Final Project Report

ECS 272 - Suchita Mukherjee

16th March, 2020

1 Original Publication

The paper proposed for implementation is ”StarGate: A Unified, Interactive Visualization of Software Projects” by Michael Ogawa and Kwan-Liu Ma, as accessed from
https://ieeexplore.ieee.org/document/4475476
This paper has described the process of creating the visualization quite lucidly and in detail. The design choices are explained extensively which was helpful in understanding how to adapt the visualization for a different dataset.

2 Dataset

This project was motivated by the need to visualize the BugSwarm datset. BugSwarm is the largest data set of its kind, having the information of thousands of real software bugs and their fixes with the ability to grow continuously. Each data point in BugSwarm is made of a fail-pass pair in the build process of a repository. For the purpose of this project, the dataset snapshot dated December 2019 is being used which contains 3140 datpoints from 186 repositories.

3 Goal & Approach

Visualization of software repositories has evoked much thought and effort from computer scientists, but when it comes to visualization of bugs, it has been limited to the software defects of a single repository. This might be the case because a dataset of the scale, reproducibility and multi-dimensionality of BugSwarm did not exist earlier, and thus did not pose to be a sufficiently challenging task. The data set captures the repository name, the programming language of the project, the test framework used, the build system integrated, the operating system of the server in which the project is deployed, the exceptions responsible for the build issue, whether the build failed completely or passed while throwing an error, the reproducibility of this build-fail pair, the commits and pull requests in this repository, the location of the fix, details about the last reproduction attempt, changes, addition and deletions required for the fix and the time stamps for creation, updation and last reproduce attempt. BugSwarm is open-source and is available for users to explore and experiment with, and aiding that is the primary goal of this visualization project. The StarGate visualization provides a great way to view multiple dimensions of a dataset from the initial view and also allows the user to employ several analytical tools. The purpose of StarGate was an instant match for BugSwarm, but the visualization had to be adapted accordingly.

4 Implementation

The system has been implemented using the web framework Flask, with the back end built using Python and the front end implemented with Javascript, HTML and CSS. The features of the visualization were all rendered using the capabilities of the D3.js library.

