Github Mining

Discover the Descriptive Metrics of the Context in Continuous Integration (CI) Project

Patrick Hibbs[1]

Supervisor(s): Sebastian Proksch[1]**, Shujun Huang**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Patrick Hibbs
Final project course: CSE3000 Research Project
Thesis committee: Sebastian Proksch, Shujun Huang, Fenia Aivaloglou

Abstract

Continuous Integration (CI) systems automate the building, testing, and possibly more. However, it is still unclear how CI should be implemented in different contexts. Therefore, this paper tries to answer the question "What metrics can be used to describe project activity", as part of a bigger study. We mined information from 500 repositories and then applied several analysis techniques to find out whether a metric can be used to describe activity or not. Among the results, we show that the activity around a release date increases, and that Java is a way more active language than other languages, with the highest amount of commits, closed pull requests, contributors, issues, and releases.

1 Introduction

Continuous Integration (CI) is a software development practice that focuses on frequent integration of code changes from multiple developers into a shared repository, followed by automated build and test processes to ensure the code quality of high standard and bug detection early in the development process [12]. CI has become the best practice of modern software development and it has already been proven that CI has positive effects on software development by for example an increase in productivity within project teams [2; 9].

Traditionally, software development followed a sequential and isolated approach, with developers working on separate code branches and integrating their changes only periodically, typically during the latter stages of a project. This approach often led to integration issues, as the codebase became more complex and conflicts emerged when merging individual contributions. Moreover, identifying and resolving these conflicts took significant time and effort, delaying the overall development process [12]. Continuous Integration emerged as a response to these challenges, aiming to streamline and enhance the software development lifecycle.

The primary objective of this study is to identify and analyze the descriptive factors that influence the implementation of CI projects. By gaining a comprehensive understanding of these factors, the research aims to provide valuable insights and guidelines for enhancing the maturity of the CI process. Concretely, the study looks at metrics for project activity, maturity, and topic classification, as well as how the project performs in the build-life cycle and how the CI pipeline is implemented. This paper focuses solely on the project activity aspect of this study, trying to answer the following research question:

What metrics can be extracted to describe project activity?

In order to address the main research question more effectively, it has been segmented into sub-research questions:

RQ1: What metrics do other research use to describe project activity?

RQ2: What other metrics can be used to describe project activity?

The rest of this paper is structured in such a way that it builds up to answer the sub-research questions and thereby also the main research question, to start with an overview of what already has been done in the area of project activity. After discussing previous work, we explain the data collection process in the methodology, followed by how the experiments were set up and the results. We conclude with the limitations during this research, a section that reflects on the ethical aspects of the research, a small discussion on the results and the conclusion.

2 Previous work

There has been some work on metric analysis for project activity, but not much. Most of the research that use activity in one way or the other just use the metrics that best fits in that particular situation [3]. The metrics that were proven in other papers to have a correlation with project activity will be selected in this study as well.

There are also some papers which made an equation that can calculate the activity level of a project. Yang *et al.* [11] is an example of that. However, since the equation is based solely on commits, it is discouraged to make use of such an equation. This goes against the principle of this study to actually find more metrics to describe activity.

3 Methodology

The data collection process will be described here. The project selection process comes first, followed by the metric selection and finally the metric extraction.

3.1 Project selection

The selected projects come from GitHub[1], a widely popular platform for hosting and managing code repositories which provides a rich source of data for mining insights into software development projects. With the help of GitHub Search[2], all the open source projects on GitHub can be traversed with a simple GUI. Many filters can be passed in the query, and for this study, the projects were selected which had a lower bound of 1 for all the filters in the *History and Activity* part (i.e., the number of commits, the number of issues, the number of branches, the number of contributors, the number of pull requests and the number of releases). Applying a lower bound of 1 for all those filters significantly reduces the number of projects and allows for more insight later on in the research. If for example the number of issues need to be closely looked at, but the majority of the projects have 0 issues, then no valid conclusion can be drawn.

After that, 500 random projects were selected that all use GitHub Actions[3] with the additional filter of 100 projects per language. This limit was also imposed to allow for valid conclusions in the experiments. Figure 1 shows the process visualized.

