Deep Insight And Analytics Into the Git Repositories

Aaron Salvo
Computer Science
University of Wisconsin-Parkside
Kenosha, WI 53144
salvo002@rangers.uwp.edu

Babita Thapa
Computer Science
University of Wisconsin-Parkside
Kenosha, WI 53144
thapa002@rangers.uwp.edu

Dr. Zaid Altahat
Computer Science
University of Wisconsin-Parkside
Kenosha, WI 53144
altahat@rangers.uwp.edu

Ryan Groves
Computer Science
University of Wisconsin-Parkside
Kenosha, WI 53144
grove021@rangers.uwp.edu

Ian Gilbert
Computer Science
University of Wisconsin-Parkside
Kenosha, WI 53144
gilbe016@rangers.uwp.edu

Abstract

Projects of any magnitude, when worked on by a team, are stored on remote repositories such as Git. Each of these repositories contains vast information about each file committed and its contributors. It can be beneficial to collect and study this information, whether by the development team or by an audit group in the future. Obtaining that information and deciphering it from the command line is unpractical. It requires a polling approach where the user must proactively look for something specific in the code history. Users could easily miss important events or alerts about the code if they manually check for events and they will certainly miss alerts that they don't know to look for. Git allowed us to check which areas of the code are changing the most. This is an indication of a possible refactoring opportunity.

1 Introduction

To solve the issue of studying the development of projects, we created the application Git Repositories Analytics. This application receives a user-chosen git repository and visualizes desired information with clear, understandable graphs. These graphs, while designed to be readable for anyone, are particularly useful for code consumers such as project managers and the developers themselves to ensure high software quality. For instance, it could allow one to quickly analyze what files are changed the most and act accordingly (perhaps the file has many bugs, or its design prevents seamless maintainability). Beyond this, it can also help managers find what files a contributor works on the most, inspect the history of a single file or contributor, or even learn and compare contributor commit styles. The current status of the application provides these services through a web application. Flask [1] was used to build the the web application and Python is used on the back-end to run the analyses of the user-chosen git repository and generate JSON files. These JSON files are then used to create graphs with D3.js [2]. The application currently lacks an automatic alert service, but it is something planned for the next phase of the application.

2 Related Works

There are other projects in development similar to this one, however, these projects often times provide raw statistics or do not have a clean interface to interact with [6, 7]. That is where our project separates itself from the rest. Our project consists of a nice, clean, easy-to-use interface with no command-line arguments to remember. The application has been divided into separate tabs, or windows, to give clear lines of separation between its abilities and the data provided. Any data the user wishes to view is presented in clean, easy to understand graphs. This limits our ability to convey all information at once, but we believe that the easier to use format will be more beneficial.

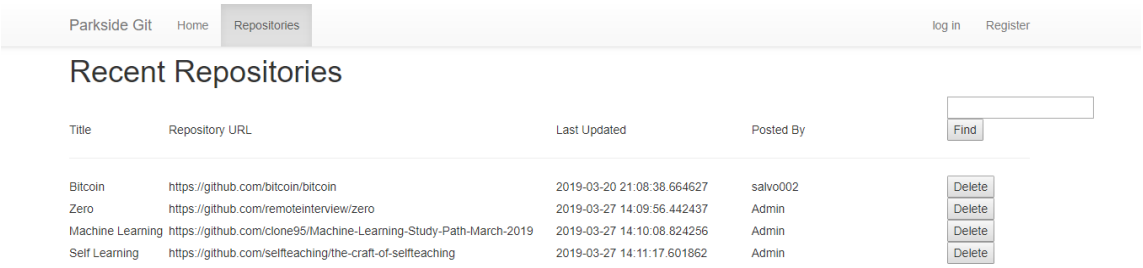

Figure 1: Web Interface

3 Method

We followed the agile development methodologies throughout the project. Specifically Scrum and JIRA were used for the tasks and project management. GitLab was used as the code repository and Microsoft team was used as a technology to communicate within the team and share file. The project can be categorized in three different sections:

3.1 Extracting the Data

The data is a key factor for the whole project. For this project, Python was used on the back-end to extract the data from the repositories. The repositories are the one user chooses providing with the link for Git repositories. The Python script then analyzes the chosen repository using the Python library gitPython [3]. The library allows Python to access the server's terminal directly for all git related commands.

