Replication Experiment: Continuous integration in a social-coding world

Empirical evidence from GITHUB

Hichame Yessou
University of Antwerp
Middelheimlaan 1, 2020 Antwerp, Belgium
hichame.yessou@student.uantwerpen.be

Abstract— ***The social coding world is an approach to software engineering mainly based on the Git version control system. In the last years these code sharing platforms had an enormous success, thanks to how easy is to share code and the usefulness of having another point of view of the problem from the online community. With this enormous number of dislocated developers the quality of the code written can get worse and the testing and building process can take a lot of time to get done. One of the most used on GitHub is Travis CI that detects automatically when a commit or a pull request is pushed to any branch of the repository that uses Travis CI. Then the service will build the project and run the test. In this replication experiment we will re examin how the type of contribution and the predisposition to run automated build and tests can affect the successfulness of a project.***

I. INTRODUCTION

The Social-coding world is an approach to software engineering mainly based on the Git version control system that integrate the social network component. Allowing multiple developers to merge their work copies in a unique main branch following and discussing the variations besides the fact that you cannot lose your work and improving the organization of it. Many successful and famous companies use social coding as an approach to improve their code and/or services. The GitHub platform is one of the most popular on the web and allows users to get in touch with large projects regardless the location or the affiliation. These platforms normally are fully integrated with the social network features. They are making visible information about the user/developer across the team project and the web and even building a portfolio. Recently GitHub took down Google code, but it's not because GitHub offers a better version control system. GitHub took over the market because the social media component on the platform is taking a huge part of the user experience. The user's GitHub accounts contains projects like a Twitter account contains tweets or a Blog contains a post. It's even usual to use integrated apps to add the chat-communication aspect to the coding. One of the most well-known is Gitter that organize the chat rooms on the repositories and letting the developers to talk about topics with an extremely easy-to-use approach. The chat-room links the contributions or other users directly to the GitHub website or to the local GitHub client. However in GitHub it possible to contribute to the main shared repository in 2 ways. With direct code modifications to the main repository, normally used only by the owner of the project and/or small groups of developers.
And with indirect code modifications, copying the locally modifies to the main repository that everyone can make. The majority of the code sharing platforms supports automated build and test systems. Travis CI is mainly used with GitHub and allow to optimize the development time and to improve the quality of the software.

II. PAPER REPLICATED

In the replicated experiment the author focused the first part one the retrieving the metadata from GHTorrent. This project is a scalable and queriable offline mirror of the data offered through the GitHub REST API and monitors the GitHub event time line. GHTorrent retrieves the contents and their dependencies storing the raw JSON data in a MongoDB database and the structure in a MySQL database. The second part was concerning the extraction of the data through the Travis CI API. Travis CI is a distributed continuous integration service used to build and test projects. To get the data of the selected projects it's needed to query the repos endpoints, incorporating the applicable criteria that follow the automatic builds and verifying if the project is configured for Travis CI and if the successfulness of the test was related with the GHTorrent data previously retrieved. Selecting the data from the GHTorrent have been done through the GHTorrent web interface, setting up some parameters to limit the number of the projects. On the replicated experiment the limitation are regarding projects that: are not forks of other repositories; have not been deleted; are at least one year old; receive both commits and pull requests; have been developed in Java, Python or Ruby; had at least 10 changes (commits or pull requests) during the last month; and have at least 10 contributors. The choice of considering projects with both pull requests and commits have been done to understand if the way

of the contribution can affect the build successfulness. The extraction of the data of the automatic build and tests have been done querying the repos endpoints of the Travis CI JSON API, using *username/repo* as an argument.
If the response was not empty it was necessary to query iteratively the build associated to the project to understand the successfulness or the failure of the contribution. After that the paper aggregate the data in a contingency table for each project and applied the $\chi 2$ and Stouffer test.

III. Go-Decision

My go decision have been taken after checking the availability of the data and tools. I found easy to check it on the GHTorrent project, but for the Travis CI API queries have been a little bit trickier.

