Analysis and Visualization of Kieker's Software Evolution

Malte Hansen
Kiel University
Kiel, Germany

Daniel Riese
Kiel University
Kiel, Germany

Johannes Klindt
Kiel University
Kiel, Germany

Wilhelm Hasselbring
Kiel University
Kiel, Germany

Abstract

Kieker is a software observability tool that has been under development for 20 years. With over 9,000 commits from more than 25 contributors to its main branch on GitHub, the Kieker codebase has been extensively maintained and continuously extended. In this work we analyze and visualize the evolution of Kieker's software architecture using our 3D software visualization tool ExplorViz. The Code Analyzer of ExplorViz runs a parallelized static analysis on file revisions in Kieker's Git repository. Additionally, data on contributor activity is requested from the GitHub API. The 3D visualization in our web-based frontend is complemented by commit charts, information on contributor activity, heat maps, and the ability to animate file changes based on commits. This enables the interactive visual exploration of Kieker's software evolution.

1 Introduction

Program comprehension of large software systems and their evolution is a challenging activity for software engineering tasks. Software visualization approaches can reduce the time needed to explore the software architecture and evolution of software systems [3].

In this work, we present an approach to analyze and visualize data collected from Git repositories. Kieker [8] is a software observability tool that is under continuous development for about 20 years and has accumulated over 9,000 commits on GitHub.[1] Thus, it makes a good case study for our approach. We employ our 3D software visualization tool ExplorViz [4, 5] and run an analysis on Kieker's GitHub repository. In the following sections, we present related visualization approaches, the tool architecture of ExplorViz, results of the performance analysis for the Kieker repository, and an exemplary visualization of the collected data.

2 Related Work

ExplorViz visualizes software systems as cities by default. The city metaphor is also prominent in previous and other visualization approaches such as CodeCity [2] or BabiaXR [6]. Occhipinti et al. visualized the evolution of various large software systems in 3D with a tool called SYN using a spiral layout [7]. Our animated visualization of Kieker's software evolution is inspired by SYN and also supports the spiral layout in addition to a city metaphor visualization.

3 Tool Architecture

A backend component called Code Analyzer enables that Git repositories are cloned and analyzed. For efficient and customizable static code analysis, we rely on lexers and parsers generated by ANTLR [1].[2]

For each supported source code file, data about its classes and functions is collected. This excludes the raw source code, as our visualization focuses on the architecture of software systems.

Utilizing ANTLR grammar files, the Code Analyzer supports a more in-depth analysis of the following programming languages: C/C++, C#, Go, Java, JavaScript/TypeScript, Kotlin, PHP, Python, Rust, and Swift. The following metrics are computed for each supported source code file: lines of code, source lines of code, commented lines of code, number of functions, and number of variables.

The resulting data is sent via gRPC to the Landscape Service that is responsible for storing it in a Neo4j graph database.

4 Analysis Performance

We executed a full file analysis on the Kieker repository's main branch with 3,877 first-parent commits. The needed analysis time for each commit and the full runs were measured on an Apple Mac mini with a base M4 chip and 32 GB of main memory. Analysis runs were repeated five times each, in two different modes: Writing the resulting data directly in the Code Analyzer to files and sending the data to the Landscape Service for storage in the Neo4j database. The results are presented in Table 1. It can be seen that all analysis runs completed in under twelve minutes.

Storage	Min	Median	Max
File	5m 28s	5m 29s	5m 48s
Neo4j	11m 07s	11m 16s	11m 42s

Table 1: Time needed for analysis of Kieker's main branch with 3,877 first-parent commits.

[1] https://github.com/kieker-monitoring/kieker

[2] https://github.com/antlr/grammars-v4

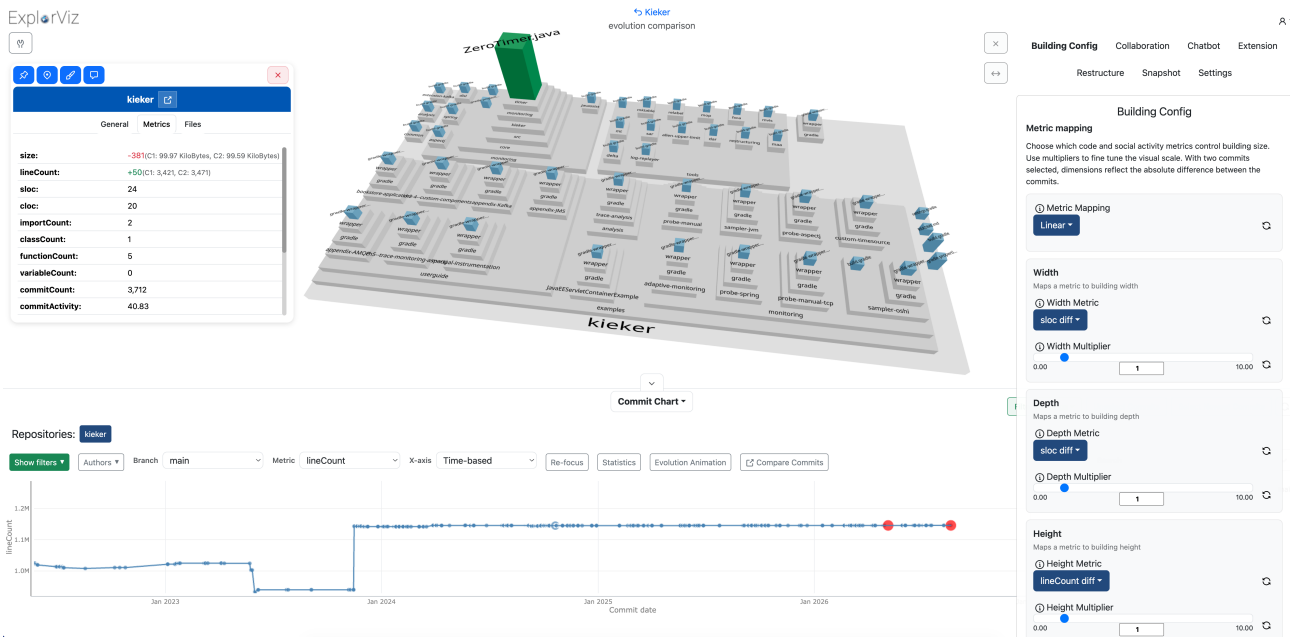

Figure 1: Visual comparison of two recent commits in the Kieker repository.

5 Visualization

The frontend of ExplorViz employs the city metaphor [2] by default to visualize the collected data on a software's structure. In this visualization, applications are represented as cities with a gray foundation and directories are mapped to nested districts in the city. For Kieker, we chose to visualize the whole repository as one city. However, the monitoring component, analysis component, and various tools that are contained in the Kieker repository could also be visualized as individual software cities.

Cuboid buildings representing files are placed on the city foundation or in the districts in accordance with the analyzed directory tree. Clicking on a city foundation or district opens a popup that displays information about the files it contains, e.g., accumulated file metrics. Clicking on a building opens a popup that displays the calculated file metrics and data about contained data structures such as classes. Raw source code cannot be displayed in ExplorViz, but a button in the popups links to the corresponding file or directory for the selected commit on the GitHub website.

Analyzed commits are displayed as a chart in the bottom bar of the visualization. An exemplary visualization is illustrated in Figure 1. In the example, the commit chart is set to display the accumulated line count metric for each commit that has been directly committed to the `main` branch. Two recent commits are selected, which triggers the evolution comparison mode of ExplorViz and adjusts the color scheme to highlight differences between the commits. Green indicates that files or folders have been added, red indicates removal, and blue represents files that have been modified between the selected software versions. In the given example, where unchanged files are filtered out, mostly Gradle files have been modified and the file `ZeroTimer.java` has been added. The dimensions of the buildings, representing files, can be mapped to the collected metrics. In this case, the area of the buildings is set to represent the difference in source lines of code and the height corresponds to the difference in the number of lines. A popup of the city foundation is opened on the left of the visualization and shows that 50 lines have been added overall to the repository. This suggests that only minor changes to the business logic were incorporated into the main branch of Kieker's software within the given timeframe.

The displayed line count metric in the chart of Figure 1 points towards a significant change in commit `7b819b3` in November 2023. This merge commit adds features from OceanDSL[3] to Kieker. In terms of added files, this is the biggest single change in Kieker's commit history, surpassed only by commit `b526bda` in November 2009, which introduced the JPetStore[4] codebase to Kieker.

The available metrics can also be displayed as a heat map, with the buildings in the city colored accordingly. Alongside metrics derived from the file analysis such as source lines of code (sloc), ExplorViz can also calculate metrics based on data from the GitHub API, e.g. based on GitHub issue activity and pull requests.

