Analyzing the Effects of CI/CD on Open Source Repositories in GitHub and GitLab

Jeffrey Fairbanks
Department of Computer Science
Boise State University
Boise, Idaho, USA
jeffreyfairbanks@u.boisestate.edu

Akshharaa Tharigonda
Department of Computer Science
Boise State University
Boise, Idaho, USA
akshharaatharigo@u.boisestate.edu

Nasir U. Eisty
Department of Computer Science
Boise State University
Boise, Idaho, USA
nasireisty@boisestate.edu

Abstract—Numerous articles emphasize the benefits of implementing Continuous Integration and Delivery (CI/CD) pipelines in software development. These pipelines are expected to improve a project's reputation and decrease the number of commits and issues in the repository. Although CI/CD adoption may be slow initially, it is believed to accelerate service delivery and deployment in the long run. This study aims to investigate the impact of CI/CD on commit velocity and issue counts in two open-source repositories, GitLab and GitHub. By analyzing more than 12,000 repositories and recording every commit and issue, it was discovered that CI/CD enhances commit velocity by 141.19% but also increases the number of issues by 321.21%.

Index Terms—CI/CD, Mining Repositories, Open-Source Software, Software Engineering

I. INTRODUCTION

It's rare for resolving code issues to be a simple task. Unforeseen problems and unanticipated test cases often arise during the project's development. This causes the project to shift its original intentions and require different inputs and outputs. Keeping track of these changes can be challenging, mainly when working with multiple developers, and it may increase the resources needed for the project.

Developing a project with a team of developers requires effective communication and a simple method for implementing new code with new test cases. Unfortunately, in many cases, testing code frequently and in small increments is disregarded until after a significant amount of code has been written. Unit testing has gained popularity in recent years and allows for more efficient testing as code is developed. However, it is often employed only as the code section conducting unit testing is nearing completion.

As testing (both unit and end-to-end) is an integral part of the development life cycle, a newer solution has come to test its validity. The software engineering community widely advocates continuous integration and Continuous Delivery (CI/CD). There are two parts to CI/CD.

CI (Continuous Integration) is a practice that helps many developers integrate code changes into a single project more quickly and easily. Developers use an automated set of tests to validate the code before it's integrated or merged into a project's main branch or a repository that is shared with other developers on the team. This allows development teams to build quickly and test their code continuously, which helps them to avoid problems later on.

On the other hand, CD (Continuous Delivery) is a great way to automate the code changes to the process. It is a better way to deliver changes to development or production environments than traditional methods. Continuous Development can also push rapid and small changes to various environments, allowing for quick rollbacks or changes and avoiding major differences or exceptions. When using and pushing small changes, deploying and testing code that could be troublesome becomes substantially more straightforward. In addition, this method allows for risk mitigation when pushing to a critical production system.

CI/CD is essential in a development pipeline as it enables a faster development cycle, facilitates the identification of imminent issues with the code, and promotes efficient and minimal code changes. By gathering and presenting essential metrics on the benefits of utilizing CI/CD in a development pipeline, developers can enhance their productivity and deliver exceptional products to users. Consequently, our research endeavors to address the following inquiries:

RQ1: Does integrating CI/CD enhance commit velocity?

RQ2: Does integrating CI/CD impact the number of issues reported for the project?

RQ3: Are there any significant variations in CI/CD issues between GitLab and GitHub?

We designed these inquiries to evaluate the effectiveness of utilizing CI/CD in projects involving developers' teams. Although many articles assert that CI/CD improves commit velocity and streamlines testing and deployment pipelines, limited formal research substantiates these claims [25]–[27]. Moreover, utilizing CI/CD requires implementing multiple tools and processes that must work together harmoniously. Correctly learning and implementing these tools can be time-consuming and challenging, leading many developers to question their value [2].

The primary focus of the third research question is to examine whether there are significant disparities in the utilization of CI/CD between GitLab and GitHub. This metric holds paramount importance as one platform may facilitate commit velocity while the other may impede the advantages of utilizing CI/CD. Similarly, the drawbacks associated with

IEEE COMPUTER SOCIETY

CI/CD, such as the time-consuming setup process, may vary between the two open-source repository platforms. This study aims to investigate these variations and determine if any noteworthy differences exist that could pose challenges in implementing CI/CD.

