TU WIEN Informatics

Identifying GitHub Trends Using Temporal Analysis

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Media and Human-Centered Computing

eingereicht von

Thomas Anderl, B.Sc.
Matrikelnummer 01427841

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Mag.rer.soc.oec. Dr.rer.soc.oec. Silvia Miksch
Mitwirkung: PhD. Roger Almeida Leite

Wien, 1. August 2021

Thomas Anderl	Silvia Miksch

TU WIEN Informatics

Identifying GitHub Trends Using Temporal Analysis

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Media and Human-Centered Computing

by

Thomas Anderl, B.Sc.
Registration Number 01427841

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Mag.rer.soc.oec. Dr.rer.soc.oec. Silvia Miksch
Assistance: PhD. Roger Almeida Leite

Vienna, 1st August, 2021

Thomas Anderl	Silvia Miksch

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Thomas Anderl, B.Sc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. August 2021

Thomas Anderl

Danksagung

Hiermit möchte ich all jenen Personen danken, die durch ihre persönliche und fachliche Unterstützung zum Gelingen dieser Masterarbeit mitgewirkt haben.

In erster Linie danke ich meiner Betreuerin Univ.-Prof. Mag. Dr. Silvia Miksch als auch Phd. Roger Almeida Leite für die kompetente Hilfe im Verlauf der Arbeit, sowie meinen Freunden, Verwandten und Arbeitskollegen bei *Accenture* für das kritische und hilfreiche Feedback.

Ein besonderer Dank geht auch an alle Freiwilligen, die an der Evaluation teilgenommen haben.

Acknowledgements

I want to thank all people that helped with their personal and professional support throughout the course of the thesis.

A special thanks goes to my advisor Univ.-Prof. Mag. Dr. Silvia Miksch as well as Phd. Roger Almeida Leite for their expertise and help as well as my colleagues at *Accenture*, friends and relatives for their input and participation in the evaluation.

Kurzfassung

Durch die *COVID-19 Pandemie* sowie dem steigenden Interesse an quelloffenen Projekten, gewinnen Versionskontrollsysteme wie *Git* an zunehmender Popularität. Durch diesen Anstieg erhöhte sich auch die Vielfalt und das Ausmaß an Daten auf Plattformen wie *GitHub* zunehmend, was zu steigendem Interesse für Soziologen und Softwareanalysten führt.

Diese Arbeit fokussiert sich auf die Visualisierung von *GitHub*-Daten mit der Hilfe von *Visual Analytics*. Die Daten stammen sowohl aus der *GitHub* API [Api] als auch dem *GitHub Archive*[Gha], sind multivariate und enthalten diverse Informationen über Projekte, Nutzer und Ereignisse. Diese Daten werden außerdem durch die zeitliche Dimension ergänzt, um potentielle Trends zu entdecken. Für die Problemdefinition und der Methodik wurde das *Design Triangle* wie von Miksch et. al [MA14] beschrieben, herangezogen.

Das Ergebnis dieser Arbeit ist ein Prototyp, der es Domänen-Experten nicht nur erlaubt typische Aufgaben in Bezug auf *GitHub* Trends durchzuführen, sondern auch visuelle Interaktionsmöglichkeiten bereitstellt, um Fokus auf speziellere Zeitbereiche zu legen. Obwohl sich grundsätzlich viele Arten von Trends visualisieren lassen könnten, fokussiert sich der hier entwickelte Prototyp nur auf eine kleinere Teilmenge von Problemen. Die generelle Zielgruppe liegt hierbei auf Analysten in technologischen Industrien.

Der Prototyp wurde durch Domänen-Experten mit verschiedenen Schwerpunkten durch eine vordefinierte Liste an Aufgaben evaluiert. Die Ergebnisse der Evaluation zeigen, dass es ein großes Interesse in der Analyse von *GitHub*-Daten gibt und das die Wahl der korrekten visuellen Kodierung und Interaktionsmöglichkeit essentiell für das Finden von Trends sein kann.

Abstract

With the increase of remote work due to *COVID-19* and the overall movement towards open source projects, distributed version control system, like *Git* gained popularity over the last years. The publicly available data on platforms (e.g., *GitHub*) therefore becomes richer and attracts sociologists and software analysts for further analysis.

This master thesis aims to visualize *GitHub* trends using *Visual Analytics*. The data used originates from the *GitHub* API [Api] as well as *GitHub Archive* [Gha], is multivariate and contains different types of information containing repositories, users and events. This data will be extended by the temporal dimension to identify potential trends. For the problem definition and further methodology, the design triangle as described by Miksch et. al [MA14] is being used.

The outcome of the thesis is a prototype, that not only enables domain experts to fulfill common tasks related to identifying *GitHub* anomalies and trends but also allows for user interaction to focus on more granular analysis. While many trends can potentially be visualized, this thesis will focus on a small subset of trends to introduce a generic approach and evaluate it on given scenarios and tasks. The general group of potential user groups is broad, but there is a strong emphasis on analysts in technology industries.

The prototype was evaluated with domain experts in different fields of expertise that were asked to perform given tasks that can be fulfilled using the developed prototype. The results of the evaluation showed, that there is a strong interest in the analysis of *GitHub* data and that the right encodings and visualization methods can help find patterns and trends significantly.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**
1.1 Background . . . 1
1.2 GitHub Terminology . . . 2
1.3 Research Question . . . 2
1.4 Structure . . . 3

2 Related Work **5**
2.1 Information Visualization . . . 5
2.2 Identifying *GitHub* Trends . . . 6
2.3 Visualization of Large Data . . . 7
2.4 Visualizing the Temporal Dimension . . . 7
2.5 Benefits and Limitations . . . 19

3 Problem Statement **21**
3.1 Data . . . 22
3.2 Users . . . 23
3.3 Tasks . . . 23
3.4 Requirements . . . 23

4 Visualization Design **27**
4.1 Issues Board . . . 27
4.2 Repositories Board . . . 29
4.3 Alternative Visualizations . . . 33

5 Prototype Implementation **37**
5.1 Backend . . . 37
5.2 Frontend . . . 39
5.3 Pipeline . . . 41

6 Evaluation **45**
6.1 Methodology 45
6.2 Results 48
6.3 Analysis 55

7 Future Work **57**

8 Conclusion **59**

List of Figures **61**

List of Tables **63**

Bibliography **65**

CHAPTER 1

Introduction

1.1 Background

With over 50 million users and around 200 million repositories, *GitHub* is a fast growing versioning platform for developers [Gitb]. Developers can collaborate on the same code all across the globe while maintaining a consistent code base. With the increase of popularity of platforms, like *GitHub* and a stronger focus on telecommuting due to *COVID-19*, the interest in analyzing technological and social trends seems to grow accordingly [BHO+20]. Many industries rely on not only identifying but also predicting such trends to drive strategic decisions within the company.

For a platform like *GitHub*, the data is obviously very technology orientated. It, however, also allows for the analysis on social aspects within the *GitHub* ecosystem, which this thesis will put a stronger emphasis on. Even though there exists a variety of different tools to analyze *GitHub* trends [Bar][Ant][Epa] [The][Gitc], none of the publicly available tools provide enough functionality and depth to answer different research questions simultaneously. They are mostly narrowed down to solve one specific problem without the option to provide user input and consequently changing the scope of the analysis. A big reason for that is that the amount of data available is very large and consists of quantitative, abstract and multivariate events. For archived data, it mostly comes in form of events, making it easier to parse but harder to evaluate, due to the lacking query potential. Using the API with *GraphQL*, however, gives a better querying and filtering potential but limits the amount of allowed calls and data.

The aim of this thesis is to find appropriate methods to visualize *GitHub* data to help analysts be more efficient in their tasks. There is a strong focus on the temporal dimension, while aiming for a high usability. The chosen visualizations will focus on specific problems while showcasing a flexible solution on analyzing trends.

A trend describes a general direction, something is moving towards. These can be of positive as well as negative nature. There are many different reasons for a trend to occur, common ones are of sociological nature (e.g., Christmas), as well as due to changes in popularity of specific topics [SK15].

1.2 GitHub Terminology

There is a glossary maintained by *GitHub* containing all the important terms [Gita]. The following list will, however, mention the most important ones for the course of this project.

1.2.1 Commit

A commit represents a documented change to one or more files. They contain the user that issued the comment alongside a commit message. Commits are identified by a unique ID an can be tracked as well as rolled back to.

1.2.2 Issue

For public repositories, every user can create an issue. An issue can point out a problem, suggest new functionality or might also just be a question. They are moderated by the collaborators of the project and can help keeping track of tasks.

1.2.3 Repository

A repository can be seen as a project on *GitHub*. It spans over all the files, configuration and history and can either be public or private. Every repository comes with it's own list of owners and maintainers who take care of the project.

1.2.4 User

Every user comes with her/his personal profile and can actively participate on the *GitHub* platform. They can create, watch and like repositories, create commits, open and close issues, collaborate with other users on repositories and do any other action available for users.

1.3 Research Question

The research question is defined as follows:

How can *Visual Analytics* assist in analyzing *GitHub* trends?

With the corresponding hypothesis:

- **H1** Social and technological trends can change rapidly over time.

- **H2** Domain experts are capable of using *Visual Analytics* software given the right design.
- **H3** Visual interaction methods provide more insight by leveraging the temporal dimension.

1.4 Structure

Chapter 2 will give insights in to the current state of visualizations for temporal data. It will also focus on state-of-the-art approaches to deal with large data-sets as well as *GitHub* data.

Chapter 3 puts emphasis on the problem characteristics, more precisely the data, users and tasks as well as the derives requirements.

Chapter 4 focuses on the chosen visualization design. This includes the boards and their components as well as the visual encodings.

Chapter 5 goes into detail for the implementation of the prototype. It will give an overview of which technologies were used and for what reason. There is also the pipeline described that was needed to parse the large amount of data into query-able tables.

Chapter 6 highlights the required steps for the evaluation as well as the results and outcomes of the performed tasks by the experts evaluating the prototype.

CHAPTER 2

Related Work

There exists a variety of literature for the individual topic of this thesis, but rarely any that combines *Visual Analytics* of *GitHub* data with a temporal context. This chapter focuses on highlighting common techniques when working with temporal data and gives an overview of existing solutions for similar problems.

2.1 Information Visualization

The modern interpretation of visualization uses computation power to facilitate the visual perception of humans. Schumann and Müller [SM00] defined requirements on visualization software. The first of them is *expressiveness* which indicated that the visualized data should include as little as possible and as much as required. *Effectiveness*, however, describes that the visualization should utilize the visual perception of humans. As a result, effectiveness depends on the user, making it difficult to establish common rules. The last requirement is *appropriateness* and expects, that the value gained from the visualization comes at a appropriate amount of cost (e.g., computation time).

Shneiderman [Shn96] mentioned seven tasks on a high abstraction level that users usually intend to perform:

- **Overview:** Provide an overview over the entire data set
- **Zoom:** Enable to zoom into specific areas
- **Filter:** Limit the result set to a more interesting subset
- **Details-on-demand:** Retrieve more detailed information of a selected element
- **Relate:** Show relationships among items

- **History:** Preserve a history of previous actions
- **Extract:** Support extraction and persistence of results and their parameters

2.2 Identifying *GitHub* Trends

Some papers, like [CLC16] show general approaches that should be used when working with *GitHub* data. While not directly evaluating *GitHub* data, it provides a solid foundation to avoid common mistakes. Another paper [KGB+14] very similarly explores the capabilities of the analysis of *GitHub* data proving, that rich amount of information can be derived from using a high amount of data. A more predictive orientated approach was chosen by [KDP16] where over 4000 issues and their projects have been analyzed. They were not visually analyzed but still showed an interesting approach using machine learning. The issues were used to predict the life cycle of a repository. These repositories and issues were then observed to evaluate the solution. A more socially oriented approach was chosen by [PMS14]. In this study, pulls and forks for different repositories were analyzed and the users where split into three groups: core, external and mutant. There three groups were then defined for different programming languages to find common patterns in the activity of developers in open source projects related to specific programming languages. For trends, however, to be explored, there needs to be metrics related to repositories that can be measured. This was done by Chatziasimidis and Stamelos [FI15] who evaluated what makes a project successful.

For identifying trends in general, Sunar and Kankanala [SK15] mentioned three important aspects that may influence trends:

- **Interest over time**: For the survival of a trend, it is important how the interest towards a topic changes over time. This change can be of positive or negative nature and is a driving factor, meaning that it is not enough to look at the interest at a specified point in time but over a period of time. Sunar and Kankanala [SK15] mentioned the example of the *ALS Ice Bucket Challenge*, that intended to increase the awareness towards *ALS (Amyotrophic Lateral Sclerosis)* which turned out to be a huge success. Even though there have been many negative reactions to that idea at given moments, the positive ones were significantly superior over time, resulting in a positive outcome.
- **Events influencing interest**: Any given event or activity, can heavily impact the course of a trend. This mostly happens from groups or individuals who connect to a large niche. Due to their followers being in the same niche, presentations, conferences or even tweets can lead to a positive or negative interest on a trend. An example for the impact of a single person on trends has been shown by Elon Musk in 2021. With only a few *Twitter* posts did he influence the return of *Bitcoin* by 18.99% and *Dogecoin* by 17.31% [Ant21].

- **Period of trend relevance**: There are different dimensions of relevance that can influence how a trend is evolving:

  - Time - A specific trend can potentially only be relevant during a specific time. Public events, like a world championship or an upcoming election could thus be a driving factor for an increased activity towards a given trend.
  - Region - There are cases when trends are only relevant on a local basis and not globally. Examples for that could be news or weather forecasts that will mostly not affect people, who do not live in the corresponding area.
  - People - The last relevant factor are the people who relate or show interest in specific trends. Some age groups or communities may show more interest towards specific trends. A trend in technology might consequently pick up more hype between developers than athletes.

2.3 Visualization of Large Data

ThemeRiver [HHWN02] as seen in Figure 2.5 introduced an interesting approach to visualize large temporal data sets and find anomalies in the change of data. For that case, different types of documents were taken and then analyzed by included keywords. For specific events in the world history, there could a specific change in keywords be derived. As the name suggests, the keywords and their occurrences were visualized as curves which look familiar to rivers. The used approach could theoretically be used to find trends and be applied to information from *GitHub* data as well. Another visualization of temporal data is described in [LS09]. The data visualized is multivariate and the authors introduce a distance metric called SUB-DTW which helps to estimate when trends occur. Since my topic tries to visualize trends as well, it seems very natural to use this paper as inspiration.

Another thesis [Cis18] makes suggestions based on sports activity with a temporal context. While the suggestions are based on similarities between given sport activities, there is still a decent amount of reusable information. Especially the concept of the Spiral Graph [WAM01] visualizing the amount of bookings for each month could be applied at some point for this work.

2.4 Visualizing the Temporal Dimension

Adding the temporal dimension to a visualization can bring many challenges. The main reason for that is that time is an abstract concept and there are consequently no natural ways to visualize it. Consequently, there exist several models and taxonomies to represent time, but each of them heavily depend on the concrete problem. To derive specific models from a concrete application, design aspects have been defined.

2.4.1 Design Aspect

Aigner et. al [AMM+08][AMST11] described the main criteria for working with time-oriented data:

- **Ordinal versus discrete versus continuous**

  For ordinal time only relative relations are taken into consideration. To be more precise, a specific time event to be happening after or before another time event is the only information available in this domain. This is a very simplistic model that is mainly suitable for problems without requirement of quantitative information. In a discrete time domain, time is mapped on small time units. These units should be as small as possible to allow for accurate mapping. Discrete time is the most used domain in information systems, with a known example being UNIX time in seconds or milliseconds. Continuous time on the other hand the mapping happens on real numbers. This means, that the data does not lose precision or accuracy when being mapped. As a result, between any two points in time there is another point in time. This also referred to as dense time.

- **Points versus intervals**

  Most techniques in visualization deal with time points, as they are easier to visualize. Intervals on the other hand represent a duration, like days, weeks or months. It is possible for a time value to have different interpretations in a point-based and an interval-based domain.

- **Linear versus cyclic**

  When talking about linear time, it is assumed that every point in time has a single unique predecessor and a different unique successor and is happening on a linear timeline. Cyclic time, however, represents the concept of reoccurring events. Many natural processes, like the day/night cycle are based on it. A typical visualization method to identify cyclic events is the Spiral Graph which uses spirals as axis as helps identifying periodic behaviour in data.

- **Ordered versus branching versus multiple perspectives**

  Ordered time describes the concept of time events that happen in a sequence, but it is possible for different primitives to happen at the same time. *ThemeRiver* (Figure 2.5) is an example for a visualization suitable for ordered time data. Branching time data, however, can be used to describe multiple branches of possible scenarios to help assist in decision-making processes. Multiple perspectives facilitate more than one point of view for given events.

There are several papers from Leite et al. related to the visual analysis of temporal data. COVIs [LSC+20] focuses on a storytelling approach in regard to known *COVID-19* related statistics. For their methodology, the design triangle was used and the tasks

are oriented towards a journalistic point of view. There are several views representing different metrics and scales. Hermes [AAS+20] compares several financial sectors to each other with a temporal context. The design triangle methodology was also used here leading to the development of prototypes, which have then been evaluated. For evaluation, a task-based approach has been chosen, where users have been observed while working with Hermes. For that, the users were given specific tasks they had to answer using the tool. In regard to temporal analysis, there has a slider been implemented, used to set the year to a given one. NEVA [AGM+20] implements an approach to visualize networks with the aim of detecting frauds. It has a strong emphasis on false positives and false negatives. There was also the design triangle applied to the methodology and for the evaluation, a mixture of interviews and making the experts fulfill given tasks was chosen.

