Visualizing Programming Languages Used at RPI

Peter Ryder

***Abstract*—What programming languages are used most at RPI? Are there languages used during certain times of a students career at RPI? What should new students expect to learn while at RPI and what languages might they learn before entering RPI to help better prepare them? This paper serves to describe a visualization tool which I created to help current RPI students or incoming students determine which languages are used most at RPI as a Computer Science student. It also describes difficulties I had in gathering good clean data and the process of determining what visualization is best for the data. As you will see, there were several challenges in gathering the data I needed and my vision for what the visualization should be changed throughout the process of creating this tool. What I came up with however is both functional and I believe useful.**

I. Introduction and Motivation

THE computer science department at RPI teaches students many different programming languages to help them learn different programming paradigms and to diversify their education. After having taken Interactive Visualization, I wanted to visualize the different languages I have used as a computer science student for my own curiosity, but this tool can also help incoming Freshman see what their four years might have in store for them. Based on data collected from my four years, this visualization makes it clear what languages I used most as a Computer Science Freshman, Sophomore and beyond.

II. Related Work

I took inspiration from the New York Times article which was an interactive streamgraph depicting Box Office receipts between 1986 and 2008. I thought this visualization was visually pleasing, informative, and unique. The flowing of the lines created by the streamgraph combined with the different colors all blending together towards the X axis of the visualization is very striking to the eye. This visualization would not work without the interactive portion of it however. The user can hover their mouse over the different data points of the streamgraph to see the movie title and details about its box office performance. After switching to D3 for my visualization toolkit, I took inspiration from Mike Bostock's Stacked Bar Chart example from the D3 website. This bar chart is visually appealing like the streamgraph, but I believe it is also easier to read. It contains dates along the X axis like the streamgraph and stacks the data per date up the Y axis with colors separating the data points. In order to create the tool tips when hovering the mouse over a data point, I referenced Justin Palmer's Tool Tip example from the D3 website.

III. Design Evolution

The visualization part of this project was initially supposed to be a streamgraph showing all the programming languages I have used at RPI and when I used them similar to the New York Times visualization. This proved very difficult however. After creating several debug graphs in Excel, I created a streamgraph using an Excel add-on from the Microsoft Research team. This add-on is very good at creating simple streamgraphs from input data but it lacks customizability. I could not change the colors save from a handful of built in ones, and I could not change the axis titles. Because of the limitations of the streamgraph tool and from suggestions from classmates, I switched to using D3 for my visualization toolkit. Within D3, I used the Stacked Bar Chart which is able to show the individual data points per time frame. I then combined several examples from the D3 site to create an interactive bar chart visualization which still shows the information the streamgraph would have.

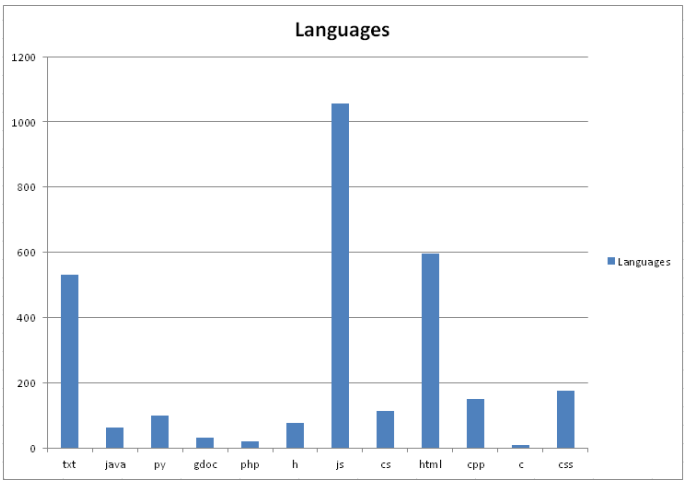

Fig. 1: Result of file whitelist system.

