As the project grows, issues become numerous and an appropriate management process, e.g., to decide which issues must be accepted or prioritize them, must be put in place. A common way to facilitate the organization of issues in projects is based on the use of labels [2]. Labels are a simple and effective way to attach additional information (e.g., metadata) to project issues. A label can give any user an immediate clue about what sort of topic the issue is about, what development task the issue is related to, or what priority the issue has. Furthermore, labels may also be useful for project administrators, since they can serve both as classification and filtering mechanism, thus facilitating the managing of the project. However, little effort has been devoted to the analysis of the impact of using labels in OSS projects.

In this paper we have analyzed a population of more than three million of GitHub projects in order to study how they use the label mechanism. GitHub has become one the most popular platforms to develop and promote OSS projects. With millions of repositories, GitHub gathers many of the important projects in the scene. Among its different functionalities, GitHub provides a light-weight and flexible issue-tracker which also includes labeling support. Our analysis focuses on the more than three million of original projects (i.e., projects that are not simply the fork of another existing project).

In particular, we have studied (1) how/which labels are used in GitHub projects and (2) how labels may influence the success of the project (mainly, in terms of the time it takes to address new submitted issues). Our results show that there exists a significant difference in terms of the number of treated issues between projects that do not use labels and projects that do. And the more different labels are used in the project the more this effect is observed. Besides this global analysis, a more fine-grained preliminary study aimed at studying how labels are used together is also conducted.

The rest of the paper is organized as follows. Section II introduces our analysis and the main results. Section III discusses the results and presents further analysis steps on the topic. Section IV reports on the related work and Section V gives some last remarks and future work.

II. Use of Labels in OSS

We perform our analysis over the population of repositories hosted in GitHub. GitHub provides an off-the-self issue-tracking system allowing projects to manage their development tasks by means of issues. Issues have a title and a description, and can be tagged with labels, linked to milestones and assigned to developers. The GitHub issue-tracking system comes with seven generic labels (i.e., *bug*, *duplicate*, *enhancement*, *help wanted*, *invalid*, *question*, *wontfix*), but more interestingly, it provides the capability to create custom labels adapted to the specificities of the project.

For our label analysis we used the dataset provided by GiLA [3], a tool which generates a set of visualizations to facilitate the analysis of project issues depending on their label-based categorization. GiLA in turn relies on GHTorrent, a scalable and offline mirror of data offered through the GitHub REST API [4], and extends it with a set of auxiliary tables and views storing semi-digested information to facilitate computations regarding projects (repositories), their users, issues and labels.

In our label analysis we targeted the following research questions:

- RQ1 Label usage. How many labels are used in Github? How many labels are used per project? What are the most popular ones?

TABLE I: Label usage in GitHub projects.

# labels	# projects	% labeled issues	avg. labels/issue	% users involved in labeled issues
1	55,561 (45.53%)	59.87% (σ = 0.37)	1.00 (σ = 0.02)	80.98% (σ = 0.30)
2	31,026 (25.42%)	61.00% (σ = 0.32)	1.02 (σ = 0.14)	72.06% (σ = 0.28)
3	13,390 (10.97%)	58.89% (σ = 0.30)	1.04 (σ = 0.21)	77.73% (σ = 0.30)
4	6,910 (5.66%)	58.94% (σ = 0.29)	1.06 (σ = 0.27)	75.81% (σ = 0.30)
5	4,206 (3.44%)	59.72% (σ = 0.29)	1.09 (σ = 0.33)	75.22% (σ = 0.30)
6	3,011 (2.46%)	56.16% (σ = 0.30)	1.08 (σ = 0.36)	72.05% (σ = 0.31)
7	1,934 (1.58%)	57.83% (σ = 0.29)	1.13 (σ = 0.41)	72.87% (σ = 0.30)
8	1,378 (1.13%)	58.99% (σ = 0.28)	1.18 (σ = 0.48)	72.52% (σ = 0.29)
9	955 (0.78%)	58.83% (σ = 0.28)	1.20 (σ = 0.51)	72.06% (σ = 0.28)
10	723 (0.59%)	55.06% (σ = 0.27)	1.25 (σ = 0.53)	69.25% (σ = 0.28)
>10	2,918 (2.60%)	55.88% (σ = 0.27)	1.52 (σ = 0.85)	70.43% (σ = 0.27)
Total	122,012	58.29% (σ = 0.30)	1.14 (σ = 0.37)	78.72% (σ = 0.30)

RQ2 Label influence. Does using labels influence the evolution of the project?