2.4.2 Techniques

There is a large variety of methods to visualize time-oriented data. Aigner et. al [AMST11] and Harris [Har99] provided a detailed survey over the common techniques and a selection of these will be presented here. The techniques are characterized along the following properties:

- **data**
  - frame of reference: abstract/spatial
  - variables: univariate/multivariate
- **time**
  - arrangement: linear/cyclic
  - time primitives: instant/interval
- **vis**
  - mapping: static/dynamic
  - dimensionality: 2D/3D

Point Plot

frame of reference	abstract
variables	univariate
arrangement	linear
time primitives	instant
mapping	static
dimensionality	2D

Table 2.1: Characteristics of the Point Plot

The point plot is one of the most basic techniques to visualize time-oriented data. It uses the Cartesian coordinate system to map data points to their temporal dimension. Figure 2.1 shows a point plot with eight data points containing time date. This can also be referred to as a point graph or scatter plot and usually focus on individual data points as they are visualized on a common scale and consequently be perceived efficiently. The single dots of the plot can also be coded (e.g., color/size) to visualize additional variables [Har99].

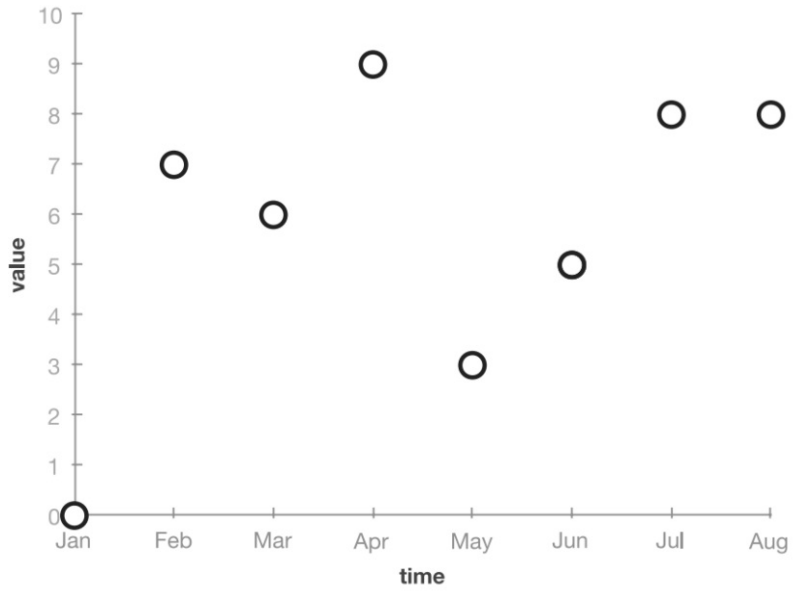

Figure 2.1: Point Plot mapping the time to the horizontal axis and the value to the vertical axis [AMST11].

Bar Chart

frame of reference	abstract
variables	univariate
arrangement	linear
time primitives	instant
mapping	static
dimensionality	2D

Table 2.2: Characteristics of the Bar Chart

Another common method to visualize time-oriented data is the bar chart. As the name suggests, the values are represented as bars here. Since the bars appear on a common scale that always starts at zero, the values are easier comparable than with a scatter plot. Depending on the problem, the individual bars can be of different width to allow for better readability if there exists many bars. In case the bars are reduced to a degree that they appear as spikes, the chart is referred to as a *spike graph* (see Figure 2.2) . In some cases it may also be useful to stack individual bars, allowing the comparison of multiple variables [Har99].

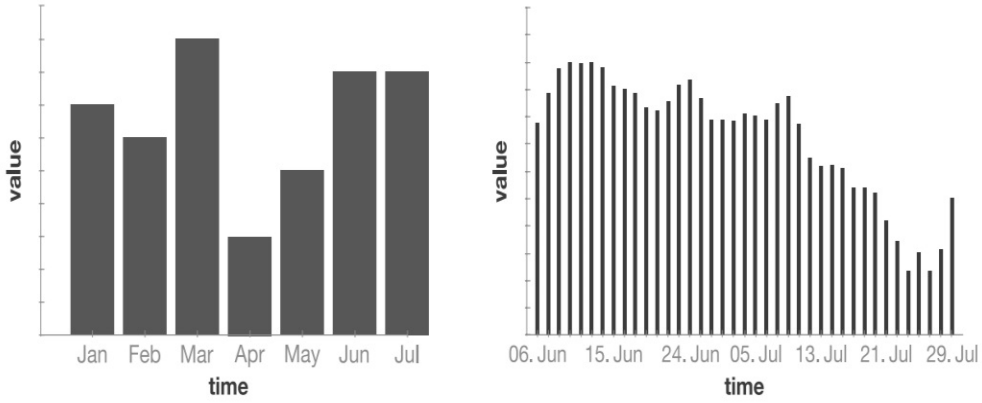

Figure 2.2: A bar chart (left) and a spike chart (right) [AMST11].

Line Plot

frame of reference	abstract
variables	univariate
arrangement	linear
time primitives	instant
mapping	static
dimensionality	2D

Table 2.3: Characteristics of the Line Plot

The line plot is considered the most common technique to visualize time-oriented data. In contrast to a scatter plot, the line plot connects the data points and can consequently help to visualize trends over time better. To create a continues line from a given list of data points, however, it should be mentioned that the values between two consecutive points are approximations and do not represent real data points. How these values are connected can happen over different methods. Two of these methods to connect two points are straight line and Bézier curves as seen in Figure 2.3. Which method to choose will vary depending on the concrete problem and desired output [Har99].

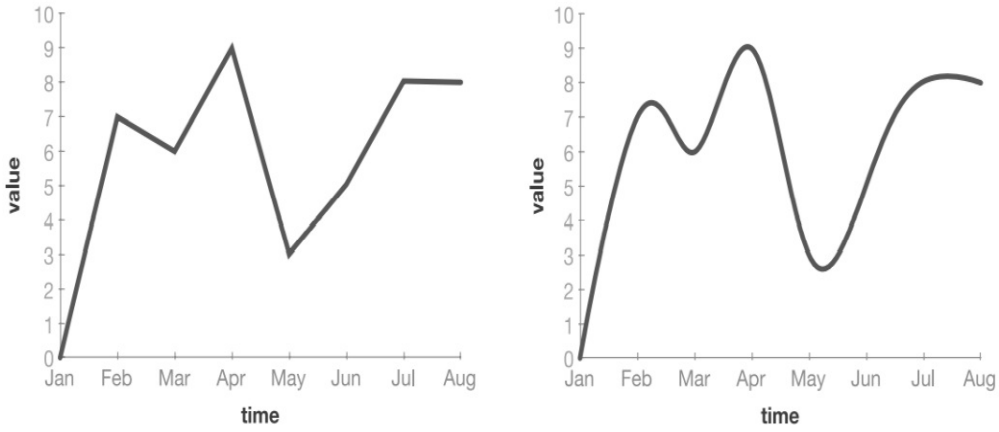

Figure 2.3: Visualization of line plots. The left plot shows straight lines, while the right plot uses Bézier curves [AMST11].

SparkClouds

frame of reference	abstract
variables	univariate
arrangement	linear
time primitives	instant
mapping	static
dimensionality	2D

Table 2.4: Characteristics of SparkClouds

SparkClouds help visualizing trends for different keywords. The keywords appear larger, the more important they are. Opposed to classical word clouds, SparkClouds can also give an indication of the evolution over time of keywords. It allows for a visual appealing overview of trends but lacks qualitative data to precisely compare keywords and their trends. As shown in Figure 2.4, the 25 most relevant keywords for a given time are shown colored, while the other keywords appear grayed out. There is also a variation, where the keywords that did not make the top 25 are left out instead of being grayed out [LRKC10].

Figure 2.4: SparkCloud highlighting the top 25 keywords for the last time point of a series while the other 50 keywords appear grayed out [LRKC10].

ThemeRiver

frame of reference	abstract
variables	multivariate
arrangement	linear
time primitives	instant
mapping	static
dimensionality	2D

Table 2.5: Characteristics of *ThemeRiver*

ThemeRiver was developed by Havre et al. [HHWN02] within the context of press article to visualize trends for keywords appearing in articles (Figure 2.5) . It contains several area plots that each represent a single topic. Consequently does the width of an area represent the number of occurrences of said topic and affect the overall appearance of the *ThemeRiver*. As a result, it helps in giving an overall view over given trends.

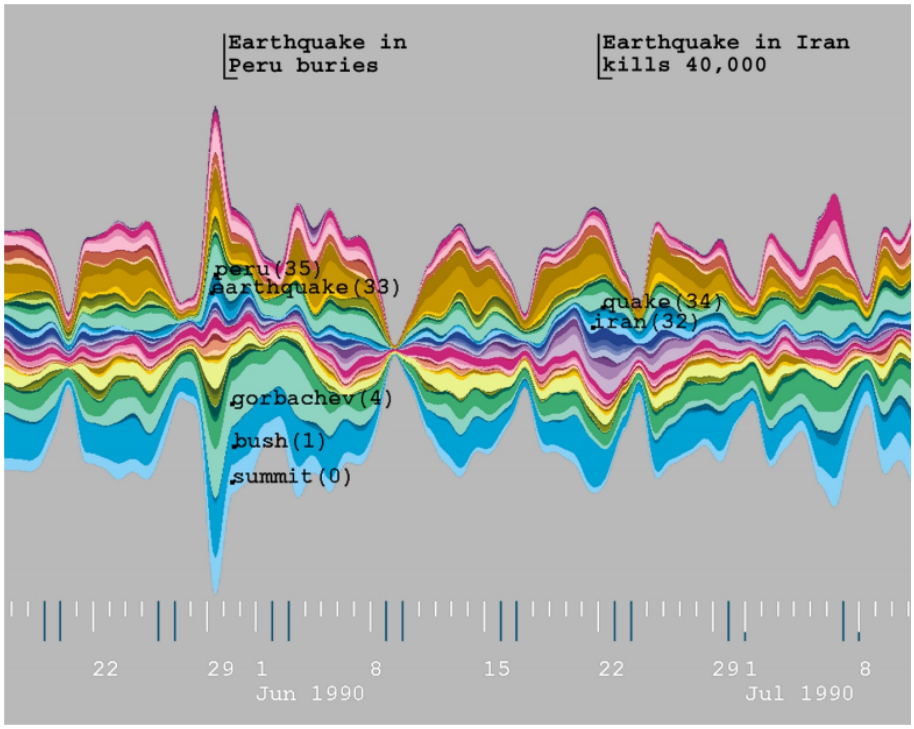

Figure 2.5: *ThemeRiver* visualizing press articles from June - July 1999 [HHWN02].

Spiral Graph

frame of reference	abstract
variables	univariate, multivariate
arrangement	cyclic
time primitives	instant
mapping	static
dimensionality	2D

Table 2.6: Characteristics of the Spiral Graph

Developed by Weber et. al [WAM01], the spiral graph represents a novel method to visualize cyclic data. As seen in Figure 2.6, the data is mapped along the spiral and coded as color, thickness or texture. Spiral graphs are especially useful when dealing with time-oriented data that shows periodic behaviour as it helps in finding patterns [AMST11].

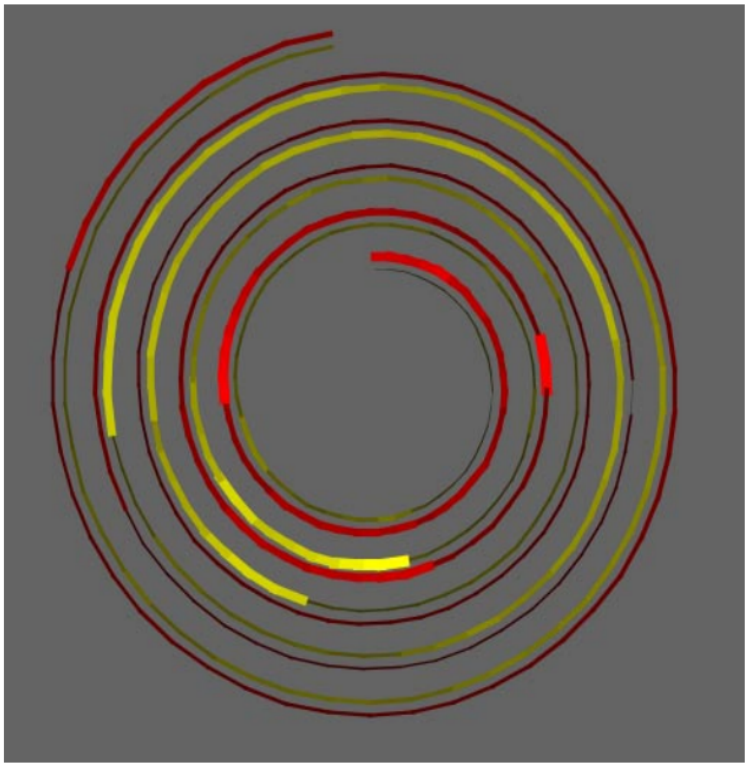

Figure 2.6: A spiral graph showing the stock prices of *Microsoft* (yellow) and *Sun Microsystems* (red) over five years [WAM01].

TrendDisplay

frame of reference	abstract
variables	univariate
arrangement	linear
time primitives	instant
mapping	static
dimensionality	2D

Table 2.7: Characteristics of the TrendDisplay

The TrendDisplay technique was presented by Brodbeck and Girardin [BG03] in 2003 and supports in analyzing large amount of information to summarize trends in both space and time. In the presented design study, TrendDisplay was used in the context of drug discovery to find anomalies within high-throughput screening data (Figure 2.7).

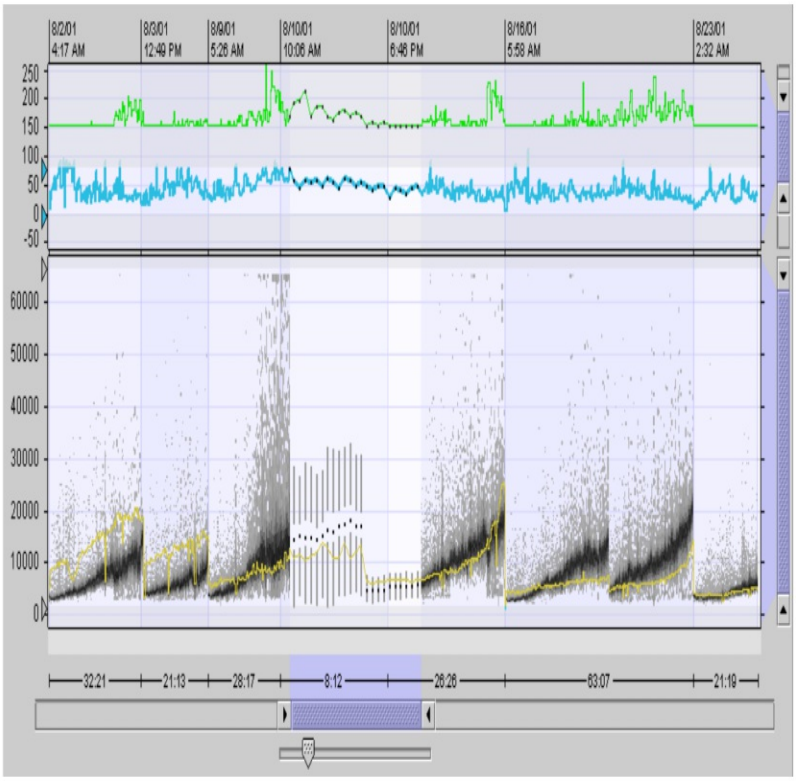

Figure 2.7: TrendDisplay visualization by Brodbeck and Girardin [BG03]. The top panel is showing derived values such as the inhibitor reactions (green) and the standard deviation (blue) while the bottom panel contains the raw data (drug discovery data).

Silhouette Graph, Circular Silhouette Graph

frame of reference	abstract
variables	univariate
arrangement	linear, cyclic
time primitives	instant
mapping	static
dimensionality	2D

Table 2.8: Characteristics of the (Circular) Silhouette Graph

Silhouette Graphs show similarities to area charts as they draw plotted lines and fill their areas. One of the major differences, however, is, that Silhouette Graphs focus on longer time spans and side-by-side comparisons. There exist different variations to present the series, as the circular silhouette graph, where the time-series are mapped on concentric circles (Figure 2.8) [Har99].

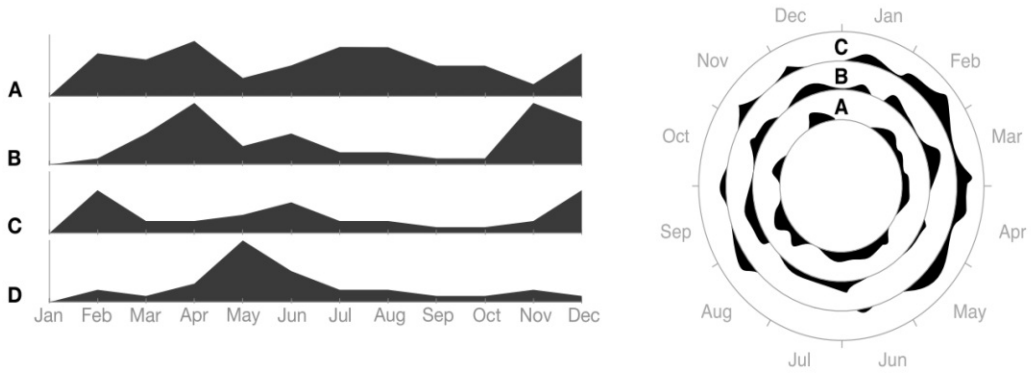

Figure 2.8: Silhouette graphs to compare multiple time-series data mapped on a horizontal axes (left) as well as on concentric circles (right). This form of visualization allows for easier comparison between multiple data sets [AMST11].