IV. Class Feedback

The class feedback was extremely helpful in creating my final visualization. During that class, I showed classmates the initial debugging graphs and the streamgraph seen in Figure 3. Their responses were that the bar graphs were easier to read, but the streamgraph looked better. They suggested I change the colors so that the different programming languages would be easier to read on the streamgraph. Several people suggested I switch to D3 for my visualization toolkit as I could then

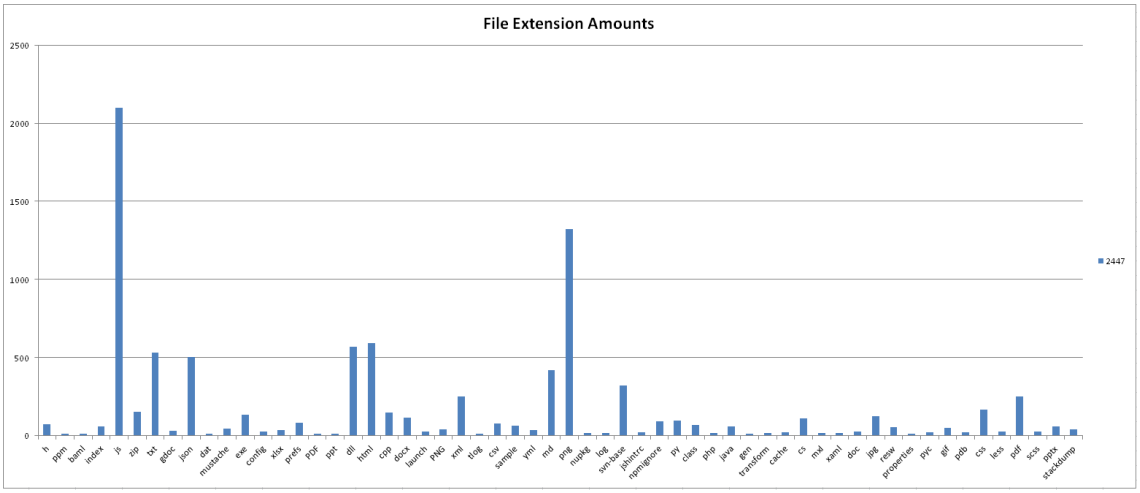

Fig. 2: First debug graph created in Excel visualizing all file types and amounts

create a more interactive experience than just looking at an Excel spreadsheet. These comments are what got me to switch to using D3 as was discussed in the Design Evolution.

V. Implementation Details

I created Python scripts to recurse through my school directory and determine all the file types and amount of file types. With this script I also collected last date modified, date created, and amount of lines within the files. The script outputs a single CSV file which, for every month, contains the file types and amount of files. Initially for debugging purposes, I used Microsoft Excel to create bar graphs of how many total files of each file type I had contained within my school directory. Three of them can be seen in Figures 1, 2 and 5 After viewing the first debug graph, I made some changes to the script. I implemented a file whitelist to help remove all files which are not programming languages as there were many files contained in the data which I did not want. Secondly, I created a directory blacklist system which would allow the scripts to ignore directories. This was important as I have many node projects which download thousands of JavaScript files. The first debugging graph I created seen in Figure 2 showed over 2000 JavaScript files, and this is not representative of the amount of JavaScript files I have written at RPI. Using the directory blacklist system, I was able to ignore all of the downloaded node package files. After doing this basic filtering, I ignored all JavaScript files which are less than six lines. This is because many downloaded or auto generated JavaScript files are minified. Minified JavaScript files are reduced to about six lines including comments at the top of the file to help reduce bandwidth usage. Therefore, if the scripts encounter any JavaScript files which are this length, they can be ignored because they are most likely not user written. As can be seen in Figure 5, this had a dramatic effect on reducing the garbage data points. I was left with C++ being the most popular object oriented programming language followed by C#, Python, Java and C. This is what I expected my results to look like before starting the project.