To answer these inquiries, we collected data from over 12,000 repositories, including those with and without CI/CD. We mined data from open-source repositories on GitLab and GitHub, gathering information on projects with multiple developers and CI/CD workflows using their respective APIs. We analyzed the data and answered our research questions using custom functions found in Google Sheets. Additionally, we utilized the programming language Python and the Google Colab platform to analysis and process the data.

II. RELATED WORK

Today's development climate is marked by research on CI/CD that focuses on transitioning to a CI/CD pipeline, pinpointing pain points in its usage, or comparing different platforms that use CI/CD (e.g., comparing the use of GitLab versus GitHub) [28]–[30].

Savor et al. [5] identified four critical elements of continuous deployment: small software updates, automatic releases, developer responsibility, and fully automatic deployment. Tools aid in managing these responsibilities by offering insights into code changes, automating repetitive and error-prone tasks, and logging every process action. Their findings suggest that CI/CD should be a continuous and automated process that continuously tests and deploys code to production. Based on this understanding, our research aims to ensure that CI/CD, as used in real-world applications, actually increases development velocity as intended.

In their study, Rangnau et al. [3] explore the integration of security tools within CI/CD pipelines. While they assume that CI/CD is widely used today, no formal research supports this claim. Their work contributes to our study by highlighting the shift towards a SaaS (Software as a Service) development model, where multiple users share instances running on cloud infrastructure. This new development environment allows software practitioners to continuously enhance product quality through frequent updates, leading to increased use of CI/CD. Our research examines the validity of this claim by analyzing a real-world and up-to-date dataset.

Nogueira et al. [4] demonstrate how Apache Kafka pipelines support integrating and normalizing event logs from multiple sources into data streams that feed process mining algorithms in real-time. They apply this to the complex CI/CD pipeline of a major European e-commerce company, highlighting how these techniques enhance the monitoring and observability of development processes. Their study focuses on a single CI/CD pipeline, whereas our research examines multiple pipelines to investigate whether velocity is improved overall and if development issues are reduced.

III. METHODOLOGY

This section discusses the data collection and analysis methodology employed in this research. The study involved mining two primary open-source repositories to obtain information on the usage of CI/CD. Specifically, we investigated the GitHub and GitLab platforms to identify any variations in velocity and CI/CD utilization between them. Despite their apparent similarities, these platforms exhibit distinct differences [16] that we aimed to uncover. Therefore, in the following discussion, we will delve into the data collection and analysis process for each platform.
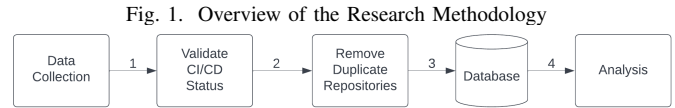
Fig. 1. Overview of the Research Methodology

Fig. 1 depicts the general process we followed for each of the open-source repositories, GitHub and GitLab. The first step involved collecting data and verifying whether each repository had CI/CD capabilities. Once this was established, we cleaned the data and eliminated any duplicate entries, if present. Subsequently, we stored the cleaned data in a database and performed further analysis to address the research questions.

A. Collecting Data from GitHub

GitHub is an open-source repository for developers to store, collaborate, and manage their code in a centralized location [17]. To collect data from GitHub, we primarily utilized the API provided by the platform. We adopted several different approaches while using the API. Initially, we collected information on repositories that implemented CI/CD and then further mined this subset of repositories to gather data on pull requests, issues, and related metadata. To achieve this, we utilized the GitHub API to search the path of each publicly available repository for the file path: ".github/workflow" [18]. Fig. 2 shows an overview of this data collection method.
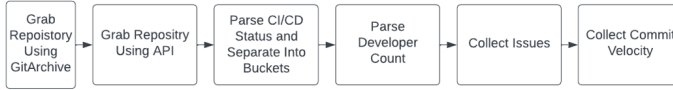
This search was conducted to identify projects that contained CI/CD since this file is required to implement CI/CD within a repository. This method was effective until we reached the limit on the number of results. As per the documentation, only 1000 results can be returned for each search [19]. Thus, we needed another approach to collect the required CI/CD projects after reaching the limit of 1000 repositories.

