# Analysis and Visualization of Unit Test Traces With Kieker and ExplorViz

Malte Hansen
Kiel University
Kiel, Germany

David Georg Reichelt
Lancaster University Leipzig &
URZ Leipzig

Wilhelm Hasselbring
Kiel University
Kiel, Germany

## Abstract

Testing software and understanding how software systems evolve are essential for maintaining code quality and identifying changes that affect program behavior. We present an approach for visually analyzing unit test execution traces in the context of software evolution. These traces are collected using the Peass (Performance Analysis of Software Systems) tool in conjunction with Kieker and OpenTelemetry. We use ExplorViz, a web-based software visualization tool, to visualize the collected trace data alongside data from static program analysis. Visualizing how unit tests interact with evolving codebases over time enables insights into changes in test coverage and the impact of refactoring or feature additions to program behavior. We showcase our approach experimentally using the OpenHAB Zigbee binding as the system under test.

## 1 Introduction

Software testing is an integral part of software development to increase software quality, document it, and increase a development's team confidence in the software under development [4]. In addition, developers spend a considerable amount of time to read and understand source code [6]. Therefore, techniques and tools to increase program comprehension such as software visualization are needed [1].

In this work, we present an approach to analyze and visualize traces from unit tests in the context of the software's evolution. To collect and analyze unit test traces, we employ the Peass (Performance Analysis of Software Systems [9]) tool in combinations with Kieker and OpenTelemetry [13, 14]. ExplorViz, as web-based 3D software visualization tool, is used to visualize the trace data with additional data from static program analysis [8, 10].

OpenHab[1] (open Home Automation Bus) is an open source home automation platform. It enables automating the use of various things in the household, such as lighting, heating, security systems, and entertainment devices. A central component of OpenHab is the Zigbee binding, which enables the use of devices that follow the Zigbee standard using the OpenHab platform. In this work, we use the OpenHab Zigbee binding as System under Test (SuT) to demonstrate the capabilities of ExplorViz for visualizing the evolution of unit tests.

## 2 Related Work

It is common for test suites to compute metrics such as code coverage or present the cumulated results in 2D visualizations [3]. Dreef et al. present an approach to display the test coverage within a software system with matrix visualizations [11]. Therein, methods that contain test code and methods of the system under test are placed on the axis on the matrix. The cells of the resulting grid can visualize the code coverage on a method level and by color coding failing tests can be easily identified. Our approach also inspects the code coverage on the level of methods but uses a 3D visualization and combines static and dynamic analysis.

Tahir et al. also combine static and dynamic analysis to visualize test suites [7]. They use a dependency graph to visualize the distribution of test code and their relation to production code in open source software systems, while we use the city metaphor.

## 3 Tool Architecture

Our tool ExplorViz is able to process and visualize data from both dynamic and static program analysis [12]. The process of data collection for a SuT is presented in the following subsections.

### 3.1 Unit Test Instrumentation

For the trace visualization, ExplorViz needs OpenTelemetry traces. These traces are created by execution of a SuT with instrumentation. This requires an automation of the instrumentation, e.g., of adding the monitoring probes to the source code, and automation of the execution.

For instrumentation and execution automation, we use the tool Peass [9], which has been built to detect performance changes in unit tests. For unit test analysis, Peass automates the instrumentation by injection of either the monitoring probes directly in the code or by automatically adding the Kieker agents into the build tools maven and Gradle. We use this

[1] `https://github.com/openhab`

Peass auto-instrumentation of unit tests in order to obtain Kieker traces of all unit tests.

Afterwards, the traces need to be imported into ExplorViz. To do so, we first replay them using Kieker's log-replayer, which can export them to any Kieker tool. For the import into ExplorViz, we use Kieker's otel-transformer [13]. Finally, ExplorViz reads the obtained trace and can visualize them. Figure 1 depicts this process.

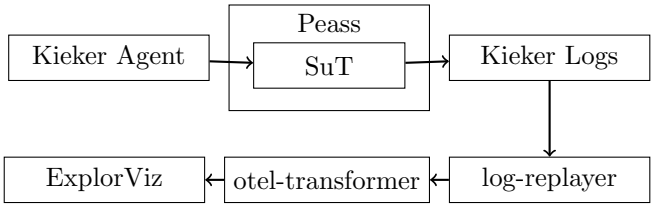


Figure 1: Trace Gathering Process

### 3.2 Analysis of Code Evolution

In addition to analyzing the dynamic properties of the unit tests, we examine the static properties of the SuT, including the package and class structure and the methods they contain for different software versions. Given the URL of a Git repository, the ExplorViz software system can analyze a given range of commits. The frontend of ExplorViz requests both the data from dynamic and static program analysis and combines them into a unified visualization.

