EvoScat: Exploring Software Change Dynamics in Large-Scale Historical Datasets

Souhaila Serbout
University of Zurich
Zurich, Switzerland
souhaila.serbout@uzh.ch

Diana Carolina Muñoz Hurtado
Software Institute
USI Lugano, Switzerland
carolina.munoz@usi.ch

Hassan Atwi
Software Institute
USI Lugano, Switzerland
hassan.atwi@usi.ch

Edoardo Riggio
Software Institute
USI Lugano, Switzerland
edoardo.riggio@usi.ch

Cesare Pautasso
Software Institute
USI Lugano, Switzerland
c.pautasso@ieee.org

***Abstract*—Long lived software projects encompass a large number of artifacts, which undergo many revisions throughout their history. Empirical software engineering researchers studying software evolution gather and collect datasets with millions of events, representing changes introduced to specific artifacts. In this paper, we propose EvoScat, a tool that attempts addressing temporal scalability through the usage of interactive density scatterplot to provide a global overview of large historical datasets mined from open source repositories in a single visualization. EvoScat intents to provide researchers with a mean to produce scalable visualizations that can help them explore and characterize evolutions datasets, as well as comparing the histories of individual artifacts, both in terms of 1) observing how rapidly different artifacts age over multiple-year-long time spans 2) how often metrics associated with each artifacts tend towards an improvement or worsening. The paper shows how the tool can be tailored to specific analysis needs (pace of change comparison, clone detection, freshness assessment) thanks to its support for flexible configuration of history scaling and alignment along the time axis, artifacts sorting and interactive color mapping, enabling the analysis of millions of events obtained by mining the histories of tens of thousands of software artifacts. We include in this paper a gallery showcasing datasets gathering specific artifacts (OpenAPI descriptions, GitHub workflow definitions) across multiple repositories, as well as diving into the history of specific popular open source projects.**

***Index Terms*—Software Evolution Visualization, Interactive Scatterplot, Large-Scale Historical Data**

I. Introduction

As software systems evolve, they leave behind a rich trail of events that captures their transformation over time. Historical datasets include changes to the software as well as its accompanying test cases, documentation, configuration files, and their version tracking metadata [11]. Researchers mine this data to analyze software evolution patterns [4, 21], uncover trends [28], and develop tools for tracking and improving software quality and maintainability [5, 9, 10, 14, 17, 22, 30].

Drawing insights from such rich historical data requires appropriate means of exploration, as the complexity and volume of versioned software artifacts can quickly become overwhelming. Among these, visualization plays a crucial role [9, 24, 30, 33]. Visualizing evolving parts of software projects at scale helps to compare various artifacts within the same view, which can facilitate the detection of similarities, divergence, and time related trends. However, a critical challenge emerges when attempting to scale these visualizations to datasets containing evolution data of a large number of projects or to projects that produce millions of data points tracking each component's evolutionary steps [25]. Scatter plots can efficiently represent a large number of data points within the same view, allowing visual comparison based on data points color, positions, shapes and density [15, 20, 31]. They allow multiple dimensions of information to be encoded simultaneously: the position of each dot can represent attributes such as time and artifact identity, and the color can encode change types, their age, or artifact categories [15, 31].

In the context of software evolution, we designed EvoScat, a large scale data visualization tool that leverages scatter plots exploiting their visual encoding flexibility. EvoScat uses this flexibility by mapping time to one of the axis, allowing the viewer to follow the temporal evolution of software components vertically. Each artifact history is represented as a vertical sequence of dots, where each dot corresponds to a specific event (e.g., commit) in the artifact's life time. The other dimension is used to organize artifacts—by name, directory structure, similarity, or logical grouping—so that patterns such as synchronized changes [2], divergence, or inactivity across components become visually salient. The key challenge of using a scatter plot to visually represent and compare the histories of thousands of artifacts concerns how these are sorted to be placed along the artifact axis.

In this paper, we showcase how EvoScat can be used to create such a visualization in the case of eight real-world large-scale software evolution datasets containing different types of software artifacts—code files, CI/CD pipeline configurations file, API documentation files—and covering two levels of granularity—across artifacts of the same kind mined from multiple open-source repositories and across all artifacts found within the same repository.

This paper makes the following contributions: 1) we discuss the design space of the evolution scatterplot addressing the use cases of empirical software engineering researchers; 2) we introduce the interactive visualization tool[1] and the filtering features it allows it supports to generate visualizations tailored to the analysis needs; 3) we showcase a gallery of diverse historical datasets visualizations generated by EvoSCat, attempting to reveal some insights about key events occurring during the evolution of these software projects.

The rest of this paper is organized as follows. In Section II we collect the use cases which drive the design of EvoSCat's visualization. In Section III we present tool's filtering features and its implementation details. We showcase the usage of EvoSCat on a selection of historical datasets mined from open source repositories in Section IV and present a reflection on the scalability and limitations of EvoSCat in Section V. Before drawing some conclusions in Section VII, we introduce the related work in Section VI.

II. Use Cases

To guide the design of the visualization tool, we identified a set of key use cases that reflect the challenges faced when analyzing large-scale software evolution datasets [21, 25]. These use cases are derived from practical analysis tasks and recurring needs in empirical software engineering research [3, 19], and they ensure EvoSCat's temporal scalability for evolution data belonging different artifact types and project contexts.

Dataset Overview & Temporal Coverage

U1. Provide an overview of large historical datasets supporting its initial exploration (Figs. 2, 4).

U2. Assess the uniform or skewed distribution of event occurrence over time and the artifacts' age across the sampling time period of the dataset (Figs. 1, 2, 5).

Artifact Evolution & Development Trends Comparison

U3. Compare the development pace or rhythm of different artifacts belonging to the same or different projects (Fig. 6).

U4. Distinguish artifacts undergoing high intensity effort vs. regular maintenance (Fig. 6).

U5. Highlight the stability or variability of metrics associated with each commit, separating the artifacts where the metric improves over time from the ones with degraded measurements (Fig. 8).

Anomaly & Pattern Discovery

U6. Help to visually spot forks, clones or duplicated artifacts in the dataset (Fig. 1).

U7. Hunt for clusters of interesting, outstanding or peculiar artifact evolution histories, to be further analyzed (Figs. 6, 7).

Lifecycle & Activity Monitoring

U8. Check the freshness of the artifacts (Figs. 9, 10): has their development been discontinued many years ago? Are there recent changes? What is the ratio between discarded artifacts, stable artifacts, and recently added ones?

Data Volume Scalability

U9. Scale to visualize millions of events collected, tracking the history of thousands of artifacts over decades (Figs. 1, 2, 4–11). The visualization is intended to represent large datasets using high resolution displays. It is not designed for small sets of short lived artifacts.

Crawler Evaluation & Dataset Discovery

U10. Compare the performance of different crawlers regarding their ability to discover previously unknown artifacts vs. how closely they track the evolution of known ones (Figs. 1, 2).

III. Visualization Design

EvoSCat uses a simple density scatter plot, where each dot represents an event in the history of an artifact evolution, as tracked by a version control repository. The events belonging to the history of the same artifacts are positioned on the same vertical line. The initial event at the bottom of the dotted line represents the creation of the artifact, which may be changed by committing new revisions to the repository at a specific timestamp, until it is removed (or no more changes occur). In this section, we define the various configuration options available in the current version of EvoSCat, enabling users to tailor the visualization to their specific analysis needs.

The vertical axis of the visualization represents time, while the other indicates the specific artifact being tracked (Fig. 1). We choose the Y axis as the time axis in order to visually represent the stratification of time from bottom to top, so that newer artifact versions layer on top of previous ones. This also leaves the X axis to somehow place the artifacts side-by-side so that their interdependent evolution histories can be tracked by visually following parallel vertical lines.

