Wait For It: Determinants of Pull Request Evaluation Latency on GitHub

Yue Yu*†, Huaimin Wang*, Vladimir Filkov†, Premkumar Devanbu†, and Bogdan Vasilescu†
*College of Computer, National University of Defense Technology, Changsha, 410073, China
†Department of Computer Science, University of California, Davis, Davis, CA 95616, USA
{yuyue, hmwang}@nudt.edu.cn, {vfilkov, ptdevanbu, vasilescu}@ucdavis.edu

***Abstract*—The pull-based development model, enabled by git and popularised by collaborative coding platforms like BitBucket, Gitorius, and GitHub, is widely used in distributed software teams. While this model lowers the barrier to entry for potential contributors (since anyone can submit *pull requests* to any repository), it also increases the burden on *integrators* (*i.e.*, members of a project's core team, responsible for evaluating the proposed changes and integrating them into the main development line), who struggle to keep up with the volume of incoming pull requests. In this paper we report on a quantitative study that tries to resolve which factors affect pull request evaluation latency in GitHub. Using regression modeling on data extracted from a sample of GitHub projects using the Travis-CI continuous integration service, we find that latency is a complex issue, requiring many independent variables to explain adequately.**

I. Introduction

The pull-based development model [1] is widely used in distributed software teams to integrate incoming changes into a project's codebase [14]. Enabled by `git`, the distributed version control system, pull-based development implies that contributors to a software project need not share access to a central repository anymore. Instead, anyone can create *forks* (*i.e.*, local clones of the central repository), update them locally and, whenever ready, request to have their changes merged back into the main branch by submitting a *pull request*. Compared to patch submission and acceptance via mailing lists and issue tracking systems, the traditional model of collaboration in open source [3], [9], the pull-based model offers several advantages, including centralization of information and process automation: the contributed code (the patch) resides in the same version control system, albeit in a different branch or fork, therefore authorship information is effortlessly maintained; modern collaborative coding platforms (*e.g.*, BitBucket, Gitorius, GitHub) provide integrated functionality for pull request generation, automatic testing, contextual discussion, in-line code review, and merger.

Pull requests are used in many scenarios beyond basic patch submission, *e.g.*, conducting code reviews, discussing new features [14]. On GitHub alone, currently the largest code host in open source, almost half of the collaborative projects use pull requests exclusively (*i.e.*, all contributions, irrespective of whether they come from core developers with write access to the repository or from outside contributors, are submitted as pull requests, ensuring that only reviewed code gets merged) or complementarily to the shared repository model [14].

While the pull-based model offers a much lower barrier to entry for potential contributors, it also increases the burden on core team members who decide whether to integrate them into the main development branch [14], [17]. In large projects, the volume of incoming pull requests is quite a challenge [14], [21], *e.g.*, Ruby on Rails receives upwards of three hundred pull requests each month. Prioritizing pull requests is one of the main concerns of integrators in their daily work [14].

From a perspective of increasing numbers of pull requests, in this paper we report on a preliminary quantitative study that tries to resolve which factors affect pull request evaluation latency in GitHub. Using regression modeling on data extracted from a sample of GitHub projects using the Travis-CI continuous integration service, we find that:

- latency is a complex issue, requiring many independent variables to explain adequately;
- the presence of CI is a strong positive predictor;
- the number of comments is the best single predictor of the latency;
- as expected, the size of a pull request (lines added, commits) is a positive, strong predictor.

II. The Pull Request Evaluation Process

Continuous Integration: The principal mechanism through which integrators ensure that pull requests can be handled efficiently without compromising quality is automatic testing, as supported by continuous integration (CI) services [14], [17]. Whenever a new contribution is received by a project using CI, it is merged automatically into a testing branch, the existing test suites are run, and the submitter and integrators are notified of the outcomes. If tests fail, the pull request is typically rejected (closed and not merged in GitHub parlance) by one of the integrators, who may also comment on why the contribution is inappropriate and how it can be improved. If tests pass, core team members proceed to do a team-wide code review by commenting inline on (parts of) the code, and including requests for modifications to be carried out by the submitter (who can then update the pull request with new code), if necessary. After a cycle of comments and revisions, and if everyone is satisfied, the pull request is closed and merged. In rare cases, pull requests are merged even if (some) tests failed. Only core team members (integrators) and the submitter can close (to merge or reject—integrators, and to withdraw—submitter) and reopen pull requests.