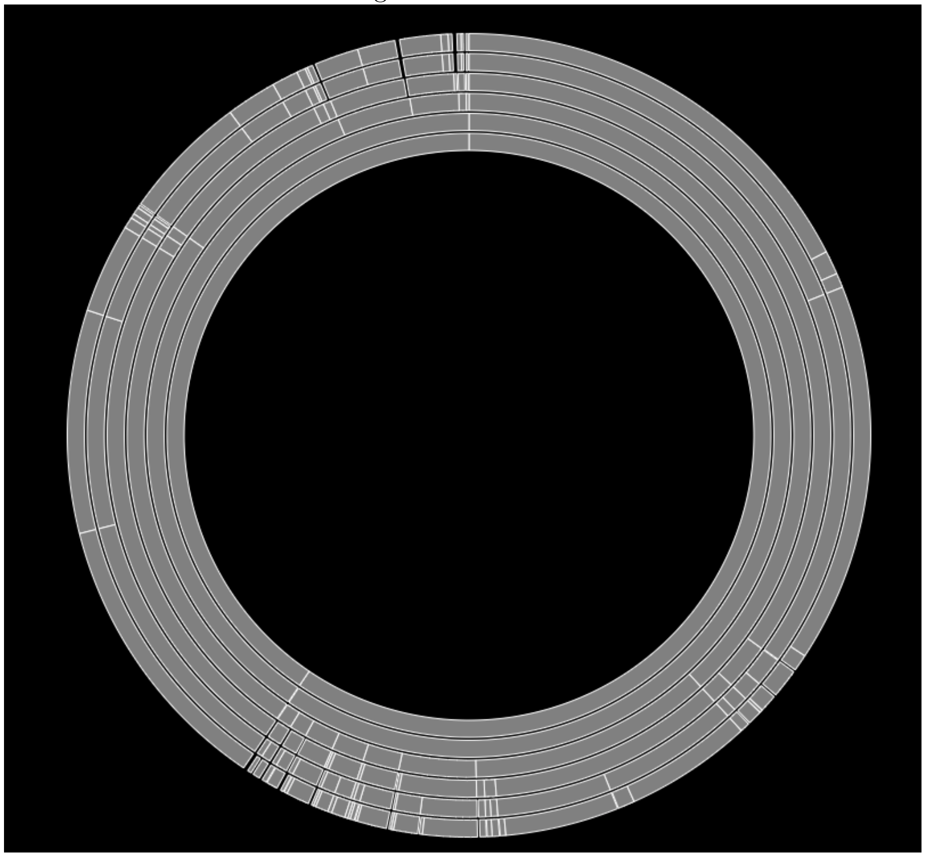
4.1 EDA & APIs

The exploratory data analysis was done using Python. The dataset has a number of features that are required for reproducing the Docker snapshots of the repositories from which the bugs are mined. These features are not an integral part of the visualization and were dropped from the dataset. Four columns in the dataset, namely Patch Location Classification, Pull Request Metrics, Reproduction Status and Failed Job details were available as jsons, from which important features were extracted into added columns like a single column for combined patch locations, number of commits, pull request difference URI, reproducibility and so on.
After preparing the data, two APIs were created to provide data to the front end. The first API '/getArcData' fetches the data required for creating the Gate of StarGate and the other '/getRepoData' provides the information for rendering the Stars and StarDust.

4.2 The Gate

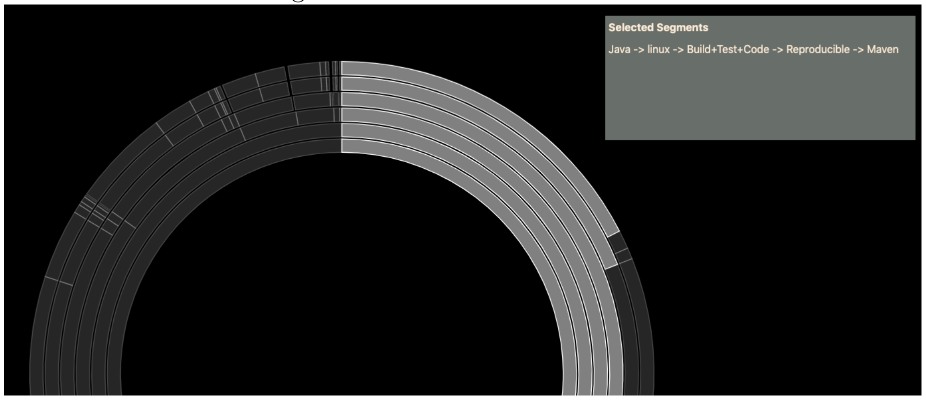
The Gate is made of hierarchical segments representing the six most important features of the repositories. The innermost layer represents the programming language of the repositories, followed by Operating System of the server, Patch Location, Reproduction Stability, Test Framework and Build System. Each segment's size is proportional to the number of repositories belonging to that feature. The Gate provides the base of the visualization and is implemented using a sunburst chart.

Figure 1: The Gate



The Gate has highlights the entire hierarchy of the arc on which the user is hovering, and breadcrumbs are added to show exactly what this hierarchy is labelled as. Moreover, the user can fix a selected hierarchy by clicking on any arc segment of the Gate. The segment labels in the original StarGate visualization was shown as labels on the arcs, for the arcs large enough to hold the label and as a tooltip for smaller arcs. In this adapted visualization, the labels are considerably larger than project folder names and may change as the dataset grows, hence the breadcrumbs have been added.