Fig. 2. GitHub Data Collection Method 1
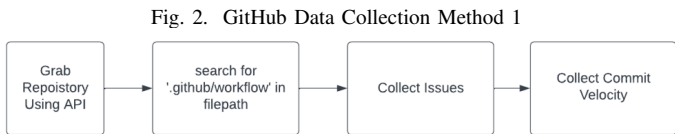
To collect data from GitHub, we adopted another approach of scraping a list of developer name/repository name pairs using GitArchive. After compiling a substantial list, we utilized the GitLab API to check each repository for CI/CD. Next, we parsed each repository into a "using CI/CD" or a "not using CI/CD" bucket. To prevent personal or school projects from skewing the data, we pruned each bucket to include only repositories with at least two developers. We also ensured that
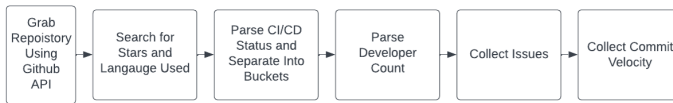
each repository was active in 2022, thus ensuring the data is current and valuable for this research. This guarantees that the results produced by this research project are relevant and applicable to the current time. Fig. 3 shows an overview of this data collection method.

Fig. 3. GitHub Data Collection Method 2

Grab Repository Using GitArchive → Grab Repositry Using API → Parse CI/CD Status and Separate Into Buckets → Parse Developer Count → Collect Issues → Collect Commit Velocity

We used another search method using the GitHub API to collect the remaining data from GitHub. We formulated the search by filtering repositories by language (e.g., python, java) and sorting them based on the number of stars. This method proved highly effective, as we observed a strong correlation between the number of stars and the implementation of CI/CD pipelines in the project repositories. The more stars a repository had, the more likely it was to have implemented CI/CD. This search method enriched our dataset significantly, boosting our confidence in the quality of the data collected. Fig. 4 provides an overview of this data collection approach.

Fig. 4. GitHub Data Collection Method 3

Grab Repository Using Github API → Search for Stars and Langauge Used → Parse CI/CD Status and Separate Into Buckets → Parse Developer Count → Collect Issues → Collect Commit Velocity

We conducted a deduplication process on all the collected repositories to ensure that each repository was collected only once and categorized correctly based on whether it implemented CI/CD. Then, we ran a validation function on each repository to ensure it was in the appropriate bucket. This step was important to ensure the accuracy and integrity of the data. After the deduplication process, it was confirmed that each repository was in the correct category and that there were no duplicates.
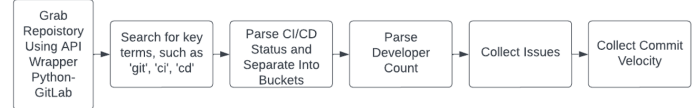
Issues With GitHub Data Collection. The primary challenges encountered during the GitHub data collection were related to the hourly rate limit imposed by the API when searching for repositories. This limitation was not ideal and prolonged the data collection process significantly. Another issue was that each API query could only return up to 1000 results, meaning multiple searches had to be conducted for each bucket, "using CI/CD" and "not using CI/CD." This led to considerable time being spent validating and de-duplicating the data.

B. Collecting Data from GitLab

The process of collecting data from GitLab differed significantly from collecting data from GitHub. Initially, we attempted to utilize GitLab's own resources for data collection [22]. However, we found that it took approximately 5 minutes to collect data from a single repository, which was deemed too time-consuming for the project's timeline. Consequently, we refined our approach by using a python wrapper

instead of the API natively [23]. Although the wrapper did not support global searching of a specific file, we devised a workaround. We used the python GitLab API wrapper to search for keywords found in the repository name across Git-Lab. Therefore, we searched for relevant data using keywords such as 'ci', 'cd', 'git', and 'workflow'. Fig. 5 provides an overview of this data collection approach.

Fig. 5. GitLab Data Collection Method

Grab Repository Using API Wrapper Python-GitLab → Search for key terms, such as 'git', 'ci', 'cd' → Parse CI/CD Status and Separate Into Buckets → Parse Developer Count → Collect Issues → Collect Commit Velocity

The method we used for data collection was not optimal in returning results. We had to spend a considerable amount of time to ensure that the collected data met the research's standards. We invested a significant amount of time ensuring that the repositories had multiple developers, were categorized correctly based on their use of CI/CD and that only unique entries were collected. This time investment was critical to ensure that we collected high-quality data. Although we attempted other implementations, this approach proved to be the only reliable way to collect the required amount of data at scale within the research's timeline.

