TEAM.NAME V2

TEAM_03

MOZILLA : CODE-COVERAGE

Deliverable 1

Members:
Alexei COREIBA
Gaganpreet KABERWAL
Harsh PATEL
Kohilan MOHANARAJAN
Kanstantsin ILIOUKEVITCH

Supervisor:
Prantar BHOWMIK
Anya TAFLIOVICH

February 26, 2020

Contents

Architecture

Context

Let us start off by introducing the concept of 'code coverage'. This is will help us understand the purpose of this side project within Mozilla and how it contributes to the overall Mozilla community. Code coverage is a term that stands for how many lines of code are covered by any testing in relation to the entire function, method, class, or repository being evaluated. In other words, given some tests, how many individual lines - in terms of percentages - are executed within the overall scope of your test.

Mozilla has automated it's testing such that it is automatically ran for any commit to their master branches for the Mozilla Firefox browser. The code coverage of individual tests are agglomerated in respects with their respective repository at which point said repository is given a code coverage score. These scores are then fetched by our web-application and are shown in an interactive fashion. The current production web-application can be found here: coverage.moz.tools. The repository for the project can be found here: github.com/mozilla/code-coverage.

This web-application is split into three main components: {frontend, backend, bot}, along with several smaller - yet important - components: {tools, report, events, addon}.

The frontend is written with JavaScript and is in charge with serving content to the end user. The backend is written in Python3, with Flask as it's backbone to accelerate development, is in charge of fetching the needed data for the frontend. The bot, written in Python3, is in charge of overseeing any code-coverage testing that is done, notifies the needed stakeholders, and stores it onto their Google Cloud.

The smaller components, even though that is how they're described above, are not all necessarily small in the sense of the word. Their purpose is to support the three main components of the application and act as 'helper' components. Firstly, we have the tools component. This module has three purposes. It established the scope of languages that are supported by this web-application, it is in charge of the logging procedure to keep a 'paper-

trail' as they define it, and finally establishing a connection with their google cloud service account. Secondly, we have the report component. This components seems to be responsible for testing the firefox browser in terms of code-coverage. Thirdly, we have the events component. This section is in charge of handling any workflow management for within the organization. Lastly, we have the addon component. This component augments other existing Mozilla web-applications with code-coverage data.

With the general understanding of what these two sets of components do we now have a good understanding of the structure and flow of data through this web-application. This will allow us to understand the generated, and hand-built UML that shows the class structure layout of the application.