Tile Maps

frame of reference	abstract
variables	univariate
arrangement	linear, cyclic
time primitives	instant
mapping	static
dimensionality	2D

Table 2.9: Characteristics of Tile Maps

Tile maps were introduced as a visualization technique by Mintz et. al [DTM97] in 1997 and encodes individual days as shade or color on a grid containing cells. As seen in Figure 2.9, each row represents a day of the week and the columns indicate the calendar weeks. By then putting the measured values into a grid, it is possible to detect long-term trends as well as cyclic behaviour.

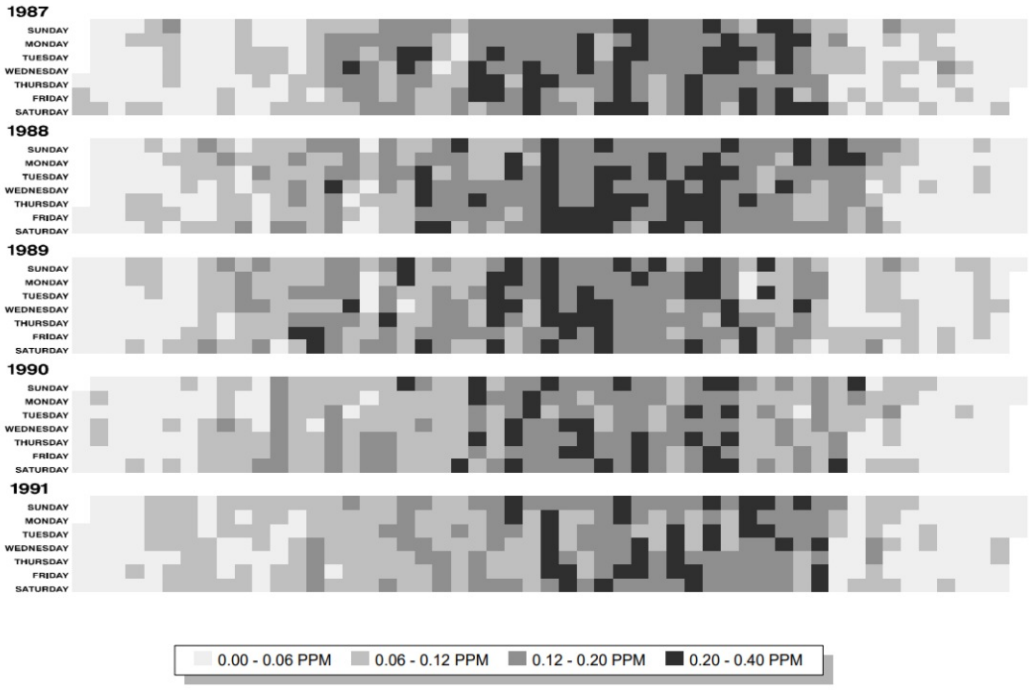

Figure 2.9: Tile maps showing the ozone measurements for Los Angeles from 1987 until 1991. Each tile represents a day of the year and the tiles are organised as a grid, representing a calendar [DTM97].

2.5 Benefits and Limitations

The techniques and technologies mentioned in this chapter are an indication for a high amount of literature in the area of visualizing trends. *ThemeRiver*, for example, outlined an intuitive solution to quickly identify spikes over a long time-span but lacks support for direct comparison of rivers. Different approaches exist for different problems and the variety of existing techniques indicates, that it might be difficult to apply a solution of a given problem onto a similar problem. New problems usually tend to rely on a modified solution as well.

A set of common visualization techniques for generic problems has, however, already been established and provides are core set of utility to built upon. The concrete implementation will then heavily depend on the data, that is being worked with. Not only is the amount and structure of data important, but also does the context play a significant role when choosing the correct type of visualization.

In the context of data originating from version control systems, the solutions are mostly based on simple charts, without much room for exploration and interaction. Especially when trying to identify trends, it is fundamental to provide more extensive tool to support users in their tasks.

To work with large data-sets within the context of *GitHub*, it is therefore needed to identify the concrete tasks and find techniques that enable the specified user group to work with said data. This might require adapting existing methods and extend their concept to fully utilize them.

CHAPTER 3

Problem Statement

This chapter focuses on the problem characterization and the definition of the major factors for the project. To do so, the Data-Users-Tasks triangle by Miksch et. al [MA14] as shown in Figure 3.1.

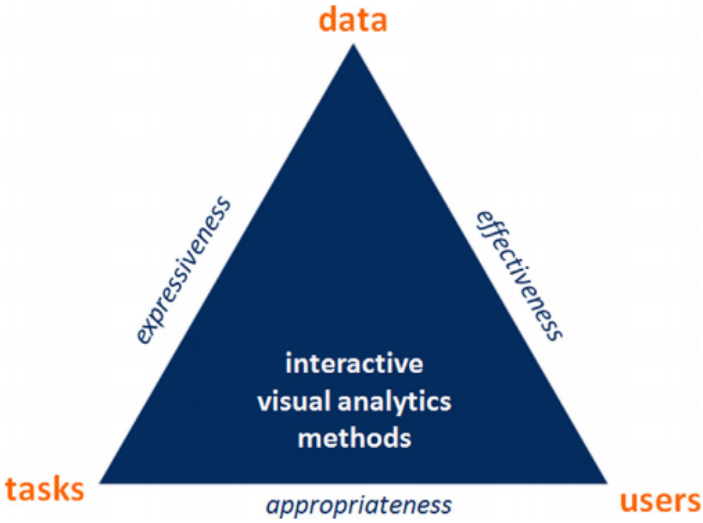

Figure 3.1: The design triangle as described by Miksch et. al [MA14].

- Data: What kind of data will the users work with?
- Users: Who is using the *Visual Analytics* solution?
- Tasks: What are the tasks the users want to perform?

Depending on the analysis of the data, users and tasks, different methods, visualizations and techniques can be derived to fulfill the requirements of the solution. In the evaluation, the tasks are then mapped against the requirements for the solution.

3.1 Data

The data used contains all the events happening on *GitHub* in the year 2020. They come as quantitative, abstract and multivariate events. They are linear and time based instants. The individual fields are described in Table 3.1 and only contains the fields needed for the course of this project. The original data contains more fields, but they are discarded for performance and memory reasons.

3.1.1 Structure

Attribute	Type
Timestamp	ordinal
Repository	nominal
Actor	nominal
Event type	categorical
Action (for issues)	categorical

Table 3.1: *GitHub* data attributes

The timestamp comes as UTC string and gives a geographically independent time of when that event happened. The repository gives the name of the repository that this event refers to. The actor is the user that issued that event. The event is one of a given list and can contain an unstructured optional payload containing additional information only relevant to that kind of event. One of the fields inside that payload is the action for issue events. As the name suggests, that action field only exists for issues and contains the activity that has been performed on that issue (e.g., open/reopened/closed). An entire list of the possible events and their fields can be found in the *GitHub* documentation [Ghe].

3.1.2 Limitations

Due to the large amount of data and limited available computation power, the events have been restricted to 2020 only. Some used metrics, like the global averages were previously calculated as they can not be calculated in real-time during the evaluation. The requirement of reading a high amount of nodes and edges made it impossible use the *GitHub* API [Api] and the huge amount of data transferred and the consequential pricing ruled out *BigQuery* as an alternative. Consequently, the archived data from *GitHub Archive* [Gha] has been used and processed locally to allow for acceptable computation times and usability. This limits the extensive data to a manageable data-set usable for a prototype.

3.2 Users

The main group of users spreads across multiple categories. The most important ones, however, are software developers, recruiters and sociologists who are interested in analyzing current and past trends related to repositories and users. The solution is consequently designed in a way, that it can also be used by users who are not familiar with the underlying technologies or *Visual Analytics* software.

3.3 Tasks

There are a variety of tasks, that can be fulfilled with this system. For the further description of the tasks, they will be organised into abstract high-level tasks and technical low-level tasks [AMST11].

For this thesis the high-level tasks include:

- details-on-demand
- gaining an overview
- filtering capabilities
- zoom on specific regions

The low-level tasks, however, include:

- finding overall trends
- direct lookup of specific repositories
- getting insight into the rate of change
- identify patterns in users

A concrete example could be the evaluation of the impact of pandemics (e.g., *COVID-19*) on developers and their software projects as well as providing assistance in predicting further development of open source projects.

3.4 Requirements

The expected outcome of this thesis is to provide a user-friendly solution for the visualization of *GitHub* data, The solution should be usable by domain experts and provide a versatile product to visualize different trends. In contrast to existing solutions, the outcome is supposed to enable user input to change relevant metrics (e.g., time span).

To be more precise, different views should enable the user to derive different information based on the task. All the used data originates from the *GitHub* API, but they come as archived data to improve performance and usability while relying on the original events.

The software is supposed to be capable of solving many different tasks, but will be narrowed down to specific problems to limit the scope of the product. The temporal aspect will, however, be included in all the views where time plays an role.

The implemented tasks should not cover all possible tasks related to trends, but provide a foundation and reference point for future research. The emphasis will rely on social and technological aspects.

3.4.1 List of Requirements

The planned requirements can be defined as follows:

R1: Comparing repositories

From a technological point of view, analyzing repositories is an interesting task. Especially for repositories that fit into similar areas, it can be useful to identify potential trends of the competitors. Such information can help in the decision-making on choosing the appropriate technology stack.

R2: Repository details

The user is able to view the temporal change for specified repositories. For this, there is a view providing insight into the activities over a given time span. This can be helpful to analyze if a given repository is getting more attention or is stagnating.

R3: Activity of users on repository

The system is capable of showing the activity of the user base in a given time span. This can potentially be used by sociologists to analyze the impact of pandemics, like *COVID-19* on open source projects or by business analysts to find patterns in productivity.

R4: Concrete user activity

It is possible to select a single user from the user base and get a more detailed view for that user over a repository. The system will visualize the different events related to said user. Doing so could help recruiters throughout the hiring process and give project managers an overview of their most active users.

R5: Issue life-cycle

Given a specific repository, the issues being opened and closed over time can be visualized. This can be used to derive if the amount of created being closed is higher than the

amount of issues being opened. That information is often useful when evaluating the development state of a project (e.g., alpha/beta releases).

R6: Issue activity

The activity related to issues can be used to indicate whether there is a lot of discussion and brainstorming going on for implemented functionality. As a result, the amount of comments can also be visualized by the tool.

CHAPTER 4

Visualization Design

This chapter gives an overview over the decisions made for the visualization design. The chosen components, controls as well as visual encodings will be highlighted in more detail as part of this chapter.

The visualization is split into two boards, the repository board and the issues board. As the names suggest, each of them focus on a different type of data and they can be seen as independent boards. Both boards share a common toolbar that allows navigation between the boards.

4.1 Issues Board

The issues board as seen in Figure 4.1 helps identifying patterns for a specific user or repository over time. This board provides controls to limit the months, the user, the repository and the weekdays. The visual components contain two coordinated views that share a common horizontal axis and zoom state.

4.1.1 Controls

There are overall four controls to limit the query:

- Time span: Separate iterations and discussions with experts have shown, that filtering for the time span in an important factor when identifying trends. To increase usability and reduce complexity, the granularity of the time span was limited to cover only a selection of months and not specific days.
- Repository: This control allows to limit the issues to a given repository. Selecting either a repository or a user is mandatory.

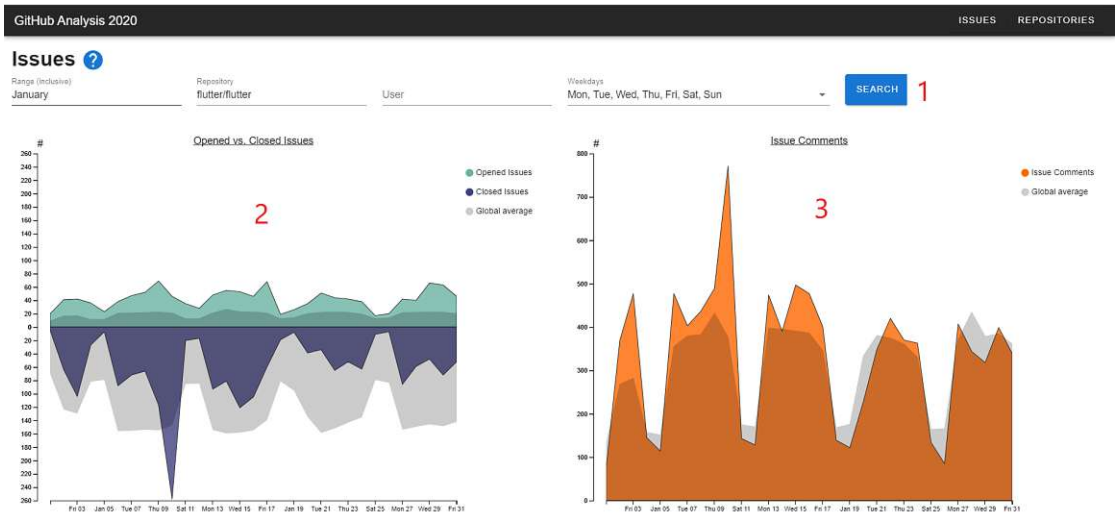

Figure 4.1: The issues board for the repository *flutter/flutter* in January 2020 with controls (1), the opened/closed issues (2) and the issue comments (3). The grayed out areas indicate the global average.

- User: This control allows to limit the issues to a given user. Selecting either a repository or a user is mandatory.
- Day of the week: This control is a multi-select that allows to filter out given days of the week. It can for example be used to only show the issues that happened on weekends.

4.1.2 Opened vs. Closed Issues

This component shows the closed and opened issues for the given parameters as an area chart. It should be mentioned that there are more actions related to issues (assigned, labeled, reopened, unlabeled, unassigned) but closed and opened issues turned out to be the most interesting and relevant ones for an effective visualization.

As seen in Figure 4.2, the horizontal axis represents the time axis and can be additionally limited by selecting the desired area in the chart, causing a zoom on this chart as well as the issue comments. The vertical axis, however, is split in two halves: The opened issues and the closed issues. Both area charts are shown in a mirrored position and always appear on a common vertical and horizontal scale. This allows for a better comparison across them and supports finding patterns. Making it an area over a simple line also improved readability and comparability.

Each of the main areas also includes a gray area that indicated the global average across *GitHub* data. This became necessary as it was difficult to tell if the identified patterns are considered to be expected behaviour (e.g., less activity on weekends) or if there was

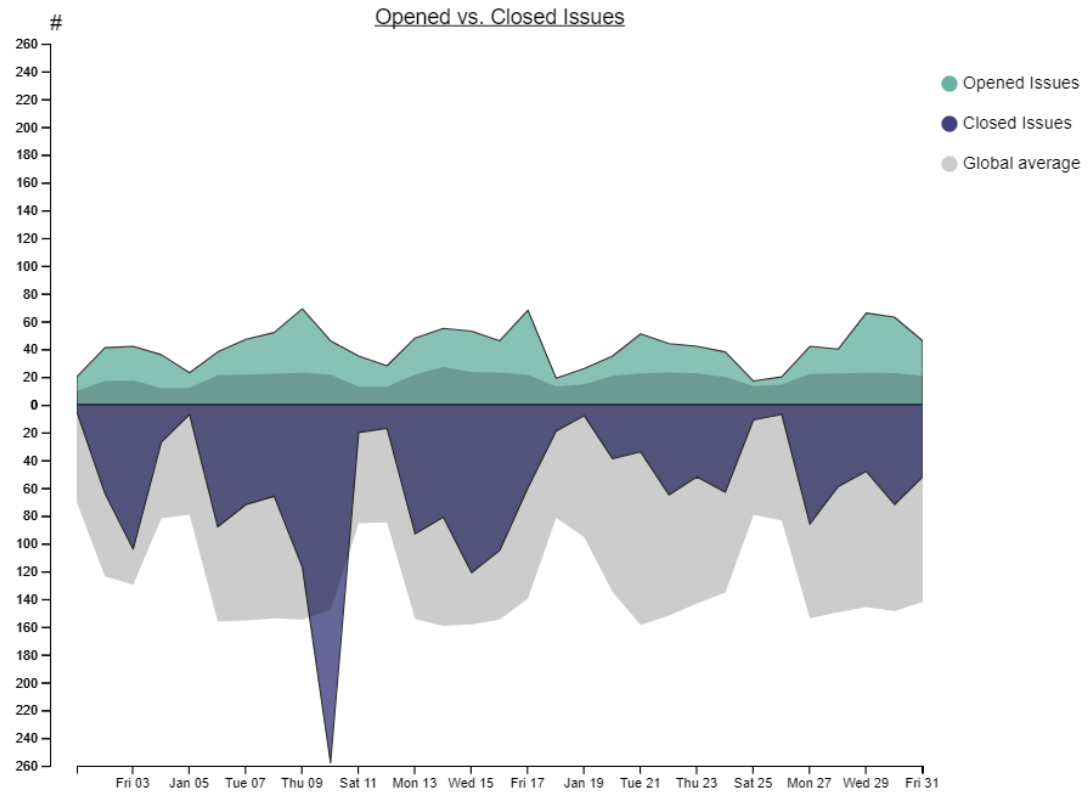

Figure 4.2: Area chart showing the closed (bottom) as well as the opened (top) issues for *flutter/flutter* in January 2020. The grayed out areas indicate the global average.

a trend. The global averages are normalized alongside the visualized data, making it a relative reference.