[1] https://github.com

[2] https://seart-ghs.si.usi.ch

[3] https://github.com/features/actions

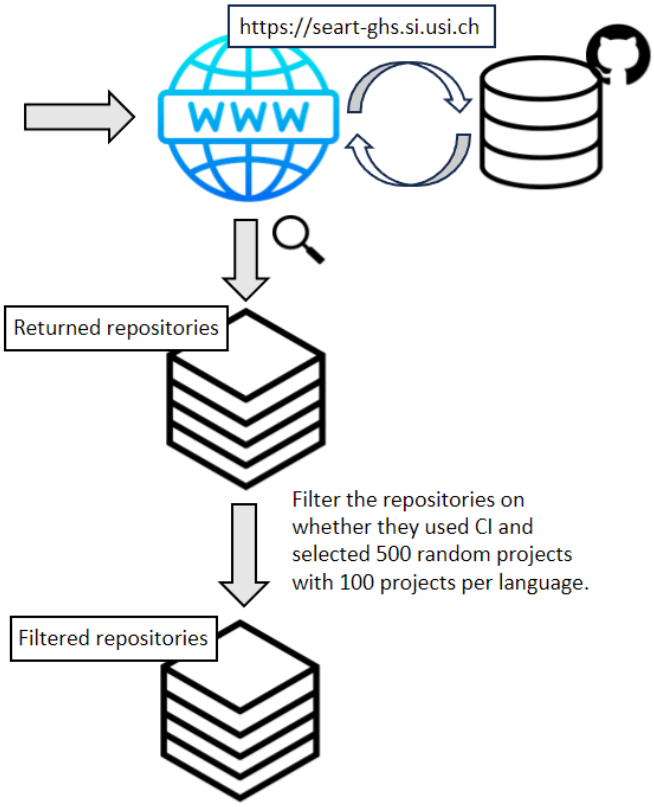

Figure 1: Visualization of the project selection process.

3.2 Metric selection

In order to establish a comprehensive and quantifiable set of metrics to assess their impact on project activity, an extensive review of relevant literature has been conducted. During the literature review, it was found that different papers use different metrics when talking about the activity within a project. Cosentino *et al.* [3] dedicate this to the different contexts in which every paper is written.

Given the subjective nature of *activity* across papers, the selection process for the current study focused on identifying the most commonly utilized metrics across multiple papers, as well as those metrics which were proven to have some correlation with activity in other papers.

The full table of metrics that were selected is visible in Table 1. The number of merged pull requests can be a good indicator for project activity because more pull-request merged commits add to the net project activity level [10]. Every merged pull request represents a contribution to the project, and hence represents project activity [6; 9].

Pull requests, commits and contributors are the strongest drivers for the project activity level, as shown by Hamilton *et al.* [7]. The total amount of issues in the repository is also a valid metric, as a higher number of issues contributes to higher participation of the community around the repository [8]. And the number and frequency of releases also influence the activity levels in the project. The activity increases exponentially towards the release date and then rapidly drops again [1].

At some point, watchers, forks, and stars were also considered to be used as metrics, however they have negligible effect on the activity level [7]. For forks, the additional reason as to why it should not be used is because forks are a copy of another user's repository where new ideas can be implemented. This then draws away the activity from the original project to the forked project [1; 7].

3.3 Metric extraction

After the projects and metrics were selected, the actual metrics were extracted from the repositories on GitHub with the GitHub API[4]. This was done with a custom-made tool in Python with the help of PyGitHub[5]. This is a library which acts as a wrapper-API around the GitHub API. By using this library, tremendous amount of implementation time was saved and low-level coding was avoided.

4 Experimental Setup

Our sample size consists of 500 repositories with 100 repositories per programming language. We already presented the 5 default metrics in Table 1, and in general we will try to find more metrics which can represent project activity.

Several different techniques were used in the analysis of the metrics to approach the research from as many different angles as possible. In three metrics, Spearman's rank correlation coefficient was used to apply correlation analysis. The decision to use Spearman's rank correlation coefficient instead of alternatives like Pearson was motivated by its suitability for small data sets and its robustness against outliers[5]. The idea behind correlation analysis is the more metrics the metric under study has a correlation with, the better it can be used to describe project activity.

5 Results

This section presents the findings of the researched metrics. The first section takes a closer look at one of the default metrics from Table 1. It studies when the activity around a release date is the highest.

The second section clusters all the repositories by their main language and studies whether certain languages show more activity than others.

