Final Report: DevOps for Github

Siyang Zhang, sz2741, Tsai-Chen Hsieh, th2990

1. Synopsis:

Our project is a Google Chrome extension that can present DevOps evaluation metrics. The extension would allow users to view DevOps-related metrics via visual charts. DevOps evaluation is a relatively new field, with no standard established at the time of writing. However, it is crucial for teams that want to improve DevOps further. In this project, we want to build some helpful metrics which allow developers to evaluate the workflow and improve better.

The novelty of this extension is to provide useful metrics regarding DevOps, but being extremely friendly for people that do not have experience in DevOps. The vanilla GitHub does not have many metrics that focus on the evaluation of DevOps, and the extension can solve that. On the other hand, we are aware of the existing visualization tools that can provide better customization and functionality to the user. However, they require software installation on the computer or additional account creation (such as Jenkins and Grafana), while the Chrome extension requires minimal installation. Another issue with these more powerful tools is that users need to spend some time configuring them before actually using them, which requires knowledge of DevOps beforehand. On the other hand, we focus on intuition rather than customization. Predefined metrics and “lessons” on the metrics can let the developers be able to start using them without much knowledge of DevOps, while also learning them on the go. In cases such as class projects or open source projects, the metrics not only help experienced developers become more efficient, but newcomers like students can also use the extension to learn more about DevOps. Current Computer Science education does not teach much about DevOps, so the project can help students understand DevOps from the evaluation perspective. In short, we view our project as a gentle introduction to DevOps rather than a sophisticated tool used in production.

In terms of availability, with the popularity of GitHub, our focus on the platform can benefit the most users. Currently, our extension would require a GitHub access token to be stored locally, which is not possible to use if we publish the project to the Chrome Web Store. Despite this, we still believe that it would not matter much because the additional processes only include cloning the repository, turning on developer mode on Chrome, and importing the repository. The relatively tedious part of the installation, which is acquiring the access token, needs to be done even if we publish the Chrome Web Store. Since it is likely that the intended audience has git already installed, we view our project as very accessible in terms of availability. More install information can be found on our repository’s readme file.

2. Research Questions

In our project progress report, we proposed two research questions, which we will explain in detail.

2.1 How can we evaluate the quality of a project from the DevOps perspective?

2.1.1 Data From GitHub

Before we dive into the metrics of DevOps, we need to first explain how GitHub’s features, especially GitHub Actions, can help us evaluate DevOps. GitHub Actions is essentially a CI/CD platform that can automate tasks such as building and testing. We apply a Yaml file to dictate

the jobs required to execute after a commit, where one or multiple jobs compose a workflow. Within each job are several steps, where each step is an automation task. Different workflows can be defined to have different purposes; in our metrics, we would only require three workflows for these specific tasks (which can be combined if fit): deployment, release, and unit test. The reason why we choose GitHub actions for deployment data is that it requires minimal installation, and users can configure workflows right away. Many popular repositories also take advantage of it and run automated tasks on them. The demonstrated example, Natrium, also utilizes GitHub actions to automate deployment and release tasks. Therefore, obtaining GitHub Actions data is an intuitive way to observe the quality of DevOps.

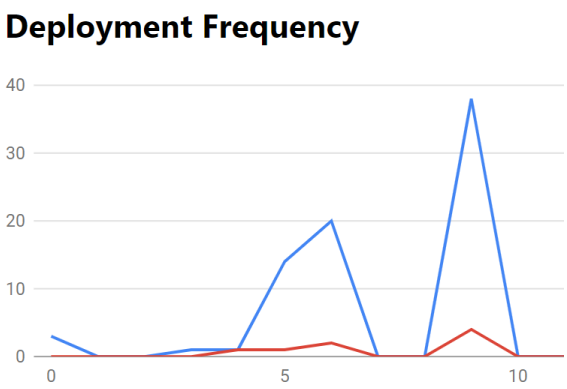
Another one is GitHub Issues, which can be seen as a tool for people to report errors observed by the user, as opposed to unit tests. Only the active issues would be counted. We mainly use them to calculate Defect Density.

