Design 3: Github Commit Graph

Group:
Houda Alberts, 10740287
Jasper Bakker, 10260250
Milou Bisseling, 10427538
David Lips, 6225950

Introduction github network graph visualizer link:
https://github.com/blog/39-say-hello-to-the-network-graph-visualizer

Part 1 Analysis

In this design workshop you will sketch multiple solutions to visualize the commit graph and the branches of a git repository.

Take a look at various Github Network Graphs. Here are two examples 1, 2.

Think about how these networks are different. Analyze the "dimensions" of these networks. What are the relevant attributes (e.g., commits, users, branches, commit size, etc.) of these representations? What other attributes could be relevant in this graph? Write a list of all the attributes your visualization could show.
Both networks show owners on the vertical axis and dates on the horizontal axis. In both graphs there are also different branches created by different users. In the CS171 network, most updates are done directly in the master branch, while in the caleydo network most updates are done in a separate branch. After the separate branch is discussed, a pull request to a higher order branch is send and the two branches are merged. Relevant attributes of these representations are commits, users, branches, pull requests, merges, commit size and date. Another attribute that could be relevant in this graph is the size of each change, which could be represented in the thickness of the lines. Colors within the graph already show a distinction between different branches which is relevant to distinguish those.

Are there different roles, i.e., different types of users who might want to achieve different things? Write a list of user roles.
There are different types of users who want to achieve different things. Every user has his own row in the Github network. In a team different users have different roles, for example in the second example one person writes code for the visualization and another person writes code for biological data manipulation. In example 1, the CS171 github, the master branch almost only gets updated by Jenkins. Other members of the network only update their names in the index. So, there could be users who are only interested in updates to get full knowledge about the

project, users that keep tasks separated to achieve the best collaboration by not doing double work, and users that want to work on the same task, but different subsections. With the last task, the merge option within Github can combine the work of both users.

Think about which tasks a user of your visualization might want to achieve. Write down a list of tasks.
The user of our visualization wants to know who commits to the project, what is committed, how much is committed and when it is committed. By seeing who committed a certain part, they know who is responsible and what this person did. Time helps to keep track of the progress during the project and the size shows the amount of work done on certain dates. A user would also like to keep track of different branches (other updates), which are visualized as well.

Identify one role that you want to design your visualization for. Prioritize your task and attribute lists based on this role's needs.
Imagine a client that wants a website made. A visualization that shows every update with details over time could reduce the amount of work of the client by following the current progress of the website in a clear manner. Since the amount of work is also shown in the graph, it can be converted to work per person, so that there exists two graphs in one (two y-axes). So, the attributes are commits, users, branches, pull requests, merges, commit size and date which will be enough for the role of the client while the people working on the website can monitor their own progress as well.

Alternatively, the visualisation could function as a to-do list for project/repository owners, allowing them to quickly see what commits and pull requests are available for merging and whom they are contributed by. To facilitate this more easily, the visualisation should allow for prioritization of open pull requests, possibly based on branches-of-interest (predefined by the user) and the size of commits. An ordered list view or hierarchical tree display of open commits/pull requests alongside (or interchangeable with) the network graph could help with this.

Part 2 Sketching

Decide on which visual variable to use for which attributes of the visualizations. Consider the strengths and weaknesses of visual variables that were mentioned in Carpendale's article Reading 2 (also briefly discussed in this week's lecture: Process). Use the strongest visual variable for the most important attributes of the data.

The visual variables are position, size, shape, value, color, orientation and texture. Colors and positions are to distinguish between branches, contributors, and timeframes within our graphs and they show the most important visual variables. The size of the nodes shows the size of the commit, which is another visual variable we used. Since we used

nodes, we kept the circular node shapes. The connections between nodes are lines to show relations between commits.

Do you think it is necessary to represent every single commit as a separate node? Could you think of ways to aggregate this?

It is not necessary to represent every single commit in a separate node, a way to aggregate those commits is using one node per branch. Interactivity could be included to show the time series of commits per branch after clicking or hovering on that node.