IV. Reproducibility

The reproducibility of the sample selection had to figure out how to access and query the data. After trying to query locally the restored MySQL dump database I decided to give up. With the DBLite web interface I could run queries in Guest mode without downloading it or asking for an SSH connections. I tried to register an account on the web interface but it's blocked. Asking the support they told me that it's allowed to access the DBLite web interface just with the Guest Mode. Getting the access to the database, the next task was concerning the retrieving of the data through a query matching the projects limitations explained before and on the paper. The export of the data can be data in HTML, XML and .CSV. Concerning the data integration I had to use Travis CI in a virtual machine environment, due to the fact that I am not running Linux on my laptop. Setting up and querying the repos and builds endpoints of the Travis CI JSON API in this environment was a very slow process. For the statistical analysis it was required to have a table for each projects listing all the pull requests/commits builds in a excel tables applying the $\chi 2$ test. Regarding the Stouffer test I didn't found enough documentation to understand and apply it.

V. Materials

To report the replication of this experiment I covered 5 main topics: Identification, Description, Availability, Persistence and Flexibility.
For each one I covered 3 branches: raw data, processed data and the tools.

A. Raw data

The GHTorrent raw data are extracted directly from GitHub public event timeline and are in a JSON format. To extract data regarding the automatic builds it's needed to query the endpoints of the Travis CI API that are also in a JSON format.

B. Processed data

The processed data are the extracted structure stored in a MySQL database.

C. Tools

Thanks to easy to use, the extraction of the data from the GHTorrent project have been done through the MySQL DBLite web portal.
To retrieve the build data it's necessary to use an API Client.

	Raw data	Processed data	Tools
Identification	X	X	X
Description	X	X	X
Availability	X	X	X
Persistence	X	X	
Flexibility	X	X	

Table 1 attributes for each type of element

VI. Methods

This replication experiment is partial procedural validation. A complete reproduction of the original one, using the different methodologies and the current dataset offered by the GHTorrent and the current data retrieved from Travis CI API.
I started from the same data source but the retrieval methodology can be different while the study parameters are the same.
The methodology might be different because the way how I retrieved the data can be different under several aspects. From the way on how the query is written, to the limitation (timing limitation generally) of the web portal on the query.
For simplicity I preferred to use just one big query that uses subqueries.
On the replicated study there was no information about the original resulting dataset so I cannot assess if the results of my replication study are correct or not.

A. *Identification*

The GHTorrent name comes from the initial way to distribute data, from the BitTorrent network. The GHTorrent project allows to download and access the MongoDB raw data and the MySQL dataset. Currently they can both be obtained from the website in different versions. There are several Travis CI API Client [1]that can be downloaded, some official and other from third party. Personally I used the command line client and a Ruby library to interface with the Travis CI service.

B. *Description*

GHTorrent provides the documentation of all the collections with the GitHub API URL from where they cache data.
The MongoDB dumps are incremental and mainly as a backup purpose. These dumps are presented as a daily and bi-monthly format and can contain duplicates.
The MySQL dump is a full dump of all the data available, it is a 1.5 billion rows for 34GB of a compressed .tar.gz file that becomes a 128GB .tar file and a final 130GB folder with one .csv file per table instead of the MongoDB dump backups. There are present restore scripts and instruction on how to do the restore.
It's available a full relational DB schema explaining how and why a certain table or attribute have been set up in that way. Instead the Travis CI API documentation is poorly detailed and for most of the issues it's needed to search the solution on the web. (I.e. Stackoverflow)