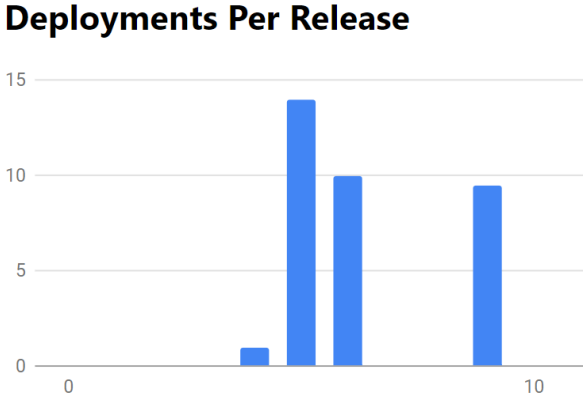
From the data provided, we did three sections that can help evaluate metrics. We only did the metrics that was proposed initially because each metric presented in papers have several ways to evaluate and present them, so we focused on making the proposed ideas more comprehensive.

2.1.2 Deployment Frequency

The first metric in our extension is Deployment Frequency, which in our implementation, needs to define a workflow that runs whenever a deployment happens. The metric can be calculated by the deployment workflows ran (blue) / time unit. Release workflows (red) are also included for reference. There are four options for the time unit: week, month, year, and custom development cycles. Each time unit starts from the current day and displays data up to 11 units of time earlier. The higher the deployment frequency, the better in terms of DevOps because the errors in the system can be minimized [1]. More deployments mean less code being altered per deployment, which also makes defects less likely to occur. Even in the case where there are errors found after deployment, developers are more likely to fix them easily because there is less code to examine. In conclusion, the metric evaluates the concept of CI/CD perfectly, which encourages users to make incremental and frequent code changes.

The first metric “Deployment Frequency” is in line with the RDF metric, which is the total number of deployments/time units (taken in hours) [1]. Due to fewer deployments of public repositories on GitHub, we adjusted to make the time frame longer. The next metric “Deployments Per Release” resembles the metric of the Four Key Metrics (FKM) more, which is explained as the “number of deployments/releases in a certain period” [2]. We believe that our calculation has the spirit of the metrics presented above.





2.1.3 Defect Density

In our current implementation, we have three metrics that help to evaluate Defect Density.

a. Issues / Deployments: This represents on average, how many unsolved GitHub Issues per deployment. The lower the better, which represents a higher software quality. The issue count reflects errors observed by the user, which cannot be detected by automated testing tools. If the metric is too high, the development team should consider doing more checks before deployment.

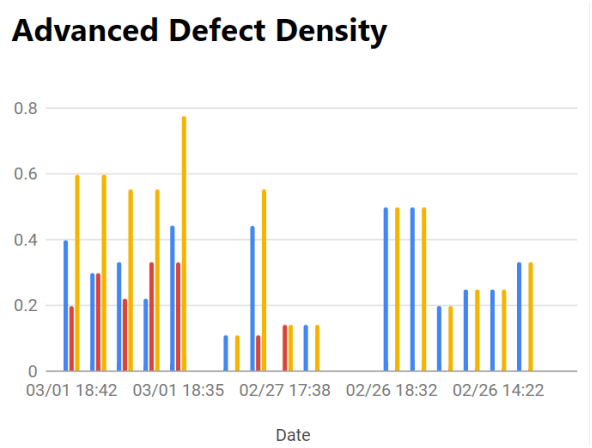
b. Issues / Successful Deployments: For this metric, the lower the better, which again represents a higher software quality. The switch of the denominator to successful deployments is useful to examine the unit test process. Since we assume that the issues are not detected during deployment (hence being a successful deployment), if the metric is too high, we would need better unit test processes to catch the errors.

c. Successful Deployments / Deployments: This calculates the percentage of successfully executing a deployment. The higher the metric, the better. If the score is too low, then the development team should consider doing more checks before deployment.

In one paper, Defect Density is listed as a metric for DevOps, calculated as the total number of defects/size of software [1]. The defects described here are not precise, so we interpreted them as issues reported by the users or failures/errors found in unit tests (the latter will be introduced in the next section). In Four Key Metrics (FKM), there is no direct corresponding metric. However, the “Change Failure Rate” is the main source of inspiration for this metric, which calculates the percentage where a change in code needs additional fixing [2]. For example, two methods proposed are “Detect failures by using monitoring metrics and divided by deployments” and “Count rollbacks divided by deployments”. In our definition, “failure” and “rollback” can be simplified as “an issue being created”, since it is not practical to monitor both server failure data and rollbacks from GitHub. We do believe that our metrics represent the spirit of the metrics because they all represent some form of software issue that needs to be addressed.