Given that the plot is meant to represent a very high number of artifacts and events in their history, we only use one dot for each event. Its color can be mapped to summarize relevant properties of the artifact or the changes applied to it in the particular commit. For instance, in the case of Figs. 1 and 2, the color is used to distinguish the commits by their corresponding years. This helps to get an overview of the widespread usage of GitHub workflow in public software repositories over the years, since the technology was introduced.

While it would be possible to also use the dot shape or size to visually map further information from the dataset, such details would not be clearly visible in the overview when the entire dataset of a large number of events is displayed. We also do not draw explicitly the axes because, unlike traditional scatter plots emphasizing the relationships between the independent variable (X) and the one depending on it (Y), whose values are read on linear or log scales, in our case we focus on the resulting visualization shape, which should visually represent the similarities (or differences) between the artifact evolution histories. It would also not be feasible to print the identifiers of a very large number of artifacts positioned side

[1] https://design.inf.usi.ch/evoscat

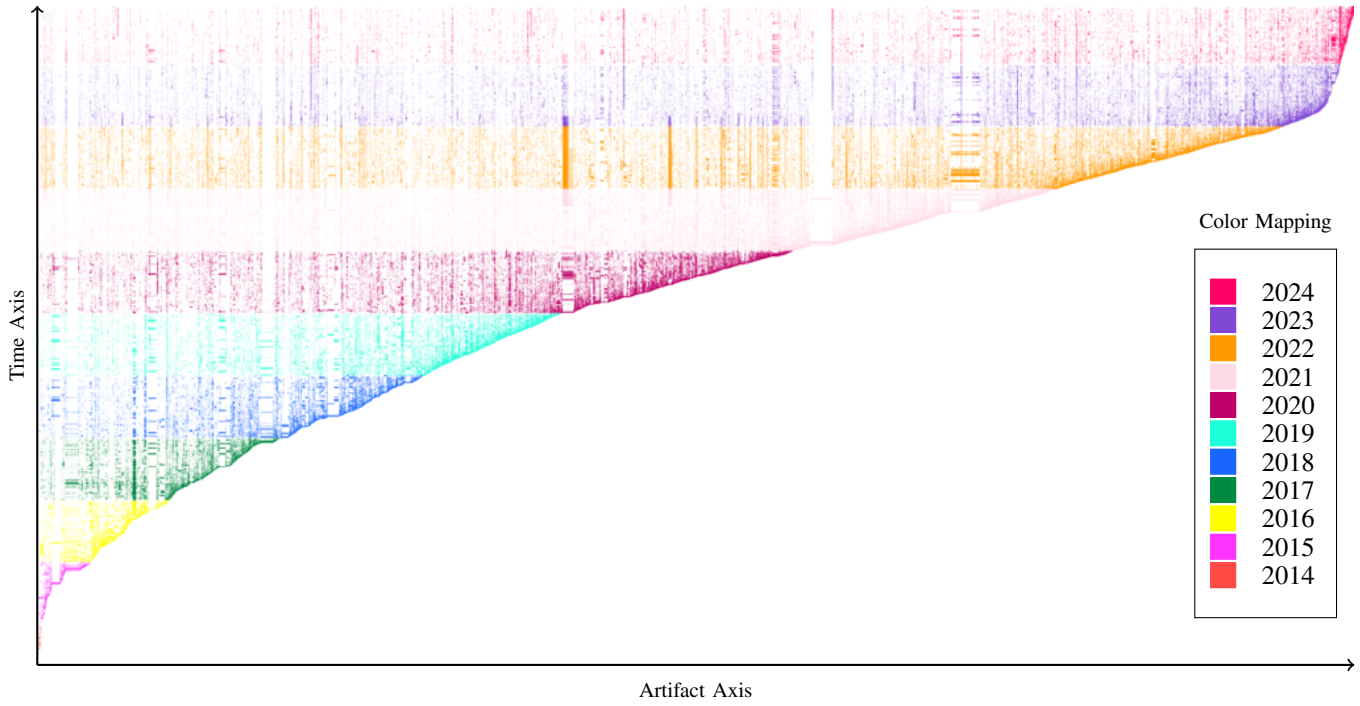

Fig. 1. Evolution Plot of the Filtered OpenAPI Dataset (Artifacts sorted by first commit timestamp, Absolute time on the vertical axis, Colored by commit year). This provides insight into the adoption of the OpenAPI standard, with early adopters positioned on the bottom left corner and highlighting the yearly increase in usage, as reflected from the OpenAPI specifications found on GitHub open source repositories.

by side along the X axis. To reify the visualization, we show the timestamp and the artifact identifier as the user interacts with its elements.

A. Filtering

To ensure a sharp visualization, a filtering mechanism is applied. During pre-processing, it is possible to specify *the minimum number of events per history* to hide the artifacts that lack a sufficient number of data points. The filtering helps to avoid a sparse, low density plot by visualizing only the artifacts with substantial evolution histories.

B. Time Axis Configuration

EvoScat supports various time scales for the Y axis. We distinguish five possible configurations:

- **Absolute:** Represents time in its natural form, preserving the actual timestamps of events. This mode is useful for examining changes in real chronological order as time flows from bottom to top, and the most recent events are layered on top of the older ones.
- **Relative to the Start:** Sets the initial point of all artifact histories to a common origin. Each artifact timeline is aligned and can be directly compared as it advances with the same speed on each vertical track (Fig. 2 (d)).
- **Relative to the End:** Aligns the end points of artifact histories, facilitating the visual comparison of the most recent changes across multiple artifacts.
- **Relative to the Median Date:** Centers the timeline around the median commit date, revealing symmetries or asymmetries in the artifact evolution histories, highlighting collective changes or synchronized evolution phases.
- **Normalized:** Scales each timestamp between 0 and 1, normalizing the time domain within which artifacts evolve, stretching it across the entire visualization regardless of their individual lifespan duration. Using this mode, one can see if, across their entire history – independently of its duration or how many commits are present – the artifacts have a similar tendency to change (e.g., grow or shrink, or to become more or less error-prone as shown in Fig. 8).

C. Artifact Axis Sorting Criteria

Artifact histories are displayed as sequences of dots vertically with each artifact side by side in the visualization. The order of the artifacts in the layout is a critical parameter affecting the resulting visualization, as their placement can highlight similarities between artifact histories. Given the high number of artifacts present in the visualization, there is a combinatorial explosion of possible orderings. Also, it is not practical to manually place thousands of artifacts along a certain order. Therefore, artifacts can be automatically sorted along the X axis according to different criteria, making the visualization very flexible so that it can be easily tailored to specific analysis needs (Fig. 2).

- **By Number of Events:** Groups artifacts by how many events are found in their history. This separates rarely changing artifacts from the ones that underwent intensive modifications during an eventful evolution history.