Issues With GitLab Data Collection. The process of gathering data from GitLab is significantly more arduous, particularly in terms of searching for relevant information, which consumes a considerable amount of time. In comparison to collecting data from GitHub, obtaining GitLab data takes considerably longer. Despite the need to employ multiple methods to collect GitHub data, it is still roughly three times faster than gathering GitLab data. The disparity in speed is largely due to GitLab's requirement to perform an API search for each issue and timestamp on commits individually, as opposed to GitHub, which can retrieve all necessary information with a single API call. Consequently, the API call-intensive process imposed by GitLab restricts the number of repositories that can be collected in a day and incurs significant waiting time as the API replenishes requests.

An additional problem encountered when searching for repositories on GitLab is that the API lacks the comprehensive functionality offered by GitHub's API. Compared to GitHub, the GitLab API is less user-friendly and has fewer resources available to facilitate the search for repositories on the platform. In addition, unlike GitHub, the GitLab API does not support the search for file paths, highly-rated projects, language-based projects, or projects that use CI/CD. Consequently, a new approach was necessary to gather data on projects utilizing CI/CD on GitLab.

While searching for a new data collection method, we discovered that many researchers began by scraping repository URLs and then enriching the data using the GitPy tool to create a temporary local clone of the project. The logs required for the research were extracted from the local clone, and the project was then deleted [20]. Another method we found

during our search involved crawling GitLab to collect the required data [21]. However, neither of these methods was ideal for collecting large amounts of data. Consequently, we decided to develop a new data collection method from scratch, as described above.

C. The Data Set

We collected data throughout this research, which comprises about 12,000 repositories. For GitHub data, there were at least 3000 repositories in each category; for GitLab data, roughly 1300 repositories were in each category. The collected data enabled us to analyze to answer the research questions. However, in this section, we will demonstrate the data to provide the reader with insight into the collected data. Table I represents the number of repositories with and without CI/CD.

TABLE I
NUMBER OF REPOSITORIES

	CI/CD	NO CI/CD
GitHub	3223	6007
GitLab	1357	1356

We collected a few attributes associated with each repository. We cataloged each repository based on its file path in GitHub or GitLab. Additionally, we stored the number of active or retired issues within the lifespan of each repository. We collected the issues to analyze whether CI/CD creates an environment where fewer issues are created. Furthermore, we collected the average time between commits. We calculated the commit velocity of the repository by determining the mean between the time of each commit, and we calculated each commit to obtain an accurate measure. As shown in Table II, we designed the layout of our collected data. We further parsed the data above to enable calculations to take place. We subsequently analyzed the data using various statistical measures, and we describe the results in the next section.

TABLE II
LAYOUT OF THE DATA

Repository Name	Issues	Commit Velocity
requilence/integram/	64	639:10:58
bueltge/WordPress-Admin-Style/	2	494:15:37
apache/airflow/	863	4:09:41
openwdl/wdl/	65	872:34:25
microg/android_packages_apps_UnifiedNlp/	97	580:26:51
Froxlor/Froclor/	32	4:10:46
cloudson/gitql/	4	251:57:46
repoze/repoze.workflow/	4	1679:15:44
john30/ebusd/	60	39:40:22
KSP-CKAN/NetKAN/	27	24:42:51
snarfed/bridgey/	95	31:35:57

IV. RESULTS

In this section, we will present and discuss the findings of this study. We used the research questions to guide the discussion and discovered several notable outcomes through this study. Finally, we will discuss the points collected between using CI/CD on GitHub and GitLab.

First, we conducted an analysis of the GitHub collection data to find the mean, median, and standard deviation within the dataset. Initially, we focused on finding the distribution of issues between the projects that utilized CI/CD and those that did not implement the process. After collecting and resourcing the open-source repositories, we conducted a more detailed analysis of each metric within the repository.