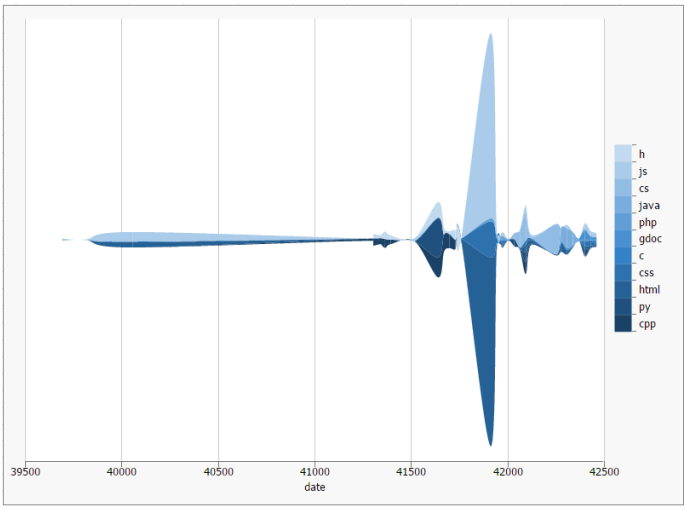

Fig. 3: The first streamgraph created in Excel using clean data

VI. Results

After creating several different debug graphs using Excel, I managed to get the data which I was expecting before starting the project. The web development languages are not representative of how long I spent using them, however this could be attributed to my method of determining what is a ”most used” language. After the web development languages, C++ was the language with the most amount of files I wrote at RPI. This was expected as it is used throughout all four years at RPI from Data Structures to Algorithms to Cryptography. The language with the second most files was C#. This is due to the fact that I spent many years in RCOS, the Rensselaer Center for Open Source Software. During these years I spent a great deal of time using C# to create a project. This might be unexpected for some people, but I believe to be a critical data point. Even though C# is not directly taught at RPI for any specific class, I used it heavily because I wanted to. I was a part of RCOS because I wanted to learn more about

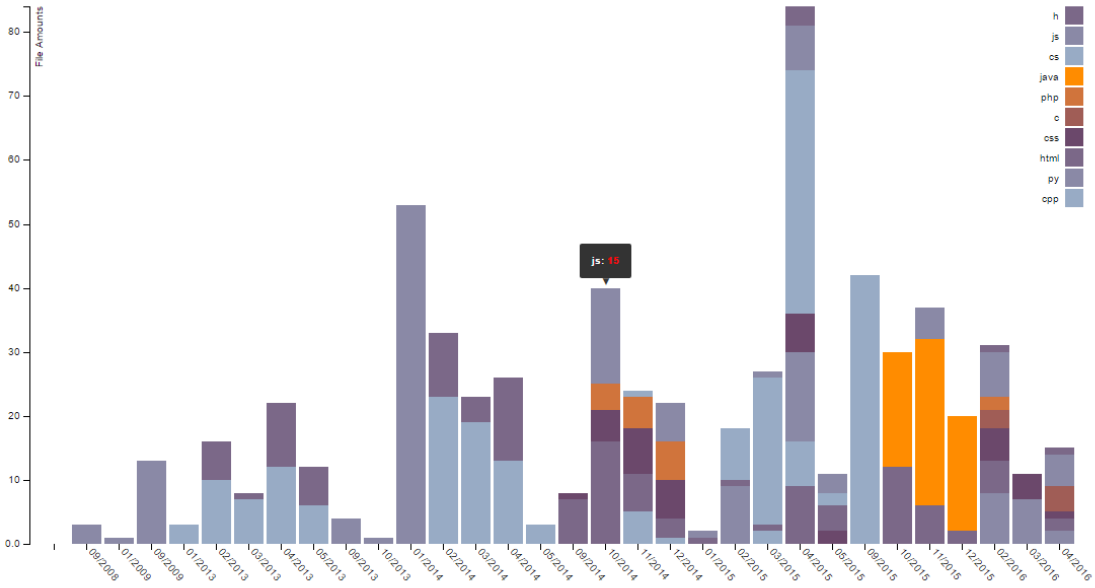

Fig. 4: Final visualization of D3 Stacked Graph implementation

C#. This is important to incoming Freshman because RPI is not about you learning Python or C#. It is about learning the fundamentals of how programming languages work. After students get this groundwork, they are able to teach themselves any programming language they desire. Following C# was Python. This was expected because Python was used through all four years, and as Python is my personal favorite language, it makes sense that I would enjoy using it whenever possible. Following Python was Java. This was unexpected to me because I only had one class which used Java, and that was the only time I used Java while at RPI. This can be attributed to the fact that to write Java programs you need to create lots of files to satisfy Java's love of inheritance. Following Java was PHP which I used for one web development class and then finally C. C was only used in the Operating Systems class, so it makes sense that this would be at the bottom of the list.