The script’s first step is to take the user given URL and process it with gitPython. Depending on whether the repository exists on the machine, the repository is either cloned or a fresh copy is pulled. Regardless of which command is performed, the script also creates a "repo object;" this object is used by gitPython to execute all future git commands on that repository. This repo object is saved and passed to the analysis functions.

The next step is to perform the user chosen analysis function. To do this, the script passes in the given repo object and a handful of user defined parameters to the corresponding function. Finally, the function will carry out its analysis by calling git log commands through gitPython. Each function spits out a human readable JSON file that is fit to graph using d3.

3.2 Visualizing the Data

Another important process was visualizing the data. D3.js was used for a nice and clear representation of the data. At first, experiments were done with various file type like csv and tsv. Then a better solution was found to work efficiently in parsing the data and visualization using JSON file. Atom and WebStorm were used as IDE to write it

The application uses the JSON file to represent the data in a quantitative way. Currently, the *File History* and the *Author History* graph options are used for the visualization. The File History graph is a histogram which is also interactive to the user and changes the color based on any touch in the bar. The X-axis is the name of the file and the Y- axis is based on the number of commits ranging from 0 to the maximum number of commits. And the Author History Bar has a little complex code. Written in JavaScript using D3.js, it implements the bubble line graph based on the date and time. The date on the X-axis and the time on the Y-axis is parsed from the JSON data field and using the substring method. The Bubble also coincides with the point in the graph. The size of the bubble is based on the number of files changed. That means if there is more file changed in the commit, the bubble will appear larger. Also, the point in the graph displays all other details parsed from the JSON file. The graph uses the dimple library available online from dimplejs.org. There are plenty of visualizations in D3 library. These can be used for future graphs and representation based on any contents.

3.3 Creating the Web

Final phase of our project was to create a web application putting everything together. Flask was used to build the web application in the front end. We have the application prompt the user for the link to the Git repositories. The page also allows the option to choose the graph type. Based on the user preference, the data is parsed from the repositories and the graph would be created representing the aspects of the data. The Flask framework integrates smoothly with the Python scripts we wrote. Since Python has strong support for git repositories and Flask is written in Python, Flask was chosen.

4 Results

Utilizing the complete application, we can generate graphs for researching purposes, useful for any team of developers. As an example, let's inspect the bitcoin repository on GitHub [4]. Our first task will be to choose a function via the "Graph Type" field and use the "Total Commit Statistics" function:

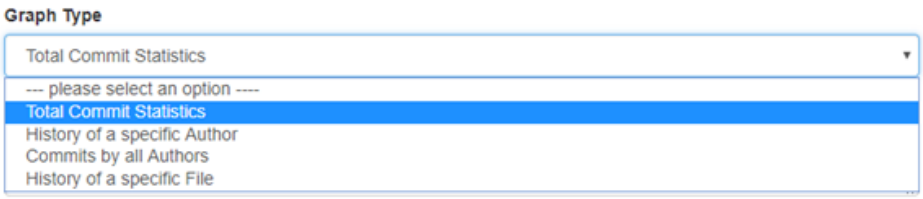

Figure 2: Total Commit Statistics

Next, we need to input the bitcoin GitHub URL:

Figure 3: Inputting GitHub URL

The generated graph shows a list of files (organized from most committed to least) with the number of commits per file shown. For scale, the average number of commits per repository is displayed with a red line:

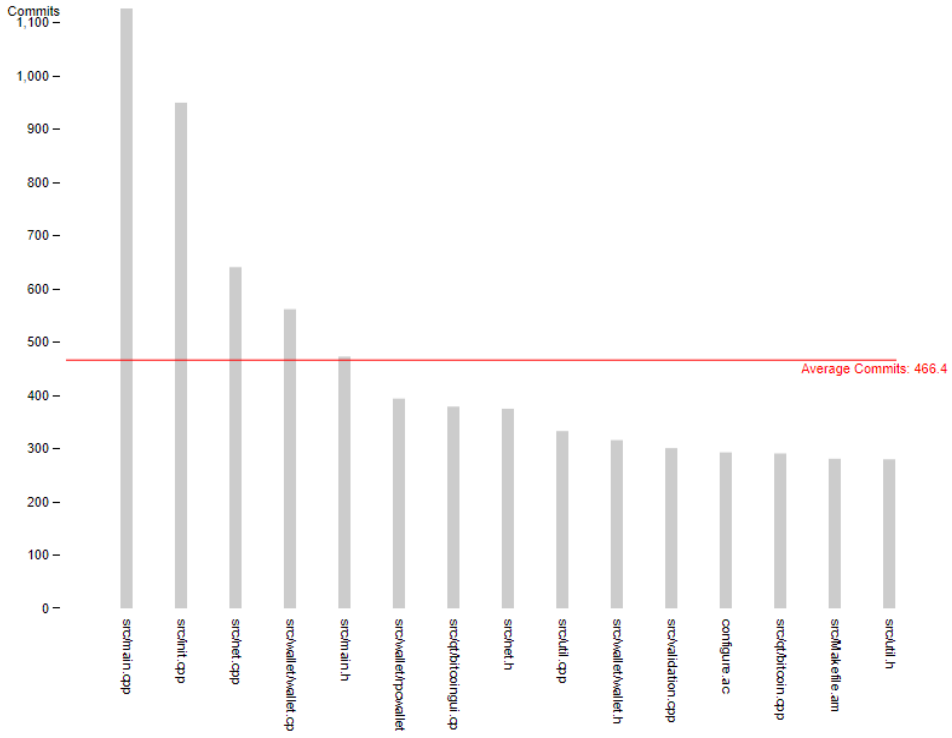

Figure 4: Commit graph for Bitcoin

For more information, the user can hover over a file's bar to see the filename, the file's exact commit count, the percentage of all commits dedicated to the file, and the size of the file in bytes:

