Graphical Lines of Code

An easy to use GUI to count lines of code

Ben Greenberg, Spencer Howell, and Daniel Troutman
Team Name: GLOC

I. INTRODUCTION

Our project is a tool called Graphical Lines of Code, or GLOC, based on the popular tool Count Lines of Code (CLOC). CLOC is a command line tool that allows developers to get a detailed breakdown of the contents of their codebase, including the programming languages used and the number of lines of each language, comments, and blank space. [1]

While the CLOC tool has proved to be popular with developers, with over 13,000 stars on GitHub, we believe it would be even more helpful if it provided an easy to parse graphical output. By using charts, graphs, and colors, we can make parsing the results of the tool quicker and easier. This would also allow developers to generate visuals for project updates. In addition, we plan on providing the tool via a web interface, allowing any developer to use the GLOC tool without downloading or installing any software on their machine.

To our team's knowledge, no fully-featured product like this is currently available to developers. While there are a variety of "count lines of code" programs, with features we can investigate adding to our project, no options with a robust graphical output exist. The closest thing that most developers currently use is the "Languages" menu on GitHub repositories, which simply shows the percentage of the codebase that consists of each language. We believe making a tool as accessible as this GitHub feature, with more detailed output, will allow developers to quickly and easily gain a better understanding of their codebase and share that understanding with others.

Our team consists of three graduate students who have experience as software engineer interns at a variety of organizations. We believe this background allows us to see the needs of developers and provide solutions that fill a gap in the current software development ecosystem.

II. CUSTOMER VALUE

A. Customer Need

Our primary customer is a software developer working on a project for their work, for a personal project, or just for fun. Software developers are always searching for ways to improve the health of their codebase - whether that be with automatic testing, code review processes, linting and formatting, or other tools. The goal for these developers is to keep their codebase clean, easy to modify and extend, and free from bugs and other challenges. There are many tools available to help with these goals, many of which are open source, free, and community maintained.

B. Proposed Solution

Our proposed tool will allow developers to measure the composition and health of their codebase with an easy-to-use tool. Using GLOC, they will be able to quickly and easily generate a report describing their codebase, and use this report to measure progress, track trends, and report their findings to their team members and clients. While tools like CLOC exist, we believe GLOC will make these reports more convenient and accessible to developers.

C. Measures of Success

We will know if customers are benefiting from our tool by a couple of metrics. The first is usage: if we see our tool is being used after it is published, then we will know that others find it useful. The second will be user feedback: we will present our completed tool to other software developers (the students in our class) and have them use GLOC on their own repositories. We will then gather their feedback on the usefulness of the product and make adjustments as needed. This feedback will include categories such as ease of use, user interface design, and detail of the output.

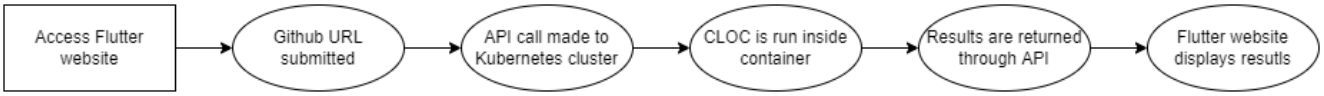
III. TECHNOLOGY

A. System

From a developer perspective, our goal is to have the user provide a link to the repositories they wish to analyze into our Flutter UI. The Flutter UI will then communicate with the back-end to send the link to our Kubernetes cluster that will host the actual CLOC application. After CLOC is finished analyzing the git repositories, it will then send the results back to the Flutter UI which will display the results to the user.

Our system will consist of a front-end UI and a server to host it, a Kubernetes cluster to host the CLOC application, and the CLOC application itself. The front-end will provide the UI for the users to interact with our application. The server will host the front-end and the back-end that will be used to communicate with our Kubernetes cluster. And the Kubernetes cluster will host the CLOC application that will perform the core analysis functionality of our system.

Our goal for a MVP is to provide a simple web-interface that users can use to run the CLOC application without having to open a terminal environment or install the applications on their own machines. Some possible enhancements



that would add value to our system are adding tools that can generate graphs based on the analysis done by CLOC. Another possible enhancement would be to add git client authorization to allow users to analyze private git repositories.

We will test our system by writing scripts to analyze data sent to the front-end to data given directly by CLOC in the terminal. We could also try to set up a CI/CD pipeline to automatically test and deploy any updates for parts or our application such as the Flutter web app.

B. Tools

We will be using a cluster of raspberry pi 4s to build our Kubernetes cluster that will be used to host the CLOC application. We will use Flutter to build our front-end and Firebase to build out our back-end. We can possibly leverage Github Actions to add a CI/CD pipeline to our development process, and we could use Ansible to better automate the configuration of the Kubernetes cluster.

IV. TEAM

A. Skills

While we have not made this kind of project before, we have some reated experience. All of us worked together on a senior design team, and we built a mobile application in Flutter. While our interface is not going to be designed as a mobile app, the cross-platform language means we can use our experience to develop for a browser. We also have experience using APIs from a previous digital archaeology class as well as in our software engineering internships.

We are familiar with some of the tools we plan to use for this project. Figma and Flutter were used in our senior design project, so we should work well with them. For container technology, we all have a little experience with Docker from the digital archaeology class. Other technologies such as Kubernetes and Perl (if we have to hook into CLOC) are less familiar to us as we have not used them before. There is extensive documentation and tutorials available that we can use to create those parts of the project.

B. Roles

We took inspiration from the scrum design process and will have three roles for our team members. The product owner will have the vision for a particular part of the product and can work with the other developers to implement their ideas. The development team will be responsible for actually implementing the project with the various technologies. Finally, the scrum master will help with tracking progress and focusing work where it is needed to make deadlines.