Figure 2: Hover and Click on Arc



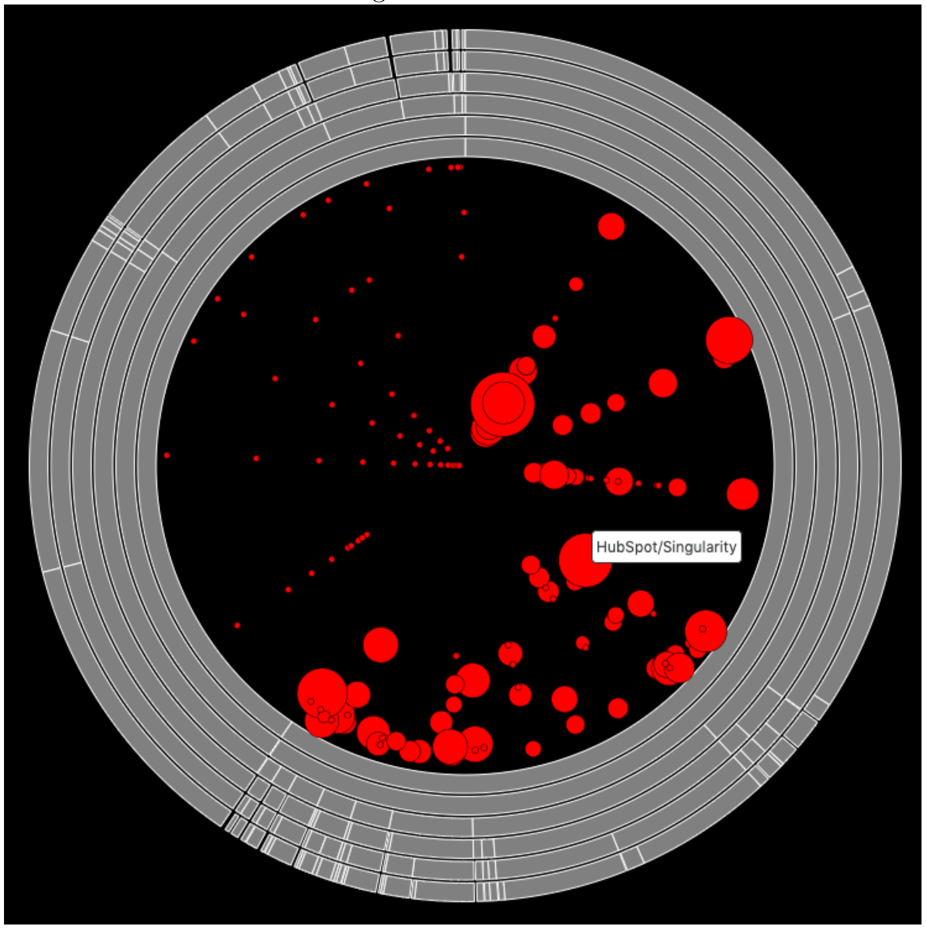
Selected Segments

Java -> linux -> Build+Test+Code -> Reproducible -> Maven

4.3 The Stars

The Stars of the adapted visualization were used to represent the repositories which are mined to create the bug data. A fundamental different of the stars from the original StarGate is that there is no social network between them, due to which that feature was omitted from the adaptation. Similar to the original visualization, the stars were placed around the weighted centroid of all the arcs that they belong to. To avoid overlap, D3's force layout was used initially. However, due to the large size of the dataset and fourth-order polynomial time complexity of the force layout algorithm provided by D3.js, the visualization was taking close to an hour to render. To avoid this exorbitant delay in rendering, a custom force-detection algorithm has been included in this visualization which has a second order polynomial runtime complexity. This increase in speed comes at a cost of accuracy and there still exists significant overlap between the stars. However, for the purpose of this visualization the spread of the stars seems satisfactory given the speed-up. After the stars are rendered, a tooltip is added on hovering over the repository bubble which shows the name of the repository.

Figure 3: The Stars



The Stars are initially all colored the same. To select a group of stars, a lasso-selection has been added. On selecting a group of stars, the group is assigned a different color from the rest and the Gate arcs relevant to the selected group is also highlighted.

Figure 4: Lasso Selection of the Stars

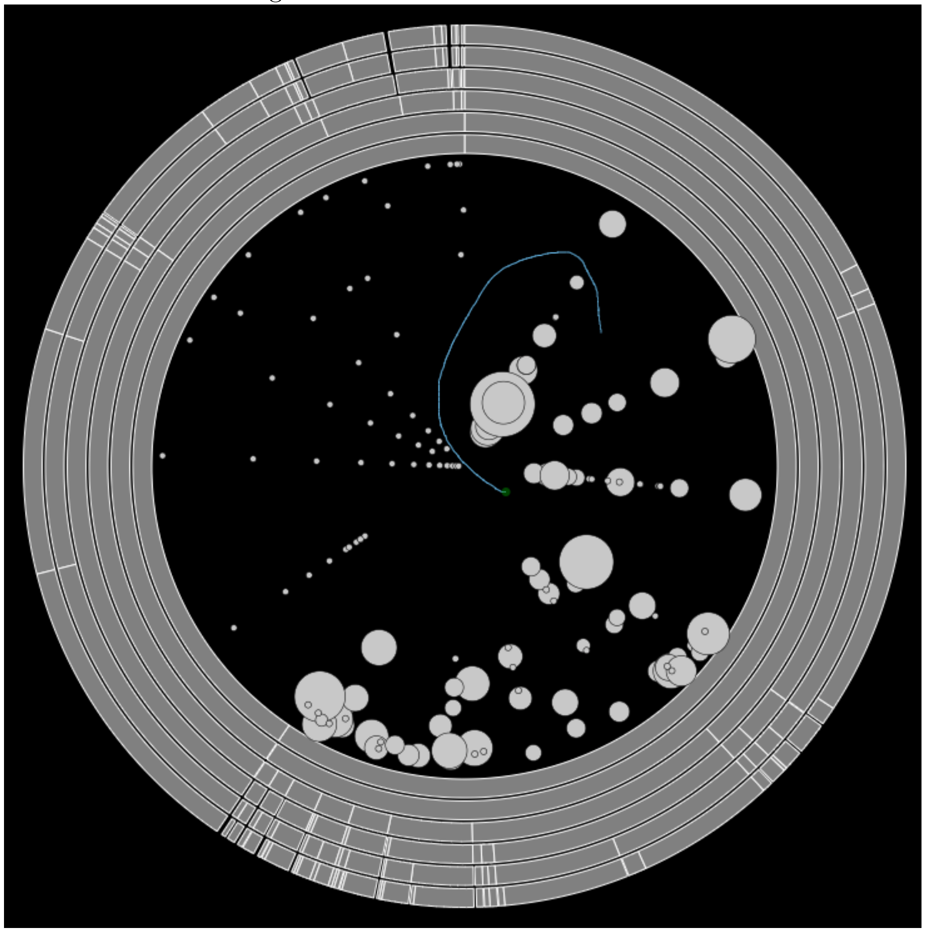
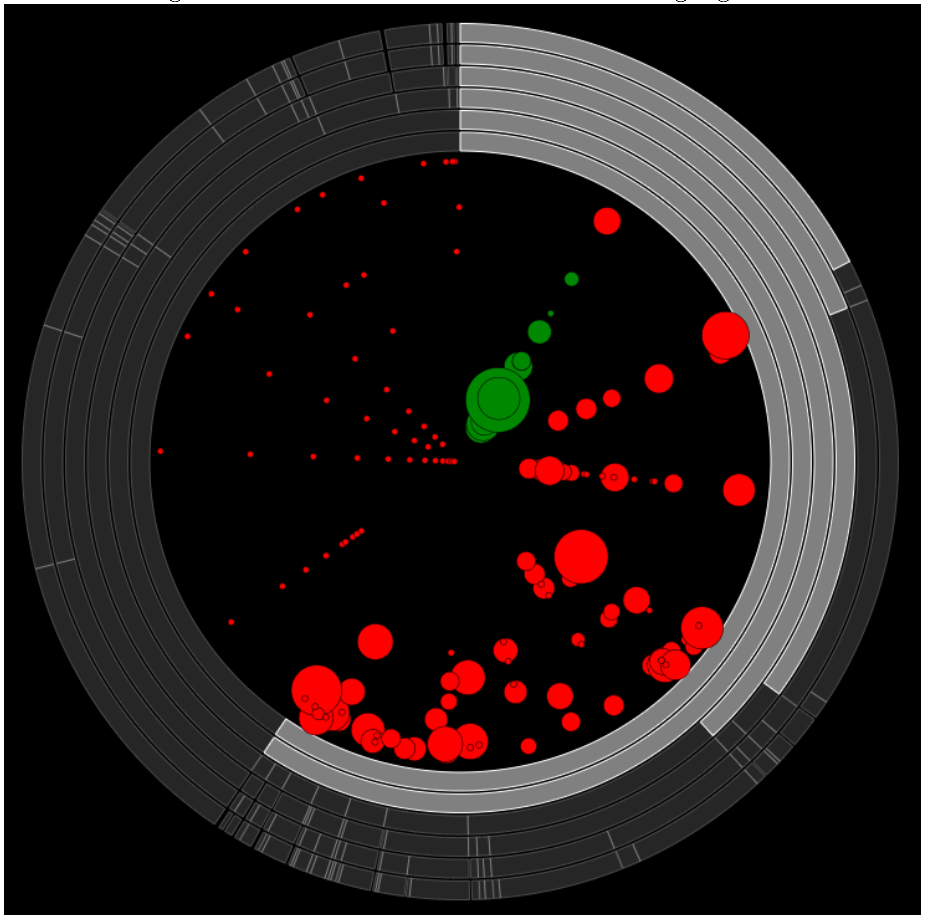
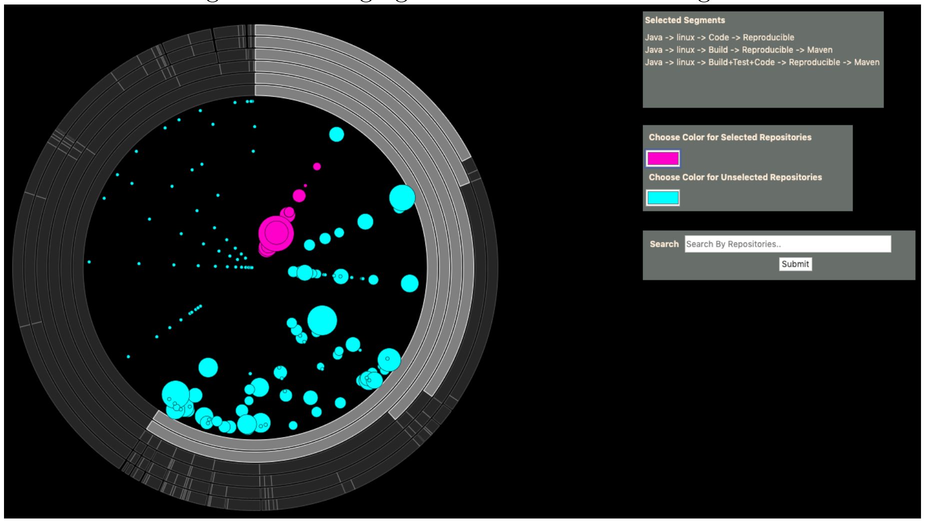


Figure 5: Selected Stars and their Arcs are highlighted



The user is able to assign color to the group of selected stars and also change the default color of the group of unselected stars through the interaction widgets provided. With the large number of repositories visualized together, and since we are visualizing a continuously growing dataset, locating a particular repository will only get harder. To address this, a search bar is provided, where the user can type in the name of the repository and if the name is found to exist in the dataset, the corresponding Star is highlighted along with the arcs pertaining to the selected star.

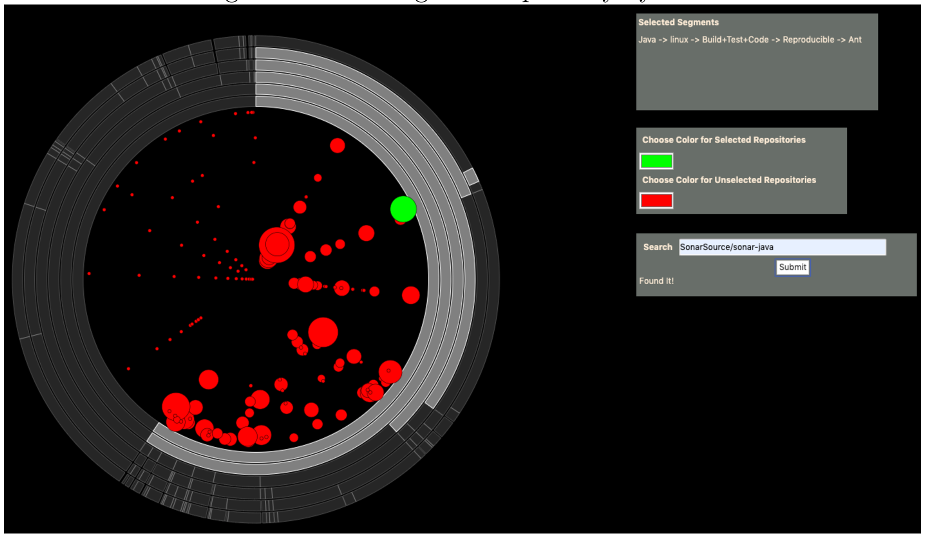
Figure 6: Changing Color of Stars with Widgets


Selected Segments

Java -> linux -> Code -> Reproducible
Java -> linux -> Build -> Reproducible -> Maven
Java -> linux -> Build+Test+Code -> Reproducible -> Maven

Choose Color for Selected Repositories
Choose Color for Unselected Repositories
Search
Search By Repositories..
Submit