## 4 Visualization

To visualize the structure of a software system, ExplorViz employs the city metaphor [2, 5]. Packages are displayed as hierarchically nested districts on a gray foundation that represents an application (see Figure 2). Classes are visualized as cuboid buildings inside those districts. Packages can be closed to hide contained subpackges and classes. Collected metrics about the SuT, such as number of methods or lines of code, can be mapped to the width, depth, or height of a building in the visualization.

The recorded method calls between objects of classes are accumulated and represented by yellow arcs. By default, the number of requests between two classes is mapped to the width of the arc. Arcs can be hovered with a mouse, to display which method calls were captured by the dynamic analysis.

**Unit Tests** The data from the execution of test cases and the data from static program analysis is combined in the frontend. As both dynamic and static program data is associated with a commit identifier, the user can first select the desired commit from a commit graph. Then, all the data points from dynamic program analysis that match the commit identifier are selectable from a timeline. Thus, the results from multiple runs of a the tests may be selectable. The resulting visualization of the software system can be searched, filtered, and offers many visual customization options.

In Figure 2, the OpenHab Zigbee binding is used as a SuT and visualized. The data for the static analysis is sourced from commit 2f0a43 in the official GitHub repository.[2] The repository contains 136 Java files and over 20,000 lines of code for this commit. Users can select the commit to be visualized. Via a configuration panel to the right, the layout and mapping of metrics can be controlled, here mapping the number of methods to the dimensions (volume) of the visual representation of a class. As it is known for each class, which of its methods were captured by static or dynamic analysis (or both), we can compute the ratio between those two categories, giving us a code coverage of the instrumented tests on a method level. We apply a heat map to the classes to visualize this metric in the given example. Blue in the heat map indicates that most or all of a class's methods were only captured by static program analysis. Conversely, red indicates that dynamic program analysis, or a combination of static and dynamic program analysis, captured most of a class's methods. When another commit is selected, the changes in classes and metrics are animated, and the applied heat map is updated. It is possible that for the OpenHab Zigbee binding many classes are only present in the static program analysis and thus not covered by the instrumented unit tests.

## 5 Summary and Outlook

We introduced a tool chain for analyzing the execution traces of unit tests in the context of software evolution, combining dynamic runtime data from Kieker with static program analysis in ExplorViz. The resulting visualization in ExplorViz was made concrete by applying our approach to the OpenHab Zigbee binding.

We plan to extend our approach in the future by integrating it into a continuous integration pipeline. In addition, the information whether a test passed can be incorporated to enable visual inspection of failed tests. In terms of performance engineering, we envision to integrate with performance benchmarking tools and make their results visually explorable.

## References

[1] J. T. Stasko, M. H. Brown, and B. A. Price. *Software Visualization*. Cambridge, MA, USA: MIT Press, 1997.

[2] C. Knight and M. Munro. "Comprehension with[in] Virtual Environment Visualisations". In: *Proceedings Seventh International Workshop on Program Comprehension*. 1999, pp. 4–11. DOI: 10.1109/WPC.1999.777733.