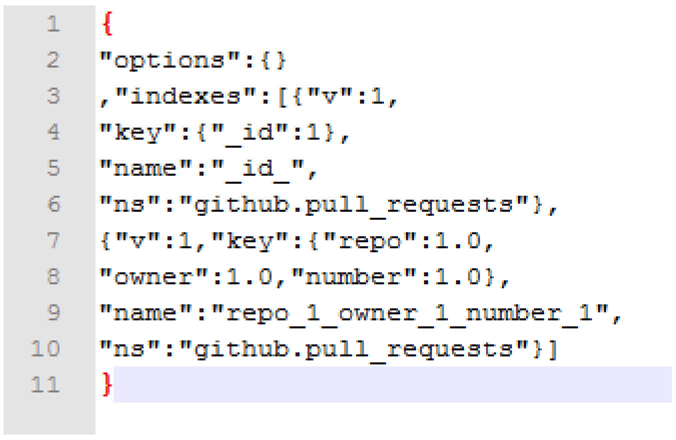
C. *Availability*

The MongoDB full dump is more than 6.5TB so the best way to access it is through their MongoDB server. The access is provided with a SSH connection, allowed only after sending them your public SSH key. It's needed only the MongoDB client and the SSH.
Accessing this dumps is doable through the restored database locally or more efficiently with the DBLite portal. The only way to access DBLite is logging in as "Guest" allowing you to do the queries. This way have some limitations. It's not allowed to use multiple tabs to make multiple queries and the web portal get stuck if it retrieve too much data and/or it is too much complicated (joins).
The Travis CI service requires to run on from a command line interface and to have installed at least Ruby 2.0

[1] https://docs.travis-ci.com/api#api-clients

D. *Persistence*

The availability of the MongoDB raw data is strictly related to the GitHub API and the publicity of the repos. The private repos cannot be retrieved, even if they are a minority as far as it is a premium service.
Concerning the dataset the availability of the accessibility of the element is associated to the "up-time" of the service offered by GHTorrent. On the usage terms is explicitly said that the service could stop in any time.



```
{
"options":{}
,"indexes":[{"v":1,
"key":{"_id":1},
"name":"_id_",
"ns":"github.pull_requests"},
{"v":1,"key":{"repo":1.0,
"owner":1.0,"number":1.0},
"name":"repo_1_owner_1_number_1",
"ns":"github.pull_requests"}]
}
```