4.1.3 Issue Comments

For the issue comments, as seen in Figure 4.3, a similar approach has been chosen as for the closed and opened issues. It is shown as an area chart and also includes a relative global average it can be compared to. The connected zoom capabilities also apply to this graph, making the user experience more consistent. A separate graph was chosen here as the values on the vertical axis of the opened and closed issues tended to be very different from the ones of the issue comments.

4.2 Repositories Board

As seen in Figure 4.4, the repository board gives an overview over selected repositories for comparison. It shall give insight in the overall progression of repositories and allow

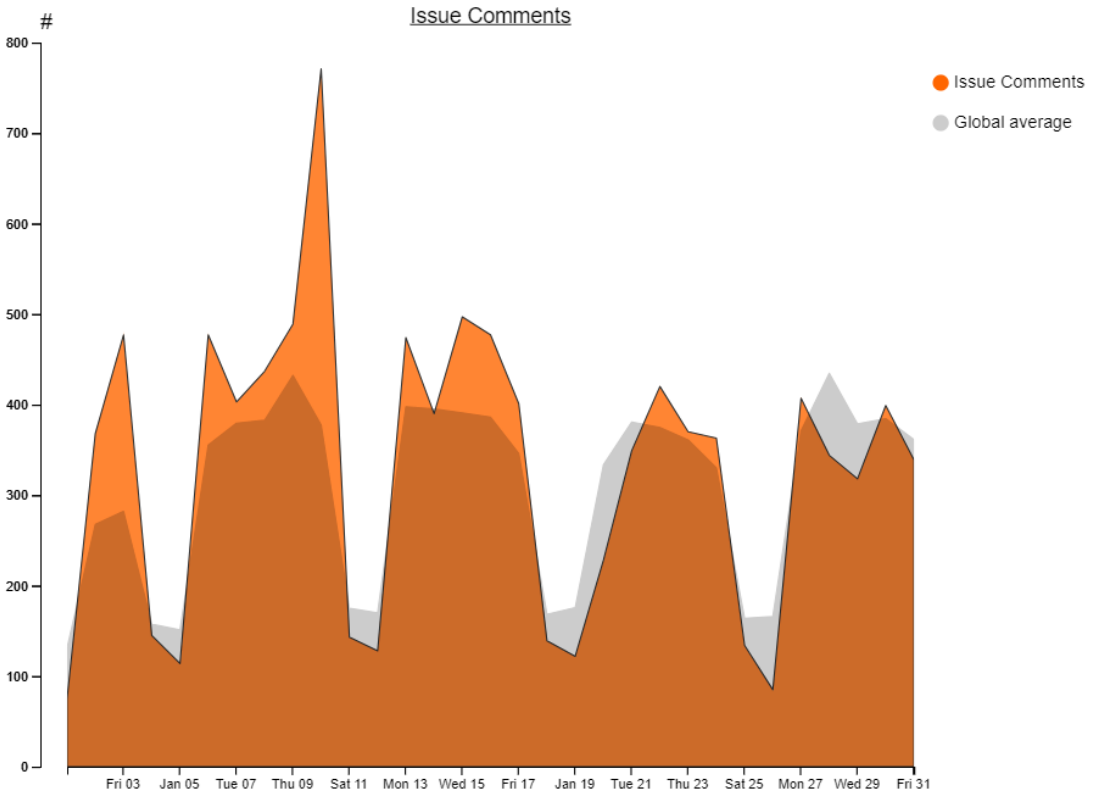

Figure 4.3: Area chart showing the issue comments for *flutter/flutter* in January 2020. The grayed out areas indicate the global average.

for more detailed information for a selected month.

4.2.1 Controls

For this board, there only exists one control which is the repository filter. That filter allows to select the list of repositories that should be visualized. The selected repositories are represented as *Chips* in the user interface to allow for easy removal and readability for the chosen selection.

4.2.2 Technical and Social Activity

This chart is a variation of the scatter plot to allow for a comparable overview of different metrics. The main idea of this component is to map the concrete *GitHub* events onto more abstract and general dimensions. To do so, every month for a repository comes with its unique scatter on the chart and all months of a repository are connected to each other. That means, that each repository comes with twelve data points that are connected from January to December. If the repository was created after the January

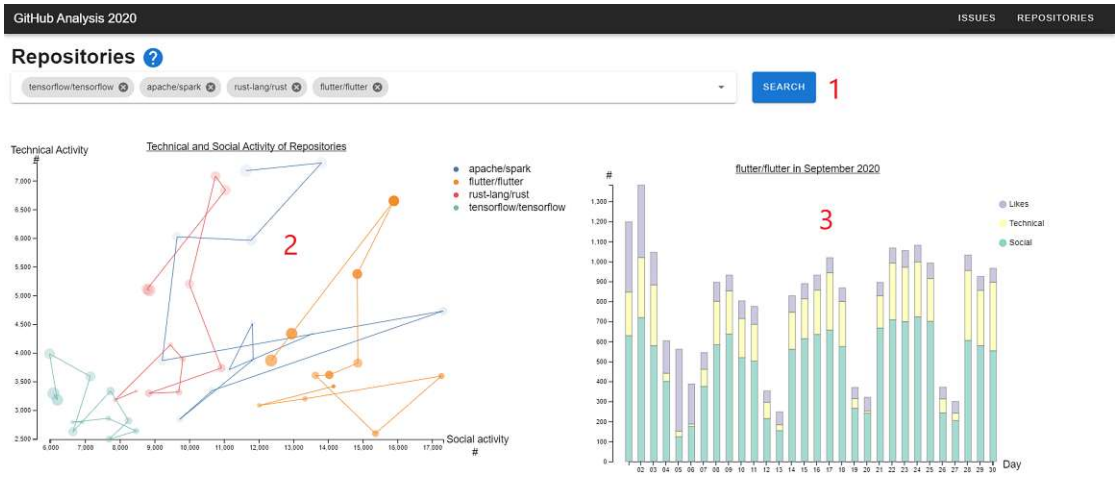

Figure 4.4: Repositories board for 2020 including the repository filter (1) comparing *tensorflow/tensorflow*, *apache/spark*, *rush-lang/rust* and *flutter/flutter* (2) with a detailed view over *flutter/flutter* in September 2020 (3).

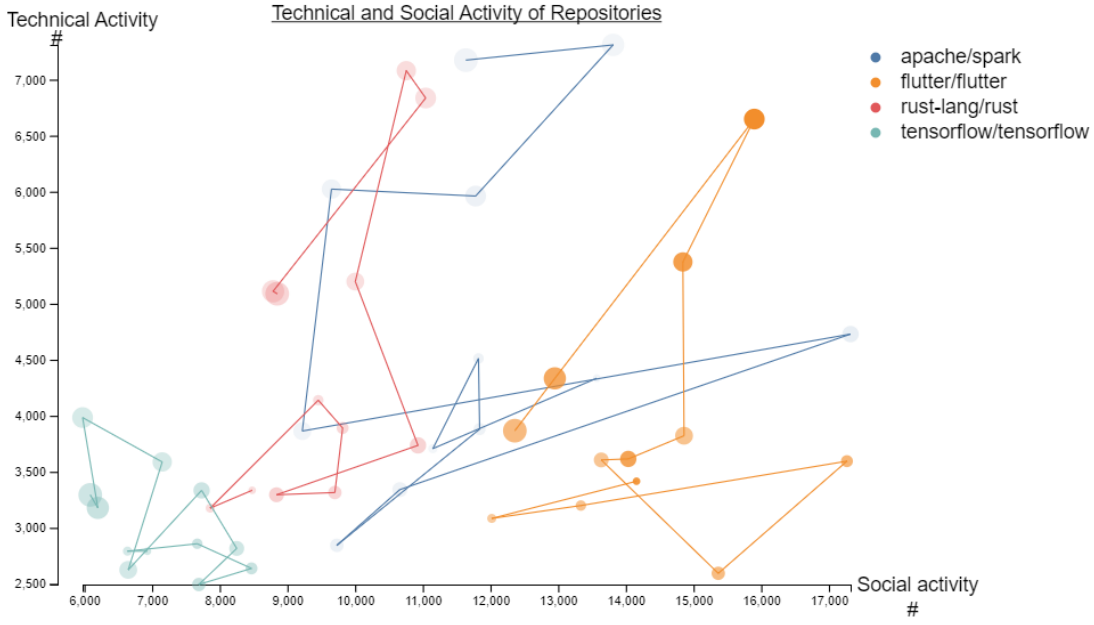

Figure 4.5: Comparison of *tensorflow/tensorflow*, *apache/spark*, *rush-lang/rust* and *flutter/flutter* in 2020 with each month being represented by one data point. The size of the data points represents actuality and the opacity encodes the relative likes.

2020, there might be less data points included in the visualization. The repositories are distinguished from each other by their color.

Usage of this component can be seen in Figure 4.5 where the horizontal axis represents the social activity of the repository during that month while the vertical axis is used for the technical activity. Details on the mapping of *GitHub* events to technical and social events are defined in Chapter 5. To easier tell the direction of the scatters for a repository, the months have been color coded by size to represent their actuality. Meaning, that data points from earlier in 2020 appear smaller than those that happened later in 2020. Another mapped dimension is the number of likes, which was excluded from the social activity intentionally. The likes are consequently mapped to the opacity of the data points.

When hovering over a data point, a tool-tip is shown to give detailed information of the social and technical activity as well as the likes, the month and the name of the repository. When selecting one of the data-points, a more detailed bar chart appears for that given repository in the selected month.

4.2.3 Monthly Repository Details

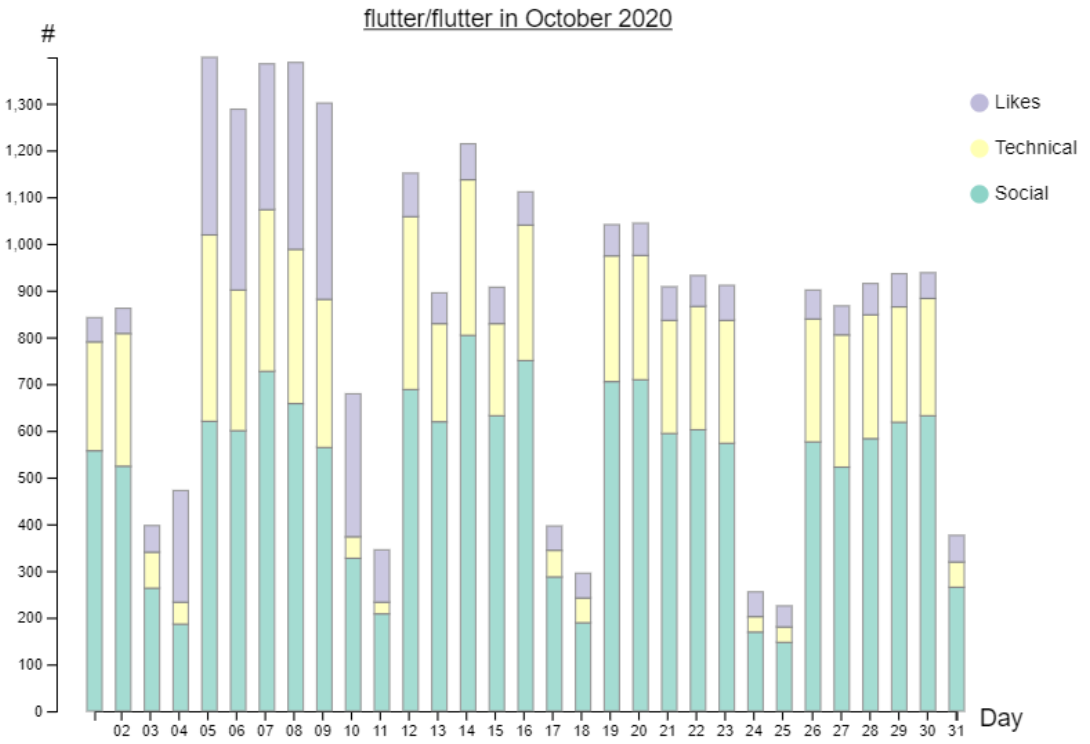

Figure 4.6: Visualization of the technical activity, the social activity and the likes for flutter/flutter in October 2020.

When selecting a data-point in the chart for social and technical activity, a detailed bar chart appears next to the existing chart. That bar chart is slightly smaller than the other

chart, as it is not as much in the foreground and should mainly help the understand why specific months might look as they do.

Consequently, this component, as seen in Figure 4.6 contains one bar for each day of the selected month. Each day, however, shows a stacked bar containing the social activity and technical activity as well as the likes. The legend on the side also indicated the color coding and its entries can we selected and deselected to remove the chosen dimension from the bar chart. So is it possible to only show the likes for the repository in this month by deselection the social and technical activity and simplify the comparison of the likes.

The stacked bars were chosen here because they allow for an easy comparison between the visualized metrics, while still being able to fit all the days of a month within the horizontal axis.

4.3 Alternative Visualizations

In earlier versions of this components, there were twelve individual scatter plots, one for each month, containing all the repositories each as seen in Figure 4.7. Feedback and discussions with experts, however, led to the decision to merge them all into one chart to allow easier comparison and easy the identification of trends for specific repositories. It felt very difficult to identify any significant change between the months. While it gave a good overview for a given month, it was hard to tell any changes as the bigger picture was missing.

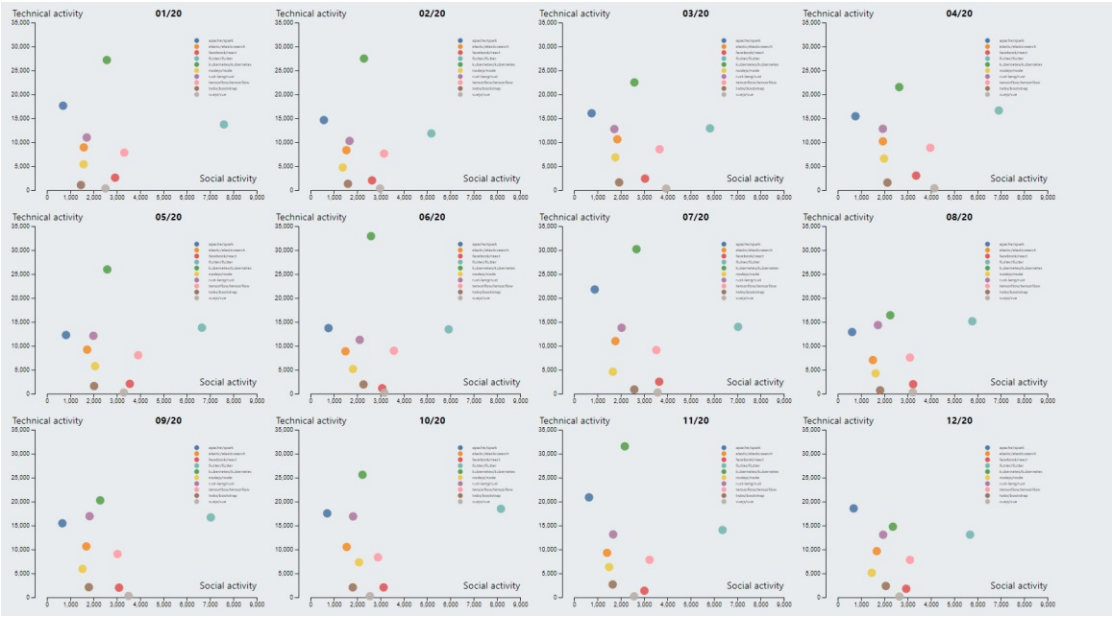

Figure 4.7: Earlier version of visualizing the monthly change of repositories.

Another scrapped version contained the *ThemeRiver* approach to visualize and directly compare the repositories in their metrics. But similarly to the individual scatter plots, it turned out that *ThemeRiver* makes it very difficult to derive precise numbers and provide a sufficient feeling for the data presented. An example of how this looked is shown in Figure 4.8 where it was difficult to identify a real trend as a single river is not fixed to the horizontal axis but instead requires to analyze the relative height of the river. It additionally turned out that finding trends is easier when the single topics can be analyzed individually and are not mixed together as one entity.

