Final Project - Proposal and Project Plan

Nathan McIntosh

Due October 26, 2020

1 A description of your topic

Something I've increasingly come to appreciate is the Homebrew package manager for Mac OS and Linux. Often I find myself wanting to test out a new software tool, or make sure that a setup "just works". It allows for easy uninstalling of packages no longer needed, and easy upgrading and downgrading of versions. As I watch it install, update, remove, etc.. packages, I'm always interested to see what dependencies are required, upgraded, downgraded. With the command `brew info <package name>` one can see statistics about the package. On my mac, Microsoft's .NET platform has the dependencies cmake, pkg-config, curl, icu4c, and openssl. At the time of writing, it has the following statistics:

- install: 622 (30 days), 676 (90 days), 678 (365 days)
- install-on-request: 414 (30 days), 468 (90 days), 470 (365 days)
- build-error: 0 (30 days)

From this we can clearly see that either .NET is not yet used on a large scale on Mac OS, or Homebrew is not the primary means by which it is installed. What could we learn if we looked at this information for all the most popular packages Homebrew serves on both Mac and Linux? What I hope to accomplish in my final project is an overview of the state of Homebrew dependencies on Mac and Linux. Which are asked for most often? Which are downloaded as dependencies most often? How are popular packages interconnected? Which versions of Mac OS are being used the most?

2 A detailed plan of how the project will get done

I plan to use Homebrew's excellent analytics APIs to get the relevant information. This includes

- Mac OS formula install events
- Mac OS formula install on request events
- Mac OS formula build error events
- Mac OS cask install events
- Mac OS versions for events
- Linux formula install events
- Linux formula install on request events
- Linux formula build error events

Furthermore, there is an API to get all information for each formula (also split into Mac and Linux), which gives the statistics again, but also information about dependencies, conflicts, etc. Documentation for all of these APIs is available here.

To get the data in a format easy for plotting, I'll write Python functions that consume the JSON produced by the APIs and produce pandas dataframes. I will then write a primary plotting function which will use the Plotly plotting library to produce an HTML file of the visualization(s).

3 A detailed time line

Breaking up the remaining time by module:

- Module 9: Write Python functions to successfully pull down JSON data about install and error events, and transform into pandas dataframes.
- Module 10: Revise project proposal. Write Python functions to get individual package JSON info and transform into dataframes.
- Module 11: Write literature survey. Write Python functions to combine all collected data into smallest possible number of dataframes.
- Module 12: Sketch examples of possible visualizations with tools like PowerPoint. Begin writing plotting code.
- Module 13: Finish writing plotting code. Put together documentation about how to run the code, and sample data that can be used to test the system. Put together outline of the final paper, with a few very rough sentences or figures that get the idea across for each section.
- Module 14: Revise final paper. Submit paper and project.

4 Your initial ideas for which course concepts are important to your project

- Network and graph visualization will be important to see how interconnected packages are within the Mac OS and Linux ecosystems.
- Human visual perception will be important to understand to best communicate any differences between Linux and Mac.
- I think it'll be easy to make visualizations for this topic that human visual perception struggles to intuitively grasp
- An important part of this project will be the ability to interact with the data; look at specific packages, highlight linked packages, etc.

5 A description of how your project differs from existing implementations, if any exist, of these data

I have so far been unable to find any existing visualizations of this set of data. Some visualizations do exist for dependencies of various pieces of software. Here are links to a few:

- https://linkurio.us/blog/manage-software-dependencies-with-graph-visualization/
- https://www.netlify.com/blog/2018/08/23/how-to-easily-visualize-a-projects-dependency-graph-with-dependency-cruiser/
- https://www.ndepend.com/

As far as I can tell, most of the existing implementations focus on the dependencies of individual packages/tools. While my project will look at dependencies of individual packages, it will also examine at the ecosystem as a whole: package popularity, reliability of packages, and the effect of packages that do not build properly.

6 A description of how your project enables better understanding of the data

My project will give a complete overview of the most popular packages in the Mac OS and Linux ecosystems, as far as Homebrew is used to install software and manage dependencies. It will allow users to answer questions like "what happens if this packages suddenly stops working for everyone?", "which packages have seen the most growth in the last year?" or "which packages are most popular and are staying most popular?"

The following is a rough idea of the visualizations I plan to make:

1. A line plot of package popularity over time. The majority of packages with low stats will be colored gray with a low opacity. The topmost packages will be labeled and in color. Packages that are not necessarily top, but show interesting trends will also be labeled and colored. There will be one plot for MacOS and one for Linux.

2. A digraph of packages and their dependence on one another. There will be a drop down menu for picking between different types of dependencies: regular, build, recommended, optional dependencies, and requirements (dependencies that may not necessarily be available from Homebrew, such as one OS or another). Hovering over a package shows how many other packages depend upon it. As before, one plot for MacOS and one for Linux.

3. MacOS and Linux bar charts for the ratio between install on request and simple install events. Packages which get most of their downloads from requests are likely "higher-level", i.e. closer to the human. And packages that get more of their installations as dependencies are likely "lower-level" or more foundational.

4. Line plots of formula build errors over time for both OSes. As for the line plots mentioned above, particularly error-prone formulas will be highlighted, while the rest will be low visibility.