Figure 2 Pull request json

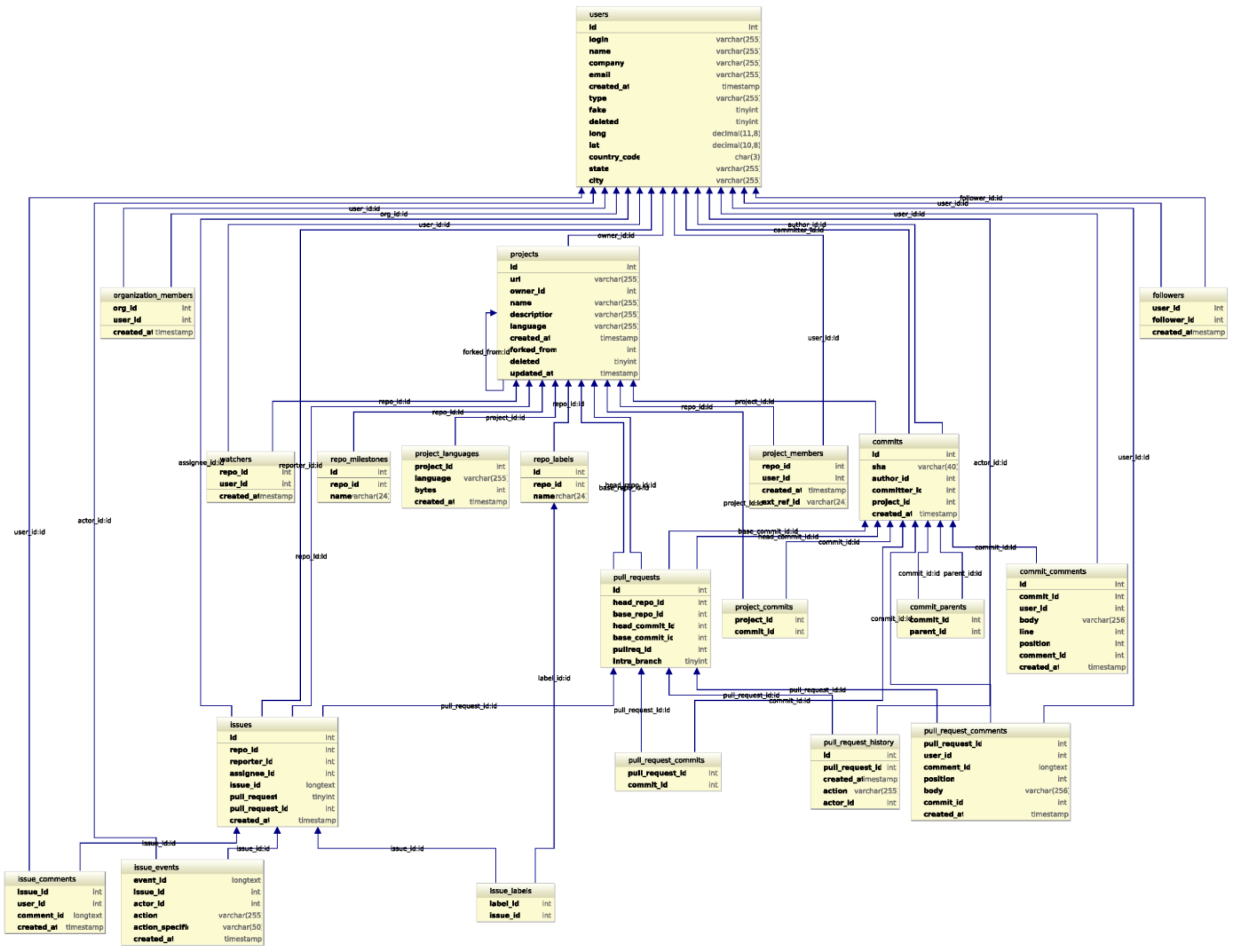


Figure 3 Relational data schema

Entity	Description	Raw Data Entity	Num Items
projects	Project repositories	repos	24044050
users	Github users	users	8791297
project	Users with commit access	repo_collabs	6365304
organization	Members of an organization	org_members	326229
commits	Commits	commits	*
commit	comments	commit_comments	2502720
watchers	users	watchers	34204412
followers	users	followers	9693732
issues	Issues	issues	23606974
issue	Events on a issue	issue_events	33377039
issue	Issue comments	issue_comments	*
pull	Pull requests	pull_requests	*
pull	Pull requests comments	pullreq_comnts	*

Table 1 Numbers of records (the * value means that the web interface was not able to retrieve the data/more than 60M records)

VII. Results

Before starting discussing the results I have to make some assumptions. Due to the enormous volume of the GHTorrent data the parameters given by the replicated experiment to limit the number of the projects was still not enough. The DBLite web interface in Guest mode enter in an infinite-loop with big queries. Adding another parameter on the selection of the project it might alter the validity of the data. The simplest way to reduce the data without altering the result is using the LIMIT. The records are stored on the database in a pseudo-random way, so taking the first "N" records can represents a sub-population of the data.

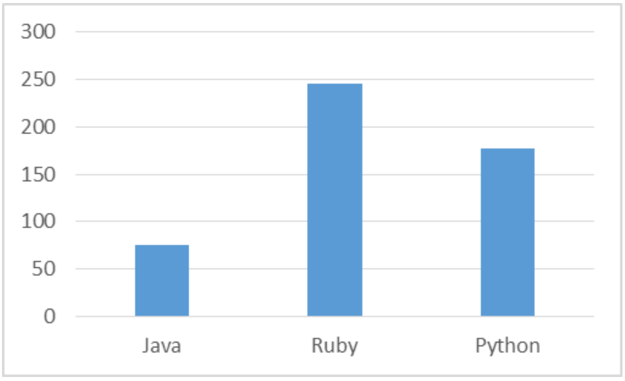


Figure 4 Distribution of the projects along the languages

A. *Direct Versus Indirect Contributions*

The result for the commits and pull request are that in my case the fewest pull requests are concerning the Python projects. And even in my case there are some projects with more pull requests than commits. The commit are more popular on Java projects.