A. Label Usage

As a first step, we studied the ratio of projects with at least one issue labeled. We focused on original projects no matter their purpose (i.e., libraries, documentation, code snippets, etc.), thus allowing us to assess the label usage in the platform. In general, from 3,757,038 studied projects, there are only 122,012 (3.25%) of them containing labeled issues. The rest includes projects that do not use labels (i.e., either do not have issues at all or, if they have, none of them are labeled). This result reveals that the label mechanism is scarcely used in GitHub.

We then studied how many labels on average are used in each project[1], both in terms of distinct labels per project and per issue in the project, and how many users are involved in labeled issues. Table I shows the main results. Second column (*# projects*) shows that the vast majority of projects only use 1 or 2 different labels (45.53% and 25.42% respectively). Third column (*# labeled issues*) gives the average ratio of issues that are classified with at least one label, showing that around 60% of the issues are labeled regardless the number of distinct labels available in the project. Forth column reports on the average number of labels per issue, always around 1, also kind of a constant value regardless of the number of project labels. Finally, last column shows that around 78% of the users participating (e.g., commenting, opening, closing, etc.) in the project issues are involved in the labeled ones. With regard to the whole GitHub user population, the ratio of users involved in at least one labeled issue is 39.93%.

We also studied how labels are used according to the size of the project (in terms of number of issues). Table II shows the main data we gathered. We identified four groups of project sizes (see the four double-columns after the column *# labels*) and in a similar way as we did before, we calculated the number of projects along with the average ratio of issues that are labeled with at least one label. As can be seen, projects with less than 100 issues use quite regularly the label mechanism (they have around 70% of labeled issues), while projects with more than 100 issues only show an increment in the issues labeled when the projects use more than 10 labels.

[1]Note that several labels can be defined at project level. We only consider those labels being used in the project issues.

TABLE II: Label usage according to the number of issues in GitHub projects ($\sigma < 0.33$ for all the values of *% LI*).

# labels	# projects with X issues and % of labeled issues (LI)							
	$X \in (0, 10)$		$X \in [10, 100)$		$X \in [100, 1000)$		$X \in [1000, \infty)$	
	# proj.	% LI	# proj.	% LI	# proj.	% LI	# proj.	% LI
1	45,150	69.55%	9,996	18.39%	407	6.03%	8	28.67%
2	17,268	75.94%	13,203	43.48%	545	11.81%	10	14.41%
3	3,915	79.65%	8,771	52.64%	694	21.15%	10	5.78%
4	1,071	82.18%	5,121	58.36%	703	28.68%	15	14.45%
5	421	84.31%	3,115	62.72%	656	30.52%	14	17.28%
6	223	78.10%	1,995	63.68%	773	31.52%	20	12.80%
7	84	84.65%	1,177	66.88%	651	39.11%	22	24.43%
8	49	84.64%	823	67.81%	481	43.11%	25	22.83%
9	19	83.57%	518	72.16%	394	41.96%	24	28.31%
10	4	85.00%	337	69.74%	363	42.89%	19	21.60%
>10	12	82.64%	780	73.85%	1,765	52.25%	361	33.95%
Total	68,216	70.10%	45,836	45.68%	7,432	34.81%	528	29.54%

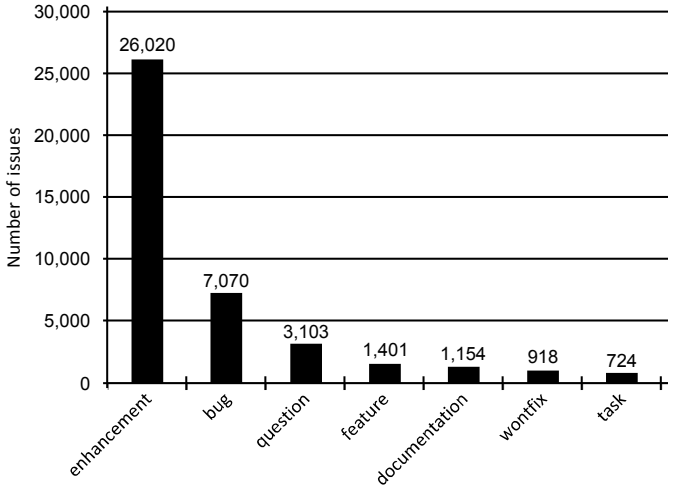

Fig. 1: Main labels used in GitHub.

Another interesting analysis is collecting the most popular labels across all projects. Figure 1 shows the seven most used labels together with the number of issues categorized with that label. As could be expected, some of the default labels are among the most used ones but, interestingly, custom labels such as *documentation* and *feature* are also broadly used, which may reveal a common need (at least from a user point of view) to demand more documentation and new features for projects.

Finally, we also studied which labels are normally used together by means of a co-occurrence matrix. We discovered that the most labels used together are *bug* and *enhancement* (around 40,000 times). Interestingly enough, other relevant label pairs are *bug-question*, *enhancement-question*, *wontfix-bug* and *wontfix-enhancement*, which are used around 10,000 times together each. The indistinct use of *bug* and *enhancement* in these pairs might reveal a misunderstanding of when they should be applied.

B. Label Influence

We have already seen that only a small percentage of projects choose to benefit from the label mechanism. In this section we look at those that do and try to see if labels have a positive impact on their evolution.

TABLE III: Label influence correlation analysis.

# labels used in the project	avg. time to solve (hours)	med. time to solve (hours)	avg. issue age (hours)	med. issue age (hours)	% solved	avg. people involved	# issues
0	786.43 (σ = 2,498.07)	26.93	7,927 (σ = 6,466.29)	6,774	22.53	3.35	2,408,224
1	795.40 (σ = 2,343.27)	46.18	4,516 (σ = 4,861.38)	2,577	43.51	3.40	111,839
2	808.60 (σ = 2,199.81)	74.92	5,867 (σ = 6,005.09)	3,747	48.76	5.44	187,705
3	937.30 (σ = 2,418.90)	101.30	7,540 (σ = 7,233.70)	6,346	53.21	9.73	158,172
4	998.50 (σ = 2,600.54)	111.80	8,646 (σ = 7,724.24)	7,427	55.27	15.61	134,617
5	1,111.00 (σ = 2,671.35)	145.70	9,196 (σ = 7,996.22)	8,081	56.30	20.93	111,186
6	1,060.00 (σ = 2,640.91)	116.40	9,729 (σ = 8,416.60)	8,335	58.82	25.91	99,749
7	1,152.00 (σ = 2,761.60)	127.20	9,330 (σ = 7,769.62)	8,268	57.95	35.18	91,584
8	1,139.00 (σ = 2,831.19)	116.40	10,610 (σ = 8,539.56)	9,154	59.28	46.79	81,657
9	982.90 (σ = 2,599.07)	70.40	10,100 (σ = 8,669.41)	8,612	63.23	51.24	70,146
10	1,425.00 (σ = 2,811.23)	306.40	8,321 (σ = 6,418.98)	7,276	47.59	71.67	84,193
> 10	1,148.00 (σ = 2,789.25)	148.10	8,302 (σ = 8,607.11)	5,918	60.19	129.18	727,024
ρ	**0.80**	**0.74**	**0.54**	**0.47**	**0.73**	**1.00**	

We measure this positive impact according to two dimensions (1) percentage of issue treatment, that is, the ratio of processed issues (indistinctly, merged or closed since both status reflect the fact that the project members considered and made a final decision on the issue's future); and (2) number of people involved in the discussion/treatment of the issues.

To evaluate this, we study the relationship between the number of distinct labels used in the project and six metrics, concretely: (1) average and (2) median time to resolve a labeled issue in the project, (3) average and (4) median age of labeled issues still pending, (5) percentage of labeled issues solved, and (6) average number of people involved (i.e., commenting, opening, closing or merging) in the project labeled issues. The Spearman's rho (ρ) correlation coefficient is used to measure the strength of monotonic relationships between the considered attributes. The values of ρ are in the range $[-1, +1]$, where a perfect correlation is represented either by a -1 or a $+1$, meaning that the variables are perfectly monotonically related (either a increasing or decreasing relationship, respectively). Thus, the closer to 0 the ρ is, the more independent the variables are.

Table III shows the results of the correlation analysis for projects using labels[2], including a column for each metric and, as reference, a column with the number of total issues. As can be seen, the ρ values regarding the average and median time to solve labeled issues and the percentage of solved labeled issues reveal high correlations. The latter result reveals that on average the percentage of solved labeled issues tends to increase together with the number of labels used in the project and it may confirm that the effort of categorizing issues is beneficial for the project advancement. The former results say that on average this might come at the cost of taking more time to solve those labeled issues. Such results are also linked with the perfect correlation value in the last column, stating that on average projects with a complex label management system in-place tend to involve more people in the discussion of those issues.

[2]The row regarding those projects not using labels is provided for the sake of comparison. Note that in this case we considered non-labeled issues to calculate the metrics.

It is important to note that the correlations have been calculated on aggregated data (e.g., average time to resolve issues in each group), although they are valid statistics [5], they cannot be used to make inferences at individual level [6], thus further investigations are needed to confirm that these correlations stand also for the single issues in each group.

III. Discussion and Further Work

While our study allows us to better characterize how labels are used in GitHub and, more importantly, to assess that the use of labels may influence positively the project success; however, looking at the standard deviations in Tab.III and quickly browsing a few projects immediately brings up the fact that projects use labels in many different ways. Some use labels to classify the kind of request, others their priority, yet others use labels to indicate the software component the issue is about. Clearly, these different label usages may bring different benefits (or drawbacks) to the projects and/or work better for different kinds of projects (e.g., depending on the project's size, language, complexity or seniority).

Therefore, we would like to dive deep down into the data to learn more about which projects benefit more from the label usage and how they do it. To this purpose, we start by analyzing if there are some specific label sets (what we call *label families*) that appear more frequently than others. These results would need to be confirmed later on with the project owners to understand better why they decided to choose that specific set of labels and how it is working for them.

To identify the families, we have performed a clustering analysis to aggregate labels commonly used together. Our aim is to identify clusters which maximize the number of common labels in their members. We applied an adapted version of the algorithm used by Tairas and Cabot [7], listed in pseudo-code in Figure 2. Although other classical clustering algorithms could be applied, we decided to employ this one due to its specific support for string-based groups (and the ease of being adapted to our problem). The input of the algorithm is a list of label groups used in GitHub projects (i.e., one label group per project). All label groups are passed through a pre-step which removes those groups that use less labels than the default ones

```
1:  selectedGroups ← ∅
2:  for group in labelGroups do
3:      if COUNT(group) ≥ 7 then
4:          selectedGroups ← selectedGroups ∪ group
5:      end if
6:  end for
7:  clusters ← ∅
8:  for group in selectedGroups do
9:      matched ← false
10:     for clusterinclusters do
11:         if MATCH(group, cluster) then
12:             cluster ← cluster ∪ group
13:             matched ← true
14:         end if
15:     end for
16:     if !matched then
17:         newCluster ← group
18:         clusters ← clusters ∪ newCluster
19:     end if
20: end for
```

Fig. 2: Clustering algorithm.

(i.e., lines $2-6$ in Figure 2) to reduce the number of groups considered and focus on the ones more potentially interesting. After this pre-step, each remaining group is evaluated for building the clusters (i.e., lines $8-20$ in Figure 2). The first iteration creates a cluster for the first label group evaluated. The second iteration evaluates the first two label groups. If they do not match, then the second label group is placed in a separate cluster. The third label group is then evaluated against the two original clusters (if the two labels groups did not match) and either added to one of them or placed in a third cluster if there is no match. The same process is iteratively applied to all remaining groups.

The comparisons between two label groups (i.e., function $MATCH$ in line 11 of Listing 2) is performed as follows. Two label groups are considered the same if either they include the very same labels or the number of different labels between them is less than two. To compare labels individually, we use the Levenshtein distance, thus only those labels with a distance of 2 or less are considered the same.

The top five clusters (in terms of number of projects in the cluster) detected by the algorithm are listed in Table IV. The detection of these clusters helped us to identify four label families (the fifth one being, as expected, the default set of labels provided by GitHub and their derivations, e.g., projects with only small changes with regard to the default set such as renaming the *bug* label as *important bug* or *ui bug*), specifically: (1) *priority labels*, i.e., labels that denote the priority of the tagged issues (e.g., *high-priority* or *urgent*); (2) *versioning labels* to indicate the version of the software to which the issue is referring to (e.g., *0.0.1* or *next-release*); (3) *workflow labels* to define the phase in the development process where the issue is located at the moment (e.g., *0 - backlog* or *1 - ready*); and (4) *architectural labels*, denoting the architectural components affected by the issue (e.g., *component-ui*, *component-emacs*). Note that the number of families is not related to the number of clusters. The output of the cluster algorithm allowed us to simplify the identification of the labels commonly used together.