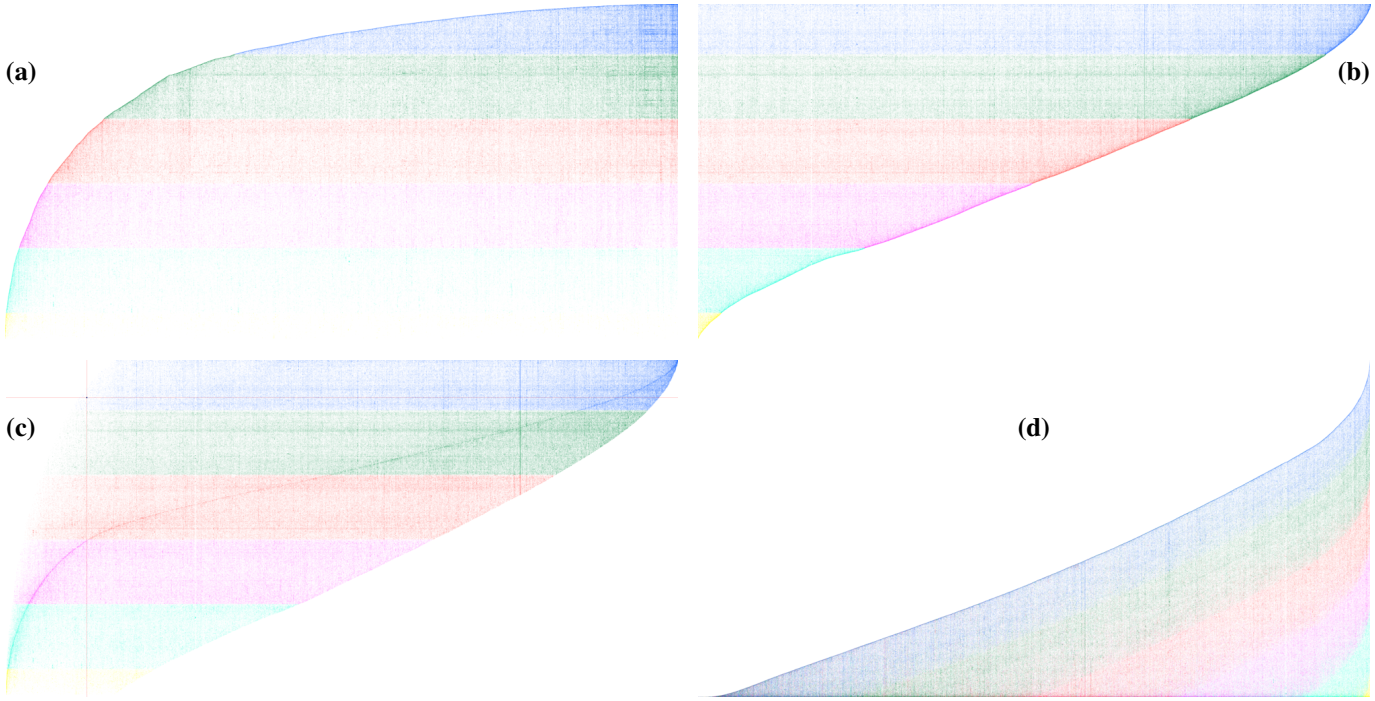

Fig. 2. Each evolution scatterplot of the GitHub Workflows Dataset shows the history of artifacts as vertical sequences of commit events (dots), with absolute time on the vertical axis and color encoding the year of each commit: 2019, 2020, 2021, 2022, 2023, 2024. Artifacts are sorted by their (a) last, (b) first, (c) median commit timestamp, and (d) age (last-first commit).

- **By Starting Time:** Orders artifacts by the time of their first recorded event, providing insight into initial development phases. This visualization indicates the rate at which artifacts were created over time, with the oldest on the bottom left and the most recent on the top right. When the curve emerging from the layout flattens, it indicates that at that time, many artifacts have been created (Fig. 1).
- **By Ending Time:** Prioritizes artifacts based on their most recent update, distinguishing fresh, recently changed artifacts. It helps to compare their proportion against older and stable artifacts, which have not been touched in a long time.
- **By Median Time:** As a hybrid of the previous two artifact orderings (Fig. 4), sorting the artifacts by their median commit timestamp reveals the boundaries of the dataset both in terms of the creation of artifacts (bottom) and their gradual abandonment (top). A horizontal top boundary would indicate the presence of many recently updated artifacts, while a horizontal bottom boundary would show that many artifacts have been present in the repository since its inception.
- **By Timestamps Similarity:** Groups artifacts with similar event distribution (comparing both absolute, relative or normalized timestamps), making it easier to detect artifacts with parallel evolution patterns, with many synchronous events.
- **By Initial Metric Value:** Sorts based on the starting metric value, useful for identifying artifacts that began with specific characteristics.
- **By Final Metric Value:** Orders artifacts by their most recent metric value, clustering artifacts with the same metric value in their latest version.
- **By Metric Value Variation:** Highlights artifacts with the most substantial changes between the first and last recorded values, drawing attention to dynamic evolution.
- **By Artifact Metadata:** Sorts artifacts by their path name, their filename extension (or both). The visualization reflects naming conventions introduced in the repository and can show refactorings involving changes in folder or filename, or the introduction of new programming languages in the code or migration across programming languages.

D. Dot Color Encoding

In EvoScat, dot colors in the scatterplot can be used to encode additional artifact attributes:

- **Time (year):** The dots can be colored to represent the year in which the corresponding event occurred.
- **Artifact type:** The color is mapped from the file extension, or other information that can be inferred from the artifact file name (e.g., documentation vs. code vs. test cases).
- **Metric:** Uses a color gradient to represent the magnitude of a chosen metric, allowing for the identification of high and low values at a glance.
- **Metric Variation:** Uses three colors to represent the positive, null or negative variation of a chosen metric, distinguishing additive changes vs. refactorings involving code removal and cleanup.
- **Author:** Coloring based on the commit author may highlight different code ownership and collaboration patterns [6, 13].

E. Interactivity

The visualization is meant to be enjoyed by interacting with it, by switching between an overview of the entire dataset and zooming into increasingly high levels of detail. To fit more data points into a limited set of pixels, it is possible to turn

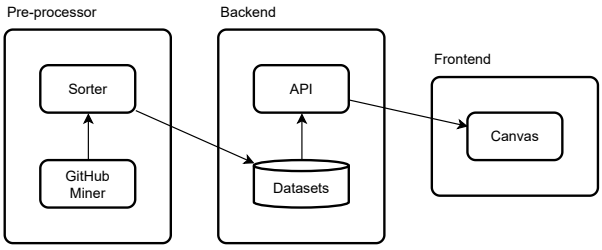

Fig. 3. Architecture of EVOSCAT

on the density plot feature, which uses the pixel opacity to visualize the density of data points under the visible pixel.

As the user changes the axis sorting criteria or time scaling configuration, it is possible to adjust the transition speed to smoothly shift between the two layouts. Tracking how the artifacts are reshuffled may reveal additional similarity patterns in the dataset. Users can control the transition duration, or turn the effect off to immediately switch between views.

The color mapping can also be changed interactively. Users can select which artifact feature (such as the file extension) or metric (such as the file size or the file size variation) is mapped to the color. The tool displays a histogram for each metric that 1) shows how many events fall under each class 2) provides a legend for the color mapping of the scatter plot. Users can also edit individual color mappings to highlight specific values in the scatter plot.

F. Tool Architecture and Implementation

The tool is composed of a data crawler component (i.e., a GitHub crawler and git repository log miner), a pre-processor, the scatter plot canvas and its surrounding interactive controls (Fig. 3).

The pre-processor takes as input commit event logs, which should contain entries with a timestamp and an artifact identifier. It is possible to feed additional attributes or metrics extracted from the specific artifact version, such as for example its size, number of defects, the author of the commit affecting it. The role of the pre-processor is to prepare the dataset so that it can be efficiently rendered. This involves positioning each event in the dataset on the scatter plot coordinate system according to the various layout and sorting criteria discussed in the previous section. While some are very simple and – even with a large dataset – could be computed as the user switches between sorting criteria, others – such as density clustering – are very time consuming and should be pre-computed. To reduce storage space and bandwidth consumption, the results of the pre-processing are compressed.

The scatter plot canvas displays the pre-processed dataset after it is downloaded from the server and decompressed. It is also responsible for rendering the density plot, where – due to the lower resolution of the display screen than what is required to paint the entire dataset, multiple events may overlap. This is reflected by their decreasing opacity level. The scatter plot also uses a spatial index to efficiently retrieve the artifact commit corresponding to the dot when the user mouse overlaps. The scatter plot canvas can also interpolate between two layouts and animate the transition with the give time. A copy of the scatter plot image – currently with the same display resolution as shown on the screen – can be saved in PNG format.

To make it easy to reproduce a visualization, its configuration parameters are stored in a URL string that can be bookmarked or shared with other researchers.

IV. GALLERY

A. Datasets

We showcase the visualization in the case of eight datasets of commits ranging from nearly half a million events to more than six million events. An extended gallery[2] containing more examples is available on EVOSCAT website.