B. *Usage of Travis CI*

Travis CI confirmed the popularity on the projects being configured in 394 projects out of 500. The result figure out the trend of the ready-to-go predisposition for the automatic builds in the majority of the projects but meanwhile just a few of them are really using it, just 244 projects. There is the possibility that the complexity of the projects are a critical factor when using the automatics builds and tests. There is also the possibility that these projects are running other types of automatic tests or trying directly deploying.

C. *Contribution Type and Build Success*

The success rate for the Contribution type is of 82% for the commits and 79.4% for the pull requests.
It can be deduced that with such little difference the contribution type cannot affect the successfulness of a build. Even if I believe that when the developers without commits permissions submit a pull request he is more careful to not break the build also for a reputation reason. Having a lot of "failed" pull requests that everyone can see can affect your GitHub profile.

D. *Impact of Project Differences on the Contribution Type/Build Success Relation*

Not being able to run the Stouffer test I cannot say anything between this relations.

Acknowledgment *(Heading 5)*

Special thanks to Serge Demeyer, the GHTorrent team and the Travis CI project.

References

[1] Vasilescu, B., Van Schuylenburg, S., Wulms, J., Serebrenik, A., & van den Brand, M. G. (2014, September). Continuous integration in a social-coding world: Empirical evidence from GITHUB. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on* (pp. 401-405). IEEE.

[2] Russell, M. A. (2013). *Mining the Social Web: Data Mining Facebook, Twitter, LinkedIn, Google+, GitHub, and More*. " O'Reilly Media, Inc.".

[3] Vasilescu, B., Filkov, V., & Serebrenik, A. (2013, September). StackOverflow and GitHub: associations between software development and crowdsourced knowledge. In *Social Computing (SocialCom), 2013 International Conference on* (pp. 188-195). IEEE.

[4] McDonald, N., & Goggins, S. (2013, April). Performance and participation in open source software on github. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems* (pp. 139-144). ACM.

[5] Gousios, G. (2013, May). The GHTorent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories* (pp. 233-236). IEEE Press.

[6] Guzman, E., Azócar, D., & Li, Y. (2014, May). Sentiment analysis of commit comments in GitHub: an empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (pp. 352-355). ACM.

[7] Yu, Y., Wang, H., Yin, G., & Ling, C. X. (2014, September). Reviewer recommender of pull-requests in GitHub. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on* (pp. 609-612). IEEE.

[8] Takhteyev, Y., & Hilts, A. (2010). Investigating the geography of open source software through GitHub.

[9] Lee, M. J., Ferwerda, B., Choi, J., Hahn, J., Moon, J. Y., & Kim, J. (2013, April). GitHub developers use rockstars to overcome overflow of news. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems* (pp. 133-138). ACM.

[10] Gousios, G. (2013, May). The GHTorent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories* (pp. 233-236). IEEE Press.

[11] M Fowler, & Foemmel, M. (2006). Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf.*

[12] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and Mozilla," TOSEM, vol. 11, no. 3, pp. 309–346, 2002.

[13] A. Hars and S. Ou, "Working for free? motivations of participating in open source projects," in HICSS. IEEE, 2001, pp. 1–9.

[14] B. J. Dempsey, D. Weiss, P. Jones, and J. Greenberg, "Who is an open source software developer?" Communications of the ACM, vol. 45, no. 2, pp. 67–72, 2002.

[15] K. R. Lakhani and R. G. Wolf, "Why hackers do what they do: Understanding motivation and effort in free/open source software projects," Perspectives on Free and Open Source Software, vol. 1, pp. 3–22, 2005.

[16] L. A. Dabbish, H. C. Stuart, J. Tsay, and J. D. Herbsleb, "Social coding in GitHub: transparency and collaboration in an open software repository," in CSCW. ACM, 2012, pp. 1277–1286.

[17] A. Begel, J. Bosch, and M.-A. D. Storey, "Social networking meets software development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder,