Visualizing programmer trends using StackExchange's public dataset

Luis Santos
IST
Lisbon, Portugal
lsantos.dev@gmail.com

Ricardo Tavares
IST
Lisbon, Portugal
rjgtav@gmail.com

INTRODUCTION

On this project we focus on programmer communities using the popular Q&A website StackOverflow.com. We can think of a programmer community as an entity on its own, a community can be for instance geared towards specific programming languages such a Python or C++. Alternatively it can be more broad scope considering for instance technology stacks or more specific considering for instance specific frameworks for a programming language.

Programmer communities are a dynamical process, communities emerge around certain technologies, they might follow a period of stagnation and eventually die out or alternatively might show strong signs of activity, sometimes even after a long time the programming language originally came about.

One of the tasks that our application has to support is to display the evolution of programming languages throughout time, this should allow us to get a detailed view for each community on how it has evolved throughout the years. We should be able to detect sudden spikes of interest in a language, or even detect languages that have just recently come out of obscurity.

Another task our visualization should support is to detect interactions among communities, going back to the example of thinking of a community as an entity, those entities are not isolated and most often than not they do interact with other entities. Since a programming community is defined based on its user base, meaning the users that engage on that community are what really what make up the community, finding communities intersections therefore becomes finding the shared user base, or in other words finding the users that engage in both communities.

RELATED WORK

Since we have used a public dataset it is only natural that other people have already used it for several uses and purposes. From what we could gather however most people build simple plots using the information contained on the dataset. These plots are mostly performed for statistical purposes rather than for creating compelling visualizations.

Some metrics people have tried plotting for statistical purposes are the average number of answers per question, average number of votes per question, time to first answer, etc. While computing dataset statistics has value on its own, it would not suffice to answer our visualization needs.

The problem of detecting programming language popularity over time is a subproblem of detecting metric variations over time, where the specific metrics are problem specific. Given the time dimension we could benefit from a plethora of visualizations specifically tailored for time series, that is visualizations that take into account the extra time dimension.

Our problem is made more difficult by the fact that we are not simply analysing how the time dimension impacts a specific programming language, instead we deal with multiple programming languages in our visualization. The number of programming languages can range from a few dozen in our small scale application to hundreds or thousands if we were to scale up our application.

The examples provided by D3.js were a great source of inspiration when it came to tailoring our own visualizations, while we always had to adapt some of the ideas and discard a few others, it was undoubtedly a good starting point.

Using a graph structure, where nodes represent programming languages and links represent a sort of relationship between programming languages was inspired from the Language Network example provided in the gallery of D3.js examples(link).

While the problem domain is not exactly the same it shares common characteristics with our problem domain. For the the problem domain of human languages using a graph structure means that each node represents a human language and links represent relationships among languages, a common relationship that can be encoded on the links is language similarity. Using concepts from graph theory the language graph is a weighted undirected graph.

The problem domain of programming languages can also be formulated in a similar way, instead of nodes representing human languages they represent instead programming languages and links represent programming language similarity. Each link has a weight associated with it, an higher language similarity would translate to higher weights.

THE DATA

Dataset Introduction

For our visualization we use data for the popular Q&A website StackOverflow.com.

StackExchange is the organization behind the website StackOverflow.com, it provides regular releases of all user contributed data openly available for the community to use.

The releases are made available under a permissive Creative Commons Share Alike license. Given that new questions and answers are posted on a regularly basis, the data is routinely updated to reflect the dynamic nature of the content. The releases are provided quarterly(four times per year), the latest release at the time of writing is from December 2017.

The StackOverflow data can be found on the Internet Archive(link), to fetch the data we simply downloaded the files that we deemed interesting for our visualization from the host provider.

Since we got our data from a single source, we did not have to aggregate multiple sources nor perform the data cleaning that usually comes with aggregating multiple sources. Overall we found the StackOverflow dataset was already quite "clean" in its raw form which enabled faster prototyping as we built the application.

If the dataset had not been provided by StackExchange and the Internet Archive, we would have needed to build a web crawler to fetch the user content and make sure the crawler adhered to StackExchange's ToS policy.

While it would still be a feasible option, it would have certainly added extra complexity to our application and so by using StackExchange's clean data dumps we managed to cut down on development time.

With that said, as discussed in subsequent sections our data processing was not exactly so straightforward for the next steps of the data pipeline.

Pentaho and Scalability

We originally used Pentaho to build a data pipeline that converted the initial StackOverflow data provided in XML files to an output format that made sense given our defined tasks and our vision of the visualization.

We tried performing the entirety of our data processing on Pentaho, including filtering operations and the calculation of derived metrics, however eventually we ran into scalability problems.

Namely we found out that Pentaho was not prepared to deal with the volumes of data we needed in a timely fashion, at one point we left the program running for a couple of hours and by the end of it the program was still running and the results were not yet available.

Our solution to the scalability issue was to move to a full fledged RDBMS and do our data processing there, by doing this we managed to achieve finer grained control on optimizing the total run time and the output files' size.

To that end we used some simple Python scripts to create the schema and load the XML data to a RDBMS, in our case we used PostgreSQL.

Dataset description

While we will not give a thorough detailed description of the StackOverflow dataset, our discussion would not be complete without mentioning the core concepts. On a high level view, the dataset has posts, users, comments and votes.

A post is either a question or an answer, which consists of the question or answer's text and associated post metadata.For each question, there may be a variable number of answers, the number of answers per question ranges from 0 to unbounded.

Vital to the understanding of our visualization is that each question is associated with tags, a tag is a label that defines the scope of the question such as "C++" or "D3.js".

Whenever a user creates a question on StackOverflow, he can choose a set of tags to label his question so that it becomes easier for other users to find questions based on tag information. Since StackOverflow imposes a max limit of tags per question of 5, the number of tags per question ranges from 0 to 5.

The number of tags per question varies with each question, as some users prefer to provide more labels to add extra information to their questions, for instance a user might tag a question with 3 labels, "JavaScript", "D3.js" and "D3-events". One label for the programmming language, another for the specific library and another for a feature or functionality of that

library. Other users might prefer using fewer tags on their questions and might stick with more generic ones, such as "JavaScript" or "Python".

Additionally each post(question or answer) has comments and votes associated to it. A comment is essentially text and some metadata, while votes can be distinguished as either upvotes or downvotes for each post, each comment or vote has an associated creation date as part of the metadata.

Each one of the concepts described so far(posts, comments and votes) have an associated user which corresponds to the StackOverflow's user responsible for that activity.

StackOverflow currently exists in 5 different languages, which in our application we call regions. Those languages are by order of magnitude(amount of data) English, Russian, Portuguese, Spanish and Japanese.

In terms of relative sizes the dataset sizes for English dwarf all other datasets, which does make sense given that StackOverflow was originally launched in the US and that English is the de facto global language.

For our application it meant that even when the datasets for the other regions worked well, we would get into scalability issues when processing or using the dataset for the English language.

Data processing

To support the tasks defined for our visualization we have created 3 different datasets, to properly distinguish among them we have named them accordingly to the role they play in the visualization.

Common to all our computed datasets is that we completely discard all textual information(such as a post or comment's content) and use only the metadata, the metadata of special relevance to our computed datasets are the post tags and creation date.

We shall describe each dataset in the subsequent sections.

Communities dataset

In the following sections we shall use the word community as a synonym for tag.
For the communities dataset, we are interested in calculating metrics for each tag along a time dimension.
The objective is to measure how each individual tag has varied through time(e.g: Python was more active 8 months ago than 4 months ago) and also how they vary between each other(Python was more active than C++ in 2017).
The results are aggregated using the creation date and tag, the smallest unit of time we support for this dataset is days.
And so for each day and tag, we use the posts to calculate the number of questions and number of answers a specific tag appeared in for a specific day.
Additionally we use the votes to calculate the number of upvotes and number of downvotes, and we use the comments to calculate the number of comments, again all is aggregated on tag and day.
In fact tags are only defined for questions, however given an answer we can find its corresponding question and get its tags, similarly for votes and comments we can find its corresponding post and if it is a question we can get its tags, if it is an answer we can get its questions and then get the tags, by following the link structure one can therefore find the tags for the questions, answers, upvotes/downvotes and comments.
One way to think of this dataset is that for each tag, we have the number of questions, answers, comments, upvotes and downvotes where that tag appears for each day ever since StackOverflow started operating.
To reduce the size of the dataset and decrease the noise we had to filter out some tags, we implemented 2 different filtering schemes for this purpose, an automatic and a manual one.
The automatic filtering scheme filters out all tags smaller than a threshold, the metrics are aggregated(we simply use the sum of all metrics) and we average out the time dimension, the result ends up being a single number for each tag which we can then use to filter out tags whose calculated values are smaller than a predefined threshold, this approach requires careful tuning of the threshold to the dataset so that important tags are not filtered out or irrelevant tags kept in.
The manual filtering scheme implies creating a tag blacklist such that tags are filtered out if they appear on the blacklist, we implemented this scheme to have further control on which tags to remove and as a way to complement the automatic filtering approach.
Essentially if a irrelevant tag still managed not to be filtered out using the automatic approach, using a tag black list is a way to make sure it will get filtered out either way.
While this approach has its advantages, it is in fact very dataset specific, additionally it is quite time consuming to create a comprehensive black list by hand.
On the latest versions of the prototype we have added an extra filter to deal with multiple regions as well, it is a simple filter that filters out all the tags that contain non ASCII characters.

While our derived metrics have strong correlations, since as we increase one metric we would expect others to increase as well(e.g: popular tags such as "JavaScript" have high values for all the metrics), we expect the metrics to be different enough to be used for different visualizations.

Skills dataset

For the skills dataset we are interested in the pairwise relationship between tags, while for the communities dataset we look at each tag in isolation along a time dimension, here we shall look at how often tags co-occur along a time dimension. The name of the dataset comes from the idea of skill intersection, which can be summarized in the following way - If a user participates in one community, what other communities is he likely to participate in?

In other words, we are interested in finding pairs of communities where users that participate in one community also participate in the other, as an example communities based on the same technology stack will share many of the same users(e.g: front end technologies such as HTML, JavaScript and CSS will co-occur often), while communities from vastly different technologies will likely not co-occur so often(that is even though each community has its own users there are distinct non overlapping sets of users, meaning less or none user sharing between communities).

Since the key concept here is the user, we must aggregate the posts and comments per user, meaning that for each user we get the complete history of all the posts and comments he participated in for the selected time unit. While we could have incorporated the votes on the calculations, they do not carry as much information for finding communities intersections as the posts and comments do and so we decided to leave them out for this dataset.

As discussed in the previous section, we can get the tags out of comments or posts by following the link structure, therefore the set of posts and comments aggregated by user can be converted into a list of tags aggregated by user. From the list of tags we then calculate pair wise counts for the tags for each user and time unit. Having the pairwise counts we can sum the counts for each user to remove the user dimension, our final dataset consists of counts for pair wise tags over a time dimension.

For the skills dataset we actually ended up using a different time unit than for the communities dataset, for the skills we aggregate by year while for the communities we aggregate by day.

The reason for that is two fold, first of all while for the communities dataset our UI required a day oriented time dimension, a day oriented or month oriented skills was not required for our UI, second we were getting huge file sizes and aggregating by year was an attempt in that direction(this problem actually became less of an issue once we stopped having accumulated values, see Section 5).

We apply a similar automatic filtering step for the skills dataset, namely if the count between 2 tags is smaller than a threshold we remove it, the threshold value again requires careful tuning.

Another filtering technique we have implemented is to remove connections for the most connected tags, we empirically found out that well known tags like JavaScript tend to connect to many other tags, these connections are not all meaningful and so if we detect a tag is a well known tag(a tag that has many connections), we actually remove some the connections with lower counts.

The last filtering technique we have attempted here has to do with the fact that we represent the data using a graph where each tag is encoded as a node(communities dataset) and each link encodes the community intersection strength(skills dataset), we shall go into much more detail about the visualizations in the next section.

However we mention it here because on the graph structure nodes that do not connect to other nodes are not too interesting(e.g: isolated nodes), and so we remove tags from the communities dataset if they do not appear too often on the skills dataset(or in other words tags that have few or no connections to other tags).

Clusters dataset

We actually did not envision this dataset initially but it made sense to include it as the application progressed.

One problem we would eventually run into is that there are simply too many tags on StackOverflow, as we filter tags out we are obviously throwing away information, we would like to find a way that could use less tags(otherwise our visualization would become too crowded and also take too long to run), while at the same time not throwing away many relevant tags that we could use.

The solution we found was to group tags in clusters using the skills dataset, the intuition is that tags that co-occured often could be grouped together into a cluster of tags, this way we could also afford richer visualizations that took into account the cluster structure and "squeeze" more tags into the visualization.

It is also interesting to look at the automatic clusters that emerge by using our clustering approach, while the algorithm could without a doubt be improved, similar tags do tend to end up on the same cluster, for instance "eclipse", "gradle" and "android-fragment" end up on the "android" cluster.

Our clustering algorithm works by progressively taking the pair of tags with the highest counts and making them a cluster, initially all tags start with their own clusters of size 1 and clusters are continuously merged at each step of the algorithm.