2.1.4 Advanced Defect Density

This is labeled “advanced” because it requires a specific unit test framework (Python’s unittest), and it also takes more time to generate due to the huge amount of log files being processed. For the chart, there are three bars for each unit test execution: blue is the failure ratio, red is the error ratio, and yellow is the ratio of the unit tests not passed. Here failure means the code gives the wrong result, error means the code does not run at all. This metric is also detecting software defects but from the unit test’s perspective, which makes it another method to evaluate DevOps. In conclusion, the two ways to measure defects in the extension are similar to the methods proposed in the papers, which are all valid ways to evaluate DevOps.



2.2 How can we let users utilize and understand DevOps metrics easily?

2.2.1 UIUX Design

To make it easier for users to understand and use our metrics, we need to deliver our ideas to our users clearly. The first thing for a developer to think about is the UIUX design because this is how we leave the first impression on our users and also where they receive the information we convey. Only if they can receive information in a comfortable way, positive feedback can contribute to the formation of a positive cycle, increasing users' trends and possibility of understanding what includes in our product.

Therefore, we mainly focus on how to make the interface clear and understandable for students who are unfamiliar with DevOps and software development. One of the essential designs we help users understand what each metric means is the text shown when users hover over the content. Because many professional concepts and definitions are mentioned in the metrics, we think of this way to provide supplementary information for users who need it while complicating the user interfaces as little as possible. Still, we leave users who are especially interested in detailed information on metrics another way to know the full explanations. Below all the charts, there is a drop-down list. Users can find complete explanations of the parameters. Only when users are ready to read them can they expand the list and read the details. With all these designs, users can face an interface with different levels of understanding cost as they want.

2.2.2 Comparison with Other Data Visualization Tools

As stated above, we aim to build a tool with more data visualization focusing on DevOps metrics and help users understand more about it. This is also the most significant difference between our product and other data visualization tools.

Compared to vanilla statistics functions in GitHub, our metrics mean more than basic and built-in metrics like the number of commits, code frequency, etc. For example, people cannot directly extract useful information from code frequency because these metrics cannot determine whether these codes fulfill specific goals or are high-quality. However, our metrics, deployment frequency and defect density, have their meanings, and as a result, they allow users to make judgments based on a reasonable standard. In this way, users can learn more about DevOps-related performance from them.

There are also many other advanced plugins, such as Jenkins and Grafana. They are such powerful tools that many developers use to monitor and analyze their data. They can customize every step of the data visualization process as they wish and extract information at a much

higher speed. But its problem is also this highly customizable feature. Professional users can make use of every feature of these plugins, but for beginners or people who are not familiar with data analysis, various sources of data and customization options will be a problem. Take the Jenkins pipeline as an example [3]. In a customized Jenkins pipeline, users can visualize job duration metrics, display the status and progress of selected jobs, support job view filters, build a view analyzer, etc. Users can write codes to customize every aspect of a project. However, it takes additional time for users to configure the environments, code, run tests, and so on. Even though it provides detailed tutorials for users to accelerate this process, its learning cost is high enough for beginners. What we do in this project is limit the scope of the data analysis to DevOps metrics and set up as many customization options as possible for our users. Therefore, users can focus on learning and getting used to DevOps.

2.2.3 User Survey

As for the differences between our project and other data visualization tools, we also did a user survey. It consists of two parts: Guide and Questions. In the former part, we help survey takers to install our extension and know how to use it. In the latter part, we ask them some questions based on two research questions. To see the raw data of the user study, please check the appendix, section 6.2.

The first section of our questions is whether our metrics can help users comprehend and learn more about DevOps. Though we receive mainly positive feedback on them, it shows that some people think these metrics cannot help them fully understand what these metrics mean in DevOps. We suppose that's because even though we put sufficient information about what each metric means in the drop-down list, they are not well organized. Maybe, we can detach information from the list and put it beside the chart to make it more accessible for users.

The second section of our questions compares our extensions and other data visualization tools. We let survey takers determine whether our product outperforms others in any specific ways. Most of them think that ours are clearer and more understandable. This is what we expect to achieve. But we also find that people think our product lacks extendability and expressiveness. We think we can explore these features more in the future.

3. Deliverables

Link to the repository: https://github.com/tsai00150/DevOps-for-GitHub
Milestone assignments are in the “milestones” folder.