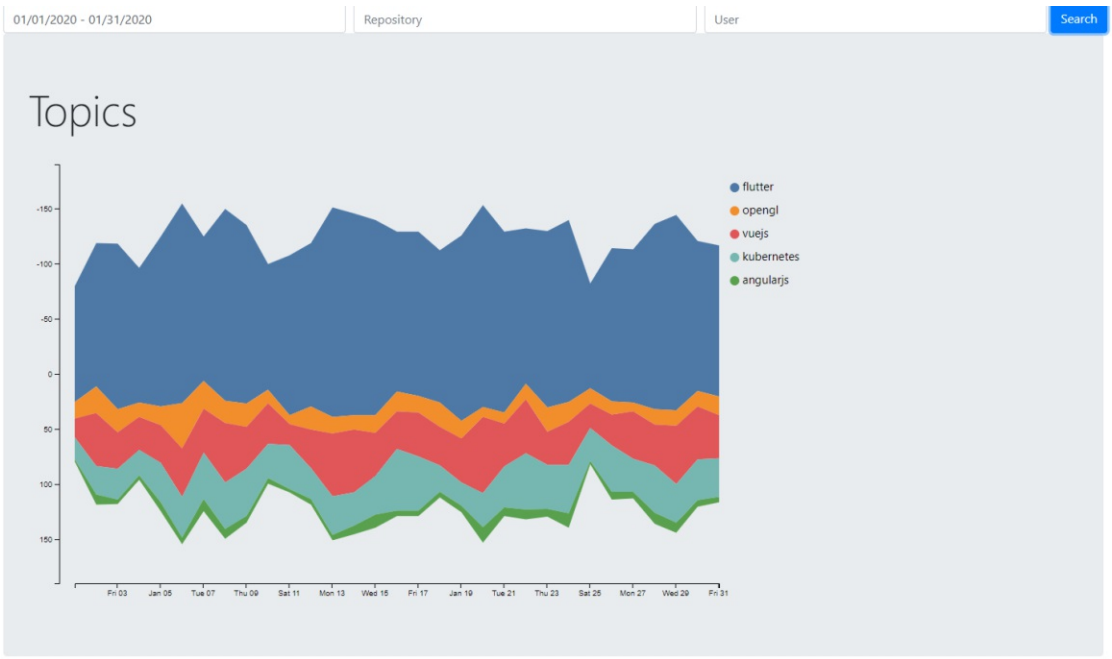

Figure 4.8: Repositories visualized as *ThemeRiver* in a scrapped version.

The last early board worth mentioning was a large scatter plot where each day of the year was represented by a single scatter. The opacity of all scatters was lowered to allow better readability and the color indicates if that day happened to be early (green) or late in the year (red). That form of chart, as shown in Figure 4.9 visualized the days of the repository *flutter/flutter* in 2020. But similar to the other scrapped visualizations did not provide a satisfying way to identify specific trends.

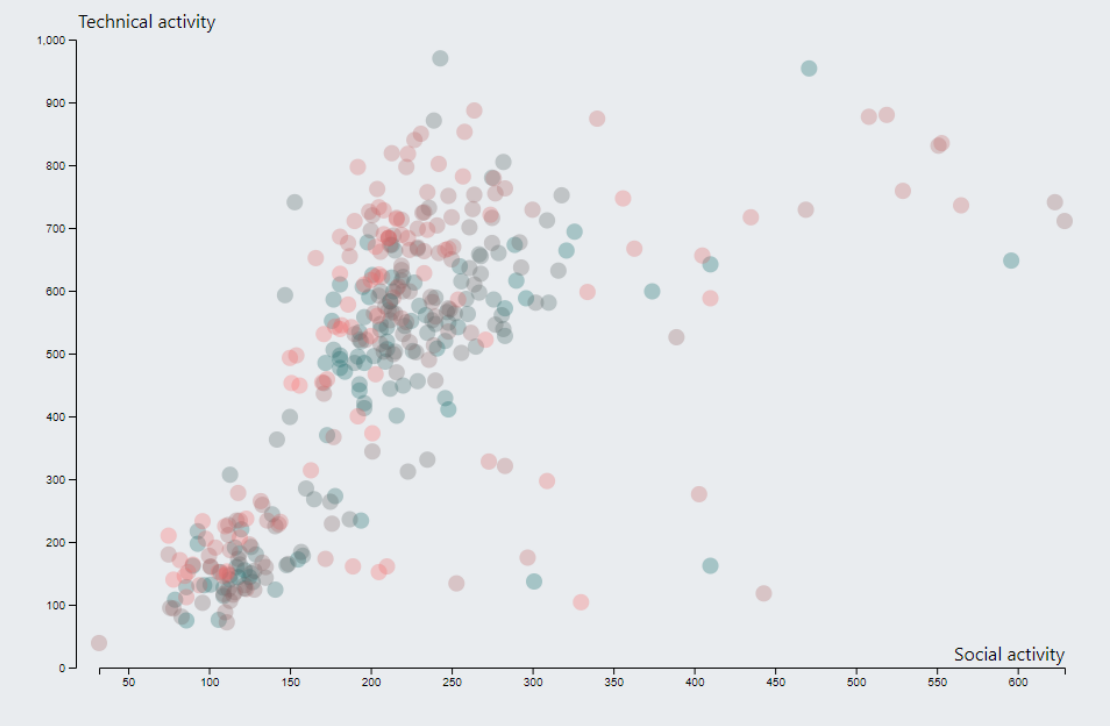

Figure 4.9: Scatter plot for number of social and technical events for *flutter/flutter* in 2020. Each scatter represents a single day in the year and the color indicates how late in the year that day way, while green was early in the year and red was later in the year.

CHAPTER 5

Prototype Implementation

The aim of this chapter is to describe the technical details for the implementation of the prototype. This includes the general architecture, the frontend and backend specific concept as well as the overall pipeline to get efficient queries from the originally large data-set.

5.1 Backend

In this section, the decisions for the backend will be mentioned and explained. This includes the original data we are working with, as well as the required technologies.

5.1.1 Data

As the year 2020 includes over 872 million *GitHub* events, the architecture for the prototype had a strong emphasis on performance and efficient processing. As the *GitHub* API allows only a limited amount of requests, the events are loaded from *GitHub Archive*[Gha]. *GitHub Archive* provides the *GitHub* events from the *GitHub* API in an archived version which simplifies processing for analytical reasons. The data is offered as a BigQuery data-set as well as *JSON* files that can be requested with HTTP. As the BigQuery data would require constant requests and process several terabytes of data, resulting in high cost, the *JSON* files have been downloaded and loaded into a local database. To keep the *JSON* files isolated and compact, they are provided on an hourly basis, meaning for the year 2000, there are theoretically 8784 files to be downloaded and processed. There, however, are single days when no data exists for different reasons, these are consequently also missing in the prototype.

An example for an entry in such a *JSON* file can be seen in Listing1, where an event for the creation of a repository can be seen. It should be mentioned, that the *payload* field

```
{
 "id": "2489651310",
 "type": "WatchEvent",
 "actor": {
  "id": 6376156,
  "login": "shenjiayu",
  "gravatar_id": "",
  "url": "https://api.github.com/users/shenjiayu",
  "avatar_url":
   ↪  "https://avatars.githubusercontent.com/u/6376156?"
 },
 "repo": {
  "id": 2325298,
  "name": "torvalds/linux",
  "url": "https://api.github.com/repos/torvalds/linux"
 },
 "payload": {
  "action": "started"
 },
 "public": true,
 "created_at": "2015-01-01T15:00:28Z"
}
```

Listing 1: Exemplary *JSON* data for the event of the user *shenjiayu* watching the repository *torvalds/linux* [Gha].

is specific to each type of event and can consequently vary heavily across the different events.

5.1.2 Technologies

Database

The chosen solution for the database was *SQLite* [sql] as it is very efficient and lightweight compared to other database systems. The main disadvantages of *SQLite*, including the missing multi-user support as well as slow concurrent writing operations, were no concern for the prototype as there are no more write operations after the initial setup and for the sake of the evaluation, multi-user support was not required.

Web Framework

For the web framework, *Django* [dja] has been used, at it provides an easy setup and allows for complex queries with decent optimization mechanisms. *Django* is based on

```
class RepositoryEvent(models.Model):
 created_at = models.DateField()
 event_type = models.PositiveSmallIntegerField()
 action = models.PositiveSmallIntegerField(null=True)
 actor_login = models.CharField(max_length=39)
 repo_name = models.CharField(max_length=100)
```

Listing 2: The data-model for the *Django* [dja] application without indices and helper functions.

Python and comes with its own database migration system, allowing generating tables from a *Python* data-model. A simplifies version of the data-model can be seen in Listing 2 as it does not include the indices and functions. Keep in mind that the *action* field is optional, because it only exists for events that are related to issues. *Django* provides a way to define *HTTP* endpoints that can be called either synchronously or by AJAX. For the course of this prototype, the *HTML* pages based on the *Django* template language are provided via *HTTP* and the functions executing the queries use AJAX and consequently return the result-set to the frontend.

5.2 Frontend

This section focuses on frontend related technologies. In case of the developed prototype, this includes the template language, the framework used for more efficient data manipulation and the library for an improved user interface.

5.2.1 Django Template Language

As already mentioned in the previous subsection, the basic template structure is provided by *Django*. It comes with a simple language to combine *HTML* with programming code and allows for concepts, like inheritance and conditional elements. The main idea of it is to provide an easy way to serve *HTML* files in a *Django* application while still allowing common scripting capabilities with JavaScript. An example of the *Django* template language can be seen in Listing 3, where one of the *HTML* files extends *base.html* which is responsible for showing the common header. Other files can consequently extend said file and access the inherited components.

5.2.2 Vue.js + Vuetify

As the *Django* template language does only come with limited functionality, *Vue.js* [vue] was chosen to extend the templates with advanced JavaScript capabilities. *Vue.js* is relatively light-weighted compared to other frontend frameworks, while still providing a sufficient tool-set for the course of this prototype. It is based on JavaScript but

```
{% extends 'base.html' %}
{% block content %}
 <div id="content"></div>
{% endblock %}
```

Listing 3: Simplified example of the *Django* template language with a file extending another file.

```
<v-combobox v-model="repos" :items="repos" chips multiple
    solo>
    <template v-slot:selection="{attrs, item, select,
        selected}">
        <v-chip v-bind="attrs" :input-value="selected" close
        click="select" click:close="remove(item)">[[item]]
        </v-chip>
    </template>
</v-combobox>
```

Listing 4: Combo-box in Vue.js[vue] and *Vuetify* [vtf] for adding and removing repositories as chips.

provides convenient methods for common problems, like two-way binding as well as state management.

Vue.js does, however, only offer tools to deal with programming logic and provides no visual elements by itself. To give the prototype a better look, *Vue.js* was extended by *Vuetify* [vtf] which is a user interface library based on material design created specifically for Vue.js. Listing 4 shows the combo-box from *Vuetify* that has been used to provide a multi-select drop-down for the repositories inside the application. It binds to the *Vue.js* field *repos* and adds functionality to add selected chips as well as removing them.

5.2.3 D3

D3 [Bos] stands for **d**ata-**d**riven **d**ocuments and is an open-source library based on *JavaScript*. It is mainly used for visualizations as it allows for *Document Object Model (DOM)* manipulation while using *SVG*. This is accomplished by providing utility functions that help to pars and visualize high amount of raw data while abstracting the complexity away from the user.

Listing 5 shows an example of how *D3* was used in the prototype to parse the original data coming from the database into a horizontal value line and styling it as an area.

```
// Create a reference for the vertical axis
const y = d3.scaleLinear().range([height / 2,
 ↪  0]).domain([0,maxOpened]).nice();

// Add the axis to the document
svg.append("g").attr("transform",
 ↪  "translate(-20,0)").call(d3.axisLeft(y));

// Create the function for the area
const valueline = d3.area().x((d) => {
    this.x1(new Date(d["created_at"]));
}).y0(y(0)).y1( (d) => {
    return y(d["opened"]);
})

// Draw the data points as an area
svg.append("path").datum(data).attr("class", "y").attr("fill",
 ↪  "#69b3a2").attr("opacity", ".8").attr("stroke",
 ↪  "#000").attr("stroke-width", 1).attr("stroke-linejoin",
 ↪  "round").attr("d", valueline(data));
```

Listing 5: Simplified example of visualizing the opened issues as an area using *D3* [Bos].

5.3 Pipeline

As the available *GitHub* data offers a large data-set, there were additional steps needed to allow for a user-friendly performance within the prototype.

The pipeline consists of few steps explained in the next subsections and shown in Figure 5.1.

Figure 5.1: Visualized pipeline for parsing original *GitHub* data for further analysis.

5.3.1 Parsing to SQL

This part involves using the original *JSON* formatted objects and bringing them into a minimalist form, so they can be persisted in a database for further querying. To do so, a SQL data model generated from the *Django* model shown in Listing 2 was created.

```
# Go over every line in the \textit{JSON} file
for line in file:

    # Load data and parse it into an object
    elem = json.loads(line.decode())

    # Create object general data
    event = RepositoryEvent(
        created_at=datetime.date(year,month,day),
        repo_name=elem["repo"]["name"],
        actor_login=elem["actor"]["login"],
        event_type=event_map[elem["type"]])

    # Check for an issue and parse the action in that case
    if elem["type"] == "IssuesEvent":
        event.action = action_map[elem["payload"]["action"]]

    # Save the object in the database
    event.save()
```

Listing 6: Parsing of a *JSON* file into a *Django* data model.

Following that, the *JSON* files containing the *GitHub* data could be parsed. The function for parsing a *JSON* file can be seen in Listing 6. First, the content from the file is read line by line and decoded into a *Python* object. The next step is to read the files from said *Python* object and map the fields on the defined database model. Once this is done, the type of the event is checked and the action is also mapped in case of the event being related to issues. The final step saves the database model in the database.

2020/05/03	**2020/06/10**	**2020/08/21**	**2020/08/22**	**2020/08/23**
1	10	15	24	16

Table 5.1: Unavailable files from the *GitHub* Archive [Gha]. Every file represents one hour

The parsing function is called for every *JSON* file which is previously retrieved from a dedicated script that iterates over every possible date and queries the corresponding file from the server. The server provides one file for each hour of every day resulting in 8784 files. However, when iterating over all the files, it turned out that not every file was represented on the server and few were missing. The unavailable files are shown in aggregated form in Table 5.1. These files were also missing in the *BigQuery* data and were consequently left out of the analysis.

5.3.2 Calculate global averages

To be able to better contextualize the visualized data, global averages can help throughout the tasks. The selected and shown data can then be compared to those expected values to support finding anomalies and patterns. As this data spreads across all repositories and will not change for the year 2020 anymore, it would be inefficient as well as unnecessary to calculate them at run-time every time. Consequently, they have been calculated beforehand and saved in a separate table. This table maps the days to the number of closed and opened issues as well as the comments on issues across all users and repositories on *GitHub*. These values can then be retrieved with a simple and fast query whenever needed.

CHAPTER 6

Evaluation

This chapter focuses on the evaluation of the prototype. It will highlight on how it was conducted and what the results of the evaluation sessions were.

The evaluation is qualitative, meaning the results can not be expressed in numbers, as the conducted tasks contain aspects of interpretation. Consequently, collected data from the evaluation sessions will involve exploratory user-computer interactions and it is required to further analyze them after the evaluation sessions [Kle15] [PSM12].

Finding the right sample size and user group for the evaluation can be a complex task. It was shown, that experts are especially useful when trying to find the fit of a visualization tool within the given domain while students can be useful when aiming to evaluate usability. As it is generally recommended having a large group of users for the evaluation, it is equally important to select the correct people [KP15] [IIC+13].

Due to the ongoing *COVID-19* pandemic during the time of this work, the evaluation was conducted remotely. Observations by Çay et al. [CNY20], however, show that people feel confident with remote evaluations as they became used to work, learn and socialize from home throughout the pandemic.

Before the evaluation, there was a trial evaluation with an expert, who was also familiar with the *Visual Analytics* domain and the context of this thesis. That trial evaluation was used to get final input and feedback for the consequent evaluation.

6.1 Methodology

For the evaluation, a total of six people have been conducted. All of them are experienced developers as well as software architects and work in areas that interact with *GitHub* on a daily basis. At the time of the evaluation, three of the participants were working for *Accenture*, while the other three participants were split among different companies working in software development.

All of the evaluations were held remotely, conducted using *Zoom*[Zoo] and recorded for further analysis. The flow of the evaluations was as follows:

1. Highlight the procedure of the evaluation as well as the background for the tasks
2. Give a small introduction to the application
3. Let the participant freely explore the functionality of the prototype
4. Present the scenario and the corresponding tasks to the user
5. Observe the user doing the tasks
6. Conduct feedback

After the evaluation the recordings have been analyzed and used to make general remarks on the findings for each task.

6.1.1 Background

To imitate a real world scenario, and provide a realistic scenario, the participants were provided with the following background:

> "You are working as a Technology Architect for a major IT company. As you like pursuing modern technologies, you came across *Flutter* and got familiar with it. The leadership now asks you to promote *Flutter* for an upcoming internal open source project and you get a tool that can help you to identify trends.
>
> First, they want you to analyze similar technologies and compare their overall trends against *Flutter*. Afterwards he wants you to focus on *Flutter* directly, so you can make sure that this technology does indeed show signs of popularity and progress. Since you have already had good experiences with *jonahwilliams* when working with *Flutter*, you are asked to check if that person would fit in your work environment."

With this background in mind, the participants were presented with the concrete tasks.

6.1.2 Tasks

The tasks where defined as follows:

- **Task 1:**
  1. Out of *django/django*, *facebook/react-native* and *ionic-team/ionic-framework* how do they compare in social and technical activity?

2. Are specific month especially standing out?
3. Add *flutter/flutter* to the list, how does it compare to the others?

- **Task 2:**

1. Only looking at *flutter/flutter* is there any visible pattern in technical activity and likes?
2. Is there a month that indicates a major release?
3. If yes, during what time of the month has this most probably happened?