A. Repository Issues Analysis

After analyzing the data on a macro level, we took a much more micro approach to find the small details hiding within the data set. We used this micro approach to mine the repository data, giving us a better look at the distribution of the number of issues found in projects that use CI/CD and those that do not use CI/CD. We analyzed the entire data set and formulated the mean, standard deviation, and median of the number of issues within it as shown in Table III.

TABLE III
GITHUB REPOSITORY ISSUES

	CI/CD	NO CI/CD
Mean	175.38	52.33
Median	29	25
Standard Deviation	549.97	90.55
Count	3223	3223

Throughout this study, we also collected and analyzed GitLab data. We demonstrate the results of the distribution between the number of issues for each in regard to CI/CD usage in Table IV.

TABLE IV
GITLAB REPOSITORY ISSUES

	CI/CD	NO CI/CD
Mean	52.04	17.56
Median	12	3
Standard Deviation	477.56	189.81
Count	1357	1357

Upon examining the data, it is evident that repositories without CI/CD processes have a lower number of issues. This conclusion is drawn from the lower mean of issues found within the repository, combined with a standard deviation that is significantly lower than that of the repositories containing CI/CD. Therefore, we can conclude that, on average, repositories without CI/CD have fewer issues compared to those with CI/CD in the development process.

B. Repository Commit Velocity

After analyzing the number of issues in repositories from both GitHub and GitLab, we further investigated the commit velocity between the two platforms. It is worth noting that the mean and median values presented in Tables V and VI are calculated in hours. Upon examining the GitLab data, it becomes apparent that the commit velocity shows a similar trend to what we observed on GitHub. On average, the commit velocity is significantly lower when CI/CD is used in the pipeline, compared to when it is not used.

TABLE V
GITHUB REPOSITORY COMMIT VELOCITY

	CI/CD	NO CI/CD
Mean	16.51	27.11
Median	18:39:28	487:37:01
Standard Deviation	52.05	31.92
Count	3223	3223

TABLE VI
GITLAB REPOSITORY COMMIT VELOCITY

	CI/CD	NO CI/CD
Mean	22.01	25.70
Median	32:39:15	120:57:14
Standard Deviation	88.57	60.21
Count	1356	1356

Upon analyzing the commit velocity data from both GitHub and GitLab, we observed that the projects using CI/CD have a faster commit velocity compared to those that do not use CI/CD. This trend is demonstrated in Tables VII and VIII, as expected.

TABLE VII
GITHUB COMMIT VELOCITY METRICS

	CI/CD	NO CI/CD
Mean	16.51 Hrs	27.11 Hrs
Standard Deviation	52.05	31.92
Median	18 Hrs	487 Hrs

TABLE VIII
GITLAB COMMIT VELOCITY METRICS

	CI/CD	NO CI/CD
Mean	22.01 Hrs	25.7 Hrs
Standard Deviation	88.57	60.21
Median	32 Hrs	120 Hrs

The most significant finding regarding the use of CI/CD and commit velocity is the median time between commits. On average, when using CI/CD in their pipelines, the median commit velocity is significantly lower compared to those that do not use CI/CD. Based on this analysis, we can conclude that implementing CI/CD results in an increase in commit velocity.

V. DISCUSSION

Through the implementation of this research, a clear outcome has been achieved and the research question are ready to be answered. Each of the questions will lead the discussion in this section of the paper.

A. RQ1: Does integrating CI/CD enhance commit velocity?

The data clearly shows that implementing CI/CD increases the commit velocity on average, which is the primary goal of CI/CD. On average, the implementation of CI/CD results in a significant improvement in commit velocity of 141.19%, as shown in Table IX.

TABLE IX
COMMIT VELOCITY DIFFERENCE BETWEEN GITHUB AND GITLAB

	CI/CD	No CI/CD
GitHub	16.51 Hours	27.11 Hours
GitLab	22.01	26.7
Velocity Increase with CI/CD: 141.19 %		

B. RQ2: Does integrating CI/CD impact the number of issues reported for the project?

In delving deeper into RQ2, as previously discussed, the use of CI/CD has an impact on the number of issues found within the project repository. Our research and analysis revealed that when CI/CD is utilized, there are generally more issues created within the repository. Although CI/CD improves and accelerates commit velocity, it is interesting to note that the number of issues within the repository would increase. However, this is not surprising as the pipeline may naturally create errors as it executes. As the project is continually deployed, the likelihood of issues increases with more commits made. The swiftness of these commits may contribute to more issues found within the repository. On average, as shown in Table X implementing CI/CD results in a significant increase in the number of issues found within the code repository by 321.21%.