D1: Web API Metrics Evolution Dataset: This dataset [26] traces the evolution of Web APIs through the changes detected in their corresponding specifications. For each specification commit, metrics related to the API size (in terms of functionality or data model) and the amount of endpoints covered by a certain security mechanism, in addition to other relevant metrics are computed. For this paper, we illustrate how EVOSCAT is used to: (1) Give an overview on the usage of OpenAPI specification in public GitHub repositories; (2) illustrate the pace and frequency in which these files are being updated over time; In existing work [17] we used the EVOSCAT to (3) visualize and classify different Web API evolution trends based on their size and security coverage over time.

D2: GitHub Workflows Evolution Dataset: The dataset is taken from the exploratory empirical analysis performed by Cardoen et al. [8] on the histories of GitHub workflows. In particular, we have used a more recent version than the one used in the paper (Version 2024-10-25 [7]). This updated dataset contains 2 686 974 workflow specifications extracted from 43 342 different GitHub repositories. Note that we kept only all workflows with more than 5 commits, which made the dataset shrink to 2 162 788 (−524 186) workflow specifications and 34 150 (−9 192) repositories.

D3: Popular GitHub open source repositories: Unlike *D1* and *D2*, which comprise evolution histories of artifacts of the same type mined from many different software repositories, *D3* contains the complete evolution history of all artifacts found within widely known open source repositories in GitHub. The goal is to showcase how EVOSCAT can be used to represent the evolution dynamics of large and complex software repositories during their entire life time.

B. Visualizations

1) Dataset overview (U1): Fig. 4 shows the overview of four different open source repositories, which have very different histories, for example, concerning the ratio of discarded artifacts (the highest in VS code) vs. the still active ones that had a long history (CPython) or a history spanning less than half of the project duration (Firefox). The four repositories

[2] https://design.inf.usi.ch/evoscat/gallery/

TABLE I
GALLERY DATASET SIZE OVERVIEW

ID	Dataset	Min. C/A	Artifacts	Commits	Events	Time Range	Years	First Timestamp	Last Timestamp
D1	Complete OpenAPI	0	256 601	680 115	1 374 430	3 868 days	10	May 8, 2014, 06:22:41	Dec 9, 2024, 15:40:37
D2	Filtered OpenAPI	5	35 441	520 754	1 045 173	3 865 days	10	May 8, 2014, 06:22:41	Dec 6, 2024, 21:41:47
D3	GitHub Workflows	5	113 816	1 369 110	2 162 788	1 903 days	5	Jul 26, 2019, 04:59:00	Oct 10, 2024, 17:17:22
D4	Node.js	0	111 728	42 664	612 055	5 933 days	16	Feb 16, 2009, 01:34:45	May 16, 2025, 14:55:50
D5	CPython	0	24 620	111 556	335 370	12 664 days	34	Sep 18, 1990, 12:47:40	May 21, 2025, 13:10:57
D6	VS Code	0	32 823	119 580	461 367	3 471 days	9	Nov 13, 2015, 15:18:17	May 16, 2025, 01:36:04
D7	Firefox	0	864 674	787 184	6 780 112	9 911 days	27	Mar 28, 1998, 04:38:53	May 16, 2025, 19:25:52
D8	PostgreSQL	0	12 098	60 946	307 195	10 548 days	28	Jul 9, 1996, 08:35:38	May 26, 2025, 13:30:01

also present a different tendency for the median commit of the artifacts. This is visible from the curve that emerges through the visualization that sorts the artifacts by their median commit time. A linear curve would indicate a regular rate of addition/removal of artifacts to the repository throughout the entire history. If the curve steepness increases (like in VS Code) the repository development pace picks up as more files are added to the repository.

Fig. 2 shows various overviews angles from which the D2 dataset of events can be seen. It shows the strong visual effect of changing the criteria according to which the artifact are sorted in the x-axis, while keeping the time axis set to represent the absolute time of the commit in all plots.

In (a), artifacts are sorted by the timestamp of the last known event in the lifecycle of the artifacts. While it shows that many artifacts have activity within four years, the bottom right part of the plot is fading, which reflects that a very few number of the early GitHub Workflows are still being updated and maintained.

In (b), the artifacts are sorted by the timestamp of the first commit instead. The visualization reveals how early adopters initiated workflows, with more recent ones appearing in the upper part. This configuration helps highlight how quickly newer workflows are being adopted and whether older ones remain actively developed.

In (c), sorting artifacts by their median commit timestamp arranges them so that each column (vertical stack of dots) represents one artifact, with its central point of activity positioned along the horizontal axis. When the dense region of points aligns along a diagonal or tight band, it implies that for many artifacts, commits are clustered around their median timestamp — meaning they are created and modified within a relatively short, consistent time frame.

In (d), the artifacts are sorted by the duration of their lifespan (age) — from the shortest to the longest. One can see how longevity correlates with activity density. Short-lived workflows have tightly packed events, while longer-lived ones may show more sporadic updates. This view can be especially useful to differentiate between ephemeral automation setups and those that are maintained over extended periods.

2) Artifact age distribution (U2): By sorting the artifacts by their age (the elapsed time between their last and first commit) and shifting the time axis to align the initial event of every artifact, we can compare the age distribution of artifacts across different datasets. For example, there is a clearly visible difference between the GitHub workflows dataset (Fig. 2 (d)) and both the complete and filtered OpenAPI dataset (Fig. 5).

Fig. 5 highlights that a very large proportion of the OpenAPI artifacts tend to be stable, as they have changed during a very short period of time after their creation or have been created but never updated afterwards. Using EVOSCAT it is possible to filter those artifacts by setting the minimum number of events in their history. Fig. 5 (bottom) shows an example of the filtering outcome, where the plot only include the OpenAPI specification with more than 5 commits in their history, also shown in Fig. 1 sorted by their first commit timestamp.

3) Development Pace and Effort Intensity (U3, U4): The regular structures found in the visualization of the VS Code dataset (Fig. 6) are due to changes applied systematically to the internationalization artifacts storing the translation of user interface elements in multiple languages. In the same Figure it is possible to spot a recently added set of files that have been the focus of a large number of changes in the most recent time period, which reflects the very intense effort invested in the integration of artificial intelligence features within the integrated development environment.

4) Stability vs. Metrics Variability (U5): Fig. 8 shows how the workflows' metric (in this case the number of security misconfigurations in the workflow) improves (i.e., less misconfigurations), stays the same, or worsens (i.e, more misconfigurations) over their histories. Each workflow evolution history is stretched along the entire plot height, the first commit being at the bottom, and the last on top. On the X axis we enumerate all the workflows in the dataset (sorted first by the variation of misconfigurations ΔM and then by the number of misconfigurations M). As we can see from the plot, we observe the formation of distinct vertical bands. The first group of bands ❶ represents all the workflows for which the metric improves over time, ❷ all those that stay the exact same, and finally ❸ where the metric worsens over time.