Figure 7: Searching for a repository by name


Selected Segments

Java -> linux -> Build+Test+Code -> Reproducible -> Ant

Search: SonarSource/sonar-java

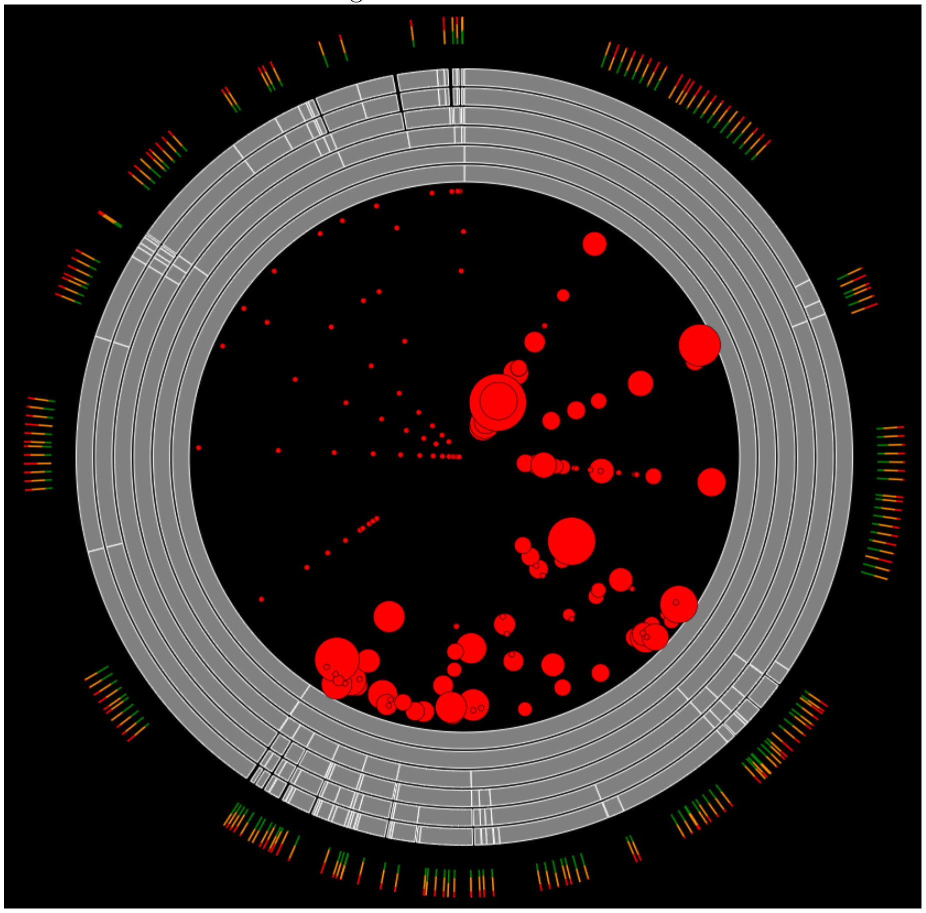
Found It!

Choose Color for Selected Repositories
Choose Color for Unselected Repositories
Submit

4.4 The StarDust

In this adapted visualization, the StarDust represents the latest fix pull request changes for each repository. This is represented along a radial line of each repository bubble that is also a radial line of the Gate. The red part of the line represents deletions, the orange represents changes and the green shows the additions. The original visualization contained dots to make up these lines, but since this adapted version is required to show the proportions of additions, deletions and changes in each repository, a solid line seemed to convey that without much effort from the viewer. Here the problem of collisions arose again. A number of repositories lie on the same radial line, hence their StarDust lines were colliding. This was handled by shifting the lines alternately to the left and the right of the original line, aligning all the lines along the periphery of Gate's outer arc.

Figure 8: The StarDust



Due to the imperfect collision detection of the repositories, their starDust is also irregularly distributed. On lasso-selection of the Stars, the corresponding StarDust is highlighted as well. The highlighting of the StarDust is also done for selecting a single star through the search widget. The other interaction feature of the starDust is that on hovering on the spokes of the starDust, a tooltip displays the name of the corresponding repository and it is a clickable link which opens the GitHub page for the pull request in a new window.