Main Components

Frontend

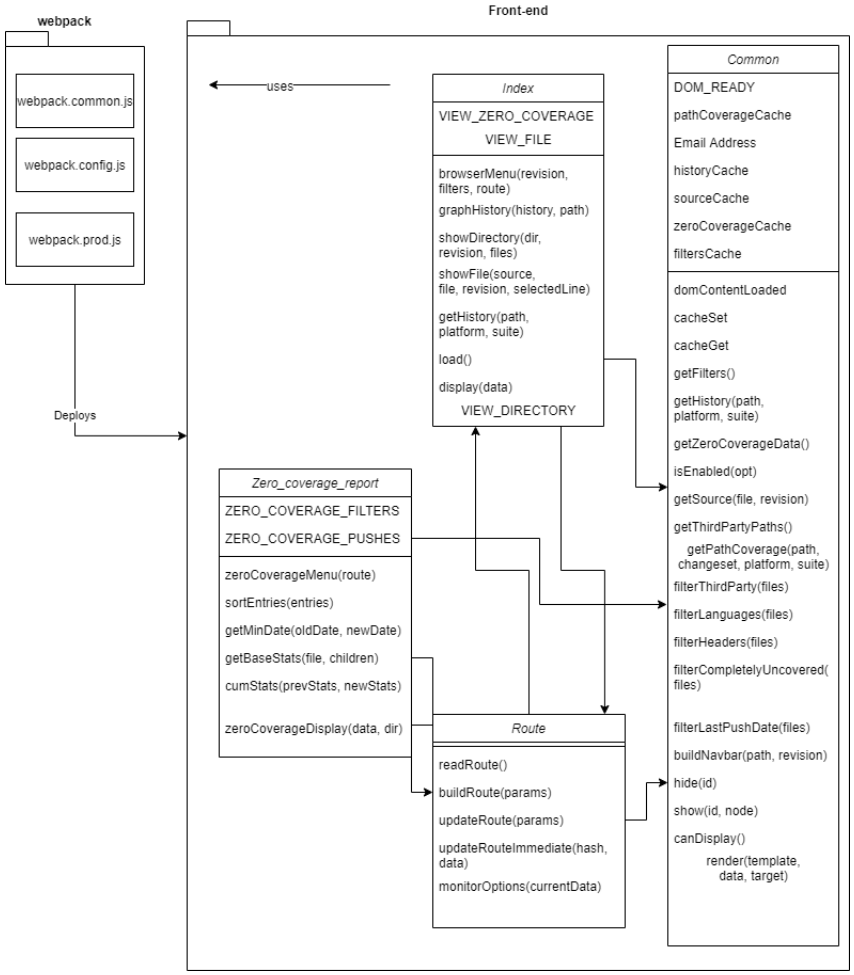

Figure 1: Generated class UML using Draw.io

The design of the front-end is very disorganized compared to more recent projects from today. There are 4 main files that work to display the data regarding the different code coverage for all the different Mozilla projects. Route.js parses and generates different routes as needed. Common.js is a large file of utility functions that are used by all the other files. Index.js renders the different components like the graph, navbar, and directories +

files. Finally zero_coverage_report displays files with no coverage. This is to allow users to see files that do not have anything reporting in the same place as the ones that do. There are 3 web-pack files that are used to deploy the front end server.

Backend

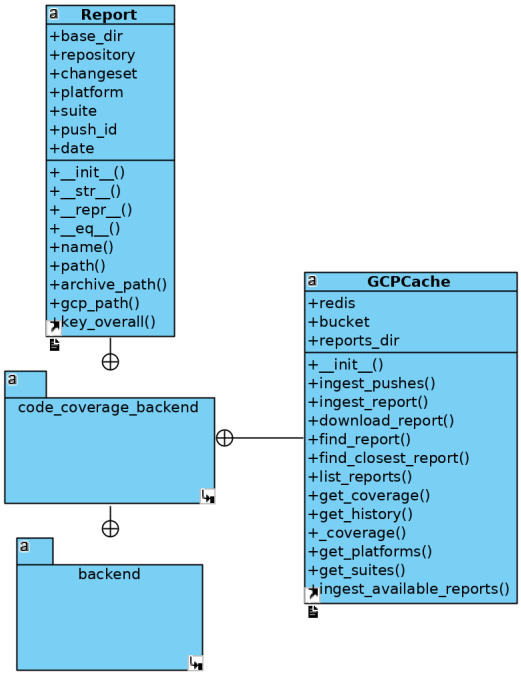

Figure 2: Generated class UML from Visual Paradigm

Upon attempting to generate some kind of UML for the backend, the only tool that we were able to get some kind of readable output with was Visual Paradigm - an enterprise modeling software - which is captioned as Figure 1 above. Sadly, this only parsed any section that was defined as a class within Python. Since Flask eliminates a lot of code necessary for running a full-fledged backend and turns it into more of a scripting environment similar to Django or Node.js, we're left with only 2 defined classes that the backend

uses: Report and GCPCache. Both are co-dependent and are coupled as GCPCache acts a controller for all Report objects.