The improvement and worsening of the metric in the workflows over time can be seen also by the color of each commit. In the case of the workflows on the left-side of the plot, we can see that the color goes from red (20 security misconfigurations) to cyan (2 security misconfigurations). Conversely, we can see that, for the workflows on the right-side of the plot, the colors

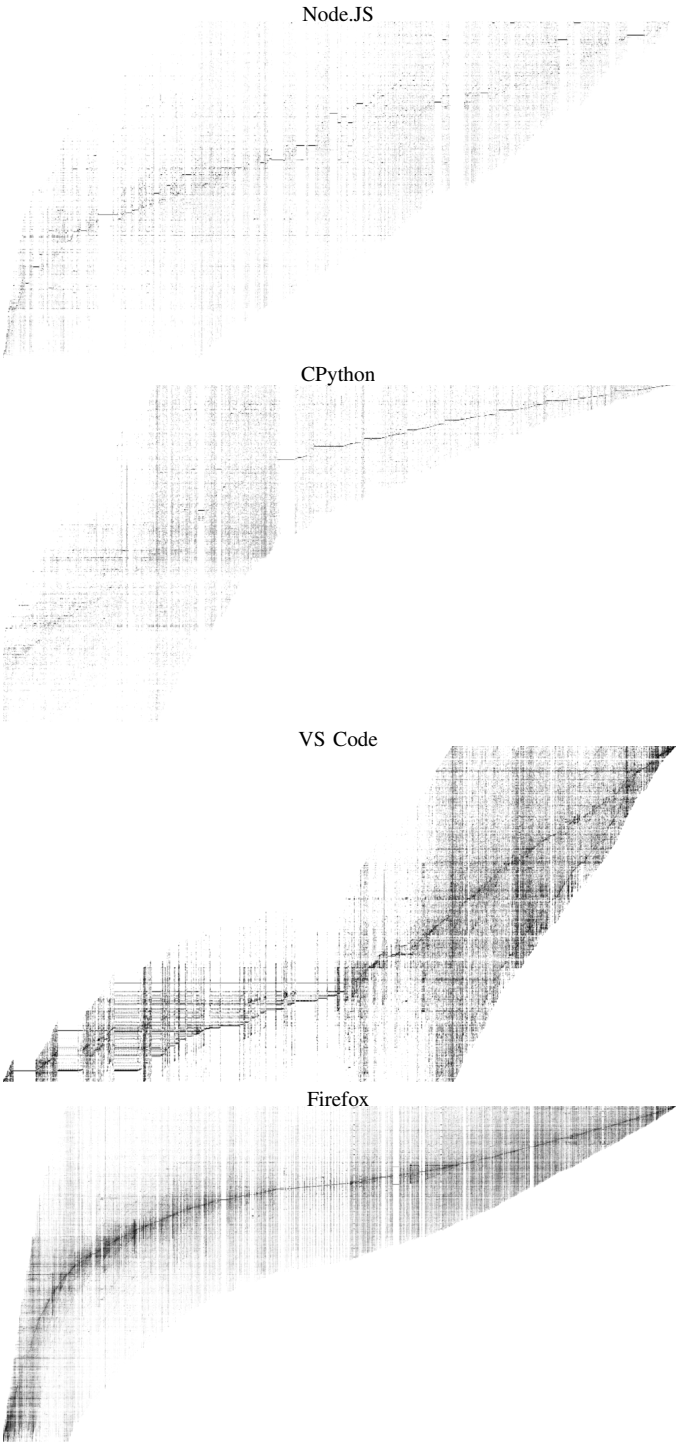

Fig. 4. Evolution Scatterplot Gallery: Artifacts sorted by median commit timestamp, Absolute time on the vertical axis. Black color intensity reflects density of commits and dataset size difference.

go from cyan to red, and sometimes to black (more than 20 security misconfigurations).

The plot also shows that there is a relatively small group of commits containing 0 security misconfigurations (in green). In the case of the green commits in the first group of bands, it shows that all the misconfigurations were eradicated from the workflow in time. In the case of the green commits in the second group of bands, instead, it shows that there are some clean workflows that never had more than 0 misconfigurations throughout their whole history.

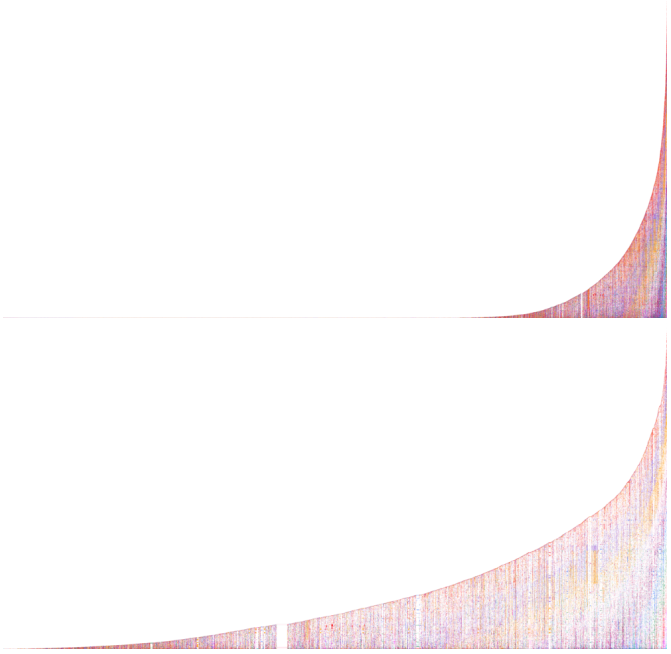

Fig. 5. Evolution Scatterplot of the complete OpenAPI Dataset (above) and filtered to include APIs with more than 5 commits (below) – Artifacts sorted by their age, Time is relative to the first commit timestamp on the vertical axis, Colored by commit year

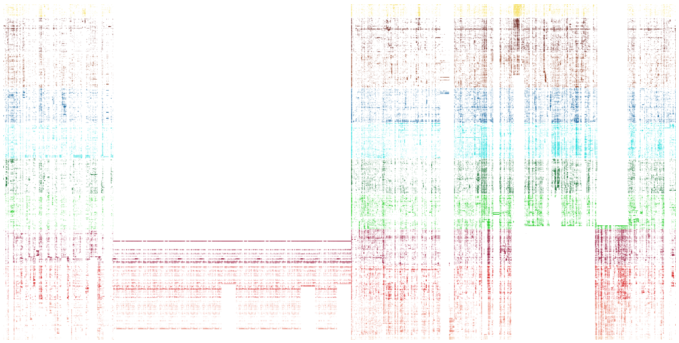

Fig. 6. Evolution Plot of the VSCode Dataset (Artifacts sorted by their pathname in the git repository, Time is absolute on the vertical axis, Colored by commit year) – The discontinued, regularly spaced commits belong to the i18n/*.json files. The high intensity blot in the current year is the copilot integration.

5) Potential clone and fork detection (U6): Fig. 1 shows how sorting by the initial commit timestamp helps to spot potential clone cases across different repositories due to the presence of repeated horizontal lines. The artifacts not only share the same initial commit timestamp (therefore, they are placed side by side in the visualization) but also share the same timestamp for many subsequent commits. In some of these cases, at some point in their history, the commits start having different timestamps, which could potentially indicate a fork.

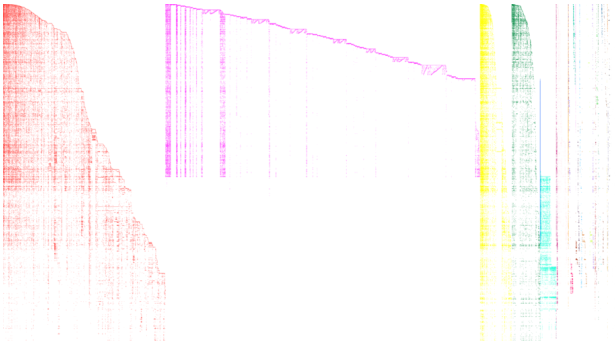

Fig. 7. Evolution Scatterplot of the CPython dataset. Artifacts sorted by 1) their extension 2) last commit timestamp, Time is absolute on the vertical axis, Colored by file extension – The documentation originally written in latex was migrated in one shot to use RST files in the middle of August 2007. Later in September 2017 the NEWS file that had been continuously updated for 17 years was discontinued. From that moment onward, every news item was included in separate documentation files.

6) Interesting Patterns (U7): Sorting artifacts by filename extension presents the history of artifacts grouped by programming language and further separating code from documentation. As shown in Fig. 7, we can spot when the Python documentation system was migrated from latex to reStructuredText markup (RST) in one transition. In general, it is possible to spot transitions between artifacts (or types of artifacts) by tracking the presence of synchronous gaps in the plot, which indicate that some artifacts have been removed while at the same time other artifacts have appeared.