Special care must be paid so that the same tag does not end up on different clusters(a tag can only belong to a cluster) and so that clusters of clusters do not emerge(by limiting the recursion depth to 1).

For this dataset we could have implemented and compared multiple clustering algorithms, however we prefered to dedicate our time polishing the visualization instead.

It turned out that our simple clustering approach returned meaningful clusters, at least good enough to be considered useful.

VISUALIZATION

Overall Description

In this section we shall give an overview of our visualization, its multiple components and what they represent. Our visualization can be loosely summarized as containing 3 main components, the header, the graph and the side bar.

The header sits at the top of the page, besides the StackOverflow logo on the left corner it contains the year selector on the middle and region selector on the right corner.

The region selector is a drop down menu that displays a list of the supported regions, currently there are 5 supported regions for 5 different languages, those languages are English, Russian, Portuguese, Spanish and Japanese. The region selector can be used to alternate between regions, in our visualization there can be only one region selected at a time. All other idioms depend on the selected region and so all get updated when the region changes.

The year selector can be used to select a specific year for the selected region, different regions have been launched in different periods and this will reflect on the year selector, for example StackOverflow in Spanish was launched at the end of 2015 while the original StackOverflow in English goes back all the way to 2008.

On the background of the year selector there is an area chart, this chart plots the activity over time for the selected region ever since it was launched up to the latest data dump release. It can be used among other things to detect spikes of activity(periods with more activity than others).

For the following discussion we use the term activity to mean an aggregation of all derived metrics for the communities dataset(meaning the number of questions, answers, comments and upvotes/downvotes).

Moving on to the next visualization is the graph, the graph takes center stage on our visualization by occupying a great portion of the screen. For the graph we apply a force layout, which is a physics simulation provided by D3.js.

As the data is updated nodes are added to random locations, the physics engine then applies different forces to calculate the nodes' position at each step and eventually after a while the graph settles down, at that moment all nodes have fixed positions.

Each node in our graph represents a tag(communities dataset), links between nodes represent the co-occurence between pairs of tags(skills dataset). The graph can be zoomed in and out and dragged around to change the point of focus.

The tags we chose to display on the graph are the clusters, meaning the tags that aggregate other tags(clusters dataset). Clicking on a tag brings up "a bird eye's view" for that tag, by showing the tags that cluster is associated with using a bubble chart to display the clustered tags.

Due to our clustering algorithm, there are tags that end up without any clustered tags inside them, for those tags we chose not to display the bubble chart. Furthermore clicking on a tag has the effect of automatically zooming in to the tag, allowing us to inspect its inner contents. Then clicking on the same tag again has the effect of zooming out to the global view.

Now we describe the sidebar. The sidebar displays useful information that complements the visualization provided by the graph.

The sidebar has 2 main views: a global view and a tag/local view, depending on the current view it will influence which charts will be shown.

The current view is controlled by the graph, when the application starts the sidebar starts on global view, whenever a tag on the graph is clicked the sidebar goes into local view, when the same tag is clicked again the graph goes back to global view.

On the upper part of the sidebar is a label field, on the local view it will display the name of the selected tag and its icon, while on the global view it will display the name of the region.

The first chart of the sidebar is a calendar heatmap, it displays the activity for each day of the year for the selected year of choice. The activity is encoded on the color of each square, more active days are encoded in darker squares.

Each column of the calendar heatmap represents a week while each square represents a day. By hovering on a square a tooltip is shown that displays the values for the derived metrics(the metrics we use here are the number of questions, answers and comments) on that specific day.

Additionally a second tooltip is shown displaying the counts for the same metrics but now aggregated by week. The next chart is the votes chart, each bar on the positive Y axis represents the amount of upvotes and each bar on the negative Y axis represent the amount of downvotes. The X axis is organized per week so that each bar aggregates all the upvotes and downvotes on that week, there is a strong relationship between the calendar heatmap and the votes chart as each unit on the X axis represents weeks for both charts, and therefore to improve the experience we aligned the weeks on the calendar heatmap with the weeks on the votes chart. Hovering over a specific week will show a tooltip with the amount of upvotes and downvotes for that week. Both the calendar heatmap and votes chart are shown on global and local views, however they display different information.

On the local view the metrics are shown for the selected tag on both charts, meaning that if we see 50 upvotes and 30 downvotes for a week those are the number of upvotes/downvotes that occurred on that week for the selected tag. On the global view however we display the metrics aggregated for all tags, meaning that metrics are summed over all tags and the resulting metrics are shown.

