Requirements Specification

Git OSS-um

Gary Baker
Van Steinbrenner
Stephen White

November 19, 2018

Project Sponsor: Dr Igor Steinmacher,
Assistant professor, SICCS, Northern Arizona University

Faculty Mentor: Ana Paula Chaves Steinmacher

Version 1.0

Accepted as baseline requirements for this Project:	
Client:	Team Lead:

Table of Contents

1. Introduction

Open Source Software (OSS) is a type of computer software in which source code is released under a license, where the copyright holder grants users the rights to study, change, and distribute the software to anyone and for any purpose. OSS may be developed in a collaborative public manner using a version control system named Git and hosting the project itself on a site known as GitHub. This version control system tracks changes made to a project and easily allows projects to have multiple contributors. It also allows for a project owner or manager to accept or decline contributions depending on their opinion of the quality of the change through what is called a pull request.

Unfortunately, a substantial amount of developers do not contribute to open source projects. We have all been there: we don't know where to start, what files are important, or what needs to be changed. The community may be friendly and quick to provide assistance to a newcomer, or the contributor may get berated for submitting a pull request that does not adhere to the project's standards. It is so daunting, that the newcomer gives up, and moves on. This experience shows that there is a need in the open source community for a tool that will allow a novice passing through to quickly and efficiently see how "newcomer friendly" a repository is.

Our client, Dr. Igor Steinmacher, is a researcher and assistant professor at Northern Arizona University. Dr. Steinmacher and his colleagues have created a web application called FLOSScoach[1] to address this need for a tool to help newcomers, where a user can investigate the skills needed to contribute to open source. Although FLOSScoach has been helpful, it has its shortcomings: FLOSScoach is manually fed, must be constantly maintained to provide updated information, and displays paragraph descriptions instead of visual representations of

[1] FLOSScoach: http://www.flosscoach.com

collected data to users. These limitations restrict use of the project to already available information and does not assist with finding new projects to contribute to.

Due to the shortcomings of the current solution Dr. Steinmacher and his colleagues have come up with, we will be spearheading a new solution. Our job is aimed at giving power back to the newcomer by creating an autonomous process where relevant data of a requested repository is presented to the user. This will allow them to decide if the specific OSS project will support their needs and help them feel at ease when picking an open source project to contribute to.

In order to develop a successful newcomer meter for open source dynamics, we will create a web application that will take a URL leading to a GitHub project page, which will then be accessed via the GitHub API. Desired data such as newcomer acceptance rates and newcomer retention, among various other data points will be collected. This data will be used to assist users in finding projects, and interactive graphics will demonstrate to the newcomer what OSS projects are best for them.

2. Problem Statement

Because Dr. Steinmacher does research in open-source, it is important to build a web application, since open-source software projects are based in online repositories and through sites such as GitHub. The built in functionality of these websites is what assists Dr. Steinmacher with his research into OSS.

FLOSS coach

Dr. Steinmacher and his colleagues have created a web application called FLOSScoach, where newcomers can find the skills needed to contribute to an OSS project. Despite this, FLOSScoach does have its flaws, including:

- Manually fed information
- Must be constantly maintained
- No interactive graphics

The three bulleted topics above are what makes FLOSScoach a flawed application for both newcomers and Dr. Steinmacher's research. Perhaps the biggest flaw of FLOSScoach is the manual input of projects to view. This makes the number of projects that newcomers can view very little. Because of the manual input of information, FLOSScoach must be maintained constantly. The absence of interactive graphics from a research area that needs a lot of data is a big fault, as the application will have a less powerful way of letting the user know how the specific project is compared to another one. We aim to fix these three issues into a more dynamic solution.