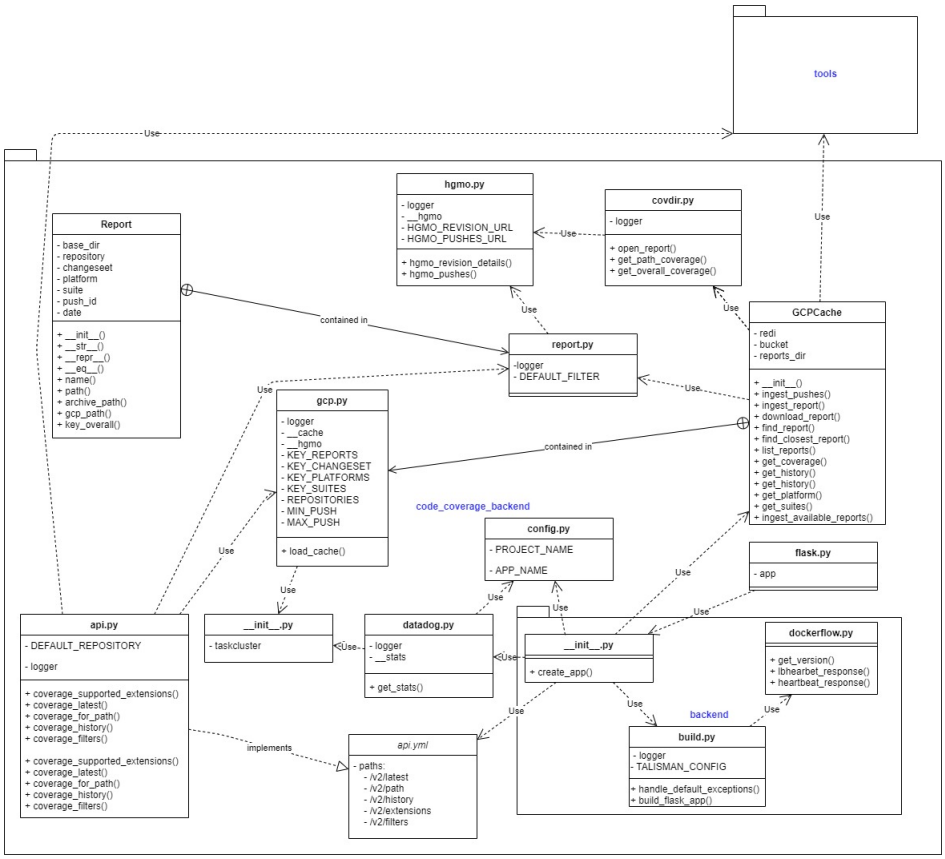

Figure 3: Hand composed UML covering full call sequencing of code_coverage_backend

A UML diagram was created to show how individual python scripts are used within the backend which can be seen at Figure 2 above. This is a more comprehensive diagram showing the dependency between python scripts and how the class diagram shown in Figure 1 is intertwined within the overall structure and dataflow of the backend.

Since this is a Flask backend, the application starts with the execution of the flask.py script which creates an instance of a Flask app. This in turn

parses api.yml which sets up the backend routing paths, and to which api.py then implements. This file is the brain of the operation as it is the endpoints of the backend. It is responsible for all calls being passed to the backend, most important one of all being gaining cached code coverage data from gcp.py, and formatting it into a readable format with the help of report.py. These two procedures take a lot of exchanges between what can be considered as 'middleware' that help to expedite and govern this process. Overall, a commentary on the design structure of the project's backend is that it is messy. Nothing is organized into some kind of readable fashion such that upon briefly scanning the name of the file, or the contents, you can decipher it's purpose. Only after the creation of this UML - Figure 2 - by hand, did we begin to understand the dataflow behind the scenes.