The next chart will change depending on whether we are on global or on local view. For the global view we display a scatter plot, where each point on the scatter plot represents a tag from the graph.

We can then choose between a set of metrics with the hope of finding correlations between metrics, the supported pair of metrics are Questions/Answers, Answers/Comments and Upvotes/Downvotes. Clicking on a metric changes the X and Y axis and moves around the data points. Hovering over a data point will display a tooltip with the name of the tag and the exact values for both metrics.

For the local view we display 2 donut charts side by side, which we name related communities and sub communities. Each one of the 2 charts displays different information, since these charts only exist exist on the local view they make sense in the context of having a selected tag. For the related communities we display all tags that are related to the selected tag, for tag similarity we use the skills dataset. Out of all links on the graph we get the links that connect at either end to the selected tag, the links are sorted by their counts and displayed on the donut chart.

For the sub communities we display all tags that belong to the cluster of the selected tag, for cluster information we use the clusters dataset. Hovering over a tag on both the donut charts displays a tooltip inside the donut chart showing the tag name and corresponding percentage.

Furthermore there is also a legend that displays the tags with the biggest slices.

The bubble chart interacts with the sub communities, by hovering over a tag on the bubble chart the tooltip on the sub communities is shown, similarly by hovering over a tag on the sub communities the corresponding tag on the bubble chart is selected, therefore there is a bidirectional interaction happening between the bubble chart and the sub communities.

Rationale

One major design decision we considered was whether to include a time slider or a year selector. All the way up to the first prototype we were thinking in terms of selecting periods, using the smallest time unit of months, one could select a start month and end month from the time slider, after the selection the data gets updated and the graph would update showing the tags and its activity for the selected period and similarly to the charts on the side bar.

One compelling reason to use a time slider is that it gave great flexibility on choosing how many months we were interested in and which ones, however using a time slider came into conflict with another one of our charts, which was the calendar heatmap. The calendar heatmap demanded data formatted in fixed range intervals, most commonly it shows a yearly view with a square for each day of the year. As we could not meaningfully have variable range intervals on the calendar heatmap, we had to change one of the idioms to consider this.

We chose to convert the time slider to a year selector which displays the region activity on the background. This had an impact on other idioms as now data was given in a yearly format rather than as a date start and date end. We changed the data to reflect this as well, by aggregating the skills dataset by year, since the communities data was still required in a day to day format for the heatmap we aggregated that dataset by day instead.

The rationale for having a global and local view for the sidebar is that we needed to pack multiple charts into a single space, by using different views we could afford some extra space as some charts disappear and others show up. It all also made sense that besides providing information for each selected tag on the graph, we could also provide information for the whole graph in general to get the bigger picture of the dataset.

Regarding the votes chart, we could have in fact displayed the upvotes and downvotes metrics on the calendar heatmap instead of having them displayed on a separate chart, we thought they had potential to become their own chart however, besides the extra struggle regarding the alignment issues of aligning the weeks on both the calendar heatmap and the votes chart we believe having it as a separate chart provides a more compelling visualization.

Regarding the scatter plot, we are using a log scale for both axis, using linear scales would not be compelling since popular tags like JavaScript have very high values for the metrics compared to the other tags, therefore a linear scale would end up with popular tags on the top right corner and all other tags squeezed on the bottom left corner, since the relationship is exponential by nature, by using a log scale we can actually spot linear trends(on the log domain) among metrics.

The scatter plot can be quite useful to detect correlations and outliers, using a tooltip here was also deemed necessary, otherwise just from the inspection of the data points we could not infer the tag name nor the specific values for the metrics, originally we had an X and Y axis with ticks but after moving the exact values to the tooltip displaying the ticks on the axis was deemed unnecessary and we removed the axis alltogether.

Regarding the donut charts, it is a fact that humans are not that good at discerning between different radial areas. Here however we are not that concerned about the exact proportions, since our goal is to measure in relative terms how strongly two tag co-occur(related communities) or the size of each tag inside its cluster(sub communities), therefore we can perfectly trade some accuracy at telling apart the sizes for the slices for an idiom that answers our visualization needs.

We have also included a legend with the donut charts because of the problem of discoverability, meaning that we do not know before hand what tag each slice corresponds to. Whenever a slice is hovered on the donut chart the corresponding tag is displayed on the tooltip, however as we move the mouse away the tooltip disappears and there is no information left about which slice corresponds to which tag. Initially we tried putting labels on the slices, and while it would solve the problem of associating a tag to each slice, the result was not too visually compelling.