Figure 9: The StarDust of selected Stars is highlighted

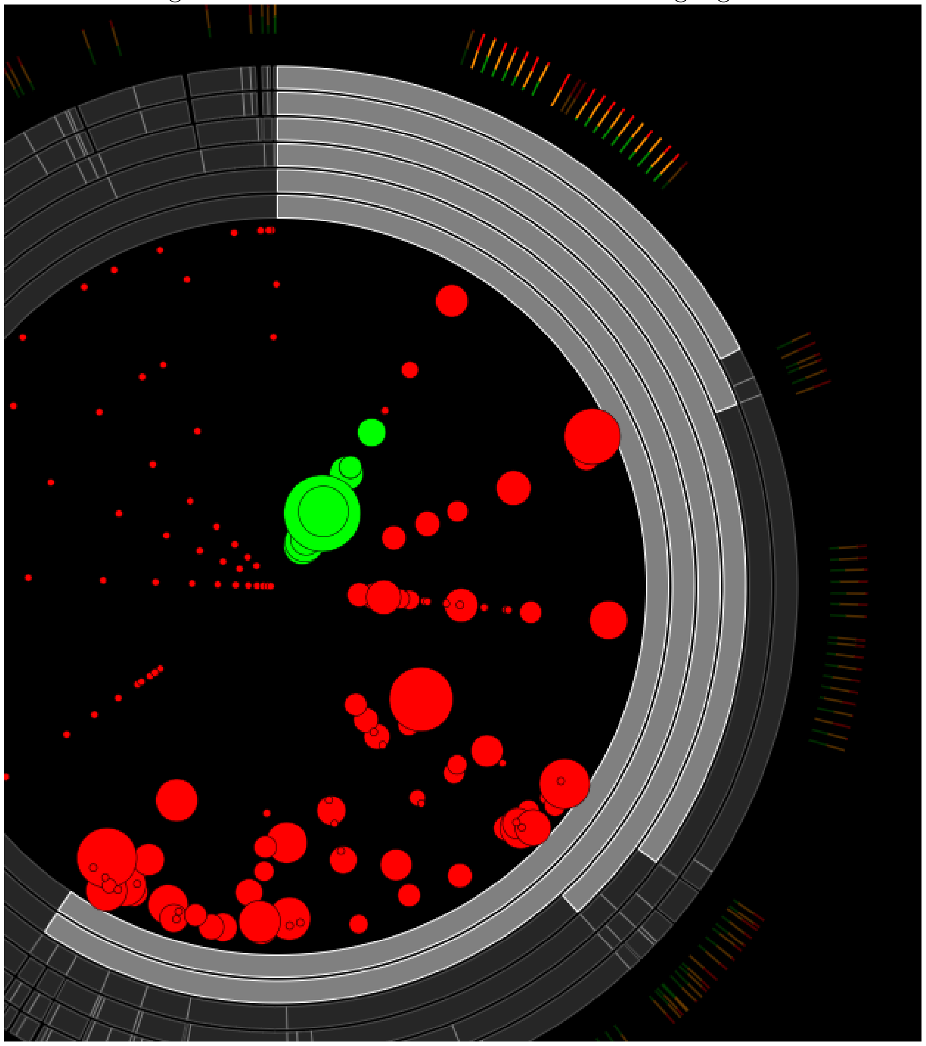
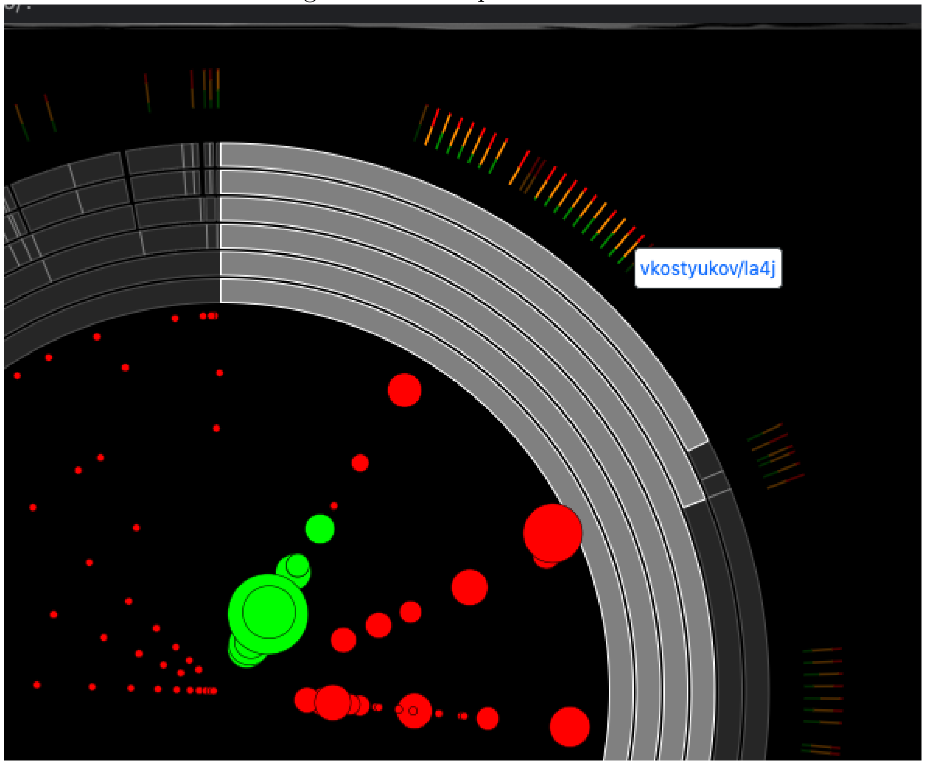