Bot

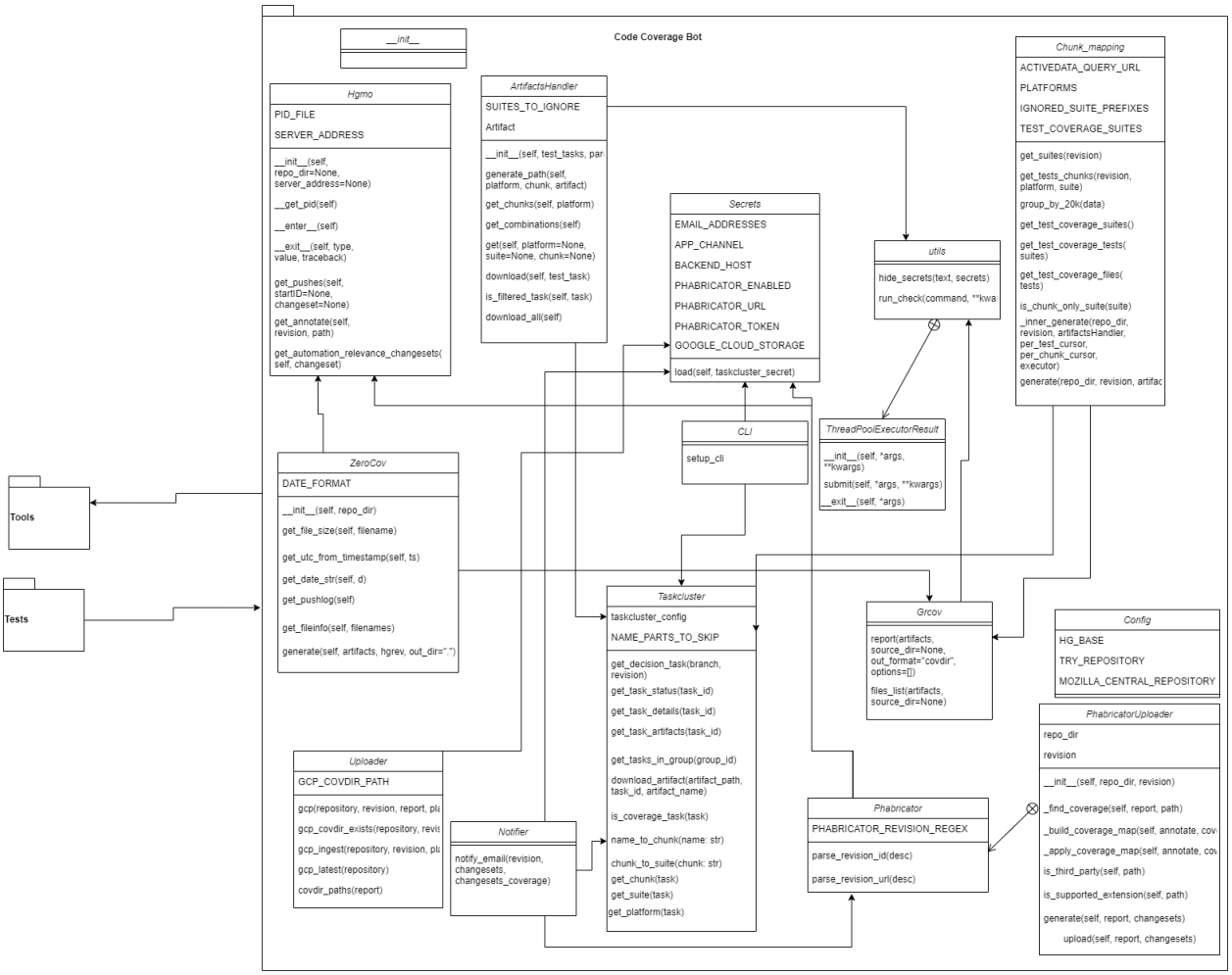

Figure 4: Hand composed UML covering full call sequencing of bot

Lastly, we have the bot component. The bot is responsible for doing the individual consolidation of all the code-coverage reports for the individual sections of the Firefox codebase. The overall process is quite complicated and is isolated with a docker container to sandbox it's actions. [idk what else to write man]