CI is becoming widely used on GitHub: 75% of projects that use pull requests heavily also use CI, either in hosted services (*e.g.*, Travis-CI [25]) or in standalone setups [14].

Pull Request Evaluation: As described above, evaluating pull requests is a complex iterative process involving many stakeholders. Recently, researchers have started investigating the pull request workflow in GitHub [11], [14], [21], [22], trying to understand which factors influence pull request *acceptance* (*i.e.*, decision to merge) and *latency* (*i.e.*, evaluation time). However, (i) these effects are not yet well understood (*e.g.*, results from different studies often diverge, as discussed below); and (ii) pull request evaluation has not been studied in the context of CI.

Pull request acceptance has been studied by Gousios *et al.* [11], [14] and Tsay *et al.* [21], [22]. For example, using machine learning techniques, Gousios *et al.* [11] found that pull requests touching actively developed (hot) parts of the system are preferentially accepted. In contrast, based on results from a regression modeling study, Tsay *et al.* [21] argue that the strength of the social connection between the submitter and integrators is the determining factor of pull request acceptance. In a follow-up qualitative study, Gousios *et al.* [14] found that target area hotness, existence of tests, and overall code quality are recognized as important determinants of pull request acceptance by GitHub integrators taking part in their survey.

In the same qualitative study [14], the authors also asked survey respondents to: (i) rate several predefined factors by perceived importance to pull request latency; and (ii) list additional determinants of latency, if any. The following ranking of predefined factors emerged (in decreasing order): project area hotness, existence of tests in the project, presence of tests in the pull request, pull request churn (*i.e.*, number of changed lines), the submitter's track record, number of discussion comments, and number of commits. Among the many other self-identified factors, only reviewer availability and pull request complexity are recognized by at least 5% of their respondents. These factors, *perceived* to be important, differ significantly from those *data mined* in an earlier machine learning experiment (three classes: merged within one hour; one day; the rest) by the same authors [11], *i.e.*, the submitter's track record, the project size and its test coverage, and the project's openness to external contributors (in decreasing importance order).

Hypotheses: In this paper we focus on pull request latency, the understudied and arguably more complex of the two facets of pull request evaluation. The literature reviewed above revealed that maintaining software quality and handling a high volume of incoming pull requests are the top priorities of integrators, who subscribe to a complex evaluation process, involving code review and automatic testing. Even though prior work [11], [14], [21], [22] has identified several social and technical factors that influence pull request evaluation, as discussed above, we believe the intricacies of this process are not yet accurately captured. Still, to establish a baseline (and replicate previous results), we hypothesize:

H1. Previously-identified social and technical factors influence pull request latency in expected ways.

There is recurring support for the importance of process factors to team outcomes in the empirical software engineering literature, *e.g.*, [2], [16]. Focusing on the integrator workflow, we expect a multitude of process-related factors to influence pull request latency, *e.g.*, current workload and availability of reviewers, pull request description quality (akin to the quality of issue reports said to impact the issues' resolution time [7], [27]), and the integrators' responsiveness (in recent work [23] we found that pull request submitters may become discouraged by unresponsive integrators, leading to communication breakdown and conflict). We hypothesize:

H2. Process-related factors have a significant impact on pull request latency.

The general literature on CI suggests that the continuous application of quality control checks speeds up overall development, and ultimately improves software quality [8]. On GitHub, the integration of CI in the pull request workflow should enable project members to find integration errors more quickly, tightening the feedback loop. Based on reported widespread use of CI and the importance of testing related factors [14], [17], [21], we expect that CI will play a prominent role in the pull request evaluation process, enabling integrators to better deal with general issues of scale. We posit:

H3. CI is a dominant factor of pull request latency.

III. Methods and Data

Dataset and Preprocessing: We compose a sample of GitHub projects that make heavy use of pull requests *and* CI. In this preliminary study we only consider projects using Travis-CI,[1] the most popular CI service on GitHub, integrated into the pull request workflow [25]. Using the 10/11/2014 GHTorrent [10], [12] dump, we identify candidate non-forked projects that received at least 1000 pull requests in total, and were written in Ruby, Python, JavaScript, Java, and C++, the most popular languages on GitHub.[2] Then, we identify which of these projects use Travis-CI, with our previous techniques [25]. Finally, in trying to assemble a relatively balanced data set, we further select, from among these, the top ten projects (ranked by number of pull requests) for each programming language.[3] For each project, we further extract data about incoming pull requests, received after 01/01/2010 and closed by the time of data collection (January 2015; *i.e.*, we ignore pull requests that are still open), from GHTorrent (metadata, *e.g.*, number of comments) and the GitHub API (description title, body, and actual contents of the pull request, *i.e.*, *diffs*). For each pull request, we extract data about the automatic builds from the Travis-CI API [25]. Lastly, we identify the integrators as those project members that closed others' issues or pull requests, using GHTorrent. We also perform identity merging [24].