3. Solution Vision

We will be building a dynamic web application for newcomer characterization that features data mining from GitHub and interactive data visualizations that the user can explore to form their own ideas of what project they want to contribute to. The web application will allow users to store the mined data and compare repositories for further analysis. Below is a list of features that our web application will have to assist the newcomer and Dr. Steinmacher's research:

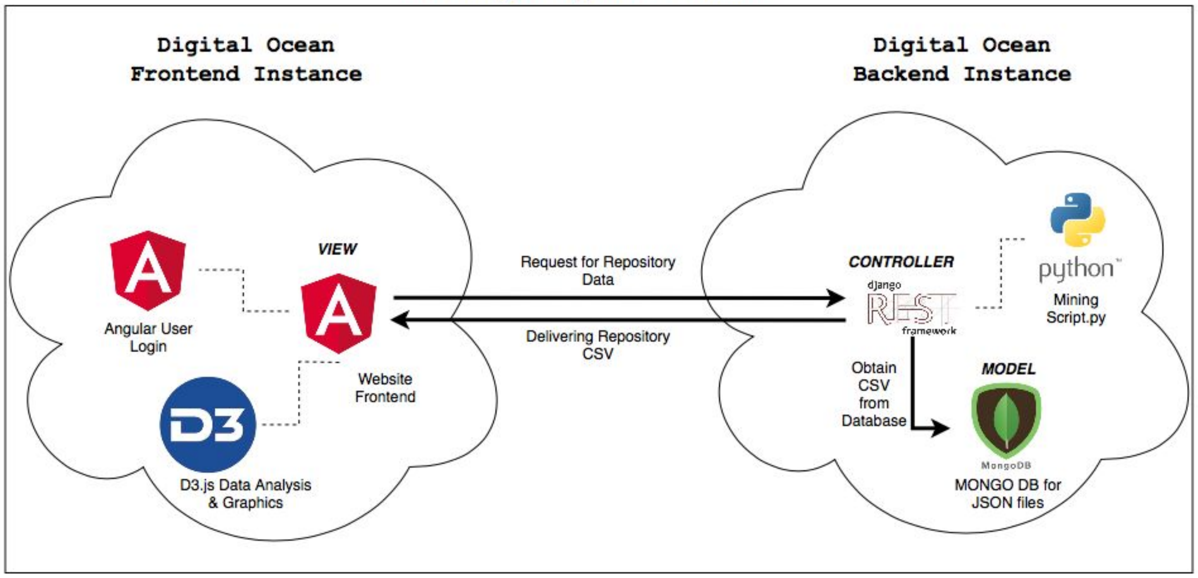
- Search for a specific repository
- Compare stored repositories, up to three at a time
- Explore interactive graphics and raw data

- Admin access to site
- View stored repositories and edit the repo status

The mined data is stored as JSON files, or JavaScript Object Notation, and are obtained from the GitHub API. From the collected JSON files, we will analyze the data and form interactive graphics and visualizations so that users can explore the data and formulate their own decisions for what project they want to contribute to.

We are hoping that our product will change the world of open-source software. With an application such as this, where users can dynamically request repositories to get mined, see interactive graphics, and compare repos, we can change how newcomers can contribute to projects.

Technology Integration Workflow



Above is an architectural diagram of our front and back-end technologies. For our front-end, we mainly use Angular for front-end work with D3.js for our data visualization. For our back-end, we have a Django REST framework that works with our mining script and database to pull data for use at the front-end. Our Technology Integration Workflow figure above highlights our technologies and requirements as well.

4. Project Requirements

In order to obtain the proper domain-level requirements and continuously refine them into much more specific requirements, our team has been meeting with Dr. Steinmacher once per week for the past three months. During the first month of meetings, the main focus was understanding the problem and learning from Dr. Steinmacher about the hole that exists in Open Source Software that is “Newcomer Support”. The second month of meetings consisted of many idea sharing and brainstorming sessions to come up with the key features as well as “nice-to-have” features, allowing us to formalize them into true requirements.

In our last month of meetings, we proposed a list of use cases to Dr. Steinmacher that included potential requirements that mostly fit the scope of our capstone project and asked him to return the list with a check by all use cases that he wanted to be included in the final product. After collecting the checked list of requirements, we then asked Dr. Steinmacher to give a priority rating for each requirement to allow us to focus our initial effort on the most important features in order to not only create a useful prototype by the end of the semester but also to know where our focus should be going into the true implementation phase.

The outcome of the past three months with our client can be summed up into four domain-level requirements. The first domain-level requirement is that the application must be able to retrieve data from GitHub. The second is that the data that has been retrieved from GitHub must be stored, by our application, for later use. The third domain-level requirement is that the application must be able to perform some analysis on the data and generate easily understandable interactive graphics to the user. The last is that the application must be hosted on a linux-based web server in the form of an attractive and intuitive web-application.

