Continuous integration in GITHUB : experiences with TRAVIS-CI

Continuous integration in GITHUB: Experiences with TRAVIS-CI

Bogdan Vasilescu, Stef van Schuylenburg, Jules Wulms, Alexander Serebrenik, Mark G. J. van den Brand
Eindhoven University of Technology, Den Dolech 2, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{*b.n.vasilescu, a.serebrenik, m.g.j.v.d.brand*}*@tue.nl*, {*s.b.v.schuylenburg, j.j.h.m.wulms*}*@student.tue.nl*

I. INTRODUCTION

Continuous integration (CI) is a software engineering practice of frequently merging all developer working copies with a shared main branch [1], e.g., several times a day, or with every commit. This continuous application of quality control checks aims to speed up the development process and to ultimately improve software quality, by reducing the integration problems occurring between team members that develop software collaboratively [1].

With the advent of social media in (OSS) software development, recent years have witnessed many changes to how software is developed, and how developers collaborate, communicate, and learn [2]. One such prominent change is the emergence of the pull-based development model [3], made popular by the "social coding" platform GITHUB. In this model one can distinguish between *direct* contributions to a project, coming from a typically small group of developers with write access to the main project repository, and *indirect* ones, coming from developers who fork the main repository, update their copies locally, and submit pull requests for review and merger.

GITHUB's implementation of the pull-based development model enables anyone with an account to submit changes to any repository with only a few clicks. This represents an unprecedented low barrier to entry for potential contributors, but it also impacts testing behavior [4]: GITHUB project owners reported scalability challenges when integrating outside contributions, driving them towards automated tests. Automated CI services, such as TRAVIS-CI[1]—integrated with GITHUB itself—or JENKINS[2], facilitate this process: whenever a commit is recorded or a pull request is received, the contribution is merged automatically into a testing branch, the existing test suite is run, and the contribution author and project owner are notified of the results.

This extended abstract summarizes the finding of our study of TRAVIS-CI, arguably the most popular CI service on GITHUB [5].[3] We quantitatively explore to what extent GITHUB developers use the TRAVIS-CI, and whether the contribution type (direct or indirect) or project characteristics are associated with the success of the automatic builds.

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

[2]http://jenkins-ci.org

[3]As supported, e.g., by the blog entries https://blog.codecentric.de/en/2012/05/travis-ci-or-how-continuous-integration-will-become-fun-again/ and https://blog.futurice.com/tech-pick-of-the-week-travis-ci, acc. 6/2014

II. METHODOLOGY

To understand usage of the TRAVIS-CI service in GITHUB projects, we extracted and integrated data from two repositories: (i) GHTORRENT [6], a service collecting and making available metadata for all public projects available on GITHUB; and (ii) the TRAVIS-CI API[4].

Due to limitations of querying the TRAVIS-CI API, we focus on a sample of large and active GITHUB projects. Using the GHTORRENT web interface[5], we selected all GITHUB repositories that: (i) are not forks of other repositories; (ii) have not been deleted; (iii) are at least one year old; (iv) receive both commits and pull requests; (v) have been developed in Java, Python or Ruby; (vi) had at least 10 changes (commits or pull requests) during the last month; and (vii) have at least 10 contributors. We choose projects that receive both commits and pull requests, since we want to understand whether the way modification has been submitted (commit or pull request) can be associated with the build success. Our choice of the programming languages has been motivated by the history of TRAVIS-CI: TRAVIS-CI started as a service to the Ruby community in early 2011, while support for Java and Python has been announced one year later. We expect therefore the use of TRAVIS-CI to be more widespread for Ruby than for Java and Python.

The data were extracted on March 30, 2014. After filtering our sample contained 223 GITHUB projects, relatively balanced across the three programming languages: 70 (31.4%) in Java, 83 (37.2%) in Python, and 70 (31.4%) in Ruby.

To extract data about the automatic builds, we started by querying the `repos` endpoint of the TRAVIS-CI JSON API to determine whether TRAVIS-CI is configured for a particular project. Then, if the response was not empty, we iteratively queried the builds associated with this project (25 at a time as per the TRAVIS-CI API) from the `builds` endpoint, collecting the `event_type` fields (that distinguish pull requests from pushes) and the `result` fields (that specify whether the build succeeded—0, or failed—1).