The last three metrics use correlation analysis with Spearman's rank correlation coefficient.

5.1 Activity around a release date

The hypothesis for activity around a release date was that there would be an increase in commits towards the release date, and then a quick drop again after the release date [3]. The findings from my experiment however, show something else and are demonstrated in Table 2.

In the table you can see the percentage of repositories which have a more than 1 and 2 times increase in commits. The amount of days spread suggest how many days were taken into account before and after the release date(s). So a

[4]https://docs.github.com/en/rest?apiVersion=2022-11-28
[5]https://github.com/PyGithub/PyGithub

Metric	Description
# closed pull requests	The total number of successfully merged pull requests.
# commits	The total number of commits.
# contributors	The unique number of contributors.
# issues	The total number of open and closed issues.
# releases	The total number of releases.

Table 1: The selected metrics to analyse project activity.

spread of 2 means that 2 days before the release date, the release date itself, and 2 days after the release date were taken into account. The increase in commits was calculated as follows: average amount of commits around the release date divided by the average amount of commits in the baseline. The baseline refers to all the days which is not also a day around a release date. So if there are 7 days with day 3 as a release date and spread 1, then day 2, 3, and 4 are the dates around the release date and day 1, 5, 6, and 7 are the baseline days.

From the experiment it can be clearly visible that on the release date itself (0 days spread), the vast majority of the repositories have at least some increase in commits, namely 66% of the repositories, and even a slight portion (almost 19%) of the repositories have an at least 2 times increase in commits.

However, including the day before and after the release date, the numbers already shrink rapidly to only 2% of the repositories having a more than 2 times increase and only 21% of the repositories having an at least 1 time increase in commits. From here on, the numbers keep decreasing, but on a more slow and steady decline than the big drop from 0 to 1 days spread.

This trend could explain that a lot of small work is done (i.e., updating documentation, a new README) that can be finished quick (and hence more commits can be pushed on the day). Furthermore, developers who have been working on their feature (for some time) have to finish in time for the release date, so they might work on it until the last possible moment to ensure no mistakes and then push their work on the release date. The small increase in activity the day before and after could suggest developers finishing their feature 'early' and bug fixing respectively.

So for developers who are looking for an active project, they might want to consider repositories which are 1-2 days away from a release date. Or preferably, if they want to tackle bigger issues, start to look a week ahead of the release date, to make sure that he / she has enough time to actually implement the feature.

5.2 Programming Language

It may not be obvious that the programming language can say something about project activity, but that does not mean that it should not be investigated. Table 3 shows the results after clustering the repositories on their main programming language.

The main programming language of a repository was decided by the amount of lines of code belonging to that programming language. The language with the most amount of lines of code was chosen to be the main language.

If you take a close look at Java, you can see that it has the highest numbers for all metrics. It has a high amount of contributors, a high amount of closed pull requests and a high amount of commits, concluding that repositories which use Java as their main language are evidently more active. This could be due to the wide variety of applications Java could be used in. Think about the gaming industry, phone applications or websites. These projects usually tend to be developed for a long time, hence giving an increased activity in that area.

# days spread	≥ 1	≥ 2
0	66.12%	18.69%
1	20.94%	2.67%
2	9.03%	0.62%
3	5.75%	0.82%
4	4.11%	0.82%
5	2.87%	0.82%
6	2.87%	0.41%
7	3.08%	0.21%

Table 2: The percentage of repositories which have a more than 1 and 2 times increase in commits with n days spread. The number of days spread refers to how many days before and after the release date were included.

5.3 Repository size

The third metric that was looked into was the repository size. In this case, the repository size refers to the amount of kilobytes (KB) worth of files in the repository.

This experiment employed correlation analysis with Spearman's rank coefficient to investigate the relationship between repository size and the 5 metrics in Table 1. The results, displayed in Table 4, revealed a decent correlation between the repository size and the number of commits, indicating that as the repository size increased, the number of commits increased as well. This finding is supported by the plot displayed in Figure 2, which visually represents the positive correlation between repository size and the number of commits.

A possible explanation for this is that the more commits are pushed in the project, the further the development process is. And the further the development process is, the more features have been added and hence the bigger the project gets.