These four domain-level requirements are very broad and have a lot more to them than meets the eye. In the following section they will be broken down into smaller, more specific pieces in the form of functional, non-functional, and environmental requirements.

4.1. Functional Requirements

The first set of requirements are the heart of our product, the functional requirements. Functional requirements basically outline exactly what the main features the application must have in order to meet the expectations of our client from an application functionality stand-point. We have broken down the many features requested by our client into the following nine functional requirements:

FR-1: Retrieve data from GitHub

FR-2: Repository search

FR-3: Compare repositories

FR-4: Request for a repository

FR-5: Administrators accept/decline requests

FR-6: Create interactive graphics

FR-7: Manage repository database

FR-8: Grant admin rights to users

FR-9: Configurable database updates

We have come up with these requirements in a specific nature in order to establish a strong understanding between our team and Dr. Steinmacher about what will be expected in the final product.

<u>FR-1: Retrieve data from GitHub</u>

Being able to mine data from GitHub is the heart of our whole operation. Mining data will bring in all the information we need to create useful interactive graphics and other important data points that will help our users choose the right repository to start their open source adventure on the right foot. In order for our application to mine data from GitHub, we will create a Python script that goes to the specified repository on GitHub, retrieves important data that is hand chosen by our client, and then stored in our database to be used later for analysis. This will all be happening behind the scenes of our web-based application and return the analyzed data to the user like magic.

FR-2: Repository Search

The user of our application having the ability to search for a specific repository is very important to the functionality of our product. In order for the user to find the data they are looking for quickly and easily, we will implement a searching function that includes various filters. This will allow the search to be narrowed down to find not only a specific repository, but also give a user that doesn't have a specific repo in mind the opportunity to come across different options that meet some criteria in which they set in the filters. The search results will appear below the search bar and filters will be in the form of check boxes on the left-hand side of the screen as well as more filters in an advanced search drop-down menu. Below is a mock-up showing the intended functionality of search in a visual representation.

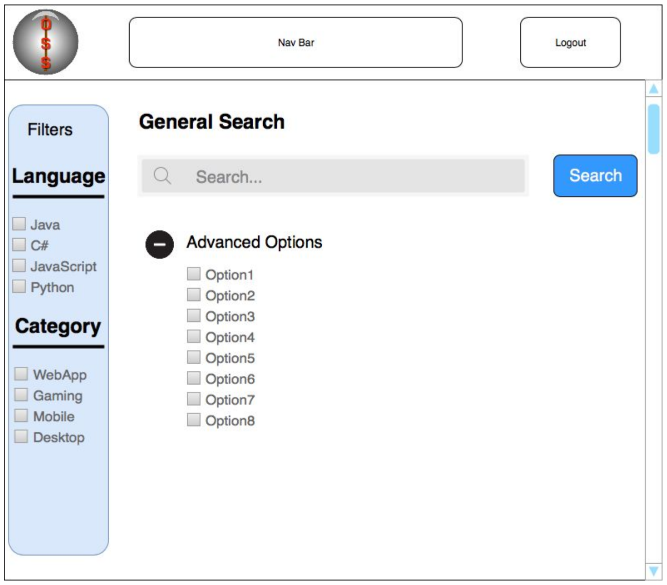


Figure 1: General Search Mock-up.

FR-3: Compare Repositories

A great way for a user to figure out what repository would be a good fit for their first contribution to open source, they should be able to conveniently view multiple repos side-by-side for quick and easy comparison. In order to keep the comparison page clean and easy to read, there will be a limit of 3 repositories per comparison. The repos will be set up side-by-side with their name at the top followed by the interactive graphics that belong to each repo ordered in columns below, and at the very bottom of the page will be a table where the repositories are compared based off of some textual data points such as newcomer retention rates, number of contributors, and whether or not the repo has a CONTRIBUTING.md, among others. Below is an example of how this comparison page may look.