Helper Components

Tools

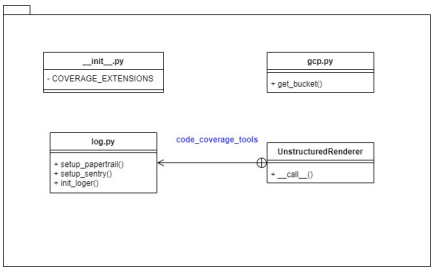

Figure 5: Hand composed UML covering full call sequencing of code_coverage_tools

The tools component, as stated in the context section above, has two main purposes: creating a logger to be used by the backend, along with, creating a connection with their Google Cloud service. As can be seen, there is no internal calls, meaning it is strictly used as a middleware thats been obfuscated away from the backend.

Report

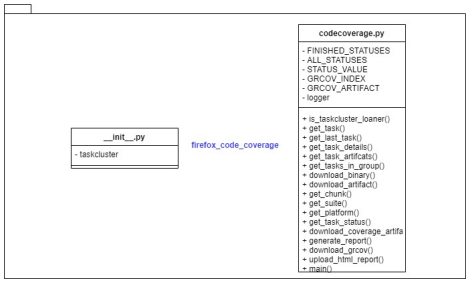

Figure 6: Hand composed UML covering full call sequencing of code_coverage_report

This component, as stated in the context section above, seems to be responsible with testing the code-coverage of individual components of the firefox browser.

Events

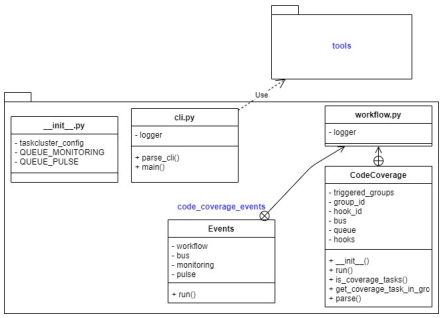

Figure 7: Hand composed UML covering full call sequencing of code_coverage_event

This component is a little more important than others. It is in charge of event monitoring. This means that this module monitors any code-coverage test and informs the necessary stakeholders that it has been completed. In other words, it is a collection of event handlers for any workflow changes that occur between testing to post-completion.

Addon

This component, as stated in the context section above, is responsible for transferring data from this web-application to other medium for other stakeholders to see. This contains all the plug-in JavaScript files needed to be added for different applications. Such an example is the searchfox web-application.

What Can Be Changed/Improved

The biggest sore point with this project is the lack of documentation. Setting up this project was a big obstacle for the team because the lack of available documentation led to confusion in regards to what to set up. There also seems to be a lot of docker compose files scattered throughout the code base. This makes it difficult to manage and further hinders the progress of the setup. Combining these docker files into one would make things much easier and simpler for developers and contributors alike.

Another thing that should be considered is a total refactoring of the entire codebase. Even though this is entirely unfeasible, it should be considered as the flow of this application is very messy. It seems to have been patched together - which makes sense as it is an 'open source' project - although it seems by commit history that it was mostly created by Mozilla developers. A particular example is the frontend which should be re-factored into sub classes that are easy to follow and more manageable.

Software Development Process

Kanban Process

The team has decided to pursue the Kanban methodology as a software development process.

Kanban is a visual system of managing how work moves throughout the process of development. The main goal of Kanban is to improve the throughput by fixing any roadblocks in an efficient manner. It helps the team gradually improve and make changes on a system they are working on. In this way, the risk to the overall system is reduced, which also leads to low resistance in the team and anyone else involved. But there are principles and practices one must follow to reap the maximum benefits of this development method.

There are four foundation principles that this team will follow:

1. Start with what you are doing now: This method focuses on not making any changes to the existing setup or process right away. Required changes will occur gradually over the course of development.

2. Agree to pursue incremental and evolutionary change: We will make small incremental changes to the processes on a pace the whole team is comfortable with.