5.4 Average closing time of issues

As for reasons noted in the **Limitations**, only issues with the label *bug* were inspected, also known as *bug reports*. The closing time is measured in seconds from the creation date of the issue until the closing time of the issue. The Spearman rank coefficients for all the five metrics with the average

Table 3: The average amount of commits, closed pull requests, contributors, issues, and releases by language.

Language	avg. # commits	avg. # closed pull req.	avg. # contributors	avg. # issues	avg. # releases
C	1103.97	34.52	23.53	93.1	7.49
Java	4142.56	1241.86	129.6	2570.47	41.27
Python	191.67	23.98	7.05	50.69	4.96
C#	415.63	32.73	11.16	59.64	8.45
Swift	181.19	34.26	7.19	64.81	9.01

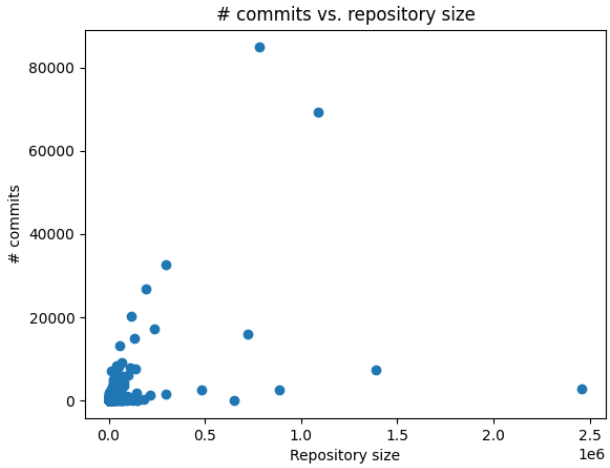

Figure 2: Scatter plot of the repository size against the number of commits.

Metric 1	Metric 2	Spearman's rank
Repository size	# closed pull requests	0.43
	# commits	0.65
	# contributors	0.43
	# issues	0.49
	# releases	0.33

Table 4: Spearman's rank coefficients for each of the five metrics were the repository size was compared against.

Metric 1	Metric 2	Spearman's rank
Average closing time	# closed pull requests	0.22
	# commits	0.26
	# contributors	0.32
	# issues	0.31
	# releases	0.12

Table 5: Spearman's rank coefficients for each of the five metrics were the average closing time of bug reports were compared against.

Metric 1	Metric 2	Spearman's rank
Average frequency change	# closed pull requests	0.23
	# commits	0.44
	# contributors	0.25
	# issues	0.27
	# releases	0.20

Table 6: Spearman's rank coefficients for each of the five metrics were the average frequency change of files were compared against.

closing time are shown in Table 5.

Looking at the Spearman coefficients, there does not seem to be any correlation with any of the five default metrics. Therefore, it seems unlikely that the average closing time of bug reports can say anything about project activity.

5.5 Frequency change of files

In the last metric we study the average frequency change of files. As described in Limitations, we only looked at the frequency change of source files of the main programming language of the repository. Then the average of all those files were taken to give the average frequency change of files in that repository. The Spearman coefficients for all the five metrics with average frequency change of files are shown in Table 6.

There seems to be no correlation with the average frequency change of files and with the five default metrics. However, upon inspection of the coefficients, it can be seen that the correlation with the number of commits is slightly higher than the other 4 metrics. This slightly higher coefficient can be explained by how the files are modified: by commits. So it makes sense that the higher the frequency change, the more commits there are, because file changes happen through commits.

6 Limitations

Despite the valuable insights gained from this study, it is important to acknowledge the limitations:

- **Small sample size**: Mostly due to time constraints, the sample size was limited to 500 repositories. Extracting metrics with the GitHub API usually took several hours, if not more. The limit imposed by GitHub to only allow 5000 requests per hour unnecessarily increased the time it took to successfully extract all the metrics. Increasing the sample size by hundreds or thousands more repositories would only make this process longer, time that we do not have during this short research period and time that is better spent on analyzing the data.
- **Only issues with a bug label**: Unfortunately, also issues were very time consuming to collect due to the 5000 requests per hour limit imposed by GitHub. That is why a filtered approach has been applied, namely only issues with a *bug* label. The idea behind this is that bugs should in general be fixed as soon as possible. If there is a short response time for bug reports, this could indicate a higher active project.
- **Only files of the main programming language**: Sometimes a repository can host many files. Not only source code files, but also images, markdown files, and GitHub-related files. Requesting the frequency change of all those files would take a really long time, and actually, the focus is not really on those files either. Source code files is where the implementation happens, so hence we only request those files. To filter it even more down, we only request the source files from the primary language of the repository.
- **Limited amount of metrics**: Due to time constraints and the imposed request limit of the GitHub API, it was not feasible to analyze all available metrics within the given timeframe. Consequently, certain results could not be obtained.