III. RESULTS

We start by investigating the preference for direct and indirect contributions among the projects in our sample. The

[4]http://docs.travis-ci.com/api/

[5]Accessible from http://ghtorrent.org/dblite/

	Prog. lang.			Age (years)			Contributors		
	Java	Python	Ruby	<2	2–4	>4	≤17	17–33	>33
# projects	10	34	40	24	42	18	29	27	28
...s.t. $p < 0.05$	3	19	23	9	25	11	18	15	12
H_0	✗	✓	✓	✗	✓	✓	✓	✓	✗
%odds ratio>1	n/a	89	87	n/a	92	82	89	80	n/a

Table I
COMPARISON OF SUBGROUPS OF 84 GITHUB PROJECTS BASED ON THE PROGRAMMING LANGUAGE, AGE AND THE NUMBER OF CONTRIBUTORS.

shared repository model (with contributors having write access to the repository) is more popular among Java projects, while Python and Ruby projects have more contributors submitting pull requests. Overall, similarly to Gousios, Pinzger, and van Deursen [3], we see that direct code modifications are more popular than indirect ones, with only a small number of projects having more pull requests than commits.

Next we observe that an overwhelming majority of the projects are configured to use TRAVIS-CI (206 out of 223 projects, or 92.3%).However, slightly less than half of the 206 projects (93, or 45%) have no associated builds recorded in the TRAVIS-CI database. This shows that while most projects *are ready* to use continuous integration, significantly fewer *actually do*. Moreover, among the projects configured to use TRAVIS-CI but not actually using it, Java projects are overrepresented, while Ruby projects are underrepresented.

We have observed that the median success rate of 79.5% for commits and of 75.9% for pull requests. To obtain a more refined insight in whether the success of a build is independent from the way the modification has been proposed, we focused on projects that had at least 5 failed and at least 5 successful builds for each contribution type, as required by the χ^2 test of independence. Out of 113 GITHUB projects configured to use TRAVIS-CI and actually using it ($206 - 93 = 113$), 84 projects had sufficient data for the χ^2 test. Among the remaining 29 projects, in most cases it was the failed pull requests cell that had insufficient data, i.e., builds fail less frequently when contributions are submitted via pull requests. We believe this is because when a developer does not have commit rights, she will try harder to make sure the change is valid change and it will not break the build. However, when instead a developer has commit rights, she can try out new things more freely, since she also has the power to reverse the change.

The 84 projects subjected to the χ^2 test have been developed in different languages, have different ages and involve different numbers of contributors. Table I summarizes differences between those languages, ages, and numbers of contributors in terms of rejecting the null-hypotheses of the χ^2 test, i.e., independence of the build success from the way the modification has been proposed. We have used the common threshold of 0.05. The thresholds of 17 and 33 contributors correspond to the 33% and 67% percentiles. Performing Stouffer tests for each group led to very small p-values, indicating that results obtained for the majority of individual experiments can be lifted to the group level. Table I indicates that null hypotheses, e.g., can be rejected (✓) for Python and Ruby projects, but cannot be rejected (✗) for Java projects. Finally, all odds ratio tests for projects where the null hypothesis has been rejected (✗) for the group level turned out to be statistically significant ($p < 0.05$) and in almost all cases the odds ratios exceeded 1, i.e., whenever build success depends on the way the modification has been performed, pull requests are much more likely to result in successful builds than direct commits.

IV. CONCLUSIONS

In this paper we have studied a sample of large and active GITHUB projects. We observed that direct code modifications (commits) are more popular than indirect code modifications (pull requests). Next, we have seen that although most GITHUB projects in our sample are configured to use the TRAVIS-CI continuous integration service, less than half actually do. In terms of languages, Ruby projects are among the early adopters of TRAVIS-CI, while Java projects are late to use continuous integration. For those projects that actually use TRAVIS-CI, we conclude that pull requests are much more likely to result in successful builds than direct commits. However, we observe differences for projects developed in different programming languages, of different ages, and involving different numbers of contributors.

ACKNOWLEDGEMENTS

Special thanks to Mathias Meyer and the Travis CI team for helping us query their API. Bogdan Vasilescu gratefully acknowledges support from the Dutch Science Foundation (NWO) through the NWO 600.065.120.10N235 project.

REFERENCES

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

[2] B. Vasilescu, A. Serebrenik, P. T. Devanbu, and V. Filkov, "How social Q&A sites are changing knowledge sharing in open source software communities," in *CSCW*. ACM, 2014, pp. 342–354.

[3] G. Gousios, M. Pinzger, and A. van Deursen, "An exploratory study of the pull-based software development model." in *ICSE*, 2014, pp. 345–355.

[4] R. Pham, L. Singer, O. Liskin, K. Schneider *et al.*, "Creating a shared understanding of testing culture on a social coding site," in *ICSE*. IEEE, 2013, pp. 112–121.

[5] 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 *ICSM*. IEEE, 2014, pp. 401–405.

[6] G. Gousios, B. Vasilescu, A. Serebrenik, and A. Zaidman, "Lean GHTorrent: GitHub data on demand," in *MSR*, 2014, pp. 384–387.