3. Initially respect roles and responsibilities: We will not make changes to existing roles (that will be decided beforehand) or functions which may already be performing well

4. Encourage acts of leadership at all levels: Any team member with any role may speak up and show leadership to help provide ideas and implement changes to improve delivery of the software.

The Kanban method also highlights a few core practices that the team will implement into the workflow:

- Visualizing the workflow:
  We will be using an online Kanban board (SwiftKanban.com) to visualize the the steps that will take place. This will help us keep all the members informed on the work that is being currently worked on, that has been finished, and that needs to be picked up.

  The current workflow sections/steps we have set up right now are:

  1. Backlog
  2. To Do
  3. Planning
  4. In Development
  5. Code Review
  6. Testing
  7. Integration Testing
  8. Completed

  We decided to segment the workflow into these eight steps in order to offer better visualization as to how far a ticket has gone, without being too compartmentalized such that maintaining the Kanban board would become a task in and of itself.

- Limiting the WIP:
  This is necessary to ensure that the team will complete work on hand before taking on anything new. The WIP limit for our team is 10. But additionally, the agreed-on limit per team member is two, therefore each team member will have a max of two items in the work-in-progress swimlane.

- Managing the workflow:
  At various stages of the workflow, one may observe either a smooth flow within the WIP limits or work piling up. We will analyze and make adjustments if any such setback occurs to help reduce the time it takes to complete each piece of work. This may involve reducing the time work remains in intermediary wait stages (i.e. waiting for someone else to complete a dependency) or any handoff stages. Improving the ability to forecast completion times will be an important aspect to focus on for the Kanban system.

- Making process policies explicit:
  We will create explicit guidelines on how the work will be done. These policies will be set at board level and for each column of the board. These will include - defining when a task is marked as completed, defining priority of items, implementation of test cases before the item moves on in the workflow etc.

- Implement feedback loops:
  Reviewing the stages in the Kanban workflow will be an integral part. This will help provide the team with continuous feedback about the work being done, or the lack of it. Getting early feedback will be crucial to figure out if one is on the right or wrong track.

- Improve collaboratively and evolve experimentally:
  Since the Kanban method is an evolutionary development process, it will help us adopt changes in a gradual manner. We will use a scientific method where we will form a hypothesis, test it and make changes based on the outcome of the test.

Why We Chose Kanban

In order to decide what direction to take, we had to consider all the software design processes we had at our disposal. First, we looked at waterfall. This method has been used for many years and by many companies in the past. The strengths lie in the structure as it is easy to understand and follow. It is a step by step instruction manual on how to conduct the development process and guides the team on what to do next. The problem is that this process makes changes very difficult to accomplish. We will be working in a fast paced environment with a lot of unknown factors where a lot of things

can change very quickly. Also, especially in the beginning, we will not be tackling large implementation projects. We also wanted to follow the team's strengths and experiences. The team had prior experience with Agile development, so the goal was to not deviate too far from that.

Our team has also had a lot of experience with Jira. We found that Kanban allows the usage of something that is similar to it (Kanban board) as we no longer have access to the Jira board from CSCC01. Kanban allows us to break down the work into small tasks and visualize them on the board. The board is shared among all the team members allowing us to assign and manage tasks easily and efficiently. The detailed overview of our plan for the use of Kanban is outlined above.

We will also make use of several XP principles and ideas, as they will help us greatly. We unfortunately, would not be able to do any pair programming to it's full extent as we all live far away from each other and do not have much overlapping free time on campus. We will however, be able to do some virtual pair programming as we share a screen through something like discord. We will also work in weekly cycles. We will have at least one meeting once per week to go over everyone's progress, challenges, and goals for the week. We opted for weekly meetings instead of daily meetings (as detailed in the Kanban process), as the amount of progress made for each team member would scale more accordingly with a weekly schedule, what with the 4-5 other courses each member of our team is currently taking. It is during these meetings that we will also revise and modify the Kanban board. We will continue to write tests for our code to make sure that we cover all cases and minimize the chance for bugs.