7 Responsible Research

In this research project, we worked with publicly available repositories on GitHub. It is important to respect the principles of ethical research. Although the data is publicly available, it is crucial to ensure that the privacy and confidentiality of individual contributors or organizations are upheld. Only anonymized data was used that cannot be traced back to specific individuals.

Additionally, this study adhered to the guidelines for responsible and ethical conduct in research, including proper citation and acknowledgment of sources and transparency in the research methodology. The code that was used in this research is publicly available on GitHub.

For the reproducibility of the methods, future engineers should also take great care in that they do not use privacy sensitive data or data that can be traced back to specific individuals. If that is the case, then that information should be left out for research.

8 Discussion

This section discusses some interesting results found in the previous section.

8.1 Average closing time revisited

Looking again at the Spearman coefficients for the average closing time in Table 5, we take the three highest coefficients and show the plots in Figure 3.

Despite the presence of a low correlation coefficient, the observed plots hint at the possibility of a potential correlation. The limited strength of the correlation could be attributed to factors such as the small sample size or the need for a broader range of issues to be considered (e.g., all issues, or bug reports and feature requests).

8.2 Correlation analysis revisited

From the three correlation analysis that were performed, two had a high correlation with the number of commits (high in this context means relatively in comparison with the other numbers). This could suggest that commits is a stronger indicator for project activity than the other metrics. This is also suggested by Alshomali *et al.* [1].

9 Conclusions and Future Work

This paper has studied metrics to describe project activity. The most kind of metrics that other researchers use are {the number of pull requests, the number of commits, the number of contributors, the number of issues, the number of releases, the number of forks, the number of stars}. However it should be noted that the number of forks and the number of stars are not representative for project activity [7]. The other kind of metrics that can be used to describe project activity are {the release date, language, repository size}. That gives us the following set of metrics that can be used to describe project activity: {the number of closed pull requests, the number of commits, the number of contributors, the number of issues, the number of releases, the release date, language, repository size}.

Moving forward, several directions for future research emerge from the findings of this study. These suggestions encompass areas that could further enhance our understanding of project activity. Firstly, we could look at release prediction. This paper showed that the release date is a good metric for project activity, but, unless the project owners specifically announce it, as an outsider it is not known when releases happen. Predicting when release dates are near could help further in finding active projects. Secondly, what characteristics do active projects share? Finding out characteristics between active projects can identify new types of metrics that can be used to describe project activity. Gautam *et al.* [4] describe a way of clustering repositories into 4 categories in terms of their activity, popularity, size, testing and stability. A similar research could be conducted solely focusing on project activity. Lastly, a new study on the correlation between the average closing time, the number of commits, the number of contributors and the number of issues could shed new light of whether there is actually a correlation or not, as described in Average closing time revisited.

