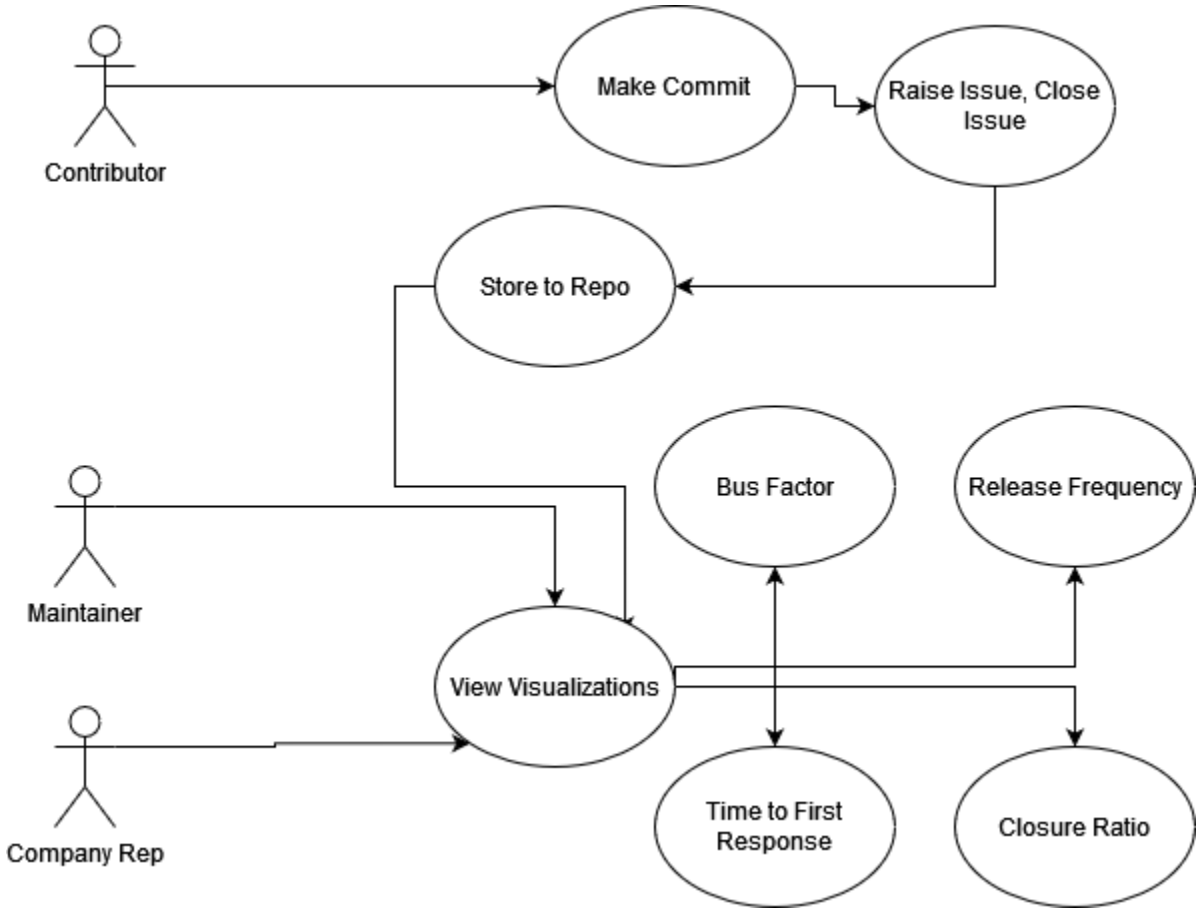
Sprint 2 Visualization Design

Eli Bollinger, Ryan Milewski, Matt Marlow, Jacob Groves, Ellie Nash

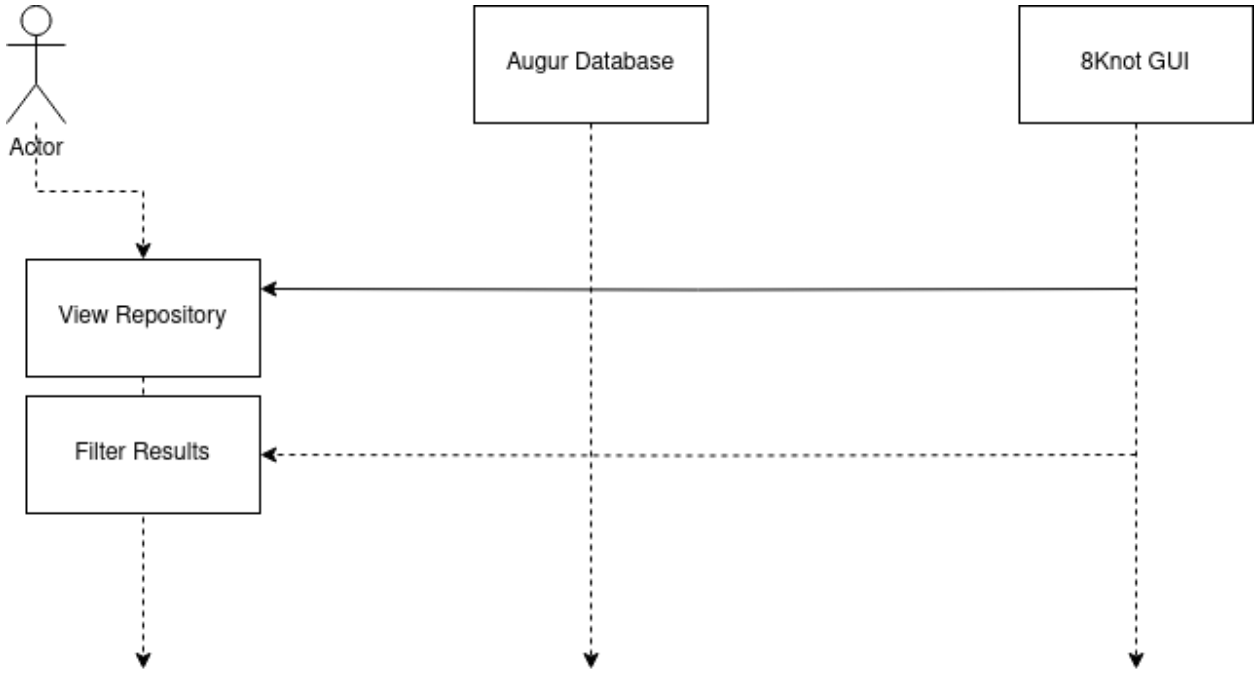
Table of Contents

Sequence Diagram of Actors



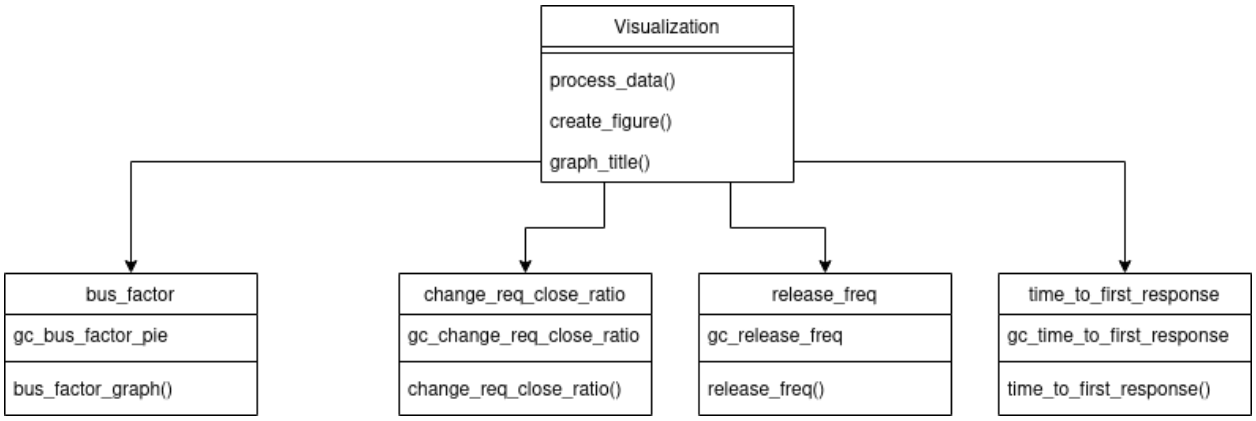
This sequence represents all three actors identified in our previous requirements document. A contributor makes commits, raises issues, closes issues, etc within the repository. The system tracks the repository and makes it available when the maintainer and company rep view visualizations. From there, the starter project health metric model is built and displayed to these actors.