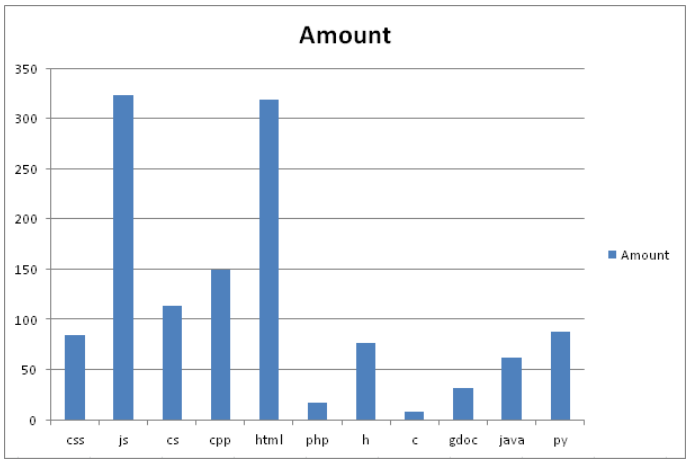

Fig. 5: Final debug graph after removing minified JavaScript files

VII. Conclusion and Future Work

After several weeks of working on this tool and visualization, I feel that I have accomplished what I envision at the beginning of the project. It is not the form I thought it would take, but this visualization shows what I wanted to show, and it does so better than what I had envisioned at the beginning of the project. This is not a complete tool however as moving the output file from the data gathering scripts to the folder which the visualization code uses as input is a manual process. This could easily be fixed however, as the file does not need to be modified once it is output. The Node server which creates the visualization can read the file immediately after it is output from the data gathering scripts. Given more time and resources there are still many improvements which could be made. Currently, an input file is used by the scripts to locate and gather data about programming files. This could be made easier with a GUI. Other improvements would be having selectors on the visualization page which could allow the user to see the data in different forms. This could be a different form of visualization entirely, or a different part of the data being gathered such as all file types in a bar graph without time to see the big picture. There is another major improvement which I would make given more time. Currently, the programming languages are being ranked by the amount of files per file type. As was mentioned earlier, certain programming languages require different amounts of files to accomplish tasks. Java uses more files than a language like Python, and for this reason, it is not a good metric for rating importance or even time spent using programming languages. In order to classify which languages are used heavily, I would add several other metrics The first metric

would be how long did a programmer spend writing the file. This would take the date created subtracted from the last date modified to determine how long a programmer spent working on the files. This has some inherit problems however such as a user creating a file, finish working on it, and then change it years later. Obviously this person did not spend years working on the file. Another metric I would take into account would be the amount of lines in a file. This also has drawbacks as some programming languages require more lines to accomplish tasks. Python and Perl are notorious for being able to accomplish tasks in very few lines. That being said, if all three of these metrics were used to create a weighted score for each language, I believe the results from the algorithm would be closer to the results I was expecting. That being said, the data I gathered very closely resembles what I would consider to be the most important and most used languages at RPI with the exception of the web development languages. These improvements were slightly out of scope for this project as I was primarily focused on gathering good data. Hopefully in the future if I have more time to work on this project I could clean it up and make it useful for people other than just myself.

REFERENCES

[1] Bloch, Matthew, Lee Byron, Shane Carter, and Amanda Cox. "The Ebb and Flow of Movies: Box Office Receipts 1986 2008."*The New York Times*. The New York Times, 22 Feb. 2008. Web. 30 Apr. 2016.

[2] Bostock, Mike. "Stacked Bar Chart." *Popular Blocks*. N.p., n.d. Web. 30 Apr. 2016.

[3] Palmer, Justin. "Using D3-tip to Add Tooltips to a D3 Bar Chart." *Popular Blocks*. N.p., n.d. Web. 30 Apr. 2016.