TABLE X
DIFFERENCE IN THE NUMBER OF ISSUES BETWEEN GITHUB AND GITLAB

	CI/CD	No CI/CD
GitHub Avg Issues	135.38	57.33
GitLab Avg Issues	52.04	17.68
Issues Increase with CI/CD: 321.21 %		

C. RQ3: Are there any significant variations in CI/CD issues between GitLab and GitHub?

This research question evaluates the usefulness of GitLab compared to GitHub based on the findings of this study. Our research indicates that GitHub provides more benefits than GitLab regarding commit velocity and issues found within the repository. In addition, the analysis reveals that GitHub has a higher commit velocity compared to GitLab.

When considering this question, it is also noteworthy to mention the difference in data collection between GitLab

and GitHub. GitLab provides fewer tools for analyzing data within the repository compared to GitHub, which offers a more extensive suite of tools to facilitate data collection and analysis.

Finally, it is worth noting that on GitHub, a higher number of stars on a project indicates a higher likelihood of the project implementing CI/CD pipelines within the repository. However, this trend was not as evident with GitLab. The number of projects using CI/CD versus those not using CI/CD did not show a significant difference based on the number of stars on the project.

VI. THREATS TO VALIDITY

During this research project, there were several potential threats to the validity of the study. Firstly, the time taken to collect repositories resulted in a small margin being collected. Although we collected more than 12,000 repositories, this is a small number when compared to the millions of users who use these open-source repository services daily.

The main limiting factor of this project was the amount of analysis done on the data. Although the analysis presented in this paper is sufficient to yield clear and accurate outcomes, it would have been beneficial to have more time to conduct additional analysis and delve deeper into the data set. Unfortunately, additional time was not available to conduct further experiments, which could have enhanced the implications and reach of this research.

VII. CONCLUSION

Many developers consider implementing CI/CD within their project pipeline because it is widely discussed. According to this study, implementing CI/CD has increased commit velocity. However, using CI/CD also significantly increases the number of issues that arise within a project. Therefore, developers must balance the benefits of faster commit velocity against the potential downsides of introducing more issues into the code base when implementing CI/CD. In the future, we would like to expand our dataset with more repositories in both sources and provide more in-depth generalized insights into our research questions.

REFERENCES

[1] Borle, N.C., Feghhi, M., Stroulia, E. et al. Analyzing the effects of test driven development in GitHub. Empir Software Eng 23, 1931–1958 (2018). https://doi.org/10.1007/s10664-017-9576-3

[2] Rossel, Sander (October 2017). Continuous Integration, Delivery, and Deployment. Packt Publishing. ISBN 978-1-78728-661-0.

[3] Rangnau, Thorsten & Buijtenen, Remco & Fransen, Frank & Turkmen, Fatih. (2020). Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines. 145-154. 10.1109/EDOC49727.2020.00026.

[4] Nogueira, Ana Filipa & Zenha-Rela, Mário. (2021). Monitoring a CI/CD Workflow Using Process Mining. SN Computer Science. 2. 10.1007/s42979-021-00830-2.

[5] T. Savor, M. Douglas, M. Gentili, L. Williams, K. Beck, and M. Stumm, "Continuous deployment at Facebook and OANDA," in Proceedings of the 38th International Conference on Software Engineering Companion - ICSE '16. Austin, Texas: ACM Press, 2016, pp. 21–30.

[6] Tsay J, Dabbish L, Herbsleb J (2014) Influence of social and technical factors for evaluating contribution in github. In: Proceedings of the 36th international conference on software engineering, ICSE 2014

[7] Grigorik I (2012) The Github archive., http://www.githubarchive.org/

[8] Laerte Xavier, Fabio Ferreira, Rodrigo Brito, Rodrigo Brito(2020). Beyond the Code: Mining Self-Admitted Technical Debt in Issue Tracker Systems

[9] Georgios Gousios; Diomidis Spinellis (2012). GHTorrent: Github's data from a firehose

[10] Bird C, Rigby PC, Barr ET, Hamilton DJ, German DM, Devanbu P (2009) The promises and perils of mining git. In: 6th IEEE International working conference on mining software repositories, 2009. MSR'09. IEEE, pp 1–10

[11] Comparing different CI/CD pipelines - theseus. (n.d.).www.theseus.fi Opinnaytetyo_Joni_Virtanen.pdf

[12] Clark, L. (2019). Deliver quality software at speed with CI/CD. ComputerWeekly.com.