Visualization Sequence Diagram



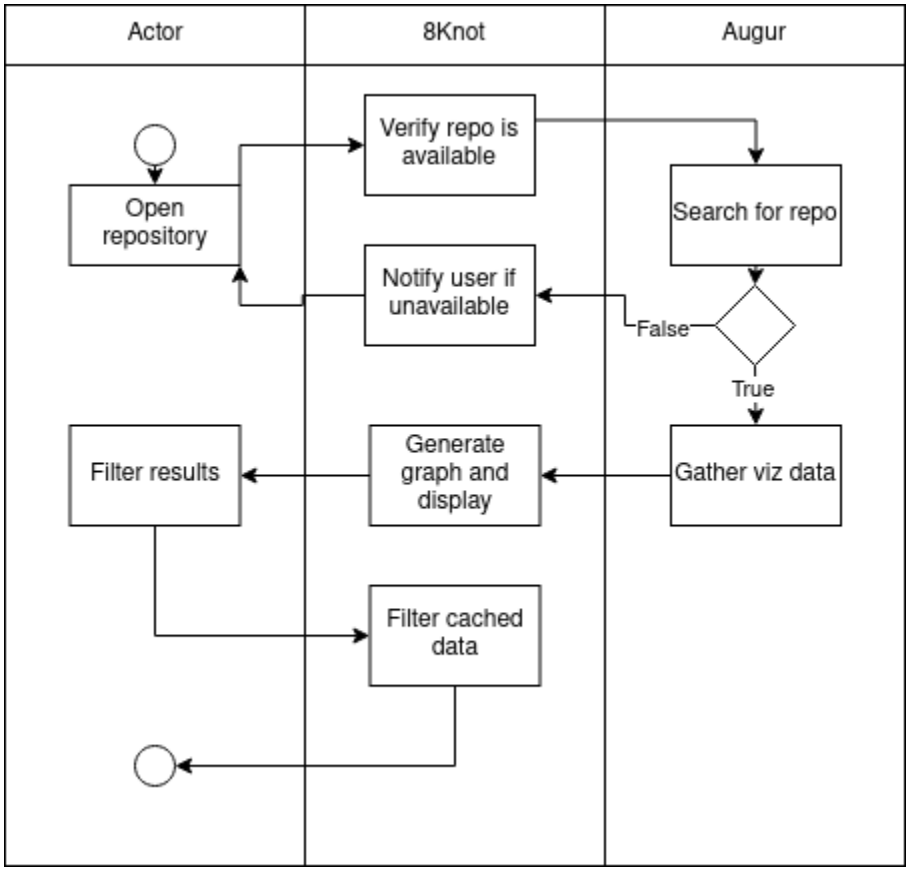
Two primary services are used in visualizing metrics - the Augur database and the 8Knot client. When an actor wishes to view a repository, both the database and GUI work in tandem but once data is cached on the page, filtering results is only interacting with the GUI.

Class Diagram



Each page is organized as a subclass because all of them process data, create a figure, and set a title. Each visualization has its own Bootstrap card as well as its own query to call within gc_*.

Activity Diagram



When opening a repository, the 8Knot GUI first verifies that the repo exists by querying the database, and if it is unavailable, returns some sort of error message. Otherwise, the relevant visualization data is gathered and returned to the GUI. Then, the GUI builds a visualization graph and displays it on the end page.

The user has the option to filter data by year or start date. If so, the 8Knot GUI will rebuild the graph based on the filter criteria.