- **Task 3:**

1. Looking at *flutter/flutter* in January 2020 until February 2020, are there visible patterns regarding the opened and closed issues?
2. Is there a specific day that has an unusual activity?
3. If yes, is this also reflected in the issue comments?
4. Looking on the issue comments from March 2020 until May 2020, is there an indication for a discussion over a weekend?

- **Task 4:**

1. Looking at *flutter/flutter* and the user *jonahwilliams* in April 2020 and May 2020, are there visible patterns regarding the opened and closed issues?
2. Is there a specific day that has an unusual activity?
3. Does that user show a different pattern than the global average? If yes, what is the difference?
4. Do issue comments and closed issues have similar patterns?
5. Looking at November 2020 and December 2020, is there a global trend for the time around Christmas? Does that user follow that trend?

These Tasks are also used to validate the requirements mentioned in Section 3.4. The tasks and their corresponding requirements can be seen in Table 6.1.

Task	Requirement
T1	R1, R2
T2	R2, R3
T3	R2, R3, R5, R6
T4	R4, R5, R6

Table 6.1: Tasks and their related requirements

6.2 Results

This section will outline each task and their findings from the participant during the evaluation. Each of these tasks will also include an exemplary solution.

6.2.1 Task 1

The idea of this task was to first compare different repositories that are related to software development. Consequently, the repositories board should have been used. Figure 6.1 shows a possible solution that compares *django/django*, *facebook/react-native* and *ionic-team/ionic-framework* to each other.

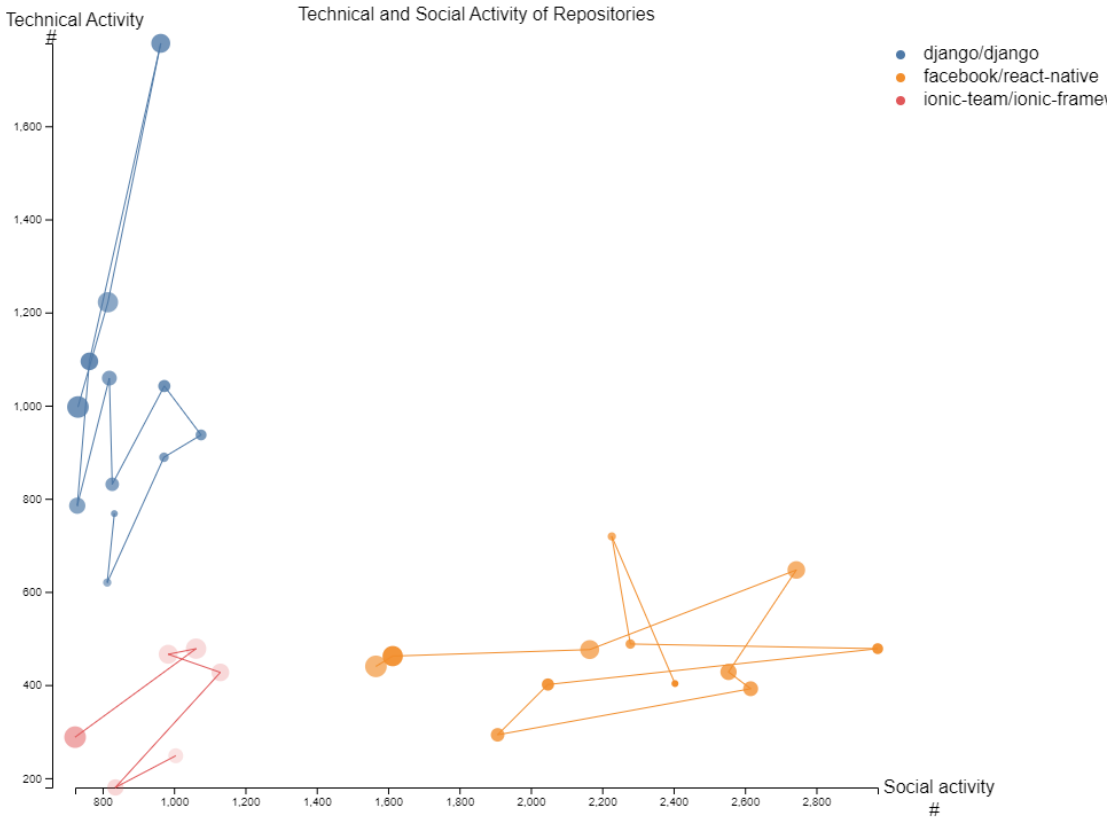

Figure 6.1: Comparison of activity of *django/django*, *facebook/react-native* and *ionic-team/ionic-framework*.

All the participants identified, that *django/django* is superior to the other two repositories in technical activity but also comes with a small social activity. There was a technical spike happening in October mentioned that resulted in a strongly visible decrease in technical activity afterwards. The repository *facebook/react-native*, however, was said to shine in social activity and appeared to be more consistent, as the values do not spread

as much. Since *ionic-team/ionic-framework* only existed starting in July 2020 it felt more difficult for the participants to find a clear pattern here, but the participants noted, that there was a better relationship between social and technical activity.

Figure 6.2 shows the same repositories with the addition of *flutter/flutter*. All the participants agreed, that *flutter/flutter* has noticeable more activity as the other repositories. Comparing the overall trend, *flutter/flutter* appeared to shine more in social activity than in technical activity relative to the others. Few participants mentioned that *flutter/flutter* shows similarities to *ionic-team/ionic-framework* but on a different scale. They justified this by stating, that *ionic-team/ionic-framework* and *flutter/flutter* are both mobile-first framework and could therefore show a similar pattern.

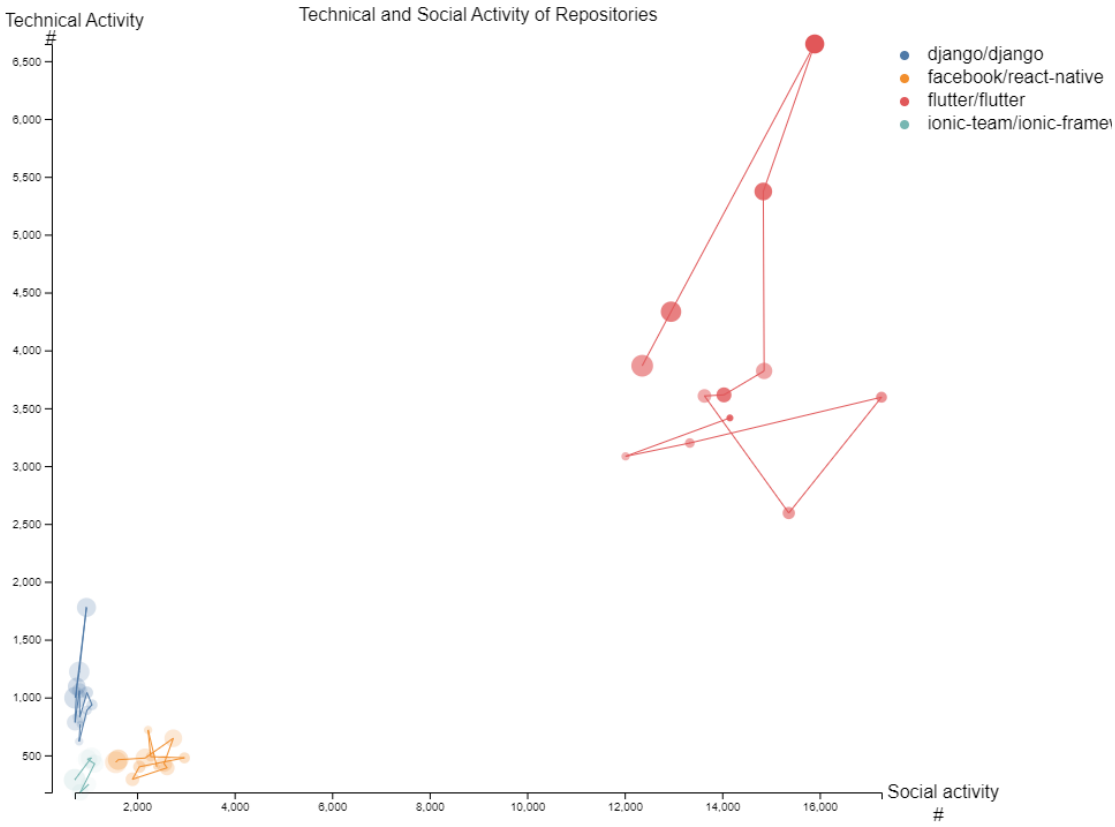

Figure 6.2: Comparison of *django/django*, *facebook/react-native.*, *ionic-team/ionic-framework* and *flutter/flutter*

6.2.2 Task 2

For task 2, the participants should take a deeper look on *flutter/flutter* and find anomalies that could indicate some specific events.

The overview of *flutter/flutter* is visible in Figure 6.3 and the participants noted, that the likes of the repository seem to increase when the technical activity spikes.

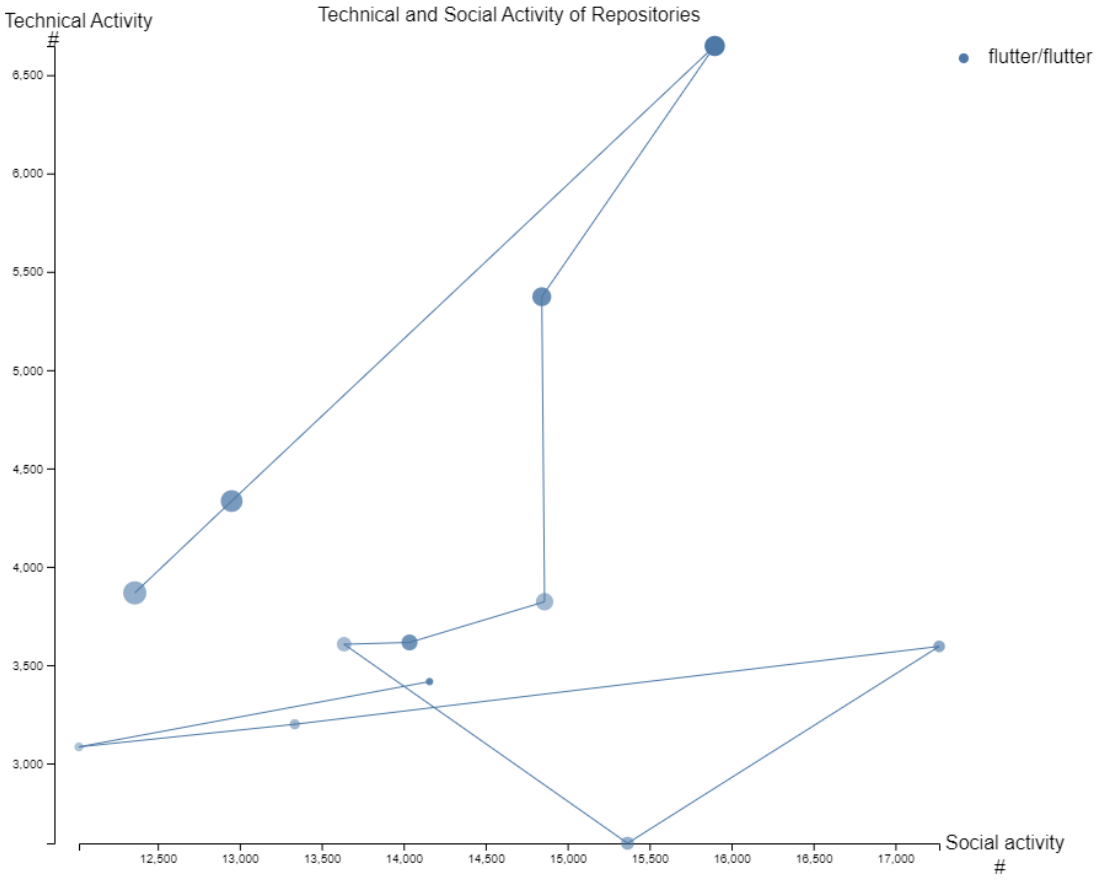

Figure 6.3: Overview of technical and social activity for *flutter/flutter*.

October appeared to be a relevant month for technical activity. The participants, however, were unsure if October had such high technical activity, because of a major release during this month or if the release happened earlier and October was used to react to the critical user feedback for potential technical defects after the major release. The participants arguing for a release happening in October also mentioned the significant increase of likes in the beginning of October while the technical activity stayed almost consistent throughout the month as seem in Figure 6.4.

Another month that participants indicated was the April, where the social activity was as its maximum, even surpassing the technically strong October. There was, however, no given day within that month that showed obvious signs for a specific event happening.

Two participants noticed, that there were many likes in July. After further investigation, they mentioned that there was a huge spike in likes at the end of July as seen in Figure

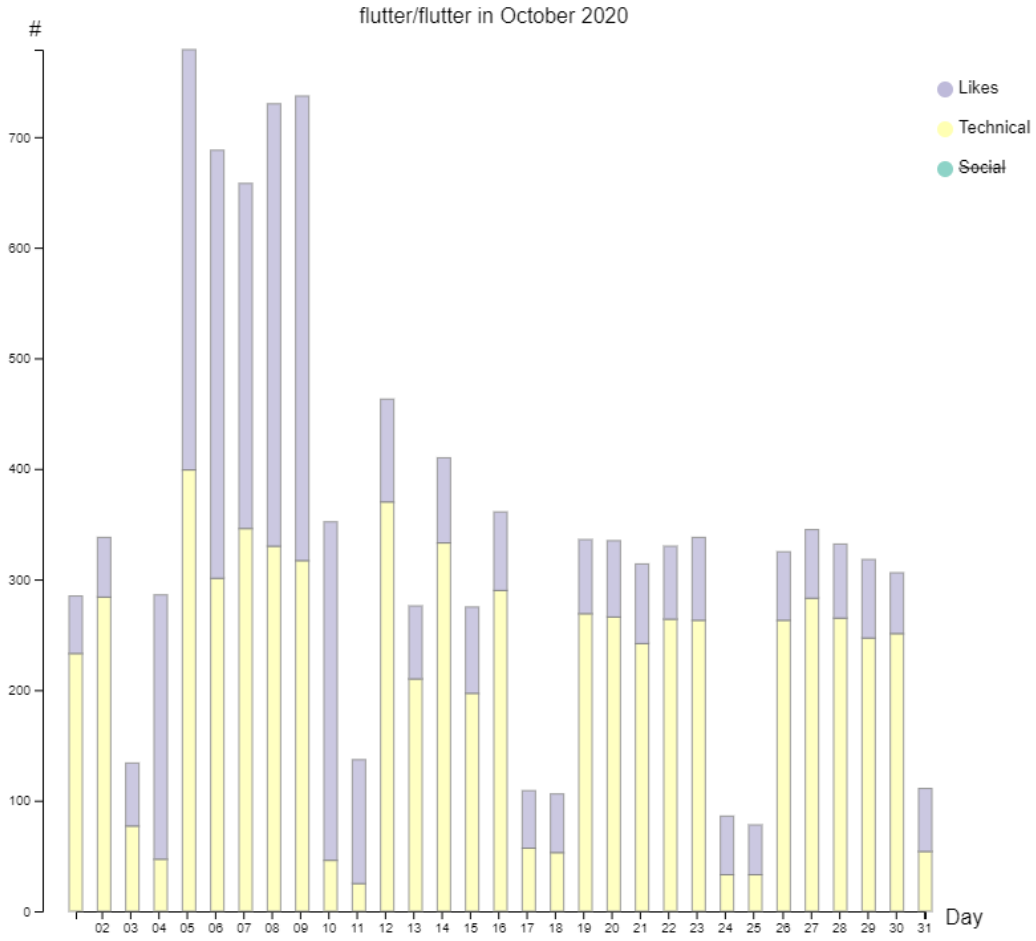

Figure 6.4: Likes and technical activity of *flutter/flutter* in October 2020.

6.5. They assumed that this anomaly indicated a public announcement or presentation from the developers leading to more popularity.

Overall the participants mentioned a very cyclic behaviour within the single months and a correlation between social and technical activity as well as a low - but still existing - activity over weekends.

6.2.3 Task 3

For the third task, a deeper look into *flutter/flutter* has been taken. To be more precise, this tasked focused on the closed and opened issues as well as the issue discussions for the January and February 2020.