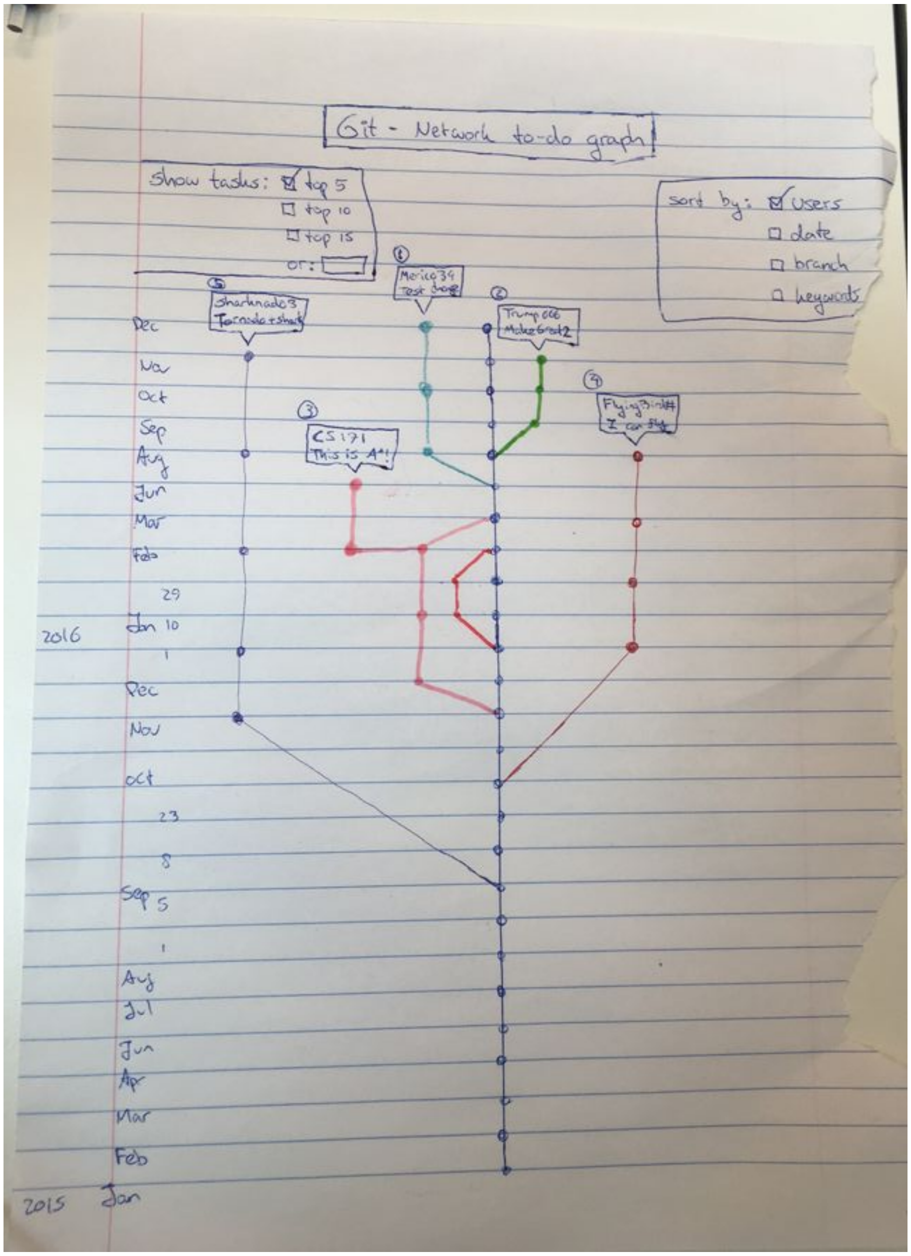
Do you think that every contributor needs a "row", as on the default network view on github? Could you think of a smarter way to summarize those?