Bus Factor Design

Our initial thoughts for design had us use a pie chart, displaying an ordered list of the biggest contributors but we quickly realized that that gives very little information and insight into the project. A pie chart would only show a snapshot in time and would give very little relative information about the project. A snapshot in time would not help any of our actors make meaningful conclusions so we scrapped that idea.

Since our metrics are starter health project health model, to fit our actors we decided change over time would be more beneficial, so we decided to implement a simple line graph as it allows us to display how the bus factor of a project is changing over time, which helps the actors make much better decisions based on the data.

To display this over time, we need to pick a unit of time and calculate the bus factor at each unit of time. Month seems to be the sweet spot of a time unit. Thus, to implement this, we will need to turn the dataframe of all commits in a repo into a dataframe of months and bus factor per month.

This transformation could be done by separating the data frame by month based on when each commit was committed, then grouping the commits by author and retaining the number of commits per author. From there, we can sort all of the authors by number of commits in that month, calculate the total number of commits that month, then traverse down the sorted list to determine how many authors it takes to reach 50 %, which is the bus factor. You can then do this for every month resulting in the dataframe to be displayed.

Time to First Response Design

Given our actors and requirements laid out for Time to First Response, we thought between two ways of displaying the data. A histogram would give good insight into the time between when an activity was opened and when it was first responded to, while it gives very little insight into the current health of the project. A line graph would give insight into how this value is changing, but very little into what's behind the average.

While having information about what is behind the average would give good insight into the value, showing outliers and how distributed those times are, you lose the very crucial information about the health of the project over time. We decided that insight into the overall health of the project was more crucial than the makeup of the response time so we went with a line graph showing the time to first response.

Looking at the queries in 8Knot, we decided to look at pull requests and pull request responses. From this we can look at each pull request, find the first response to it by the request ID and timestamp, then from there we just need to combine the data by month into an average to be visualized.

Release Frequency Design

Given the required actors and requirements in Release Frequency, we only contemplated a scatter plot showing when each release occurred. This allows for each release to be marked as a separate point on the visualization and will give insight into how frequent the releases are.

With our release frequency, we allow the user to pick the date interval to allow them to pick between day, week, month, and year to allow them to see the optimal view for whatever their dataset allows.

Change Request Closure Ratio Design

Given the actors and requirements for the Change Request Closure Ratio, we thought about a couple of different ideas on how to display the data for this visualization. We thought about using a line graph, but we were not sure if we should display just the two lines for closed and total change requests and allow the viewer to manually calculate the ratio, or actually display the ratio on its own line. We settled on the latter because it would be easier for the user to see how the visualization is calculating the ratio.

Similar to release frequency, we allow the user to pick between the day, week, month, and year to let the user pick the optimal view for their dataset.

Contributor Count

First, we reviewed the contributor count visualization examples. There was one that offered an immediate count and displayed that, and one that showed counts overtime. We decided to show contribution count as a line graph over time as we felt it was more useful to the end user and we had the existing infrastructure to implement. Similar to our other line visualizations, the user can enter in a date to narrow the visualization and hover over the line to see exact counts.

Issues Closed

We looked at the issues closed example, and for most of the examples, they used a histogram to display the data. We chose a similar approach and chose to use a histogram as well as allowing the user to also choose the timescale on which they would like to see.

Issue Assignments

Originally, we wanted to represent comments left on an issue, but there were two problems with this. First, comments don't necessarily indicate anything productive is going on. Second, the query for pulling issues did not include comments left under those issues. Instead, we opted to go with a histogram of people assigned to issues, to fit with the Issues Closed visualization. These visualizations complement each other, as they show the ratio between issues that are fixed vs. issues that are being actively worked on.

Code Review Count

CHAOSS defines this metric as the average number of comments a change request receives during the last 90 days. While that gives a good view of the project currently, it doesn't show the status of the project in comparison to itself, so instead, this version of the code review count metric shows the average number of responses a change request receives by month over the entire history of the project, which allows you to see the relative health and status of a project over time.