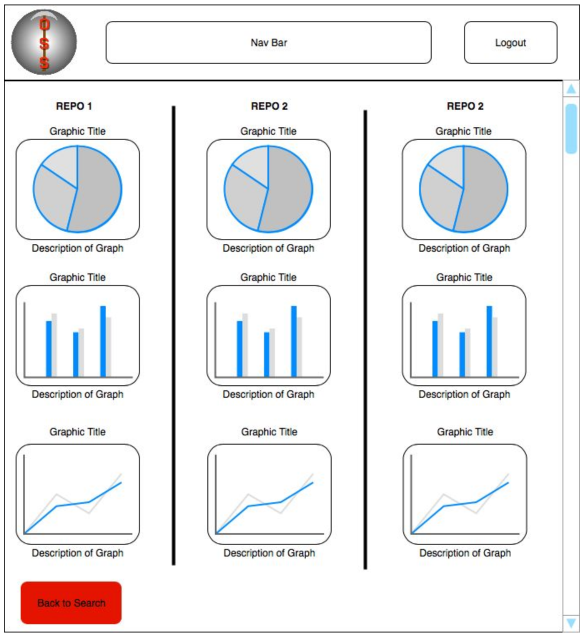
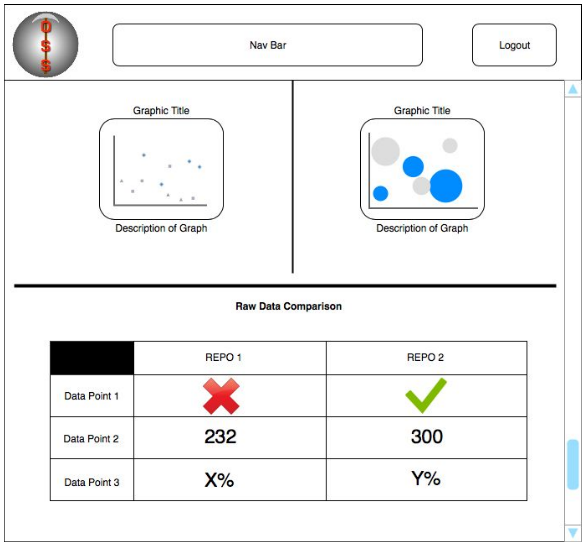


Figure 2: Repo Comparison Top of Page.



	REPO 1	REPO 2
Data Point 1	✗	✓
Data Point 2	232	300
Data Point 3	X%	Y%

Figure 3: Repo Comparison Bottom of Page.

FR-4: Request for a Repository

In the event that a repository that the user wants to view our data analysis on is not already in our database, they will have the ability to request that our application go out and mine the data from GitHub for them. This is a process that will allow our users to get any information they want as long as the requested repository is appropriate and accepted. Once the requested repository is mined, the user will then have access to interactive graphics and textual data points that our application generates for any repository in our database. Our application will have a request button that opens a small window where the user enters the repository name and link to the repository’s GitHub page and then clicks a submit button that sends the request to the admin.

FR-5: Administrators Accept/Decline Requests

As mentioned in the previous functional requirement description, the users of the application must send a request to the admin of our application in order to have a specific repository mined if it is not already in our database and available for analysis. For the specified repository to be mined from GitHub, an admin must view the request and accept it to be mined and added to the database. There are multiple reasons for this, the most important being that repositories containing inappropriate content or malicious content should not be able to be added to the database without any screening of them at all. The other main reason is to avoid bots that may try to overload our application and database with a constant stream of data mining and storing. The users with admin access will be able to navigate to a page which displays a full list of requests where they can view the repository name, click a link that takes them to the repository page on GitHub, view the name of the user requesting, and click a button to either accept or decline the request. The following image is a preview of what we feel this page may look like in the application as far as layout goes.

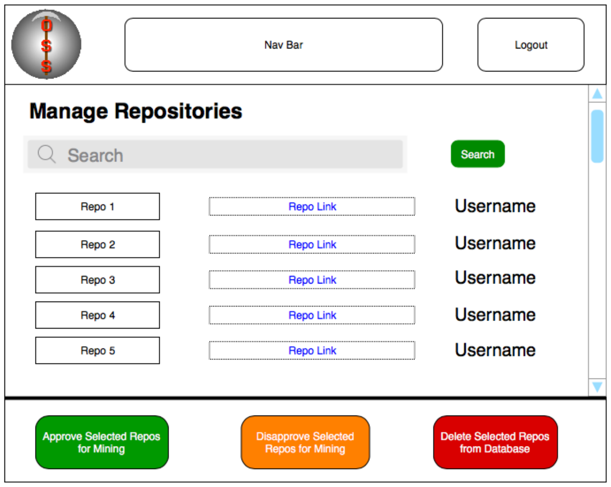


Figure 4: Admin Accept/Decline Mining Page.