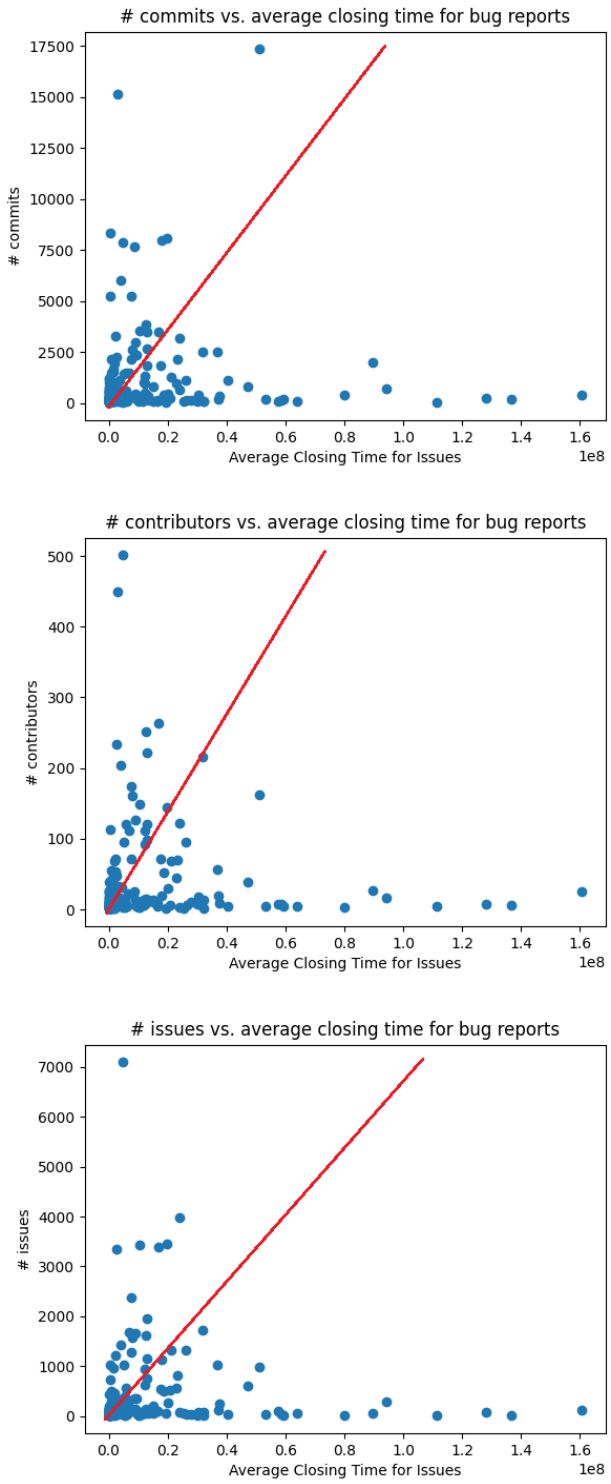

Figure 3: The 3 scatter plots of the number of commits, the number of contributors, and the number of issues respectively against the average closing time.

References

[1] Mohammad Azeez Alshomali, John R Hamilton, Jason Holdsworth, and SingWhat Tee. Github: factors influencing project activity levels. 2017.

[2] Moritz Beller, Georgios Gousios, and Andy Zaidman. Oops, my tests broke the build: An explorative analysis of travis ci with github. In *2017 IEEE/ACM 14th International conference on mining software repositories (MSR)*, pages 356–367. IEEE, 2017.

[3] Valerio Cosentino, Javier L Cánovas Izquierdo, and Jordi Cabot. A systematic mapping study of software development with github. *IEEE Access*, 5:7173–7192, 2017.

[4] Aakash Gautam, Saket Vishwasrao, and Francisco Servant. An empirical study of activity, popularity, size, testing, and stability in continuous integration. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 495–498. IEEE, 2017.

[5] Thomas D Gauthier. Detecting trends using spearman's rank correlation coefficient. *Environmental forensics*, 2(4):359–362, 2001.

[6] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. Work practices and challenges in pull-based development: The integrator's perspective. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 358–368. IEEE, 2015.

[7] John R Hamilton, SingWhat Tee, Jason Holdsworth, and Mohammad Azeez Alshomali. Analysing big data projects using github and javascript repositories. 2017.

[8] Khadija Osman and Olga Baysal. Health is wealth: Evaluating the health of the bitcoin ecosystem in github. In *2021 IEEE/ACM 4th International Workshop on Software Health in Projects, Ecosystems and Communities (SoHeal)*, pages 1–8. IEEE, 2021.

[9] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in github. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, pages 805–816, 2015.

[10] Joicymara Xavier, Autran Macedo, and Marcelo de Almeida Maia. Understanding the popularity of reporters and assignees in the github. In *SEKE*, pages 484–489, 2014.

[11] Chengrong Yang, Hao Li, Yan Kang, Chenyang Lu, and Junsong Liu. Evaluation model on the usage of java project deployment tool in open source community. *Electrical Engineering and Computer Science (EECS)*, 2:59–63, 2019.

[12] Ravi Teja Yarlagadda. Understanding devops & bridging the gap from continuous integration to continuous delivery. *Understanding DevOps & Bridging the Gap from Continuous Integration to Continuous Delivery',*

International Journal of Emerging Technologies and Innovative Research (www. jetir. org), ISSN, pages 2349–5162, 2018.