[1] https://travis-ci.com

[2] http://githut.info/

[3] We only select five projects for C++ and Java due to insufficient samples.

Table I
BASIC STATISTICS FOR OUR DATA SET. WE ONLY MODEL THE EVALUATION TIME OF PULL REQUESTS (PRS) TESTED BY CI AND MERGED.

Attributes	Ruby	Python	JavaScript	Java/C++	**Total**
#Integrators	220	177	190	103	690
#PRs received	28,409	28,903	26,983	18,989	103,284
#PRs merged	20,755	24,039	17,920	13,456	76,170
#PRs merged&CI-tested	11,562	11,955	11,821	5,510	40,848

Table I presents basic statistics about our dataset. In total, we collected 103,284 pull requests from 40 different projects. We found that 74% of pull requests have been merged (using heuristics similar to those in [13]), and that 59% have been submitted after CI was adopted (measured, per project, as the date of the earliest pull request tested by Travis-CI). In this preliminary study we only model the evaluation time of pull requests that have been tested by Travis-CI *and* eventually merged. Rejected pull requests and pull requests that do not undergo automatic testing may be subject to different processes; we will address these in future work.

Measures:

1) Outcome: The outcome measure is the ***pull request latency***, *i.e.*, the time interval between pull request creation and closing date, in minutes (in case of "reopened" pull requests, we only consider the date when they are first closed).

2) Predictors: We compute project-level, pull-request level, and submitter-level measures, as discussed in Section II.

Project age: At time of pull request creation, in minutes. Older projects are likely to have different contribution dynamics.

Team size: Number of integrators active (*i.e.*, closed at least one issue/pull request, not their own) during the three months prior to pull request creation. Larger teams may be better prepared to handle higher volumes of incoming pull requests.

Project area hotness: Median number of commits to files touched by the pull request relative to all project commits during the last three months.

Commits: Total number of commits part of the pull request. Travis-CI tests each commit separately.

Churn: Total number of lines added and deleted by the pull request. Bigger changes may require longer code reviews/testing.

Test inclusion: Binary variable measuring if the pull request touched at least one test file (based on file name/path regular expressions). Integrators prefer pull requests containing tests.

Comments: Total number of overall and inline comments part of a pull request discussion. Pull requests with lots of comments tend to signal controversy [6].

Submitter's success rate: Fraction of previous pull requests merged, relative to all previous pull requests by this submitter.

Integrator: True if the submitter is an integrator.

Strength of social connection: The fraction of team members that interacted with the submitter in the last three months (computed using *comment networks* [26]). Integrators may favor contributors more strongly connected to them.

Followers: Total number of GitHub developers following the submitter at pull request creation, as a measure of reputation.

Description complexity: Total number of words in the pull request title and description. Longer descriptions may indicate higher complexity (longer evaluation), or better documentation (facilitating evaluation akin to issue reports [27]).

Workload: Total number of pull requests still open in each project at current pull request creation time.

Integrator availability: The minimum number of hours $(0 \ldots 23)$ until either of the top two integrators (by number of pull requests handled the last three months) are active, on average (based on activity in the last three months), during 24 hours. Two reviewers find an optimal number of defects during code review [18], [19], hence our choice for top two.

Friday effect: True if the pull request arrives Friday [20].

#Issue tag and ***@mention tag***: Binary variables to encode the presence of "#" tags (links to issues) and "@" tags (to notify integrators directly) in the pull request title or description.

First human response: Time interval in minutes from pull request creation to first response by reviewers, as a measure of the project team's responsiveness.

Total CI latency: Time interval in minutes from pull request creation to the last commit tested by CI. The team-wide code review typically starts after all commits have been tested.

CI result: Binary variables to encode the presence of *errors* while running Travis-CI (most often, branch already deleted) and test *failures* across the different pull request commits.

Analysis: We use multiple linear regression to model the latency of evaluating pull requests. We build three models, the first only with predictors previously used in the literature (***H1***), and the subsequent two by adding groups of variables corresponding to ***H2*** and ***H3***. The age of the project, the team size, and their interaction were added to all models as control variables. All numeric variables were first log transformed (plus 0.5) to stabilize variance and reduce heteroscedasticity [5], then standardized (mean 0, standard deviation 1). To test for multicollinearity, we computed the variance inflation factors (VIFs) for each predictor (all remained well below 3, indicating absence of multicollinearity). We use the adjusted R^2 statistic to evaluate the goodness-of-fit of our models. For each model variable, we report its coefficients, standard error, and significance level. We consider coefficients important if they were statistically significant ($p < 0.05$). We obtain effect sizes from ANOVA analyses. The resulting multivariate linear regression models are shown in Table II.

IV. RESULTS

Model 1 has a relatively low goodness of fit ($R^2 = 36.2\%$). As expected, the pull request churn, size, and length of discussion play a dominant role in explaining the variance in the data. All three effects are highly significant, and together account for 85% of the variance explained. Pull requests with more discussion, consisting of more commits, and adding more lines of code are associated with longer evaluation latencies. Effects related to the submitter's track record, reputation, and social connection to project members are also highly significant, with smaller but still sizeable contributions to explaining the data variance. Pull requests by the core team members, contributors with more followers, more ties to project integrators, and higher previous pull request success rates are associated with

Table II
PULL REQUEST LATENCY MODELS

	Model 1		Model 2		Model 3	
	Coeffs(Errors)	Sum Sq.	Coeffs(Errors)	Sum Sq.	Coeffs(Errors)	Sum Sq.
(Intercept)	0.072 (0.009)***		0.045 (0.009)***		0.155 (0.008)***	
scale(log(proj_age))	0.022 (0.004)***	276.96***	−0.014 (0.004)**	276.96***	−0.028 (0.004)***	276.96***
scale(log(team_size))	−0.055 (0.004)***	7.92***	−0.108 (0.004)***	7.92***	−0.108 (0.004)***	7.92***
scale(log(n_additions + 0.5))	0.064 (0.005)***	3354.64***	0.065 (0.005)***	3354.64***	0.035 (0.004)***	3354.64***
scale(log(n_deletions + 0.5))	−0.016 (0.005)**	54.92***	0.001 (0.005)	54.92***	−0.000 (0.004)	54.92***
scale(log(n_commits + 0.5))	0.147 (0.005)***	3789.65***	0.130 (0.005)***	3789.65***	0.028 (0.004)***	3789.65***
scale(log(hotness + 0.5))	0.016 (0.004)***	74.31***	0.001 (0.004)	74.31***	0.016 (0.003)***	74.31***
pr_includes_testsTRUE	0.108 (0.010)***	194.91***	0.076 (0.009)***	194.91***	0.009 (0.008)	194.91***
scale(log(n_comments + 0.5))	0.409 (0.005)***	5482.16***	0.189 (0.005)***	5482.16***	0.037 (0.005)***	5482.16***
scale(submitter_success_rate)	−0.037 (0.005)***	432.86***	−0.023 (0.004)***	432.86***	−0.016 (0.004)***	432.86***
scale(strength_social_connection)	−0.072 (0.005)***	494.32***	−0.037 (0.005)***	494.32***	−0.052 (0.004)***	494.32***
scale(log(n_followers + 0.5))	−0.090 (0.004)***	358.74***	−0.108 (0.004)***	358.74***	−0.064 (0.004)***	358.74***
submitter_is_integratorTRUE	−0.129 (0.011)***	56.10***	−0.095 (0.010)***	56.10***	−0.078 (0.009)***	56.10***
scale(log(proj_age)):scale(log(team_size))	−0.074 (0.004)***	234.23***	−0.012 (0.004)**	107.07***	−0.016 (0.004)***	70.27***
scale(log(description_complexity))			0.115 (0.004)***	960.00***	0.087 (0.004)***	960.00***
scale(log(availability + 0.5))			0.037 (0.004)***	124.96***	0.033 (0.003)***	124.96***
scale(log(n_open_pr + 0.5))			0.166 (0.005)***	908.33***	0.151 (0.004)***	908.33***
Friday_effectTRUE			0.068 (0.010)***	34.68***	0.062 (0.009)***	34.68***
issue_tagTRUE			0.096 (0.009)***	56.72***	0.081 (0.008)***	56.72***
mention_tagTRUE			−0.060 (0.013)***	14.11***	−0.020 (0.012)	14.11***
scale(log(first_rsp + 0.5))			0.274 (0.005)***	1892.64***	0.243 (0.004)***	1892.64***
scale(log(team_size)):scale(log(workload+0.5))			−0.071 (0.004)***	163.80***	−0.041 (0.004)***	60.39***
scale(log(total_ci_time))					0.481 (0.005)***	3855.79***
ci_errorTRUE					−0.401 (0.009)***	977.41***
ci_failTRUE					−0.016 (0.009)	0.27
scale(log(first_rsp+0.5)):scale(log(total_ci_time))					−0.102 (0.003)***	434.71***
Adjusted R-squared	0.362		0.461		0.587	

$^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$

shorter evaluation latencies. Perhaps more surprisingly, project area hotness and test case inclusion have highly significant positive effects, *i.e.*, pull requests touching active parts of the system, and including tests, are associated with longer evaluation latencies. Since all predictors suggested by prior work are highly significant, we confirm ***H1***.

Model 2 offers a significantly better fit ($R^2 = 46.1\%$). Pull request churn, size, and length of discussion, all highly significant, remain the most prominent predictors, together explaining 67% of the variance explained. However, the new process-related factors are all highly significant, and have sizeable effects. Pull requests with later initial reactions from integrators (10% of the variance explained) tend to also be closed later, suggesting that the initial priorities integrators assign to pull requests very early in the evaluation process (the median first comment time is 16min) are already indicative of the (much later) closing time (median 11.2h). The description length (5%) seems indicative of a pull request's complexity (impact) rather than its legibility, since the effect is positive (longer closing time). The integrators' workload is another sizeable positive effect (5%), moderated by team size. Other positive, albeit smaller, effects are integrator availability (pull requests submitted outside "business hours" and on Fridays tend to be closed later) and links to issue reports; @mention tags have a small negative effect (pull requests assigned to reviewers early in the process, *i.e.*, at creation, tend to be processed quicker). Therefore, ***H2*** is confirmed.

Model 3 achieves the best fit among our models ($R^2 = 58.7\%$). The CI-related factors are highly significant and cover more than 20% of the variance explained, on par with the main social and technical effects (pull request churn, size, and length of discussion). The prominence of the total CI latency effect (16%) supports the process description in Section II: integrators wait for the automatic testing phase to end (median 39min) before proceeding to do a team-wide code review and eventually close the pull request. The total CI latency effect is moderated by first human response. As discussed above, CI errors will occur when the pull request has already been merged (then the branch has been deleted), hence the negative significant effect on latency. Therefore, ***H3*** is confirmed.

V. CONCLUSIONS AND FUTURE WORK

Allowing greater inclusivity in contributions can result in a deluge of pull requests, which, if unchecked, can significantly increase the burden on integrators in distributed software development projects. Our preliminary models show that pull request review latency is complex, and depends on many predictors. Naturally, the size of the pull request matters: the shorter it is the faster it will be reviewed. Other actionable strong predictors are the delay to the first human response and the availability of the CI pipeline. Improving on both may hasten the review process.

This preliminary study suffers from at least the similar threats that other preliminary studies do [4], [15]: possible issues with data gathering, no validation, and unrefined models. We are working on addressing all of these in a more mature study of this subject, that will also elaborate on the impact of CI on the distributed software development process.

VI. ACKNOWLEDGEMENTS

YY and HW acknowledge support from NSFC (grants 61432020, 61472430). VF, PD, and BV are partially supported by NSF (grants 1247280, 1414172).

REFERENCES

[1] E. T. Barr, C. Bird, P. C. Rigby, A. Hindle, D. M. German, and P. Devanbu. Cohesive and isolated development with branches. In *International Conference on Fundamental Approaches to Software Engineering*, FASE, pages 316–331. Springer, 2012.

[2] N. Bettenburg and A. E. Hassan. Studying the impact of social structures on software quality. In *International Conference on Program Comprehension*, ICPC, pages 124–133. IEEE, 2010.

[3] C. Bird et al. Open borders? immigration in open source projects. In *International Working Conference on Mining Software Repositories*, MSR, page 6. IEEE, 2007.

[4] C. Bird, P. Rigby, E. Barr, D. Hamilton, D. German, and P. Devanbu. The promises and perils of mining git. In *Working Conference on Mining Software Repositories*, MSR, pages 1–10. IEEE, 2009.

[5] J. Cohen, P. Cohen, S. G. West, and L. S. Aiken. *Applied multiple regression/correlation analysis for the behavioral sciences*. Routledge, 2013.