To inspect the evolution of the Kieker repository over a longer period, ExplorViz provides the ability to animate the software evolution. By default, a spiral

[3] `https://oceandsl.uni-kiel.de/oceandsl/`
[4] `https://github.com/mybatis/jpetstore-6`

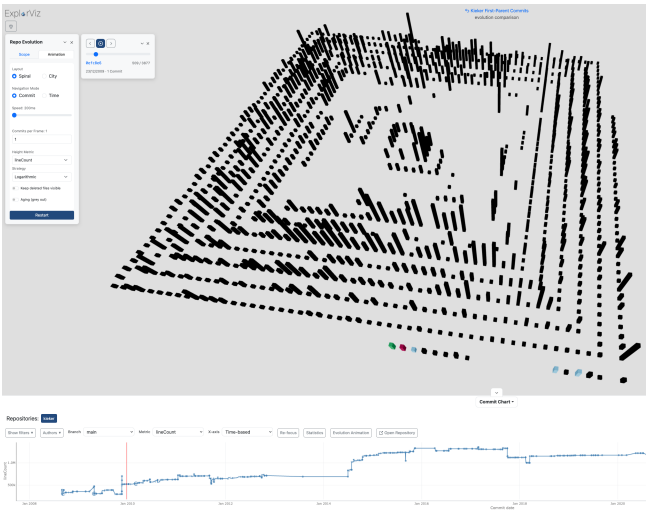

Figure 2: Animation of Kieker's software evolution.

layout is used that places the file cuboids in a rectangular spiral, sorted by their first appearance in the commit history. Added files are placed at the current end of the spiral and color-coded just like for the regular comparison of commits. Unchanged files are colored black. An example for the animated software evolution is presented in Figure 2. A red bar in the commit chart indicates at which commit the animation currently is. The speed of the animation, the number of commits that are visualized per animation step, and other visualization options can be configured via a panel on the left of the screen. For the animation, it is recommended to consider only first-parent commits of a given branch to get a coherent linear commit history. Gaps in the spiral represent files that have been deleted or moved. As files are placed in accordance with their first appearance in the repository, files close to the center of the spiral that have not been deleted represent long-lived files. An example of such a file is the `HISTORY` file, which contains a textual overview of the Kieker version history.

6 Summary and Outlook

We presented a software visualization approach with our tool ExplorViz to analyze and visualize data collected from GitHub repositories. We applied our approach to Kieker to show its applicability to long-lived software systems. ExplorViz is available as open source software[5] and we provide a reproduction package with Docker images/volumes, source code of ExplorViz, benchmark results, and a video [9].[6]

For future work, we plan to extend our analysis to include data on file dependencies and further improve the overall performance of ExplorViz.

[5] `https://github.com/explorviz`
[6] `https://youtu.be/lvM3yuuVkUs`

References

[1] T. J. Parr and R. W. Quong. "ANTLR: A predicated-LL(k) parser generator". In: *Software: Practice and Experience* 25.7 (1995), pp. 789–810. DOI: `https://doi.org/10.1002/spe.4380250705`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.4380250705`.

[2] R. Wettel and M. Lanza. "CodeCity: 3D Visualization of Large-Scale Software". In: *Companion of the 30th International Conference on Software Engineering*. ICSE Companion '08. Leipzig, Germany, 2008, pp. 921–922. DOI: `10.1145/1370175.1370188`.

[3] R. Wettel, M. Lanza, and R. Robbes. "Software Systems as Cities: A Controlled Experiment". In: *Proceedings of the 33rd International Conference on Software Engineering*. ICSE '11. Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, pp. 551–560. DOI: `10.1145/1985793.1985868`.

[4] F. Fittkau, A. Krause, and W. Hasselbring. "Software landscape and application visualization for system comprehension with ExplorViz". In: *Information and Software Technology* 87 (July 2017), pp. 259–277. DOI: `10.1016/j.infsof.2016.07.004`.

[5] W. Hasselbring, A. Krause, and C. Zirkelbach. "ExplorViz: Research on software visualization, comprehension and collaboration". In: *Software Impacts* 6 (Nov. 2020). DOI: `10.1016/j.simpa.2020.100034`.

[6] D. Moreno-Lumbreras et al. "CodeCity: On-Screen or in Virtual Reality?" In: *2021 Working Conference on Software Visualization (VISSOFT)*. 2021, pp. 12–22. DOI: `10.1109/VISSOFT52517.2021.00011`.

[7] G. Occhipinti et al. "SYN: Ultra-Scale Software Evolution Comprehension". In: *2023 IEEE/ACM 31st International Conference on Program Comprehension (ICPC)*. 2023, pp. 69–73. DOI: `10.1109/ICPC58990.2023.00020`.

[8] S. Yang et al. "The Kieker Observability Framework Version 2". In: *Companion of the 16th ACM/SPEC International Conference on Performance Engineering*. ICPE '25. ACM, May 2025, pp. 11–15. DOI: `10.1145/3680256.3721972`.

[9] M. Hansen et al. *Analysis and Visualization of Kieker's Software Evolution*. 2026. DOI: `10.5281/ZENODO.22093304`.