FR-6: Create interactive graphics

In order for our data that is mined from GitHub to be useful, we must display it to the users in an easy to understand fashion. The number one way that our client wants this data to be shown to the users is in the form of interactive graphics. These interactive graphics will be graphs that display certain information that should be helpful to the user in deciding whether or not this repository is right for them. These graphs will have the ability to be adjusted by hovering or clicking the mouse on specific points in the graph to show a specific moment or data point in the graph. The graphics will be displayed on a page that is reached by double clicking a repo name in the search results of our application. They will be laid out in rows of three in order to ensure that there is no cluttering of the screen and to keep the graphics large enough to read as they are displayed.

FR-7: Manage repository database

In order to keep our database clean and exactly what the client wants, admin users will have the ability to manage the repositories in the database from an admin management page. On this page will be a list of all the repositories in the database along with buttons to “Delete Repository,” “Pause Mining” and “Resume Mining.” This page will allow the admin to delete a repository from the database, pause the mining of a repository in progress, and resume paused mining of a repository. We plan on potentially combining this functionality on the same page as the admin accepting or declining mining requests, this being the case, please refer to Figure 4 above in the FR-5 section to view a visual representation of our current prototype of how this page may look in the product.

FR-8: Grant admin rights to users

As the size of the application itself as well as the user-base of the application increases, there will surely need to be more than the initial one admin that there will be at launch. Dr. Steinmacher being the only initial administrator, he would like to grant admin access to other trusted users to help in the management of the application. This will not only free up Dr. Steinmacher to handle different activities but it will also allow users to get better response times from admin on mining requests and any other contact to admin.

FR-9: Configurable database updates

A major shortcoming of FLOSScoach, the current application created by Dr. Steinmacher and his colleagues for helping newcomers to open source, is that it must be manually updated. An important element of our solution is the ability to automate the process of updating the database of repositories with the latest and greatest data from GitHub. This will allow our users to get analysis on very current data from the repository they are looking into. The admin of the application will have access to a script in which automatically updates the data for all repositories in the database. Configurable options for updates will be setting specific days of the week, a specific time of day, and the frequency (daily, weekly, monthly, etc…) in which the updates occur.

4.2. Non-Functional Requirements

As you can see from the section above, functional requirements are all about "what" the application will be doing. Non-Functional requirements on the other hand are all about "how well" the application does those things. For our application, the non-functional requirements are all about response time to the user with their desired data. As far as our client is concerned, there is really only one non-functional requirement and that is that if the repo that the user wants to view analysis of is already in our database, it should take less that one minute for them to be returned to with all analysis and graphics generated. If the repo in not already in the database and the user has to submit a request, due to the nature of this response being up to the admin getting to the request rather than being an application performance issue, this is not necessarily a non-functional requirement of the application that needs to be addressed.

4.3. Environmental Requirements

Our environmental requirements are a similarly short list to the non-functional requirements. According to Dr. Steinmacher, this application must be optimized for desktop and laptop browser use, making this our only main environmental requirement. Our client is not particularly concerned with mobile use of the application at this time, therefore, mobile implementation is not a requirement but it has been stated that the application should function on mobile, just not specifically optimized for mobile. The only other environmental requirement specified by our client is that the application be hosted on a linux based web-server which is satisfied through our use of the DigitalOcean web-hosting service. Now that we have our requirements sorted out, it’s time to talk about some potential risks that may evolve into actual issues with our application in the future if neglected.

5. Potential Risks

Throughout the requirements acquisition phase, the team has identified and considered four main risks. The first risk involves displaying information in a way that does not present an opinion of a repository, while the remaining three assess potential risks associated with GitHub itself.

5.1 Presenting Facts, Not Opinions

From the very beginning of this project, the team spoke to our client about how information will be presented to the end user, and what this information would look like. After an iterative process, we have determined that it is in the team’s best interest to provide soley facts about a repository, and to stay away from presenting opinions. The reason for this is quite simple, as rating a repository as good or bad, five stars or 1 star, or any other variation of this kind could actually prevent newcomers from attempting to contribute to a repository that would be a great fit for them. It should also be known that what one user might find newcomer friendly is not necessarily going to be the same for another, therefore it is best for the team to avoid providing that type of analysis altogether.