Since we all have different specialties in front-end, API, back-end, and automation, we will go with a rotating product owner approach. This works well with our project schedule because we focus on different parts of the app throughout the semester. Since our team is just composed of three members, we will all take on the developer role and work to create the product. For scrum master, we decided one person can have that fixed role so high-level progress is not lost when transitioning to someone else on the team.

V. PROJECT MANAGEMENT

A. Schedule

We plan to complete the project and have a working prototype that can be useful to developers. Our current schedules allow us to meet between classes twice a week. We can discuss the project face to face during this time. Outside of this break, we can reach each other through messaging or voice calls.

- 2/12/22 - 2/18/22: Proposal
- 2/19/22 - 2/25/22: Initial Designs & User Story
- 2/26/22 - 3/04/22: Start Back-End Setup & Architecture
- 3/05/22 - 3/11/22: API Development & API Testing
- 3/12/22 - 3/18/22: More Prototype Work
- 3/19/22 - 3/25/22: Finish Prototype
- 3/26/22 - 4/01/22: Configuration updates
- 4/02/22 - 4/08/22: UI Updates
- 4/09/22 - 4/15/22: UI Updates
- 4/16/22 - 4/22/22: Start Presentation & Work Wrap-up
- 4/23/22 - 4/29/22: Finish Presentation

B. Constraints

Since this project does inspect code, we would only want open-source code to start with. We will only run code on a private server and won't store any of the actual source material. If someone did want to run measurements on their code, they should own the material or have the right to distribute it for this purpose. If they then decide to use it with our tool, they should know the risks of sending source code to a third-party such as us. There are currently not any ethical or social concerns with building or using our product.

C. Resources

For testing and demonstrations, we can use Github to find and download open-source projects. To find these projects, we can use Github or World of Code. Since World of Code lets us filter by language, we can find a wide variety of projects and analyze them with our product.

D. Descoping

While we hope to add many features to GLOC, there exists risks with any project. If we cannot add all the features, there should still be a few things it can still do. If the automation and server cannot be implemented, a visual interface to a local version of CLOC could still be useful. On the other hand, if we cannot get a nice front-end to display the results, it might still be useful if someone wants to interact with it

through an API. Even though it may not look any better than running CLOC locally, lightweight machines could still use web requests to gather insights without having to clone the repositories themselves. Since CLOC is the tool doing the work underneath the features we build, it should not take as long to get a working product.

VI. PROCESS SELECTION

To give us the best chance at success with this project, we decided to use an iterative design process. In the iterative design process, you create a prototype and release betas with more features in each version as you complete the concept. We feel this process aligns with our work style and project because we have an idea for a prototype and a few features. While other processes like scrum may work for different groups, our small team did not want the overhead of backlogs, reviews, and retrospectives. While flexibility is a great benefit for larger teams, we feel that three members naturally gives us this side-effect. Even though we are not using the scrum process, we did adopt the three roles of product owner, developer, and scrum master. Another option we could have used instead of iterative design is the waterfall process. In the waterfall process, the entire product is planned and takes place in a very linear fashion. We are unfamiliar with some of the technologies we plan to use so it is hard to estimate how much each part would take. If we estimate wrong and the development takes longer than expected, we may not be able to implement the minimum viable product by the end of class. With iterative design, we can get a working prototype, then add features and get feedback. If we are not able to complete all the extra options we had in mind, we will still have the core features that satisfy our original goals and be useful as a development tool.

VII. USE CASES

This section describes the primary use case of our application. The primary actor is a software developer working on a non-trivial project. The user goal for this developer is to gain a better understanding of the contents and health of their codebase, and to share that information with others. For the basic flow the user will take to gain this understanding, they will perform the following steps with GLOC:

1) Copy the URL of their repository hosted on GitHub
2) Go to the GLOC website, paste their URL into a text box, and click the “Analyze” button
3) Wait on a loading screen while our servers run the CLOC utility and generate a graphical representation of the results
4) Once done, the user can view these graphs and interpret the results
5) The user can click a “copy” or “download” button below each graph to save an image to easily share with others in presentations, emails, or instant messages
6) The user closes the website

Our goal is to make these steps as intuitive and easy to perform as possible.

VIII. ARCHITECTURAL VIEWS

A. Development View

One part of our application is the front end that the users will interact with. Here, there are several different structures that will be developed in Flutter to give users a complete and easy experience. One of these structures is the results page where the charts will interact with data acquired through the API. Another interaction is between a configuration page and the web request code. This interface creates customized commands that are sent to be run by our back-end servers. A third structure is the interaction between Flutter and a user’s local computer. If someone wishes to save their code statistics, they can download it so this to will need to be supplied through web requests. All of these structures and interactions are part of a development view to help solve the problem of effective code exploration.

B. Deployment View

Another view that we are implementing is a deployment one. The entirety of the deployment of our application lies within a private Tailscale VPN, and it will remain within the VPN for the development of the prototype. The CLOC API is deployed in a Docker container based on the Shell2HTTP image that allows us to run CLI commands via HTTP requests. This container is deployed on a Kubernetes pod in a Raspberry Pi K3s cluster with 4 replicas to distribute any incoming requests. A load balancer service is used to expose the CLOC API pods through a single IP address in the K3s cluster. The CLOC API cluster-IP is then exposed from the cluster over the VPN with a Tailscale subnet router. This way any application can access the CLOC API as long as they are connected to the VPN. While we use many tools to support the API, each one serves to make the processing faster, more secure, or more robust so we can have a strong deployment.

REFERENCES

[1] AlDanial, cloc. 2022. Accessed: Feb. 17, 2022. [Online]. Available: https://github.com/AlDanial/cloc