When using the absolute time axis, the presence of horizontal lines (Fig. 6) indicates commits affecting a large number of files, changing at the same time (e.g., upon a major release or refactoring). Vertical lines indicate artifacts which change with a high frequency (e.g., at the end of every working day).

When using the time axis centered around the median commit and the color mapping based on the metric variation, one can expect that the events representing artifact removal (or decrease in size) are found above the central line, while the events representing artifact creation (or changes that do not affect size) tend to be placed below.

7) Artifact Freshness (U8): If we sort the VS code (Fig. 9) or PostgreSQL (Fig. 10) artifacts by their size as measured in the last commit and by the last commit timestamp. We can clearly distinguish two groups of artifacts. On the left, the ones which have been removed from the repository, while the currently active ones – a much smaller group in case of VS Code – are shown on the right. This plot helps to visually estimate not only the proportion of the two kinds of artifacts but also the pace of artifact removal over the project lifetime. This visualization may help to highlight major code rewriting efforts and support decision making on whether a historical analysis includes or excludes no longer present artifacts.

8) Crawler Performance (U10): Empirical software engineering researchers build crawlers to gather artifacts from open source repositories. Given the rate limit imposed by GitHub and other sources, the crawlers need to balance two different goals: 1) discovery of new relevant artifacts, 2) tracking the artifact evolution history. By comparing Figs. 1 and 2 (b), we can see that the GitHub workflow crawler discovered new artifacts at a constant rate throughout the years, while in case of the OpenAPI dataset, the priority switched in mid 2023 from new artifact discovery to tracking the history of existing artifacts, as reflected in top-right part of Fig. 1.

V. Limitations

The main limitations of the visualization concern the source of events, which are currently taken from the main branch of a repository. While one could easily try to represent branches as additional artifacts side by side with the ones taken from the main branch, git repositories do garbage collect commits belonging to deleted branches, which are either discarded (and lost from the perspective of empirical software evolution researchers) or eventually merged with the main branch (and therefore visible in the current visualization).

Since each artifact is identified by its path name, we do not explicitly track renaming of artifacts, which is shown as the removal of the previously named artifact followed by the addition of a new artifact with the new name.

We also assume that the timestamp provided by the git log is reliable, which may not always be the case [12]. The visualization has also been used to spot the presence of out of range commit timestamps (too old, or in the future) and may thus help to motivate the need for data cleanup or inspect the results against the input of the data cleanup phase.

There are some fundamental limits on the raw size of the datasets that can be visualized with the current tool implementation. These concern the maximum amount of data that can be fetched as JSON strings by Web browsers, as well as the increasingly long amount of time necessary to pre-process the data. The limited resolution of the scatter plot canvas also imposes a limit on the maximum number of artifacts that can be shown on the same plot.

VI. Related Work

A. Visualizing large datasets

Different solutions are presented to address scalable scatterplot visualizations as Tao et al. presented Kyrix-S[29], a tool that relies on multinode database storage and the use of multinode spatial indexes to achieve interactive browsing of large scatterplot visualizations, which can scale to represent datasets with one billion Reddit comments. Other projects such as Hilasaca et al. [15], presented a strategy to remove overlaps from scatterplot layouts with dimensionality reduction, to enhance readability. However, they consider that the elimination of overlaps affects the distribution represented in the plot, an issue due to overlap removal techniques, such as DGrid and ReArrange, and also can lead to information loss in the visualization. The work by Liu et al. [18] presented a visual analytics system to support evolution analysis using topic modeling. They employ the Latent Dirichlet Allocation (LDA) technique to analyze source code from open-source projects on GitHub as JavaScript libraries, D3.js, and Vue.js.

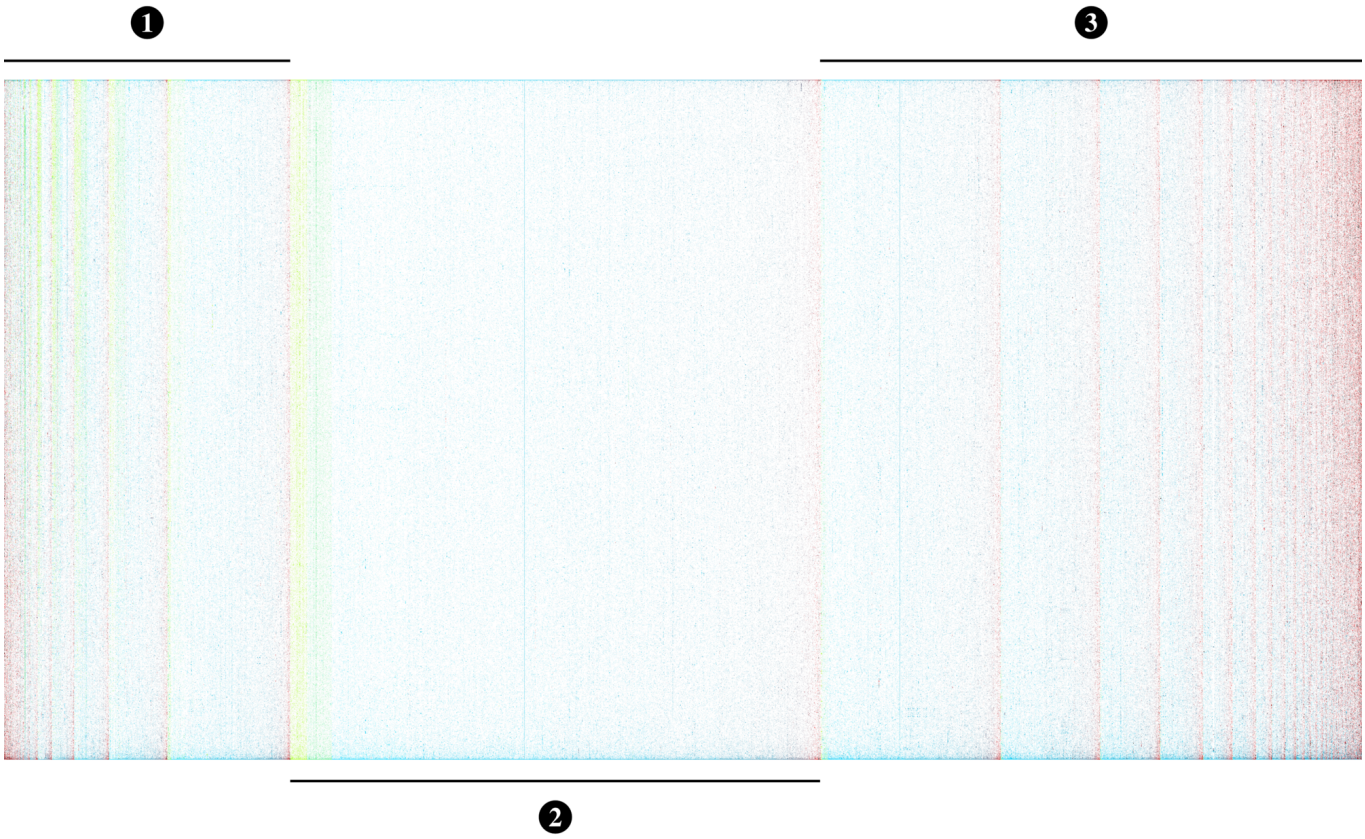

Fig. 8. Evolution Scatterplot of the Github Workflows dataset: Artifacts sorted by their 1) misconfiguration variation, 2) initial, and 3) final misconfiguration count, Time is normalized on the vertical axis, Colored by misconfiguration count: 0, 2, 20, > 20

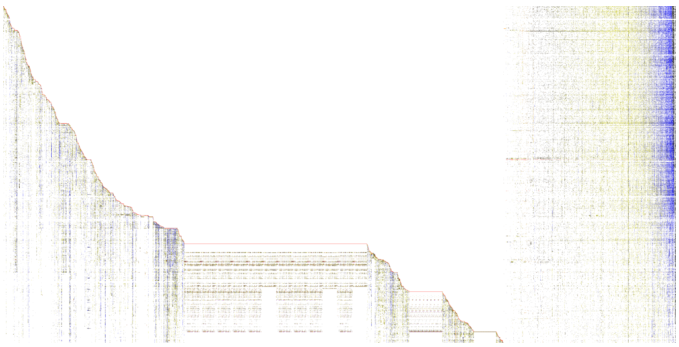

Fig. 9. Evolution Plot of the VS Code Dataset (Artifacts sorted by 1) their size measured in the last commit 2) last commit timestamp, Time is absolute on the vertical axis, Colored by size - means the file has been deleted) – The discarded artifacts throughout the project history pile up on the left, while artifacts that are still present in the repository are placed from small to large on the right side.