[13] Zaworski, R. J. (2021, November 30). Anatomy of a high-velocity CI/CD pipeline. Medium. Retrieved December 10, 2022, from https://medium.com/developing-koan/anatomy-of-a-high-velocity-ci-cd-pipeline-43a1ae3b798b

[14] InstaCI. (2021, January 30). Is your CI slowing you down? Medium. www.instaci.medium.com/is-your-ci-slowing-you-down-afe3de1bc865

[15] Continuous delivery: Overcoming adoption challenges. Journal of Systems and Software. Retrieved December 10, 2022, from https://www.sciencedirect.com/science/article/pii/S0164121217300353

[16] GitLab. (n.d.). Retrieved December 10, 2022, from https://about.gitlab.com/devops-tools/github-vs-gitlab/

[17] GitHub's products. GitHub Docs. (n.d.). Retrieved December 10, 2022, from https://docs.github.com/en/get-started/learning-about-github/githubs-products

[18] Search. GitHub Docs. (n.d.). Retrieved December 10, 2022, from https://docs.github.com/en/rest/search

[19] Search. GitHub Docs. (n.d.). Retrieved December 10, 2022, from https://docs.github.com/en/rest/search

[20] Amores, D. A. S. (2022, February 2). Estudo Geral. Mining GitLab repositories for software development activities. Retrieved December 10, 2022, from https://estudogeral.sib.uc.pt/handle/10316/97983

[21] Hadisfr. (n.d.). Hadisfr/gitlab_crawler: A gitlab.com network crawler: Codes of "An analysis of gitlab's users and projects networks", IST2020, doi: 10.1109/ist50524.2020.9345844. GitHub. Retrieved December 10, 2022, from https://github.com/hadisfr/gitlab_crawler

[22] Instance-level CI/CD variables API. GitLab. (n.d.). Retrieved December 10, 2022, from https://docs.gitlab.com/ee/api/instance_level_ci_variables.html

[23] Gitlab. python. (n.d.). Retrieved December 10, 2022, from https://python-gitlab.readthedocs.io/en/stable/index.html

[24] How to conduct the Wilcoxon Sign Test. Statistics Solutions. (2021, May 5). Retrieved December 10, 2022, from https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/how-to-conduct-the-wilcox-sign-test/

[25] Benefits of Continuous Integration Delivery: CI/CD benefits. katalon.com. Retrieved December 10, 2022, from https://katalon.com/resources-center/blog/benefits-continuous-integration-delivery

[26] Roddewig, S. (2022, February 8). CI/CD: What is it why is it important for devops? HubSpot Blog. Retrieved December 10, 2022, from https://blog.hubspot.com/website/cicd

[27] What are the benefits of CI/CD?: Teamcity CI/CD guide. JetBrains. (n.d.). Retrieved December 10, 2022, from https://www.jetbrains.com/teamcity/ci-cd-guide/benefits-of-ci-cd/

[28] Google. Setting up a CI/CD pipeline for your data-processing workflow. Retrieved December 10, 2022, from https://cloud.google.com/architecture/cicd-pipeline-for-data-processing

[29] continuous delivery: Overcoming adoption challenges - researchgate. (n.d.). Retrieved December 10, 2022, from www.researchgate.net/publication/313874428_Continuous_Delivery _Overcoming_Adoption_Challenges

[30] A comparison study of managed CI/CD solutions. (n.d.). Retrieved December 10, 2022, from https://git.cubieserver.de/jh/cs-e4000-seminar/raw/commit/a9b2cba6ec2c187f7a9b34c1f4a9ab8dc5097f71/presentation/presentation_slides_2020-04-24.pdf

[31] How gitclear estimates time used (minutes, hour, days) per commit. GitClear. Retrieved December 10, 2022, from https://www.gitclear.com/help/estimating_time_used_per_commit