As far as the facts that we will be displaying to our users is concerned, the team has met with the sponsor on numerous occasions to determine what facts will be summarized in our graphical representations of the data we have collected. Examples of this data include newcomer retention rates, and the length of time that a newcomer typically waits for their pull request to get accepted. In doing this, we hope to avoid labeling any repository in a negative away, and allow users to make an informed decision for themselves as to whether they would like to contribute to a given repository.

5.2 Maintaining Correct Information

Considering GitHub contains an ever-growing number of projects and innovators, there is always the possibility that a project owner may wish to change the name of their repository to better fit what it is that they are working on. When this happens, GitHub updates their API to reflect the new project name, and if the team were to search for it, we would be returned an error. This represents the risk of having out-of-date information being stored in our repository database, regardless of how minimally people do change their project names. In order to mitigate the risk of having incorrect information in our database, we aim to produce a robust administrator portal that will allow an admin to remove all instances of the incorrect repository name throughout our database so the new, correct information may be collected and stored. By having an up-to-date database, we reduce the possibility of presenting incorrect information to the end-user, and build trust with the community we hope will use our tool.

5.3 Changes to the API

Very recently, the Microsoft Corporation acquired GitHub, and considering the transition is still occurring, there is always a possibility that Microsoft may make changes to the GitHub API to better suit their business needs. In the event that this were to happen, the team would have no other choice than to read up on what is different, and adjusting the way that we collect information from the API appropriately. By being proactive and keeping up to date with what Microsoft plans to do with GitHub, the team may avoid potentially severe problems when it comes to mining information, and this will keep us on track to produce a successful product.

5.4 Errors During Data Collection

The final risk that we have assessed is the potential for anything to go wrong with the collection of the data itself as it is occurring. There are so many variables to account for when mining information from an API, that something such as maxing out the 5,000 requests per hour we are granted, or an internal server error on GitHub's end may happen at any time. In order to mitigate this risk, the team will implement a try-catch system that will ensure that the mining tool does not crash with an absurd amount of errors, and may continue as scheduled if possible. We also aim to notify an administrator via some kind of log so manual investigation may take place if needed, and provide the admin with the ability to pause or cancel the collection of a repository as they see fit.

With potential risks notified and assessed, the team needs to have a solid development plan in place in order to understand the timeline of the project through its entirety.

6. Project Plan

Our overall plan for implementation is iterative development for features over the span of the Spring 2019 semester. Below is a Gantt chart of our five milestones that we plan to complete for our project.

6.1 Create Mining Scripts

Our mining scripts are the backbone of our project. If these scripts are not finished and working as intended, than our whole project will fail. The mining scripts are essential for the whole project as the scripts are responsible for pulling data, in turn, must be stored in a database and visualized as interactive graphics for the user to explore. Over our plan for development, our scripts will be constantly tested to see if they can be improved upon.

6.2 Repository Database

When data from GitHub is mined, the data will be stored in a database where users can access the information. The database will be managed by an administrator, whom will authorize the actions of repositories stored, will have commands to interact with how a repository will be mined. The database functionality is our second milestone, so that we can prepare an adequate environment for the data to be mined.