Visualized in Figure 6.6, the participated pointed out, that the opened issues seem to

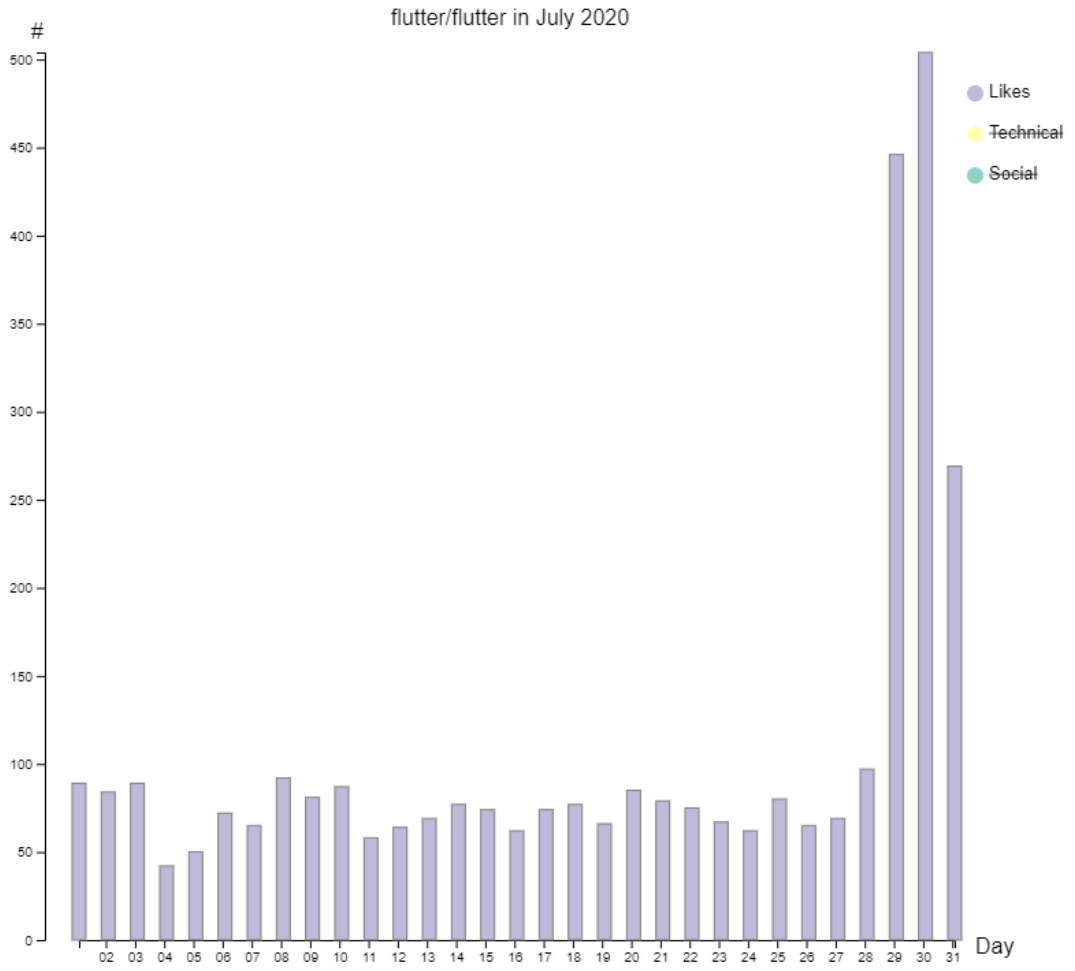

Figure 6.5: Likes of *flutter/flutter* in July 2020.

be below the global average while the closed issues are generally lower than the global average. They found a visible spike for closed issues on the 10th of January, where the value got up to more than double of the next higher value. This value also reflects in the issue comments. Some participants assumed that there happened an automatic closing of issues while others proposed the idea of the initial cleanup after the Christmas holidays.

One of the participants also mentioned, that there is usually a higher activity on Fridays than on the other days of the week.

The last sub-task focused on the discussions on weekends between March and May. Figure 6.7 shown an exemplary chart limited to only Saturdays and Sundays.

The participants noted two significant spikes during this time span occurring on the 25th

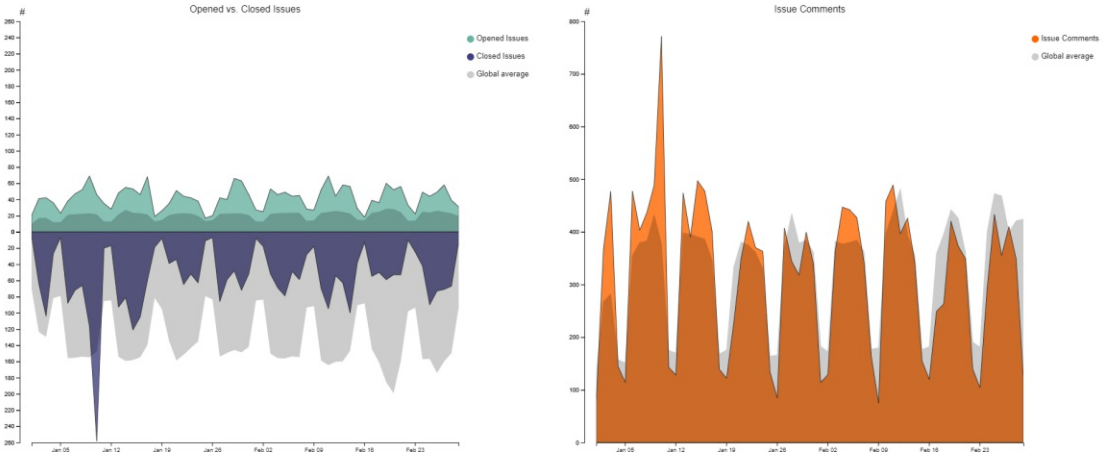

Figure 6.6: Closed and opened issues as well as issue comments for *flutter/flutter* in January and February 2020.

of April as well as the 4th of April. Both of these spikes happened on a Saturday and show a similar pattern than usual weekdays. One of the participants assumed, that those spikes might be due to *COVID-19* as more people worked remotely starting around this time.

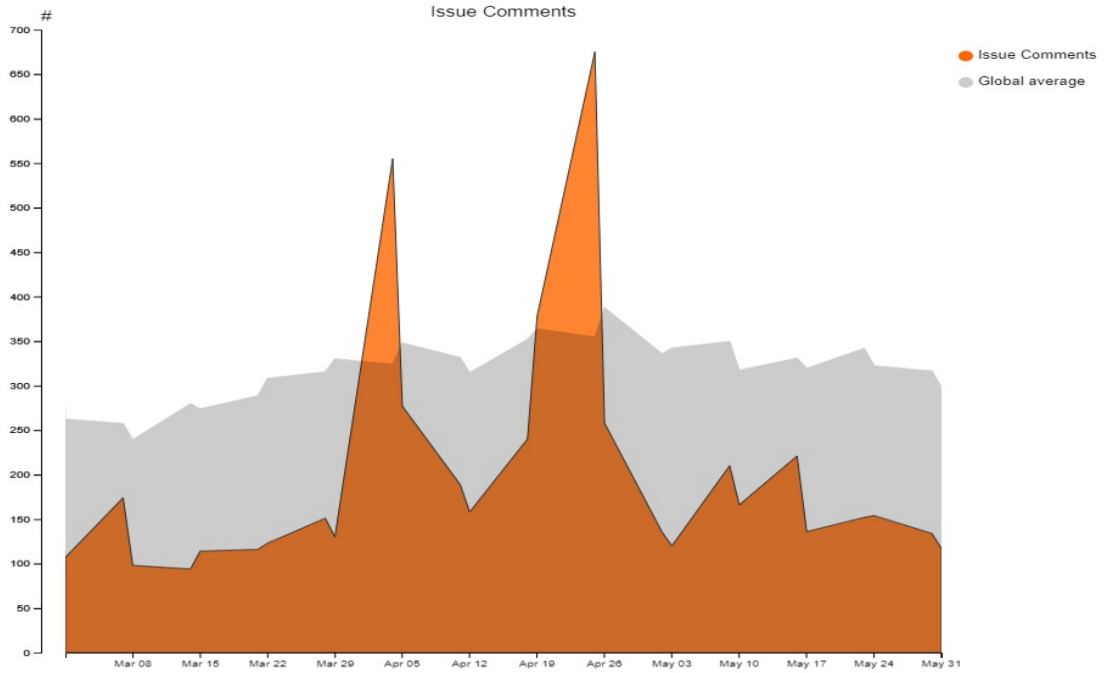

Figure 6.7: Issue comments on weekends for *flutter/flutter* between March 2020 and May 2020.

6.2.4 Task 4

The last task identified patterns and trends for the user *jonahwilliams* in the repository *flutter/flutter*.

For the activity of *jonahwilliams*, the participants pointed out, that his activity especially spikes on weekends. Throughout the week he, however, seems to be below the average activity as shown in Figure 6.8.

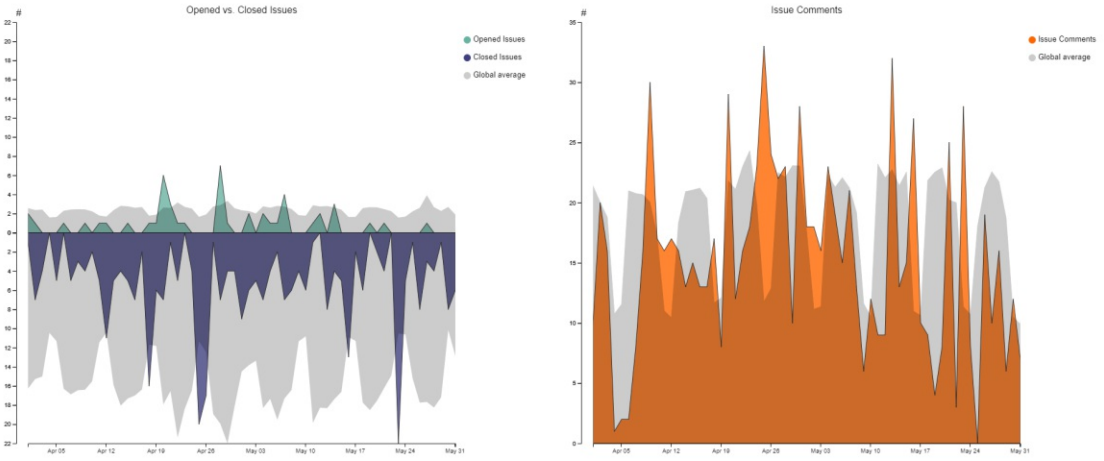

Figure 6.8: Activity of *jonahwilliams* in *flutter/flutter* between April 2020 and May 2020.

It was also mentioned, that after his spikes in closed issues, there usually follows a small spike in opened issues directly after. This was especially obvious on the 18th of April as well as the 25th, which both were Saturdays. On these Saturdays then followed a unusual amount of opened issues. One of the participants assumed, that this user is first taking care of existing issues and is finding new issues while testing his own solution. Another spike was on the 23rd of May which also fell on a Saturday but was not followed by a higher amount of opened issues.

Another thing pointed out was, that closed issues apparently lead to a higher amount of issue comments, this, however, is not always reflected in the other direction. For the opened issued there seems to be no correlation of that degree.

For the last task, the time around Christmas was analyzed by the participants to find global trends around the usual holidays. As shown in Figure 6.9, it was noted, that there was a huge global trend before Christmas which *jonahwilliams* did not follow. This user did not open any more issues close to the Christmas time and had a few spikes during December in the closed issues. It was assumed that this user went on vacation earlier than the usual *GitHub* user and decayed in activity starting on the 2nd of December 2020.

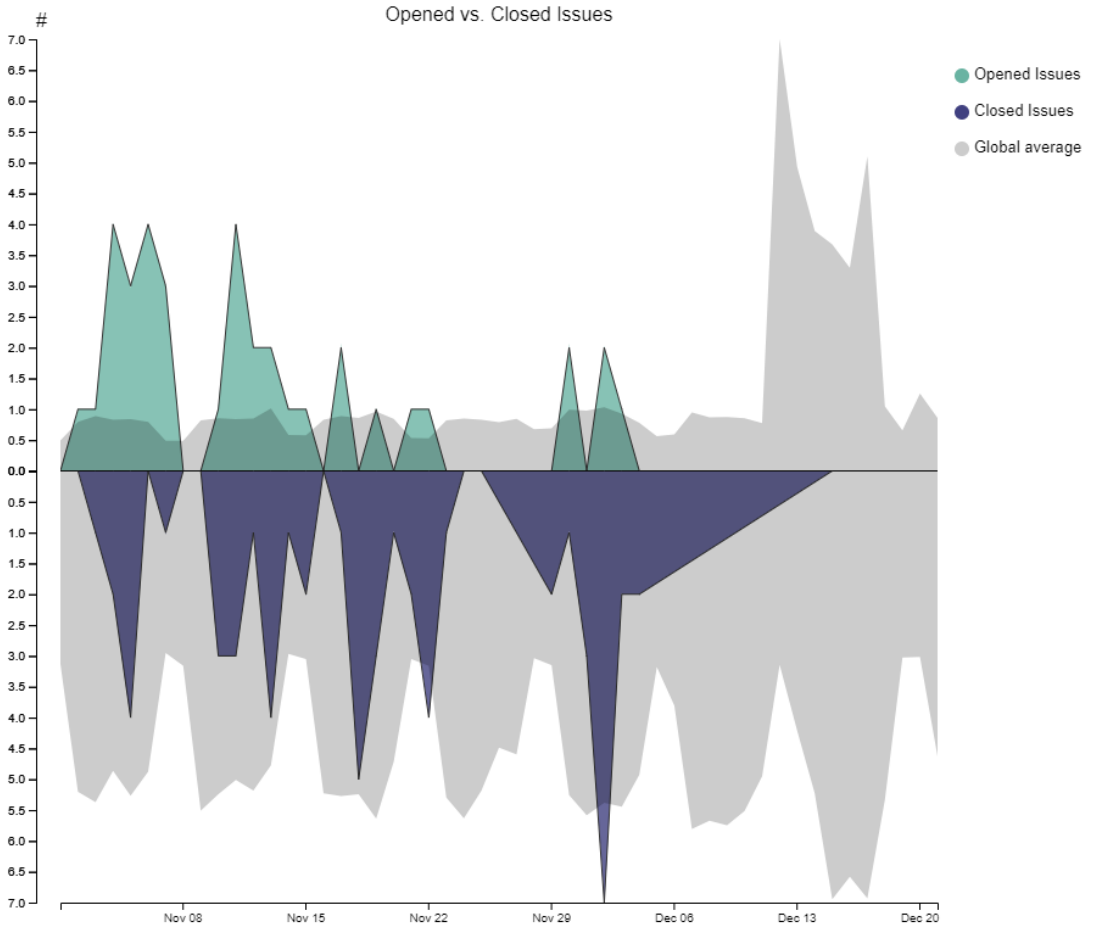

Figure 6.9: Activity of *jonahwilliams* in *flutter/flutter* in November 2020 and December 2020.

6.3 Analysis

The overall response was very positive and the participants stated that the prototype helped them to gain interesting insights into *GitHub* data. The average time to complete the tasks was 29 minutes, the shortest evaluation took around 14 minutes and the longest took around 48 minutes.

Participants stated, that they did not have any problems in knowing how to get the data they were looking for. They also showed a high interest in exploring before the tasks, indicated overall curiosity towards such a tool.

The overall interaction was very well-received as the filters provided a simple and understandable interface towards the data and the coordinated zooming as well as the details-on-demand functionality have helped the participants significantly to find the

values that they were looking for.

The identified weaknesses were mainly concerning usability, as it was sometimes difficult to derive the concrete date. For the repository board this was due to the fact, that mouse-over tool-tips appear below the component and not directly next to the mouse cursor. In the issues board, however, zooming often led to the horizontal axis labels being way to granular and showing the concrete hour instead of the day.

Another point mentioned was, that the repositories board contained encodings that were not self-explanatory enough. Especially the opacity reflecting the likes and the size displaying the month was not well-received as the participants assumed the size to reflect popularity.

The recordings also showed patterns in the fulfillment of tasks as seen in Figure 6.10. The measured times show, that the time spent on a single task heavily depends on the participant. A participant who took longer than a different participant for a given task usually also took longer for all the other tasks. This can mainly be justified by the participants capabilities in perceiving visual representations of data. As the evaluation was qualitative, some users could therefore identify more patterns and anomalies. The analysis has also shown, that the participants spending longer usually took more advantage of the coordinated views and went more into detail when finding a potential trend.

Looking at the overall distribution, *Task 4* has taken the longest time on average with 472 seconds, closely followed by *Task 1* with 445 seconds. *Task 2* and *Task 3*, however, usually went slightly faster with 412 and 411 seconds respectively.

Another visible pattern indicated, that participants that usually took more time, also tend to show a higher variance in between the individual tasks. As *Participant 1* and *Participant 6* have similar values for the tasks, *Participant 3*, *Participant 4* and *Participant 5* have larger gaps between time spent on tasks.

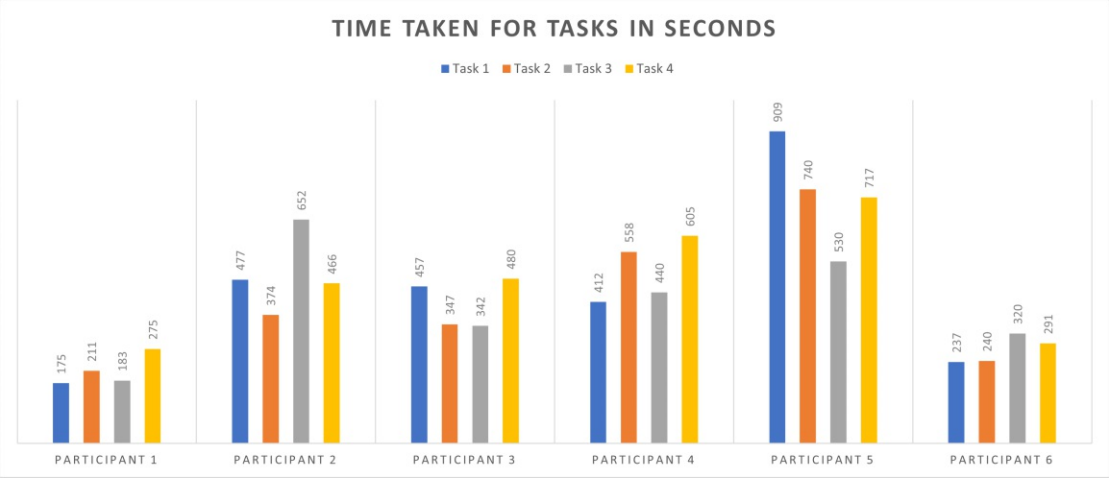

Figure 6.10: Statistical overview for the time taken of individual participants.