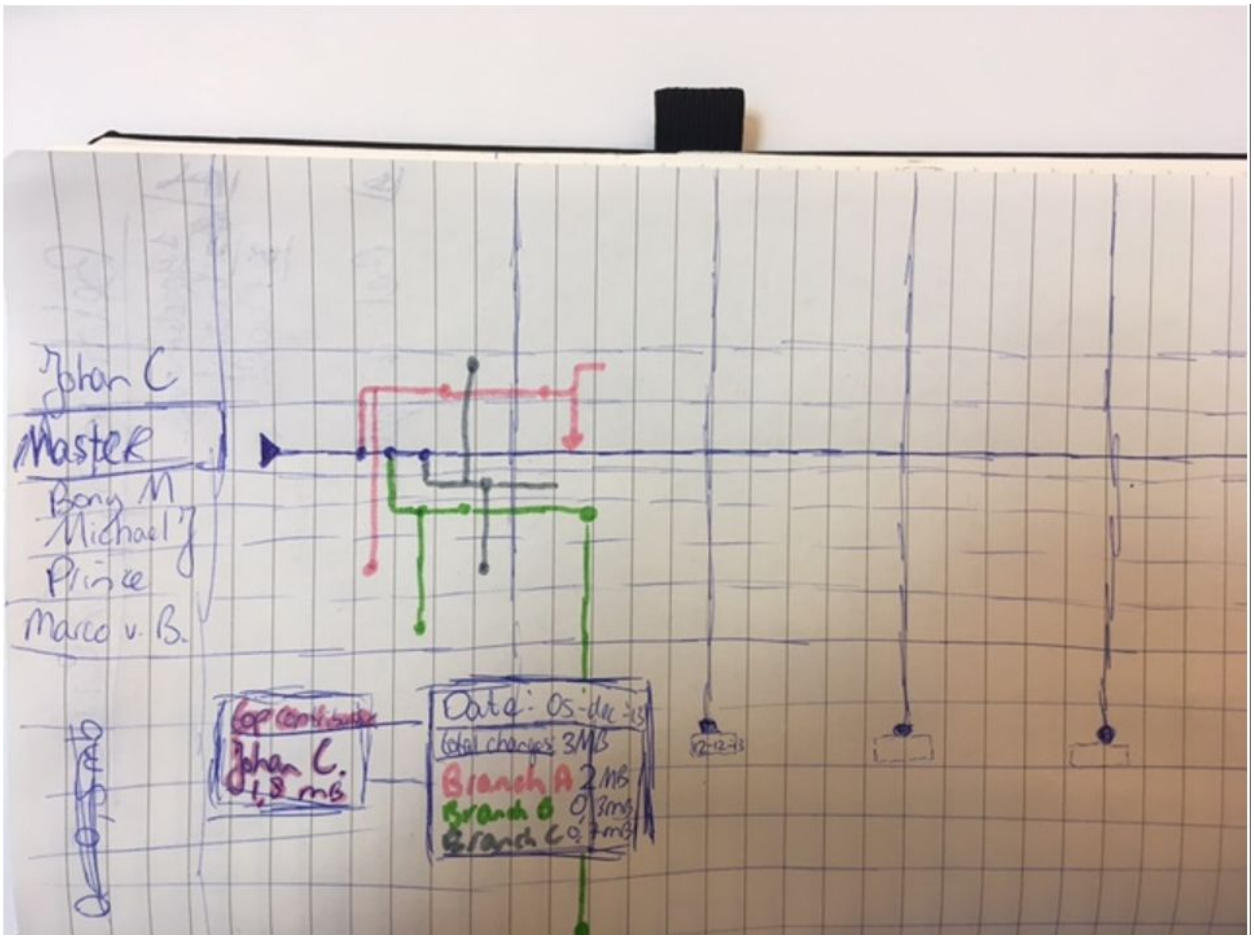
Another possibility to show the different contributors is to use different symbols (shapes) for each contributor, for example rectangle is user 1, circle is user 2, etc. It's also a possibility to categorize pieces of the project and give each category/part a different row.

So, it is not necessary that each contributor had its own row, since there are other representations (like shapes) that could show different contributors. However, the row presentation of each contributor is not wrong either, so it is a possibility to leave that in as well.

Is a node-link diagram the appropriate representation? Or should you consider alternative graph representations?

It is always worth considering alternative visualisations to generate ideas for improved representations, but given the network like nature of the data - different repository branches with commits, pull requests, and merges by various users - a node link diagram seems particularly useful to show how various commits linked together make up branches, the subsequent merging of these branches, and the origin of the commits and pull requests in terms of users and time. So, due to the network like nature of the data, the node-link diagram would be the appropriate representation.


Git - Network to-do graph
show tasks: top 5
top 10
top 15
or:
sort by: users
date
branch
keywords
1
Merico34
2
Trump666
MakeGreat2
3
CS171
This is A+!
4
5
Sharknado3
Dec
Nov
Oct
Sep
Aug
Jun
Mar
Feb
29
Jan 10
2016
1
Dec
Nov
Oct
23
8
Sep 5
1
Aug
Jul
Jun
Apr
Mar
Feb
2015
Jan



Part 3 Group Reflection

The two designs are made for different purposes - one as a way to keep track of web product development and one as a customizable Git to-do list for repository owners. Both are heavily inspired by the original network graph and use visual variables of colors and positions to distinguish between branches, contributors, and timeframes. The process we went through to come up with these designs did not match the design process described in the lecture closely, which referred more to data collection, exploration, analysis, and visualisation. Rather, our process was more user-oriented (rather than data-oriented), and mostly involved exploring different user goals that might be achieved with a Github Network graph. We then proceeded by describing product features required to meet those goals, before translating these to the visual representations shown above. (Note that given the fact that these are designs/products for different user goals rather than two alternative designs for the same goal, finding a consensus or ‘ideal’ visualisation does not apply).