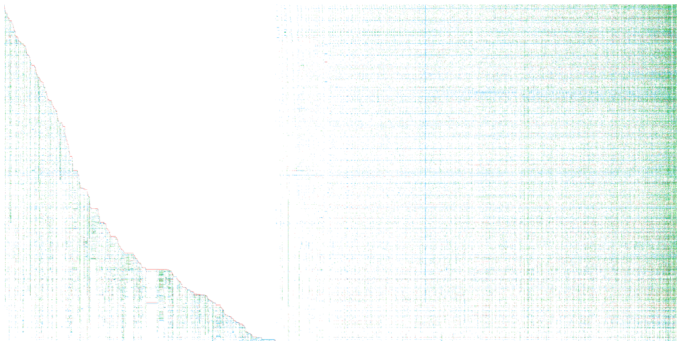

Fig. 10. Evolution Plot of the PostgreSQL Dataset (Artifacts sorted by 1) their size measured in the last commit 2) last commit timestamp, Time is absolute on the vertical axis, Colored by their size change shrink or remove, stable, grow) – Compared to Fig. 9, there are less discarded artifacts and a more "punctuated" evolution style with changes spanning across many artifacts.

After characterizing the software features and classifying the source files, they quantify differences between versions to reveal evolution patterns and development pace.

In contrast to these projects, which are focused on large-scale historical data but are limited to scaling within specific experimental contexts, we introduce EvoScat, a tool that displays large-scale visualizations that dynamically adapt to diverse datasets with rich historical data.

B. Software evolution visualization

Pfahler et al. [23] extended the city metaphor, in which buildings represent classes and districts represent packages by Wettel and Lanza [32] to visualize evolving software systems. To do so, they introduced new ways to traverse time through

a "History-Resistant Layout", in which a fixed position in the 3D space is given to each element of the city. By doing so, they managed to remove the unpredictable jumps of buildings and districts during the progression of its history, making for a more cohesive and easy-to-follow animation. The evolution scatterplot needs to deal with similar layout issues to position artifacts along the X axis, a mono-dimensional space. As the time dimension is explicitly represented, we use animations for the transition between different layouts.

Alcocer et al. introduced the Performance Evolution Matrix [1], an interactive matrix visualization where software components are arranged as rows and software versions as columns. Each cell encodes run time metrics, enabling developers to trace performance regression or improvement across versions. Listing components along the x-axis provides a comparative view, helping to spot the component that changed performance the most in absolute or relative terms. This comparative view can be also provided by EvoScat at scale: Fig. 8 provides a visual comparison of how many CI/CD workflows reduce, increase or have a stable number of misconfigurations detected throughout their whole history.

Similar comparisons can be made with CCEvovis [16], a tool that uses stacked barcharts to track clone changes across versions. Each bar represents a specific version of the software. The height of the bar corresponds to the total number of clone sets detected in that version. The bar is divided into colored segments that indicate the number of clone sets belonging to each evolution category within that version: added clone sets (red segment), changes clone sets (green segment), and deleted clone sets (dark blue). Looking at the bars next to each other allows comparing the clone sets across software versions. When using EvoScat to visualize artifact histories mined from different repositories (Fig. 1) the presence of potential clones with identical evolution histories can be visually spotted through repeated horizontal lines.

Pioneering work on the visualization of software evolution includes RelVis [24] by Pinzger et al. and evolution spectrographs [33] by Wu et al., providing graphical representations of source code and release history data. RelVis represents software modules, in this case, the nodes represent source code entities, edges the relationships between them, and the evolution of the metrics by highlighting the resulting polygons. Evolution spectrographs are used to visualize and track architectural changes, specifically modifications at the file level, in response to new entry requirements. They introduced evolution spectrographs to study changes in specific open source systems: OpenSSH, PostgreSQL (Figs. 10 and 11), and Linux. The goal was to gain a better understanding how software evolves by looking for evidence of specific changes. In contrast with our study based on the visualization of software artifact histories, our work not only represents the temporal evolution of the artifacts but also gives the possibility to explore their similarity and long-term evolutionary trends. Particularly regarding the PostgreSQL dataset, we are glad to report that the original finding from more than 2 decades ago of a "punctuated evolution" with a linear rate of growth [33] appears to have been maintained during most of the entire project lifetime (Fig. 11).

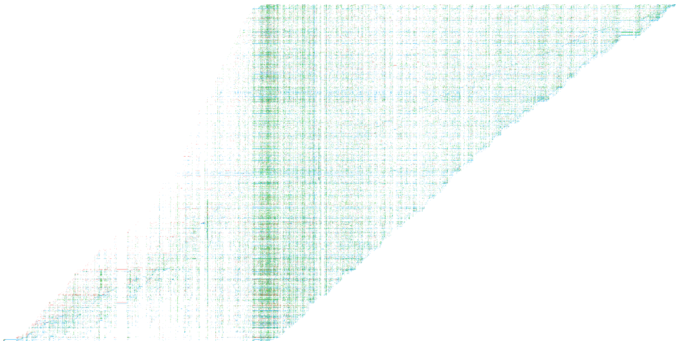

Fig. 11. Evolution Plot of the PostgreSQL Dataset. Artifacts sorted by their median commit time, Time is absolute on the vertical axis, Colored by their size change shrink or remove, stable, grow – Through the entire project there has been a long lasting core of frequently changing artifacts to which new ones are added and removed following a regular pace.

VII. Conclusion

In this paper we introduce EvoScat. A tool that allows producing interactive and scalable visualizations attempting to help researcher compare historical datasets, tracking the evolution of a large number of artifacts over long periods of time. EvoScat's visualization is designed to contain millions of events, such as commits that create, change or remove artifacts from one or multiple code repositories. Users can not only zoom to focus on specific artifacts, but also can rearrange how they are sorted along the artifact axis and configure how colors are mapped based on different metrics or features associated with the artifacts. We included a gallery of visualizations illustrating the history of multiple popular open source repositories as well as comparing specific artifacts (such as GitHub workflows and OpenAPI specifications) mined from thousands of repositories.

In the future we plan to feature a side-by-side comparative visualization of different repositories as well as to make it easier to merge datasets, e.g., to study all repositories managed by the same organization. We have also started to experiment with replacing the time axis with a different one, using a different metric to position the events along the history of artifacts, revealing an interesting relationship between the size of an API and its security coverage [17]. We are also working on the integration of the evolution scatter plot with existing detailed visualizations of specific artifact evolution histories [27].

VIII. Supplementary Material

A demo video of EvoScat is available at https://youtu.be/z_SLstI1mx8. The source code and datasets are published at https://zenodo.org/records/15525004.

References

[1] Juan Pablo Sandoval Alcocer, Fabian Beck, and Alexandre Bergel. Performance evolution matrix: Visualizing performance variations along software versions. In *IEEE Working Conference on Software Visualization (VISSOFT)*, pages 1–11. IEEE, 2019. doi: 10.1109/VISSOFT.2019.00009.