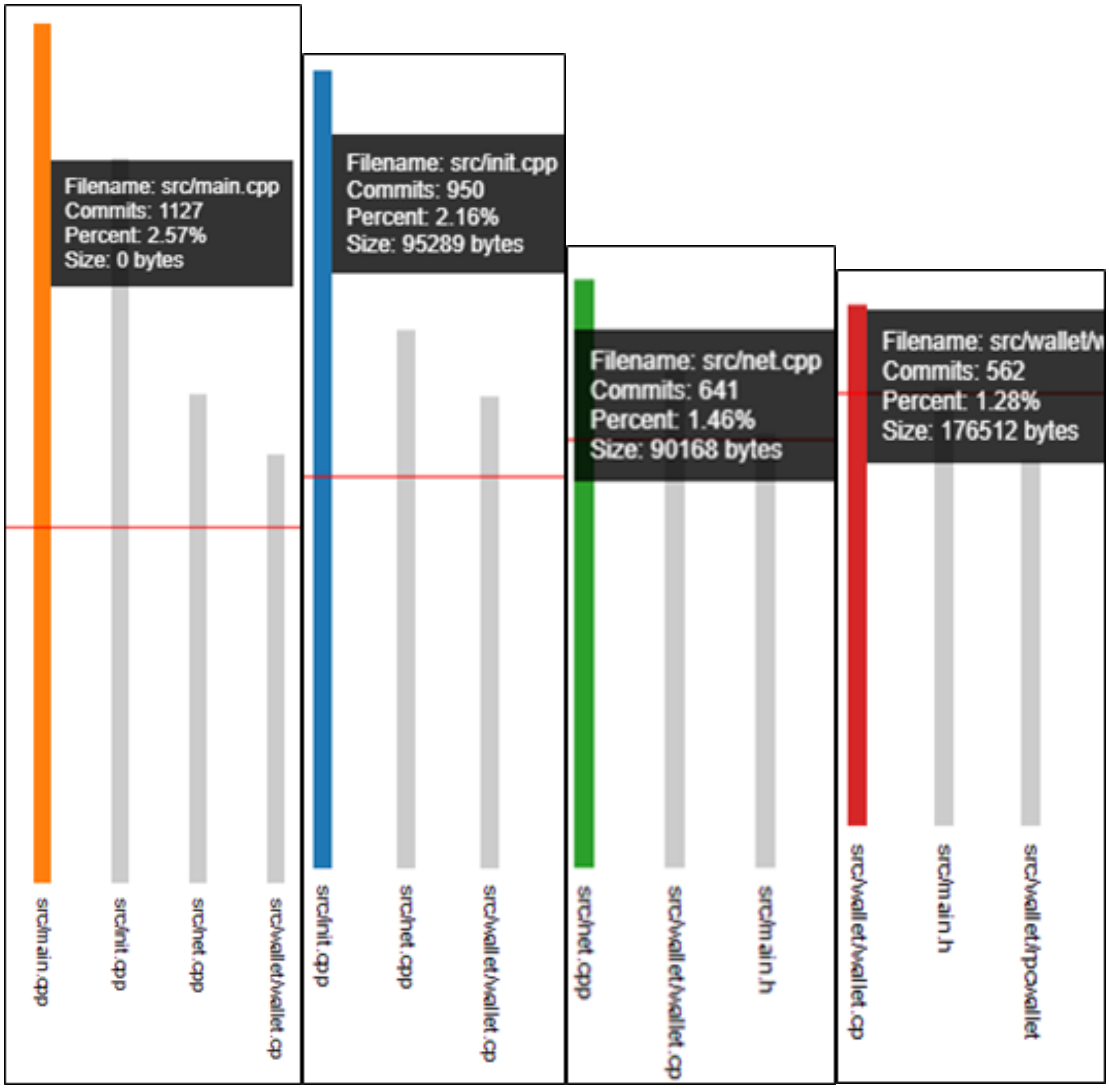

Figure 5: Hovering over a bar gives more information

Already we can learn a few things from this graph. For one, we know the top fifteen most changed files at a glance (fig. 3). Two, we can view the size of each file and try to notice oddities. Generally, larger files will have more commits than smaller ones, but we can see that isn't always the case (see fig. 4.4 vs. fig. 4.2/4.3). This could invite further research, especially if one found a very small file that was being changed constantly; this could imply a volatile, buggy file. The third piece of information is that we can tell main.cpp has been removed due to its size of 0 (fig. 4.1).

Using the same repository, we can parse the authors to see who is most active:

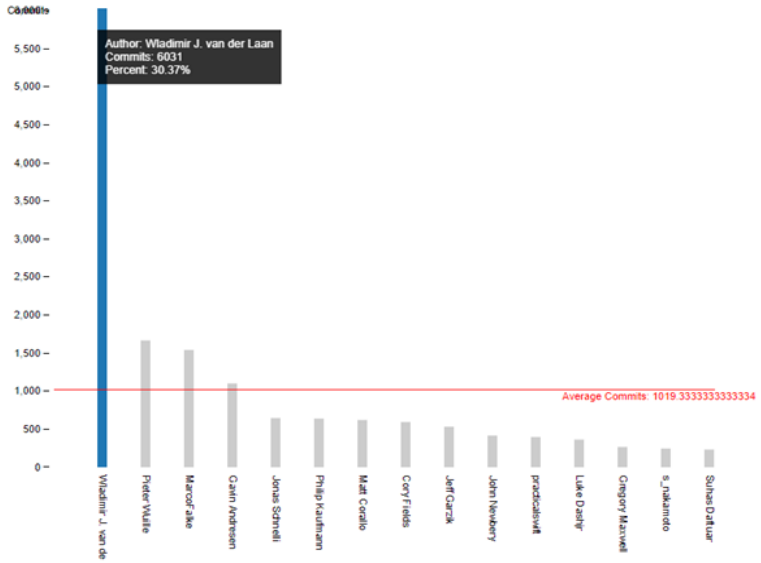

Figure 6.1Author List of 15

Something apparently obvious is the number one contributor is *far* more active than anybody else. While no other contributor has exceeded 2000 commits, Wladimir J. van der Laan has more than 6000 to his name. If we expand our search to the top 200 authors, the difference becomes even more obvious:

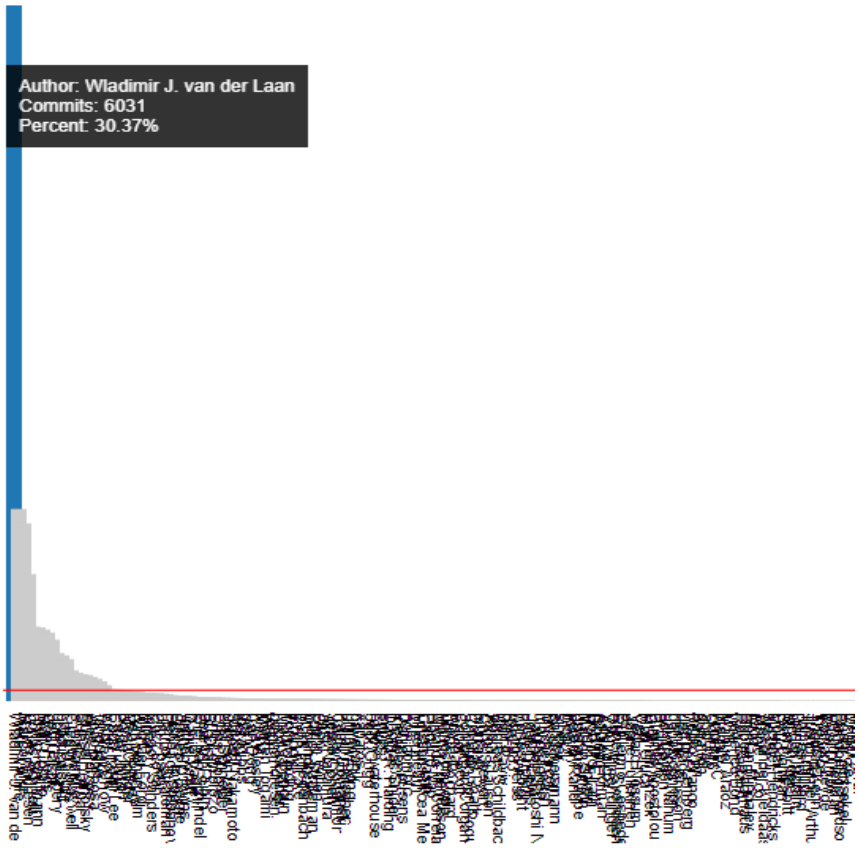

Figure 6.2: Author List of 200

It should be noted that this application does not play well with big data. When the number of authors is increased to such a high amount, the graphs become increasingly confusing to read. As another example of this, let's use the application to inspect the complete commit history of the main author, Wladimir J. van der Laan:

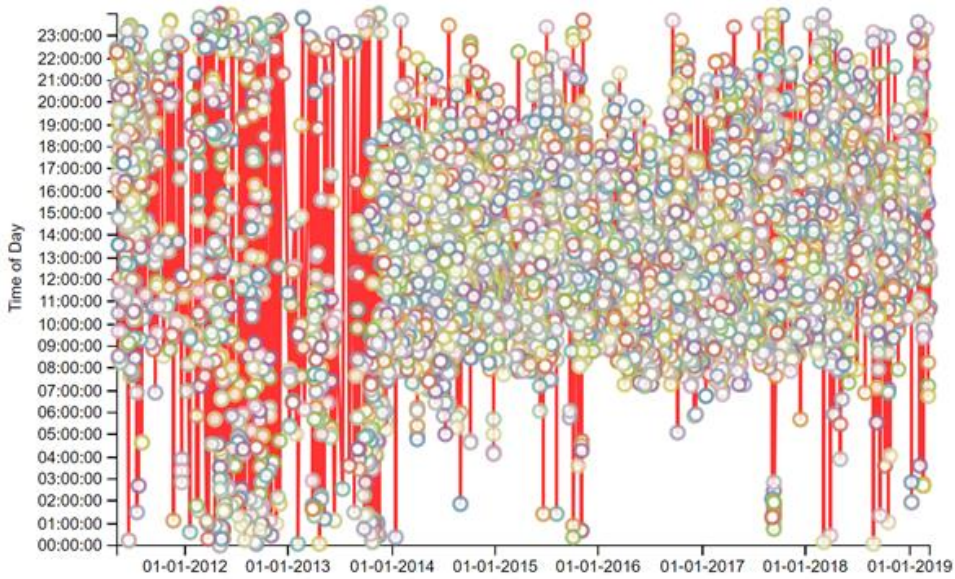

Figure 7: Too Much Data

This is practically unreadable for most. Future updates can help remedy this by allowing the user to zoom or scroll through the graph. Currently, it is best to instead limit the search to something much more manageable. In this case, let's view Wladimir J. van der Laan's commit history during the past week:

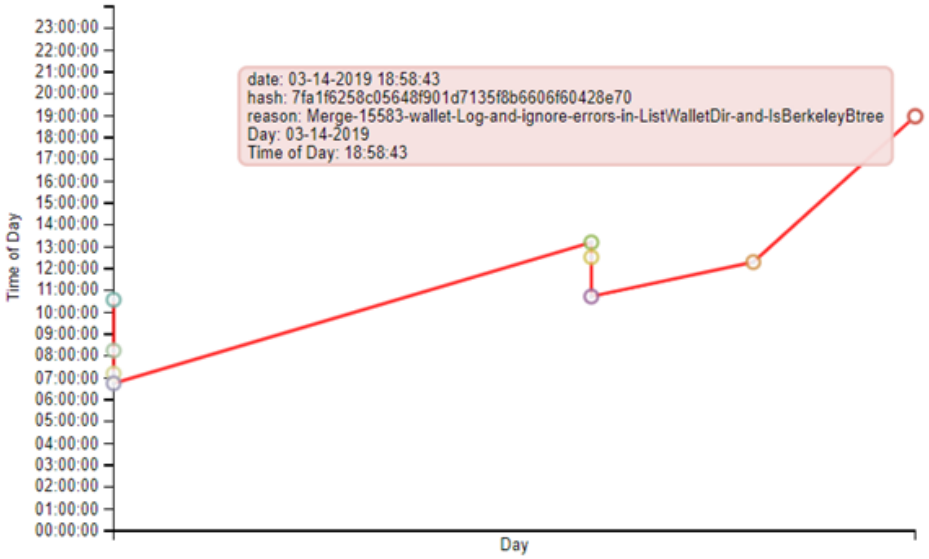

Figure 8: Last week's commits

Now we have a digestible timeline. We can tell the date and time with each commit's x/y coordinates and hovering over the commit gives more precise information such as the exact

date and times, the commit hash, and the reason for the commit. This graph also works for file's as well, allowing users to see how a file has been changed over time.

5 Future Work

Future work will consist of adding additional graphs so users may visualize and present data in a way that accentuates the data that is being explored. This includes different graph types as well as radial options for manipulating the data that has already been stored. This will also include an option for storing and saving this data for later use, as well as a search function for sorting through the stored repos. An alert system will be built which will require integration with a git repository. Each time a commit is made, our application kicks in and run the analysis and generate any alerts.

6 Conclusion

Results summary: The application's graphs are helpful for identifying active contributors, inspecting what files require the most changes, and researching an author or file's history over a set period. The data obtained through use of the application will be useful in identifying trends and identifying abnormalities over the lifetime of a repository.

7 References

[1] http://flask.pocoo.org

[2] Bostock, Mike. "Data-Driven Documents." *D3.Js*, d3js.org/.

[3] https://gitpython.readthedocs.io/en/stable/

[4] Bitcoin. "Bitcoin/Bitcoin." *GitHub*, 25 Mar. 2019, github.com/bitcoin/bitcoin.

[5] https://d3js.org

[6] "Mining Software Engineering Data from GitHub"; Georgios Gousios, Diomidis Spinellis; : IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C); 2017

[7] "A Dataset of the Activity of the Git Super-repository of Linux in 2012"; Daniel M. German, Bram Adams, Ahmed E. Hassan; 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories; 2015