4. Self-Evaluation

4.1 Tsai-Chen Hsieh

I wrote the backend of the project, specifically the functions that call GitHub’s API to retrieve data from them. One of the challenges is to write code that is readable so that Siyang can understand how to use my functions to complete the front-end projects. I did this by actively thinking about better variable naming and providing descriptions for every function that Siyang needs to use. Another challenge to that is Chrome development itself, which I have no previous experience in. I spent a lot of time figuring out how to import libraries, which do not work the same as server-side languages. Although the majority of the functionality can be developed with a front-end website in mind, there are also extra Chrome-specific configurations to learn on top of it. Overall, I feel that the technical side of the project is not easy but still manageable, and I

learned a lot of new technologies along the way.

I also designed the metrics that are presented in the extension, and subsequently the first research question in the report. The main challenge was that the DevOps metrics that I found could not be directly applied to the data provided by GitHub. I need to do some form of tweaking to make the visualization possible while staying true to the metrics presented by the papers. This gave me a deeper understanding of the metrics, and allow me to reflect more on the key attributes to define a good DevOps execution. The previous midterm paper gave me a broad view of DevOps; this project gave me an in-depth review of the components that make DevOps successful.

4.2 Siyang Zhang
I'm mainly responsible for the front end of the project, including data processing, data visualization, and UIUX.

This is the first time I have built a google chrome extension, so I encountered many technical problems setting up the project. For example, which components should I include in my manifest file, and what are they? Especially for some policies, chrome requires a few additional settings for developers to use functions like Google charts and importing external packages. It took me lots of time to solve them before developing the project. I need to further my ability to look up and understand open-source tools' documentation.

Another practical problem I need to solve is that Google charts require specific data structures as inputs. To visualize the data, I must first process them into the format I need. Though I've fulfilled all the functions we expected, there is still room for improvement in the efficiency of these codes.

I'm not good at designing an awesome user interface, so I focused on making it neat and clear. I reached the passing line, but there are still many possible improvements: Now, input fields for updating charts are placed at the bottom of the window. If they are moved to a new popup window triggered by a button beside the corresponding chart, they should be more straightforward, and the interface will be more organized. Also, although I divided the charts with different titles of different fonts, they look uncoordinated for some reason. Maybe I could expand the window's width and put more charts in a row. But to solve this problem, I need to see more products to learn what a good design is in the future.

5. References

[1] Poonam Narang and Pooja Mittal. 2022. Performance Assessment of Traditional Software Development Methodologies and DevOps Automation Culture. Engineering, Technology & Applied Science Research. 12, 6 (Dec. 2022). DOI: https://doi.org/10.48084/etasr.5315 https://etasr.com/index.php/ETASR/article/view/5315

[2] Marc Sallin, Martin Kropp, Craig Anslow, James W. Quilty, and Andreas Meier. 2021. Measuring Software Delivery Performance Using the Four Key Metrics of DevOps. In Agile Processes in Software Engineering and Extreme Programming, Lecture Notes in Business Information Processing, vol 419 (2021). Springer, Cham. DOI: https://doi.org/10.1007/978-3-030-78098-2_7 https://link.springer.com/chapter/10.1007/978-3-030-78098-2_7

[3] Gursimran Singh. 2017. Jenkins Pipeline and Data Visualization Dashboard. Retrieved from https://www.xenonstack.com/use-cases/jenkins-pipeline-visualization