To get a better idea of how representative are these families, we followed it up with a coverage analysis to determine how many projects fit into one of the identified label families. Note this number cannot just be taken from the clustering algorithm due to the high variability of label terms. A project can have a set of labels that are not syntactically equal to any of the families but convey the exact same semantic meaning than one of them and therefore should be considered to be covered by it. Therefore, for the coverage analysis, we first semi-automatically classified each label as semantically equivalent to one of the four label families, if the label name could fit in that group. In a second step, projects are assigned to one of the families if its labels have been regarded as equivalent to those in the family. We only focus on four families and discard the family of default ones since, being the default GitHub option, the results we could get with that family would not be so relevant.

TABLE IV: Excerpt of the clusters detected.

Cluster	Labels
A	*0 - backlog 1 - ready 2 - working 3 - done breaking bug build contribution documentation duplicate easy fix enhancement invalid jump in p1 p2 p3 question refactoring removal taken wontifx*
B	*bug change docs duplicate enhancement feature fixed invalid migrations question wontfix*
C	*activity bug discuss documentation duplicate enhancement feature improvement incompatible invalid milestone question remove task urgent wontfix*
D	*bug component-keymap-emacs component-keymap-vim component-logic component-mode-haskell component-mode-perl component-notyi component-ui component-ui-cocoa component-ui-gtk component-ui-pango component-ui-vty duplicate enhancement frontend-gtk frontend-pango frontend-vty imported invalid keymap-emacs keymap-vi milestone-release0.4 milestone-release0.7 opsys-all opsys-osx osx performance priority-critical priority-high priority-low priority-medium type-cleanup-refactor type-other type-task usability wontfix*
E	*0.0.1 0.0.3 0.1.0 0.2.0 0.3.0 0.4.0 0.5.0 1.0.0 1.0.0.rc1 breaking bug doc enhancement invalid new update*

TABLE V: Coverage analysis of the label families.

Family	**# labels**	**% projects**
Priority	1,027 (2.33%)	4.33%
Version	2,703 (6.14%)	1.68%
Workflow	1,972 (4.48%)	5.67%
Architecture	1,104 (2.51%)	2.00%

Table V shows the coverage of each of the four label families with regard to the label population in GitHub (second column) and the project (third column). The results show that around a 15% of labels are related to one of the families. It is important to note that some labels are very specific and only used in a small set of projects (e.g., labels not written in English) while others are used in a considerable number of projects. Results also indicate that around 13% of the projects adhere to one of the families.

Considering that the easiest option for projects is to limit themselves to use the default label set instead of creating their own labeling strategy we believe the above numbers are significant and highlight that projects doing the effort of accu-

rately managing issues by means of the label mechanism ended up following only a few different strategies corresponding to the four clusters identified before. Nevertheless, as mentioned above, more research needs to be done to confirm and expand these preliminary results.

IV. Related Work

Tagging is at the core of social networks such as Twitter, Flickr, Facebook, etc., where users can annotate their contributions. The very own nature of the labeling mechanism allows people to freely define new labels in contrast to other formal classification mechanisms. Several works have analyzed how labels are used in these platforms (e.g., [8], [9], [10]).

The use of labels has also been extended to code hosting platforms where tools like issue-trackers (as the ones provided in popular platforms like GitHub or BitBucket) offer this functionality. Nevertheless, little effort has been done on analyzing how labels are used in a population of software projects and on whether their use has any impact on the project evolution and success.

Other works ([11] and [12]) study the label usage but in only two projects reporting on the label adoption, label categories and their purpose. Our study however focuses on the set of GitHub projects, thus providing results at a much larger scale.

The use of issue-tracking systems have been studied to assess whether they have an impact in the project, concluding that such impact is disputed [13]. On the other hand, Giger et al. [14] presents an approach to predict the fix time of bug issues. Our work could complement both studies by factoring in the label dimension in them.

The study presented by Posnett et al. [15] reports that releases with high activity in improvements and new features often come with reduced effort in defect fixing. We believe it would be interesting to generalize this work by considering all kinds of labels.

Storey et al. [16] present an empirical study that explores how task annotations embedded within the source code play a role in managing personal and team tasks. The work relies on the tool called TagSEA [2] which supports programmers in defining semantically rich annotations to source code comments. Our approach is focused on analyzing issue labels but it would be interesting to extend it to deal with labels directly embedded in the source code.

GitHub data has been the target of a number of research papers. Our approach relies on the GHTorrent dataset [4] but there are other options such as GitHub Archive [17] which aim to simplify the mining of GitHub data. Additionally, recommendations on how to mine GitHub data are provided by Bird et al. [18]; we have taken them into consideration when performing our analysis.