Figure 10: Tooltip for StarDust



5 The Experience

At the start of the implementation phase, the rendering of the Gate was easily achieved after the data was created in a format for the consumption of D3.js' sunburst chart. The positioning of the Stars and StarDust proved to be the real challenge. The Stars and the StarDust at a granular level comprise of basic shapes, that is circles and lines respectively. However, the features as a whole cannot be modelled on existing forms of visualization in the same way that the Gate is modelled on a sunburst diagram. This exercise turned into an elaborate geometry problem, which was quite unexpected. The Stars are not a part of a scatterplot in the true sense. The initial position of the Star circles was assigned using the centroids of the outermost arc of the Gate that they belong to. The outermost arc is chosen in this case as the spread of the outermost arc is a subset of all preceding hierarchical arcs. The second task was to scale the positions to make sure that all circles lie within the innermost arc of the Gate, while at the same time maintaining the relative positions with the relevant arcs. For this, the angle from the center is maintained while decrementing the radial distance. The natural next challenge was to spread out the Stars to avoid overlap while trying to retain as much accuracy of positions as possible. For this purpose, the initial node positions were used to create a force layout. But that presented an unexpected turn in this project. The BugSwarm dataset has information about 186 repositories with 3000 data points. The fourth-order polynomial runtime complexity of D3's force-layout algorithm was taking over an hour to complete. This seriously hampered the development process and it was not feasible to include such a time consuming operation in the application. Thus, unable to use the available force-layout algorithm, a custom collision detection algorithm was devised to spread out the Stars radially, when they overlap. There could be considerable improvement in this algorithm however, it provides sufficient accuracy in the meanwhile.

Finally the Stardust was going to be rendered but had a similar overlapping problem. Since the Stars were mostly spread out radially, the spokes of the StarDust were overlapping. The spokes were alternatively shifted to the left and the right by keeping the radial distance the same, while slightly altering angle, so that the spokes are rendered along the outer arc of the Gate. The interactions for all parts of the visualization were implemented at the end and no specific roadblock was faced.

Overall, the implementation and adaptation of StarGate proved to be challenging in unexpected ways. It required manipulation to be done at pixel position levels which I have never done before and found quite difficult at the start. The difficulty of designing and implementing a customized visualization was revealed gradually through the timeline of this project.

6 The Adapted vs. The Original

There are in fact quite a few deviations from the original StarGate in this project. The segment labels in the original StarGate was shown as labels on the arcs, for the arcs large enough to hold the label and as a tooltip for smaller arcs. In this adapted visualization, the labels are considerably larger than project folder names and may change as the dataset grows, hence the breadcrumbs have been added. Moreover, the color assignment to the Gate arcs has not been implemented.

StarGate did not highlight relevant parts of the gate on selection of the Stars. This feature has been included in this project. The interactions with Stars was a click-based interaction in StarGate. A simpler approach has been taken in this project, by providing a widget to assign color to selected and unselected group of stars.

The BugSwarm dataset has no network between it's repositories. Hence the social network between stars is omitted in the final visualization. Since there is no network to explore, the multi-lasso selection is also excluded, instead only a single lasso selection is implemented as an interaction method with the Stars in this project.

The original visualization contained dots to make up the StarDust lines, but since this adapted version is required to show the proportions of additions,

deletions and changes in each repository, a solid line seemed to convey that without much effort from the viewer.
The final feature excluded in this project is "time-travel". Due to the BugSwarm dataset maintaining dates of reproduction rather that the date of addition of that datapoint in the dataset, having a time slider to show the evolution of the dataset lost a lot of it's meaning. But it can still be implemented, although it was not completed within the timeline for this project.

7 Limitations & Future Work

To improve on this visualization, the first thing that stands out is multi-lasso selection. Although, there is no network to explore between the stars, the users might find it useful to select multiple groups of stars and highlight their features. The seperate color assignment of the Gate's arcs is also not implemented in this project. Another feature that can be implemented is time travel. This feature would be especially useful if the BugSwarm dataset has timestamps of including the data point in the dataset, and it will be interesting to observe the evolution of the dataset. Finally, the collision detection algorithm can be made more accurate than its current performance.

References

[1] M. Ogawa and K. Ma, "StarGate: A Unified, Interactive Visualization of Software Projects," 2008 IEEE Pacific Visualization Symposium, Kyoto, 2008, pp. 191-198.

*https://ieeexplore.ieee.org/document/*4475476

[2] D. A. Tomassi et al., "BugSwarm: Mining and Continuously Growing a Dataset of Reproducible Failures and Fixes," 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), Montreal, QC, Canada, 2019, pp. 339-349. doi: 10.1109/ICSE.2019.00048

*https://ieeexplore.ieee.org/abstract/document/*8812141