[6] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in GitHub: Transparency and collaboration in an open software repository. In *ACM Conference on Computer Supported Cooperative Work*, CSCW, pages 1277–1286. ACM, 2012.

[7] N. Duc Anh, D. S. Cruzes, R. Conradi, and C. Ayala. Empirical validation of human factors in predicting issue lead time in open source projects. In *International Conference on Predictive Models in Software Engineering*, page 13. ACM, 2011.

[8] P. M. Duvall, S. Matyas, and A. Glover. *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007.

[9] M. Gharehyazie, D. Posnett, B. Vasilescu, and V. Filkov. Developer initiation and social interactions in OSS: A case study of the Apache Software Foundation. *Empirical Software Engineering*, pages 1–36, 2014.

[10] G. Gousios. The GHTorrent dataset and tool suite. In *Working Conference on Mining Software Repositories*, MSR, pages 233–236. IEEE, 2013.

[11] G. Gousios, M. Pinzger, and A. v. Deursen. An exploratory study of the pull-based software development model. In *International Conference on Software Engineering*, ICSE, pages 345–355. ACM, 2014.

[12] G. Gousios, B. Vasilescu, A. Serebrenik, and A. Zaidman. Lean GHTorrent: GitHub data on demand. In *Working Conference on Mining Software Repositories*, MSR, pages 384–387. ACM, 2014.

[13] G. Gousios and A. Zaidman. A dataset for pull-based development research. In *Working Conference on Mining Software Repositories*, MSR, pages 368–371. ACM, 2014.

[14] G. Gousios, A. Zaidman, M.-A. Storey, and A. Van Deursen. Work practices and challenges in pull-based development: The integrator's perspective. In *International Conference on Software Engineering*, ICSE. IEEE, 2015. to appear.

[15] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining github. In *Working Conference on Mining Software Repositories*, MSR, pages 92–101. ACM, 2014.

[16] N. Nagappan, B. Murphy, and V. Basili. The influence of organizational structure on software quality: An empirical case study. In *International Conference on Software Engineering*, ICSE, pages 521–530. ACM, 2008.

[17] R. Pham, L. Singer, O. Liskin, and K. Schneider. Creating a shared understanding of testing culture on a social coding site. In *International Conference on Software Engineering*, ICSE, pages 112–121. IEEE, 2013.

[18] P. C. Rigby and C. Bird. Convergent contemporary software peer review practices. In *SIGSOFT Foundations of Software Engineering*, FSE, pages 202–212. ACM, 2013.

[19] C. Sauer, D. R. Jeffery, L. Land, and P. Yetton. The effectiveness of software development technical reviews: A behaviorally motivated program of research. *IEEE Transactions on Software Engineering*, 26(1):1–14, 2000.

[20] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? *ACM Sigsoft Software Engineering Notes*, 30(4):1–5, 2005.

[21] J. Tsay, L. Dabbish, and J. Herbsleb. Influence of social and technical factors for evaluating contribution in GitHub. In *International Conference on Software Engineering*, ICSE, pages 356–366. ACM, 2014.

[22] J. Tsay, L. Dabbish, and J. Herbsleb. Let's talk about it: Evaluating contributions through discussion in GitHub. In *ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE, pages 144–154. ACM, 2014.

[23] B. Vasilescu, V. Filkov, and A. Serebrenik. Perceptions of diversity on GitHub: A user survey. unpublished, 2015.

[24] B. Vasilescu, D. Posnett, B. Ray, M. G. J. van den Brand, A. Serebrenik, P. Devanbu, and V. Filkov. Gender and tenure diversity in GitHub teams. In *CHI Conference on Human Factors in Computing Systems*, CHI. ACM, 2015. to appear.

[25] B. Vasilescu, S. van Schuylenburg, J. Wulms, A. Serebrenik, and M. G. J. van den Brand. Continuous integration in a social-coding world: Empirical evidence from GitHub. In *International Conference on Software Maintenance and Evolution*, ICSME, pages 401–405. IEEE, 2014.

[26] Y. Yu, H. Wang, G. Yin, and C. Ling. Reviewer recommender of pull-requests in GitHub. In *International Conference on Software Maintenance and Evolution*, ICSME, pages 609–612. IEEE, 2014.

[27] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss. What makes a good bug report? *IEEE Transactions on Software Engineering*, 36(5):618–643, 2010.