6.3 Hosting The Website

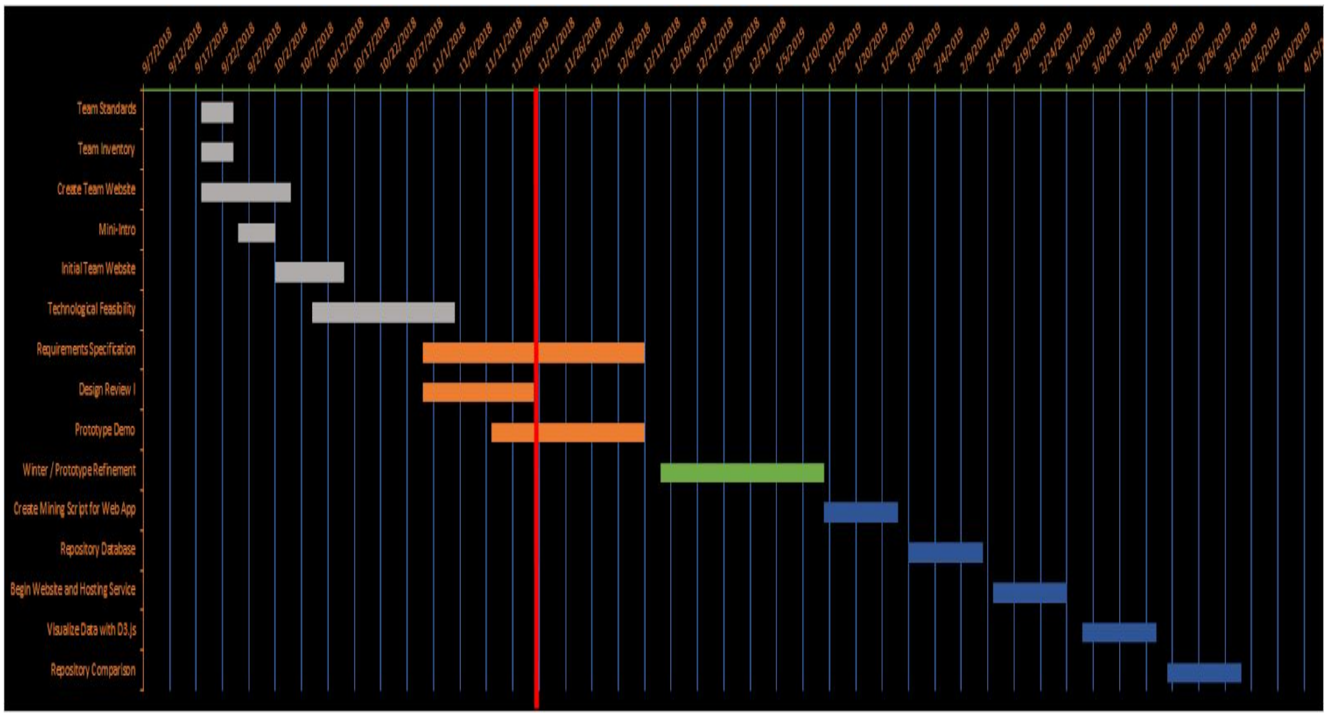
Because our web application will be performing a load of functionality from the mining scripts, it is important for us to have a powerful hosting service to perform all necessary functions. We believe that our web app will take on substantial traffic, making the overall process of the web app quite slow. With the use of the mining scripts, the web app will also take a hit on speed as well. These two factors are why we need a powerful hosting service to perform at the web app's best. We will begin hosting the web app once our mining scripts are finished and our database setup.

6.4 Data Visualization

One of the main components of our web app is data visualization. We will be using the pulled data from the scripts and database to provide the user with interactive graphics for them to explore. Data visualization will assist the user in finding a project to contribute to, as the user can interact with the visuals to see data points at a specific time. We plan to create our data visualizations after our web is hosted, this is because our framework for visualizing data must be done on a website.

6.5 Repository Comparison

Our last milestone is a repository comparison, where users can compare and contrast up to three different repositories to see the statistics of each selection. This is our last milestone because we need all the previous functionality to work properly before we can implement this milestone. After this milestone, we are open to further development and refine past milestones.



Above is a Gantt chart of the entire school year. Tasks that are complete are gray, orange is in progress, green is Winter, and blue are our five milestones described above. As you can see from the chart, the milestones are evenly separated, this is because we want to complete our first two milestones, data mining scripts and database management, so that our back-end technologies are working efficiently. After these two milestones, we will begin building our web application starting with hosting the product and the visualizations for data. Keep in mind that our schedule is subject to change as time goes on.

7. Conclusion

Open-source software is not the easiest field to be involved in, with our product, we aim to bring the power to the newcomers so that they can contribute to their desired project. Our web application will be able to mine data straight from GitHub, store the data in a database for future use, visualize that data in the form of interactive graphics, and compare repositories so that the user can filter out other options. In this document, we have narrowed down our requirements so that we can begin working on our minimum viable product with our five milestones stated in our “Project Plans” section. Through gathering requirements from our sponsor, Dr. Igor Steinmacher, we are confident that we can deliver a product that not only assists our client with his research, but to bring the power back to the users so that they can learn and find open-source projects to contribute to.