CHAPTER 7

Future Work

This thesis showed an approach for the exploration and identification of trends within large, time-oriented, multivariate data. The prototype presented offers a sufficient set of tools to find patterns as well as anomalies in the events provided by *GitHub*. The extensive amount of data, however, allows for several improvements in this field.

Scalability

As the overall resources and processing power was limited, the original events were processed beforehand and limited to the year 2020. That made it impossible to allow complex queries during run-time, restricting the visualizations to a simple level of complexity.

The granularity of the events was truncated to dates instead of timestamps to save on a significant amount of memory and storage. Removing that limitation opens new options for exploration, as dynamic loading of more precise data when zooming as well as broader details-on-demand capabilities.

Another extension is the analysis of keywords. In early versions of the prototype existed a way to analyze repositories for concrete technologies based on keywords. This was, however, discarded as a keyword search has shown to be very slow, resulting in very bad usability. With a more sophisticated backend solution, the areas for exploration could therefore even be widened to include technologies and even entire technology stacks.

Trend Visualization

The evaluation has shown, that finding trends can be difficult even to experts. For future improvements, a visual indication for potential trends could be implemented. Examples for such indications could be, but are not limited to:

1. Indicate weekends in the charts
2. Add an optional layer containing the kernel density estimation
3. Group values by their magnitude. This could exemplary be done by splitting the repository component into quadrants and naming those accordingly
4. Limit the global average to other repositories in the same area

It turned out, that finding trends is heavily relying on finding patterns first, meaning it might be difficult to distinguish trends from expected user behaviour. A typical example was seen in the evaluation where few participants could not tell if a high amount of likes is directly connected to a trend of if it is just the expected normal distributions.

This is, why indicating known factors, like weekends, public holidays or large conventions can help the experts significantly.

Spatial Dimension

Another limitation of the original data is, that there was no spatial information included. The timestamp is in *ISO 8601* format and provided in *Coordinated Universal Time (UTC)*, meaning there is no reliable way to derive the location.

With additional analysis, however, it could be possible to predict a timezone of a user from all of the issued events and their corresponding time. This, however, would still only be a prediction and not of certainty.

Another option is to load the public information from the *GitHub API* [Api] and use that information to retrieve the location of a user. This, however, would require this information to be provided by the user, as well as to be accurate and up-to-date.

Adding the spatial dimension could be very interesting when dealing with trends and allow for a higher variety of visualizations.

CHAPTER 8

Conclusion

This thesis presented a tool to work with *GitHub* data and proposed an *Visual Analytics* approach to allow domain experts to gain insight into this data using visualization techniques. The large data set and multivariate as well as temporal data required effective visual encodings to optimize exploration tasks within the prototype.

The original research question was **"How can *Visual Analytics* assist in analyzing *GitHub* trends?"** and we derived the following hypothesis:

- **H1: Social and technological trends can change rapidly over time:** Trends have shown to be capable of having a very unpredictable nature. It is often up to interpretation if a given anomaly is a trend or just expected behaviour. The prototype has shown, that identifying trends is heavily relying on finding patterns first. Once these patterns have been identified, the participants of the evaluation had an easier time in identifying trends.

  As the data and evaluation shows, already small events, like a convention or a major release can result in a positive trend which can slowly decay or stop directly after the occurrence of the event. Different types of events have shown different patterns of trend behaviour.

- **H2: Domain experts are capable of using *Visual Analytics* software given the right design.** Even though the prototype included a variety of visual encodings, the domain experts felt confident when using the prototype. Most of the experts have a development or software architecture background and have not worked with *Visual Analytics* software often before.

  Since all the tasks could have been fulfilled by all the participants showed, that *Visual Analytics* experience is not a prerequisite when working with such software. Choosing intuitive visual encodings, however, proved to be of high importance when using domain experts in the evaluation process.

- **H3: Visual interaction methods provide more insight by leveraging the temporal dimension.** Working with time-oriented data comes with its own difficulties. The prototype developed as part of this thesis put emphasis on simplifying the temporal dimension as much as possible while providing interaction methods that allow for more details and granularity. The main techniques integrated were details-on-demand as well as zoom.

  The participants made extensive use of the implemented visual interaction methods during the evaluation to explore specific points in time. This allowed for first finding patterns and anomalies in the bigger picture of data and then explore and identify potential trends with a more granular set of data.

List of Figures

2.1 Point Plot mapping the time to the horizontal axis and the value to the vertical axis [AMST11]. 10
2.2 A bar chart (left) and a spike chart (right) [AMST11]. 11
2.3 Visualization of line plots. The left plot shows straight lines, while the right plot uses Bézier curves [AMST11]. 12
2.4 SparkCloud highlighting the top 25 keywords for the last time point of a series while the other 50 keywords appear grayed out [LRKC10]. 13
2.5 *ThemeRiver* visualizing press articles from June - July 1999 [HHWN02]. . 14
2.6 A spiral graph showing the stock prices of *Microsoft* (yellow) and *Sun Microsystems* (red) over five years [WAM01]. 15
2.7 TrendDisplay visualization by Brodbeck and Girardin [BG03]. The top panel is showing derived values such as the inhibitor reactions (green) and the standard deviation (blue) while the bottom panel contains the raw data (drug discovery data). 16
2.8 Silhouette graphs to compare multiple time-series data mapped on a horizontal axes (left) as well as on concentric circles (right). This form of visualization allows for easier comparison between multiple data sets [AMST11]. 17
2.9 Tile maps showing the ozone measurements for Los Angeles from 1987 until 1991. Each tile represents a day of the year and the tiles are organised as a grid, representing a calendar [DTM97]. 18

3.1 The design triangle as described by Miksch et. al [MA14]. 21

4.1 The issues board for the repository *flutter/flutter* in January 2020 with controls (1), the opened/closed issues (2) and the issue comments (3). The grayed out areas indicate the global average. 28
4.2 Area chart showing the closed (bottom) as well as the opened (top) issues for *flutter/flutter* in January 2020. The grayed out areas indicate the global average. 29
4.3 Area chart showing the issue comments for *flutter/flutter* in January 2020. The grayed out areas indicate the global average. 30
4.4 Repositories board for 2020 including the repository filter (1) comparing *tensorflow/tensorflow*, *apache/spark*, *rush-lang/rust* and *flutter/flutter* (2) with a detailed view over *flutter/flutter* in September 2020 (3). 31

4.5 Comparison of *tensorflow/tensorflow*, *apache/spark*, *rush-lang/rust* and *flutter/flutter* in 2020 with each month being represented by one data point. The size of the data points represents actuality and the opacity encodes the relative likes. 31
4.6 Visualization of the technical activity, the social activity and the likes for flutter/flutter in October 2020. 32
4.7 Earlier version of visualizing the monthly change of repositories. 33
4.8 Repositories visualized as *ThemeRiver* in a scrapped version. 34
4.9 Scatter plot for number of social and technical events for *flutter/flutter* in 2020. Each scatter represents a single day in the year and the color indicates how late in the year that day way, while green was early in the year and red was later in the year. 35

5.1 Visualized pipeline for parsing original *GitHub* data for further analysis. . . 41

6.1 Comparison of activity of *django/django*, *facebook/react-native* and *ionic-team/ionic-framework*. 48
6.2 Comparison of *django/django*, *facebook/react-native.*, *ionic-team/ionic-framework* and *flutter/flutter* 49
6.3 Overview of technical and social activity for *flutter/flutter*. 50
6.4 Likes and technical activity of *flutter/flutter* in October 2020. 51
6.5 Likes of *flutter/flutter* in July 2020. 52
6.6 Closed and opened issues as well as issue comments for *flutter/flutter* in January and February 2020. 53
6.7 Issue comments on weekends for *flutter/flutter* between March 2020 and May 2020. 53
6.8 Activity of *jonahwilliams* in *flutter/flutter* between April 2020 and May 2020. 54
6.9 Activity of *jonahwilliams* in *flutter/flutter* in November 2020 and December 2020. 55
6.10 Statistical overview for the time taken of individual participants. 56

List of Tables

2.1 Characteristics of the Point Plot 10
2.2 Characteristics of the Bar Chart 11
2.3 Characteristics of the Line Plot 12
2.4 Characteristics of SparkClouds 13
2.5 Characteristics of *ThemeRiver* 14
2.6 Characteristics of the Spiral Graph 15
2.7 Characteristics of the TrendDisplay 16
2.8 Characteristics of the (Circular) Silhouette Graph 17
2.9 Characteristics of Tile Maps 18

3.1 *GitHub* data attributes 22

5.1 Unavailable files from the *GitHub* Archive [Gha]. Every file represents one hour 42

6.1 Tasks and their related requirements 47

Bibliography

[AAS+20] Roger A. Leite, Alessio Arleo, Johannes Sorger, Theresia Gschwandtner, and Silvia Miksch. Hermes: Guidance-enriched visual analytics for economic network exploration. *Visual Informatics*, 4(4):11–22, 2020.

[AGM+20] Roger A Leite, Theresia Gschwandtner, Silvia Miksch, Erich Gstrein, and Johannes Kuntner. Neva: Visual analytics to identify fraudulent networks. *Computer Graphics Forum (CGF)*, 39(6):344–359, 2020.

[AMM+08] Wolfgang Aigner, Silvia Miksch, Wolfgang Müller, Heidrun Schumann, and Christian Tominski. Visual methods for analyzing time-oriented data. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 14(1):47–60, 2008.

[AMST11] Wolfgang Aigner, Silvia Miksch, Heidrun Schumann, and Christian Tominski. *Visualization of Time-Oriented Data*. Human-Computer Interaction Series. Springer London, London, 2011.

[Ant] Analysing commits on github by @.gouv.fr authors – antoine augusti. https://blog.antoine-augusti.fr/2019/04/analysing-commits-on-github-by-gouv-fr-authors/ accessed November 11, 2020.

[Ant21] Lennart Ante. How elon musk's twitter activity moves cryptocurrency markets. *SSRN Electronic Journal*, 2021.

[Api] Github rest api - github docs. https://docs.github.com/en/rest accessed November 11, 2020.

[Bar] Baresquare. Github devtrends | baresquare. https://www.baresquare.com/github-devtrends/ accessed November 10, 2020.

[BG03] Dominique Brodbeck and Luc Girardin. Trend analysis in large timeseries of high-throughput screening data using a distortion-oriented lens with semantic zooming. In *IEEE Symposium on Information Visualization 2003*, pages 74–75. IEEE, 2003.

[BHO+20] Erik Brynjolfsson, John Horton, Adam Ozimek, Daniel Rock, Garima Sharma, and Hong-Yi TuYe. *COVID-19 and Remote Work: An Early Look at US Data*. National Bureau of Economic Research, Cambridge, MA, 2020.

[Bos] Mike Bostock. D3.js - data-driven documents. https://d3js.org/ accessed May 05, 2021.

[Cis18] Anca Cismasiu. *Sports Activity Suggestions: A Visual Analytics Approach*. Master Thesis, TU Wien, 2018.

[CLC16] Valerio Cosentino, Javier Luis, and Jordi Cabot. Findings from github. In Miryung Kim, Romain Robbes, and Christian Bird, editors, *13th Working Conference on Mining Software Repositories - MSR 2016*, pages 137–141, Piscataway, NJ, 2016. IEEE.

[CNY20] Damla Cay, Till Nagel, and Asim Evren Yantac. Understanding user experience of covid-19 maps through remote elicitation interviews. In Anastasia Bezerianos, editor, *Evaluation and Beyond: Methodological Approaches for Visualization*, pages 65–73, Piscataway, NJ, 2020. IEEE.

[dja] The web framework for perfectionists with deadlines | django. https://www.djangoproject.com/ accessed May 29, 2021.

[DTM97] David Mintz, Terence Fitz-Simons, and Michelle Wayland. Tracking air quality trends with sas/graph. *Proceedings of the 22nd Annual SAS User Group International Conference (SUGI97)*, pages 807–812, 1997.

[Epa] Epam solutionshub. https://solutionshub.epam.com/OSCI/ accessed November 10, 2020.

[FI15] Fragkiskos Chatziasimidis and Ioannis Stamelos. Data collection and analysis of github repositories and users. 2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA), pages 1–6, 2015.

[Gha] Gh archive. https://www.gharchive.org/ accessed December 17, 2020.

[Ghe] Github event types - github docs. https://docs.github.com/en/developers/webhooks-and-events/events/github-event-types accessed December 17, 2020.

[Gita] Github glossary - github docs. https://docs.github.com/en/github/getting-started-with-github/quickstart/github-glossary accessed May 24, 2021.

[Gitb] GitHub. Build software better, together. https://github.com/ accessed November 08, 2020.

[Gitc] GitHub. vitalets/github-trending-repos. https://github.com/vitalets/github-trending-repos accessed November 10, 2020.

[Har99] Robert L. Harris. *Information graphics: A comprehensive illustrated reference - visual tools for analyzing, managing, and communicating.* Management Graphics, Atlanta, Ga., 1999.

[HHWN02] S. Havre, E. Hetzler, P. Whitney, and L. Nowell. Themeriver: visualizing thematic changes in large document collections. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 8(1):9–20, 2002.

[IIC+13] Tobias Isenberg, Petra Isenberg, Jian Chen, Michael Sedlmair, and Torsten Möller. A systematic review on the practice of evaluating visualization. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 19(12):2818–2827, 2013.

[KDP16] Riivo Kikas, Marlon Dumas, and Dietmar Pfahl. Using dynamic and contextual features to predict issue lifetime in github projects. In Miryung Kim, Romain Robbes, and Christian Bird, editors, *13th Working Conference on Mining Software Repositories - MSR 2016*, pages 291–302, Piscataway, NJ, 2016. IEEE.

[KGB+14] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. The promises and perils of mining github. In Sung Kim, Martin Pinzger, and Premkumar Devanbu, editors, *11th Working Conference on Mining Software Repositories : proceedings : May 31 - June 1, 2014, Hyderabad, India*, pages 92–101. ACM, 2014.

[Kle15] Gary A. Klein. *Seeing what others don't: The remarkable ways we gain insights.* Public Affairs, New York, first edition edition, 2015.

[KP15] Simone Kriglstein and Margit Pohl. *Choosing the Right Sample? Experiences of Selecting Participants for Visualization Evaluation.* The Eurographics Association, 2015.

[LRKC10] Bongshin Lee, Nathalie Henry Riche, Amy K. Karlson, and Sheelash Carpendale. Sparkclouds: visualizing trends in tag clouds. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 16(6):1182–1189, 2010.

[LS09] Teng-Yok Lee and Han-Wei Shen. Visualization and exploration of temporal trend relationships in multivariate time-varying data. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 15(6):1359–1366, 2009.

[LSC+20] Roger Leite, Victor Schetinger, Davide Ceneda, Bernardo Henz, and Silvia Miksch. Covis: Supporting temporal visual analysis of covid-19 events usable in data-driven journalism. *IEEE VIS 2020*, 2020.

[MA14] Silvia Miksch and Wolfgang Aigner. A matter of time: Applying a data–users–tasks design triangle to visual analytics of time-oriented data. *Computers & Graphics*, 38:286–290, 2014.

[PMS14] Rohan Padhye, Senthil Mani, and Vibha Singhal Sinha. A study of external community contribution to open-source projects on github. In Sung Kim, Martin Pinzger, and Premkumar Devanbu, editors, *11th Working Conference on Mining Software Repositories : proceedings : May 31 - June 1, 2014, Hyderabad, India*, pages 332–335. ACM, 2014.

[PSM12] M. Pohl, M. Smuc, and E. Mayr. The user puzzle—explaining the interaction with visual analytics systems. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 18(12):2908–2916, 2012.

[Shn96] B. Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. In *IEEE Symposium on Visual Languages, 1996. Proceedings*, pages 336–343. IEEE / Institute of Electrical and Electronics Engineers Incorporated, 1996.

[SK15] D. Sam Sundar and Mila Kankanala. Analyzing and predicting lifetime of trends using social networks. In *2015 International Conference on Computer Communication and Informatics (ICCCI 2015)*, pages 1–7, Piscataway, NJ, 2015. IEEE.

[SM00] Heidrun Schumann and Wolfgang Müller. *Visualisierung: Grundlagen und Allgemeine Methoden*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[sql] Sqlite home page. https://www.sqlite.org/index.html accessed May 28, 2021.

[The] The State of the Octoverse. The state of the octoverse. https://octoverse.github.com/ accessed November 10, 2020.

[vtf] Vuetify — a material design framework for vue.js. https://vuetifyjs.com/en/ accessed May 31, 2021.

[vue] Vue.js. https://vuejs.org/ accessed May 31, 2021.

[WAM01] M. Weber, M. Alexa, and W. Muller. Visualizing time-series on spirals. In *Information Visualization, 2001. INFOVIS 2001. IEEE Symposium on*, pages 7–13. IEEE, 2001.

[Zoo] Zoom Video Communications. Video conferencing, cloud phone, webinars, chat, virtual events | zoom. https://zoom.us/ accessed May 16, 2021.