Therefore we decided to add a legend to the graph with the tag name and slice color, the order of the legend follows the order of the donut chart and is ordered by slice area. We have to cut off the legend height after a specified number of tags shown otherwise it would disappear from the side bar. This approach is a good compromise because although we display just a subset of the total tags(the ones that can fit in the legend), we no longer have the issues of discoverability while using a visually compelling approach.

Early on in the project we decided a graph structure was a compelling way to visualize the structure of StackOverflow, we were inspired by other work that similarly used a graph structure applied to human languages. Using a graph structure also ends up supporting our main tasks, furthermore it helps with the problem of discoverability, meaning if there are dozens or hundreds of tags, how do you discover which ones exist rather than simply looking at a list of tags? By using a graph we can pack structure we can hopefully display that information.

Initially we were thinking in slightly different terms, we hoped that the distance between tags also encoded meaningful information, essentially if tags were close together they were similar and if they were far away they were less related.

This proved quite troublesome to implement using a Force Layout diagram, furthermore after discussing this idea with the teacher we then moved away from it.

Instead we use clusters to aggregate similar tags together, getting similar tags close together therefore happens on the data processing side and not on the running simulation encoded in the distances between tags.

We believe a meaningful way to show the cluster structure is to use a local view and a global view, whenever a tag is selected we show the local view, showing the tags that belong that that cluster using a bubble chart, the size of the tag on the bubble chart encodes the number of co-occurences that tag has with the tag that forms the cluster.

The size of the cluster tag on the graph encodes the activity of that tag for the selected region and on the selected year, more active tags will end up being larger on the graph.

We have manually added icons for the well known tags to improve the user experience, for the tags without icons we chose to display labels over the tag, otherwise it would have hurted understandability if we could not tell from a glance what tag a certain node corresponded to.

On the latest versions of the project tags without icons became less and less of an issue as we filtered away many tags and kept adding icons to the tags, we might expect that the final version has no tags without icons.

IMPLEMENTATION DETAILS

We chose to implement most of the visualizations from scratch, we would look online for D3 examples that accomplished what we wanted to do and copied some of the code to our project, but in the end we always had to adapt the code to fit our purposes which did not match the simple examples provided by D3.js.

We use D3 events to communicate between charts, whenever something interesting happens in a chart and other charts need to be notified, we create a D3 event that other charts can listen to. Data updates are also implemented using D3 events, changing the selected region launches an event that loads the data for the selected region from CSV files to the application, furthermore selecting a year on the year selector has the effect of indexing by year.

Data change events pass as arguments the new updated data, aggregated by day(as required on the heatmap) and by week(as required on the upvotes/downvotes chart). We try to use D3 transitions when possible to provide a pleasing user experience, it allows us to visualize the data gracefully changing from the old data position to the new position instead of simply jumping to that position.

CONCLUSION AND FUTURE WORK

For this project we have implemented a web app using visualization techniques learnt at the Information Visualization course at IST using StackExchange's public datasets containing all user contributed data for StackOverflow.com.

One problem we had since early on when explaining people our dataset and visualization purposes is that there is a common perception that purely textual datasets can not lead a good visual experience. We realized however that even disregarding the textual contents, there is great potential using solely the associated metada for visualization purposes. Metadata is indeed rich in structure and can be used to calculate several derived metrics, perhaps the most known example that uses solely metadata information for textual contents to infer structure and relevance is Google's PageRank algorithm. Given more time and resources, we would likely scale up our visualization to handle several thousands of tags, even though we are working with huge datasets we had to restrict the problem domain by discarding several tags and clustering some others to reduce the number of tags.

The major bottlenecks of the process were the processing time to calculate our derived datasets, the laborious process of manually filtering out tags, the derived datasets' sizes and the amount of tags we could fit into the visualization without filling up the display. By adding extra resources and applying some optimization strategies we could effectively reduce the development bottleneck for the first 3 problems. For the latter problem we would likely need to apply changes to our visualization idioms to be able to effectively visualize hundreds or even thousands of tags. In particular using a Force Layout on the front end becomes too expensive as the number of nodes grows, we would either have to calculate it on the back end or use other idioms that use fixed position encoding.

It would be indeed interesting to see what patterns could be infered from a visualization containing a large number of tags, additionally it would allow us to cluster tags on several levels, ideally the first level cluster would be programming languages, the second level cluster would be features of the language or frameworks for that language, the third level would be framework features and so on. A cluster dendogram would be a likely candidate to visualize the hierarchical structure.