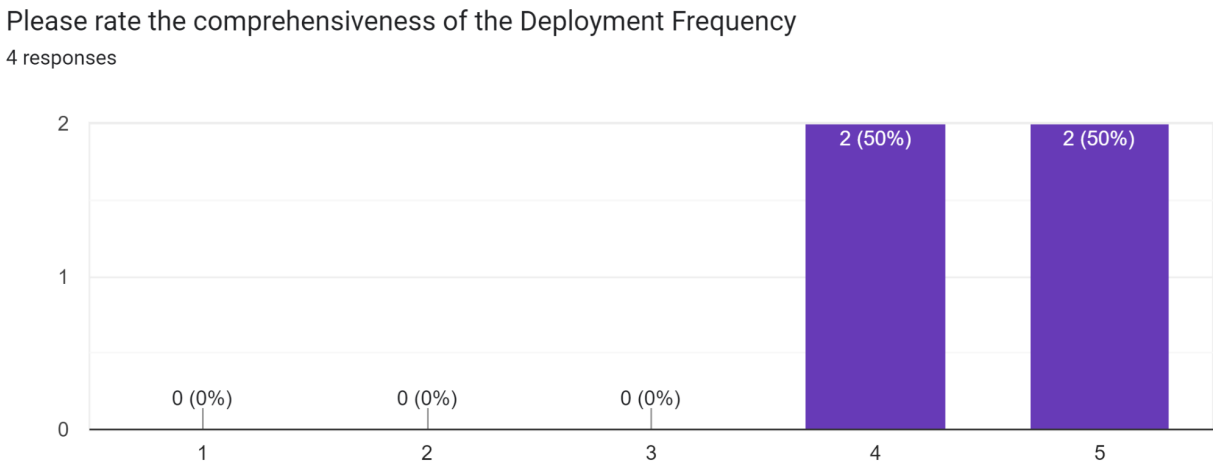
6. Appendix

6.1 Used resources (Acknowledgement section in repository's readme)

GitHub REST API - All data are obtained via the GitHub REST API.
JSZip - The project uses JSZip, a Javascript library, to help unzip the log files obtained through GitHub API. The library is stored locally to comply with the Content Security Policy for Chrome Apps, located at "scripts/jszip.min.js".
Google Charts - The charts are drawn using Google Charts.
Natrium - The instructions for parameter input uses the GitHub Actions data of the repository as an example.
Icon - Currently using the player icon from Among Us.

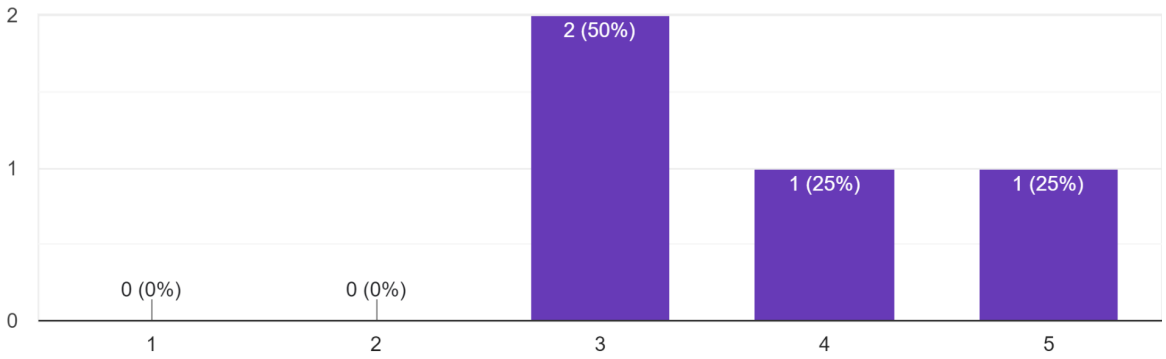
6.2 Survey Raw Data

Link to Survey: https://forms.gle/cVp6LxFthF7CY23fA (Requires a Columbia email account to access)



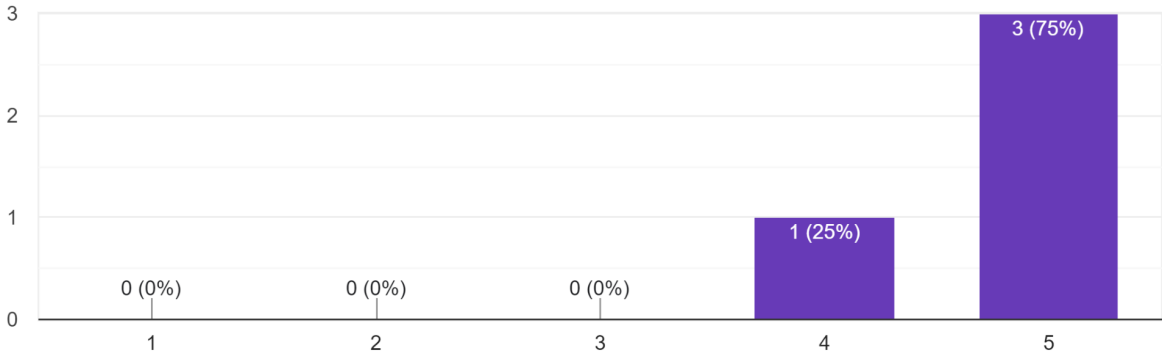
Do you think this metrics is helpful for you to understand and learn DevOps?