[2] Saimir Bala, Kate Revoredo, Joao Carlos de AR Gonçalves, Fernanda Baiao, Jan Mendling, and Flavia Santoro. Uncovering the hidden co-evolution in the work history of software projects. In *International Conference on Business Process Management*, pages 164–180. Springer, 2017.

[3] Hani Bani-Salameh and Ayat Ahmad. Software evolution visualization tools functional requirements: a comprehensive understanding. *ICSEA 2016*, page 209, 2016.

[4] Shawn A. Bohner, Denis Gracanin, Troy Henry, and Kresimir Matkovic. Evolutional insights from uml and source code versions using information visualization and visual analysis. In *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 145–148, 2007. doi: 10.1109/VISSOF.2007.4290713.

[5] David Broneske, Sebastian Kittan, and Jacob Krüger. Sharing software-evolution datasets: Practices, challenges, and recommendations. *Proceedings of the ACM on Software Engineering*, 1(FSE):2051–2074, 2024.

[6] Stefano Campanella and Michele Lanza. Hidden in the code: Visualizing true developer identities. In *2024 IEEE Working Conference on Software Visualization (VISSOFT)*, pages 24–35, 2024. doi: 10.1109/VISSOFT64034.2024.00013.

[7] Guillaume Cardoen. A dataset of github actions workflow histories. Zenodo, doi: 10.5281/zenodo.13985548, October 2024.

[8] Guillaume Cardoen, Tom Mens, and Alexandre Decan. A dataset of github actions workflow histories. In *Proceedings of the 21st International Conference on Mining Software Repositories*, page 677–681. Association for Computing Machinery, 2024. doi: 10.1145/3643991.3644867.

[9] M. D'Ambros, M. Lanza, and H. Gall. Fractal figures: Visualizing development effort for cvs entities. In *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 1–6, 2005. doi: 10.1109/VISSOF.2005.1684303.

[10] Michael Fischer and Harald Gall. Evograph: A lightweight approach to evolutionary and structural analysis of large software systems. In *2006 13th Working Conference on Reverse Engineering*, pages 179–188, 2006. doi: 10.1109/WCRE.2006.26.

[11] Michael Fischer, Martin Pinzger, and Harald Gall. Populating a release history database from version control and bug tracking systems. In *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.*, pages 23–32. IEEE, 2003.

[12] Samuel W Flint, Jigyasa Chauhan, and Robert Dyer. Pitfalls and guidelines for using time-based git data. *Empirical Software Engineering*, 27(7):194, 2022.

[13] T. Girba, A. Kuhn, M. Seeberger, and S. Ducasse. How developers drive software evolution. In *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*, pages 113–122, 2005. doi: 10.1109/IWPSE.2005.21.

[14] Geoffrey Hecht, Omar Benomar, Romain Rouvoy, Naouel Moha, and Laurence Duchien. Tracking the software quality of android applications along their evolution (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 236–247. IEEE, 2015.

[15] Gladys M Hilasaca, Wilson E Marcílio-Jr, Danilo M Eler, Rafael M Martins, and Fernando V Paulovich. A grid-based method for removing overlaps of dimensionality reduction scatterplot layouts. *IEEE Transactions on Visualization and Computer Graphics*, 30(8):5733–5749, 2023.

[16] Hirotaka Honda, Shogo Tokui, Kazuki Yokoi, Eunjong Choi, Norihiro Yoshida, and Katsuro Inoue. Ccevovis: A clone evolution visualization system for software maintenance. In *Proc. IEEE/ACM 27th Int. Conference on Program Comprehension (ICPC) – Tools Demo Track*, pages 122–125, 2019. doi: 10.1109/ICPC.2019.00026.

[17] Diana Carolina Munoz Hurtado, Souhaila Serbout, and Cesare Pautasso. Mining security documentation practices in openapi descriptions. In *2025 IEEE 22nd International Conference on Software Architecture (ICSA)*, pages 119–130. IEEE, 2025.

[18] Huan Liu, Yubo Tao, Yining Qiu, Wenda Huang, and Hai Lin. Visual exploration of software evolution via topic modeling. *Journal of Visualization*, 24:827–844, 2021.

[19] Joy Lowe and Machdel Matthee. Requirements of data visualisation tools to analyse big data: A structured literature review. In *Responsible Design, Implementation and Use of Information and Communication Technology: 19th IFIP WG 6.11 Conference on e-Business, e-Services, and e-Society, I3E 2020, Skukuza, South Africa, April 6–8, 2020, Proceedings, Part I 19*, pages 469–480. Springer, 2020.

[20] Adrian Mayorga and Michael Gleicher. Splatterplots: Overcoming overdraw in scatter plots. *IEEE transactions on visualization and computer graphics*, 19(9):1526–1538, 2013.

[21] Renato Lima Novais, André Torres, Thiago Souto Mendes, Manoel Mendonça, and Nico Zazworka. Software evolution visualization: A systematic mapping study. *Information and Software Technology*, 55(11):1860–1883, 2013.

[22] Michael Ogawa and Kwan-Liu Ma. Stargate: A unified, interactive visualization of software projects. In *2008 IEEE Pacific Visualization Symposium*, pages 191–198, 2008. doi: 10.1109/PACIFICVIS.2008.4475476.

[23] Federico Pfahler, Roberto Minelli, Csaba Nagy, and Michele Lanza. Visualizing evolving software cities. In *IEEE Working Conference on Software Visualization (VISSOFT)*, pages 22–26. IEEE, 2020. doi: 10.1109/VISSOFT51673.2020.00007.

[24] Martin Pinzger, Harald Gall, Michael Fischer, and Michele Lanza. Visualizing multiple evolution metrics. In *Proceedings of the 2005 ACM symposium on Software visualization*, pages 67–75, 2005.

[25] Gaëlle Richer, Alexis Pister, Moataz Abdelaal, Jean-Daniel Fekete, Michael Sedlmair, and Daniel Weiskopf. Scalability in visualization. *IEEE Transactions on Visualization and Computer Graphics*, 2022.

[26] Souhaila Serbout and Cesare Pautasso. Apistic: A large collection of openapi metrics. In *21st IEEE/ACM International Conference on Mining Software Repositories (MSR)*, pages 265 – 277, Lisbon, Portugal, April 2024.

[27] Souhaila Serbout, Diana Carolina Muñoz Hurtado, and Cesare Pautasso. Interactively exploring api changes and versioning consistency. In *11th IEEE Working Conference on Software Visualization (VISSOFT 2023)*, pages 28–39, Bogota, Colombia, October 2023. IEEE, IEEE.

[28] Bruno L Sousa, Mariza AS Bigonha, Kecia AM Ferreira, and Glaura C Franco. A time series-based dataset of open-source software evolution. In *Proceedings of the 19th International Conference on Mining Software Repositories*, pages 702–706, 2022.

[29] Wenbo Tao, Xinli Hou, Adam Sah, Leilani Battle, Remco Chang, and Michael Stonebraker. Kyrix-s: Authoring scalable scatterplot visualizations of big data. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):401–411, 2020.

[30] A. Telea and L. Voinea. Interactive visual mechanisms for exploring source code evolution. In *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 1–6, 2005. doi: 10.1109/VISSOF.2005.1684304.

[31] Yunhai Wang, Fubo Han, Lifeng Zhu, Oliver Deussen, and Baoquan Chen. Line graph or scatter plot? automatic selection

of methods for visualizing trends in time series. *IEEE transactions on visualization and computer graphics*, 24(2):1141–1154, 2017.

[32] Richard Wettel and Michele Lanza. Visualizing software systems as cities. In *Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 92–99, 2007. doi: 10.1109/VISSOF.2007.4290706.

[33] Jingwei Wu, Claus W Spitzer, Ahmed E Hassan, and Richard C Holt. Evolution spectrographs: Visualizing punctuated change in software evolution. In *Proceedings. 7th International Workshop on Principles of Software Evolution, 2004.*, pages 57–66. IEEE, 2004.