V. Conclusion

In this paper we have studied how labels are used in GitHub projects. Our results show that even if they are used scarcely in the platform, when they are they may have a positive impact on the project. We have also reported that projects employ labels in different ways classifying issues according to priority, affected component, workflow process, etc., according to our preliminary analysis of the top "label families". Tool support for a deeper analysis of the label usage in specific projects is also provided.

Next steps will focus on continuing our investigations around the concept of label families. In particular, we would like to study how each family may influence the project success, according to various definitions of success, and to better understand why projects choose a specific label set instead of another (or none at all) depending on the project characteristics (size, domain, language, community organization, etc). Besides this, we would like to study how labels evolve during the life-cycle of the project, including the profile of users adding/removing them. Finally, we plan to extend our approach to other web-based code hosting services (e.g., BitBucket, SourceForge, etc.) to see if there are significant differences across them.

References

[1] C. Bird, A. Gourley, P. Devanbu, U. C. Davis, A. Swaminathan, and G. Hsu, "Open Borders? Immigration in Open Source Projects," in *MSR conf.*, 2007.

[2] M. Storey, J. Ryall, J. Singer, D. Myers, and M. Muller, "How Software Developers Use Tagging to Support Reminding and Refinding," *IEEE Trans. Soft. Eng.*, vol. 35, no. 4, pp. 470–483, 2009.

[3] J. L. Cánovas Izquierdo, V. Cosentino, B. Rolandi, A. Bergel, and J. Cabot, "GiLA: GitHub Label Analyzer," in *SANER conf. (To appear)*, 2015.

[4] G. Gousios, "The GHTorrent Dataset and Tool Suite," in *MSR conf.*, 2013, pp. 233–236.

[5] B. Monin and D. M. Oppenheimer, "Correlated Averages vs. Averaged Correlations: Demonstrating the Warm Glow Heuristic Beyond Aggregation," *Soc. Cogn.*, vol. 23, no. 3, pp. 257–278, 2005.

[6] S. Piantadosi, D. P. Byar, and S. B. Green, "The Ecological Fallacy," *Amer. J. Epidem.*, vol. 127, no. 5, pp. 893–904, 1988.

[7] R. Tairas and J. Cabot, "Cloning in DSLs: Experiments with OCL," in *SLE conf.*, 2011, pp. 60–76.

[8] S. A. Golder and B. A. Huberman, "Usage Patterns of Collaborative Tagging Systems," *J. Inf. Sci.*, vol. 32, no. 2, pp. 198–208, 2006.

[9] M. Ames, "Why We Tag : Motivations for Annotation in Mobile and Online Media," in *CHI conf.*, 2007, pp. 971–980.

[10] X. Xia, D. Lo, X. Wang, and B. Zhou, "Tag Recommendation in Software Information Sites," in *MSR conf.*, 2013, pp. 287–296.

[11] C. Treude and M. Storey, "Work Item Tagging: Communicating Concerns in Collaborative Software Development," *IEEE Trans. Soft. Eng.*, vol. 38, no. 1, pp. 19–34, 2012.

[12] C. Treude and M. Storey, "How Tagging Helps Bridge the Gap between Social and Technical Aspects in Software Development," in *ICSE conf.*, 2009, pp. 12–22.

[13] T. F. Bissyande, D. Lo, L. Jiang, L. Reveillere, J. Klein, and Y. L. Traon, "Got issues? Who Cares about it? A Large Scale Investigation of Issue Trackers from GitHub," in *ISSRE symp.*, 2013, pp. 188–197.

[14] E. Giger, M. Pinzger, and H. Gall, "Predicting the Fix Time of Bugs," in *RSSE conf.*, 2010, pp. 52–56.

[15] D. Posnett, A. Hindle, and P. Devanbu, "Got Issues? Do New Features and Code Improvements Affect Defects?" in *CSMR conf*, 2011, pp. 211–215.

[16] M. Storey, J. Ryall, R. I. Bull, D. Myers, and J. Singer, "TODO or To Bug : Exploring How Task Annotations Play a Role in the Work Practices of Software Developers," in *ICSE conf.*, 2008, pp. 251–260.

[17] GitHub Archive. http://www.githubarchive.org.

[18] C. Bird, P. Rigby, and E. Barr, "The Promises and Perils of Mining Git," in *MSR conf.*, 2009, pp. 1–10.