4 responses



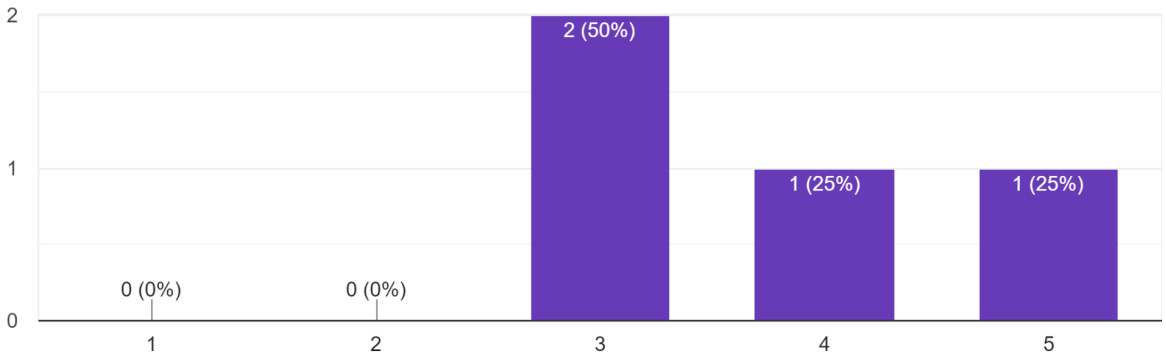
Please rate the comprehensiveness of the Defect Density

4 responses



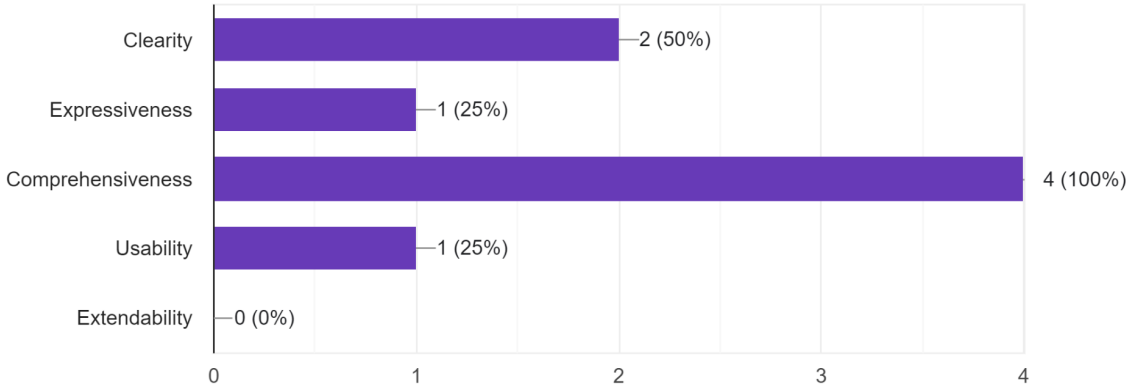
Do you think this metrics is helpful for you to understand and learn DevOps?

4 responses



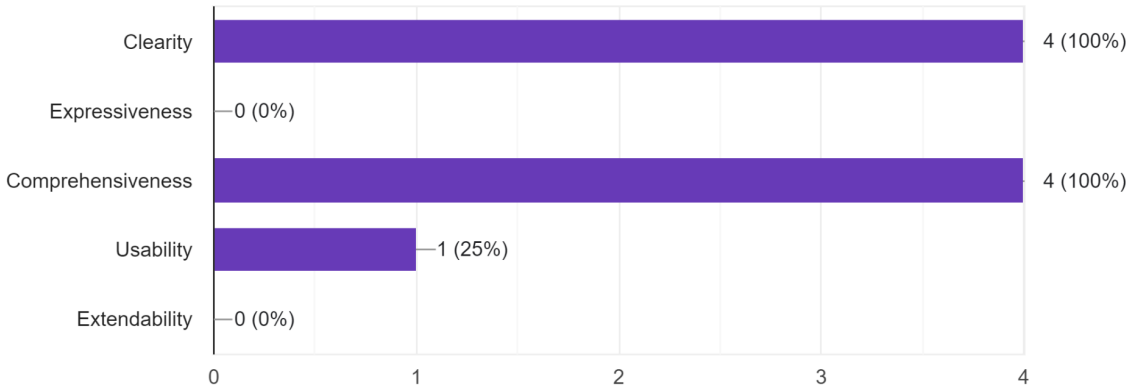
Compared to Vanilla GitHub metrics, in which aspects do you think our extension performs better?

4 responses



Compared to other plugins, in which aspects do you think our extension performs better?

4 responses



Do you have any more comments or questions about our project?

1 response

Hard to use input field