[2] https://github.com/openhab/org.openhab.binding.zigbee

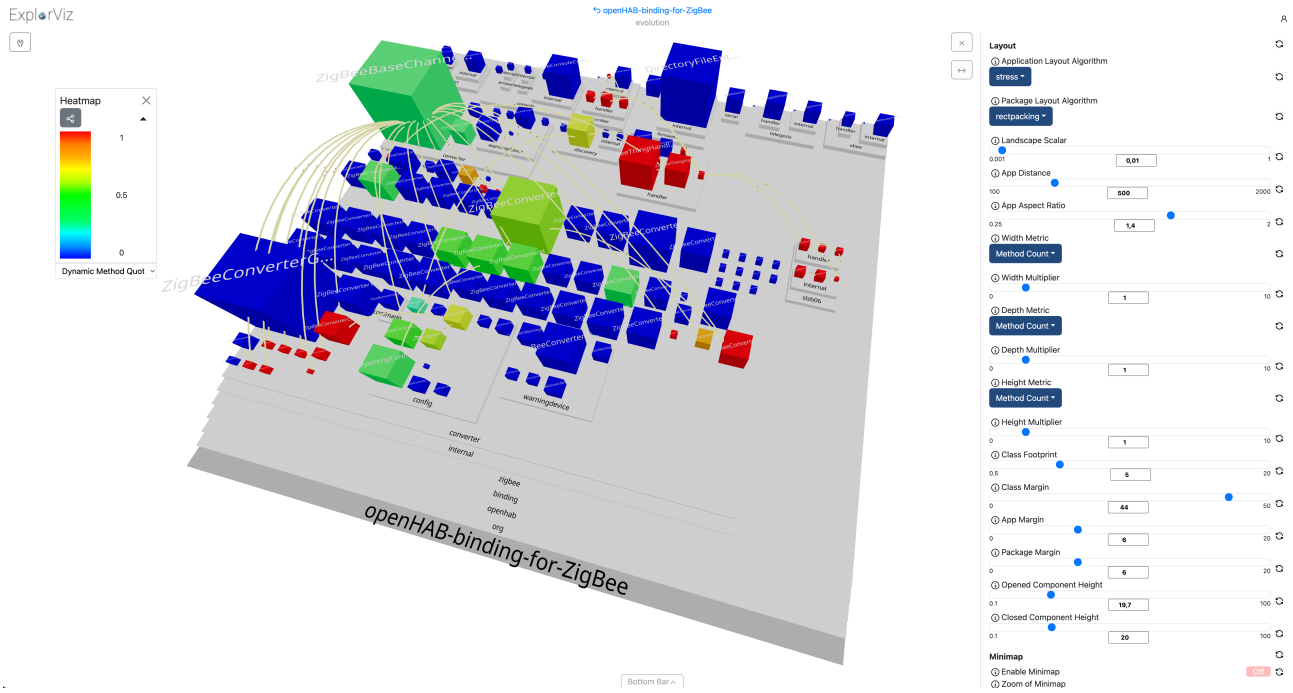


Figure 2: Visualization of the openHAB binding for ZigBee in ExplorViz.


[3] J. Jones, M. Harrold, and J. Stasko. "Visualization of test information to assist fault localization". In: *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*. 2002, pp. 467–477. DOI: 10.1145/581396.581397.

[4] L. Moonen et al. "On the Interplay Between Software Testing and Evolution and its Effect on Program Comprehension". In: *Software Evolution*. Springer Berlin Heidelberg, 2008, pp. 173–202. DOI: 10.1007/978-3-540-76440-3_8.

[5] R. Wettel, M. Lanza, and R. Robbes. "Software systems as cities: a controlled experiment". In: *ICSE*. 2011, pp. 551–560. DOI: 10.1145/1985793.1985868.

[6] R. Minelli, A. Mocci, and M. Lanza. "I Know What You Did Last Summer - An Investigation of How Developers Spend Their Time". In: *ICPC*. 2015. DOI: 10.1109/ICPC.2015.12.

[7] A. Tahir and S. G. MacDonell. "Combining Dynamic Analysis and Visualization to Explore the Distribution of Unit Test Suites". In: *2015 IEEE/ACM 6th International Workshop on Emerging Trends in Software Metrics*. 2015, pp. 21–30. DOI: 10.1109/WETSoM.2015.12.

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

[9] D. G. Reichelt, S. Kühne, and W. Hasselbring. "PeASS: A Tool for Identifying Performance Changes at Code Level". In: *ASE*. 2019. DOI: 10.1109/ASE.2019.00123.

[10] 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.

[11] K. Dreef, V. K. Palepu, and J. A. Jones. "Global Overviews of Granular Test Coverage with Matrix Visualizations". In: *2021 Working Conference on Software Visualization (VISSOFT)* 00 (2021), pp. 44–54. DOI: 10.1109/vissoft52517.2021.00014.

[12] A. Krause-Glau et al. "Visual Integration of Static and Dynamic Software Analysis in Code Reviews via Software City Visualization". In: *VISSOFT*. 2024, pp. 144–149. DOI: 10.1109/VISSOFT64034.2024.00028.

[13] D. G. Reichelt et al. "Interoperability From Kieker to OpenTelemetry: Demonstrated as Export to ExplorViz". In: *SSP 2024*. PID: 20.500.12116/46200. 2025, pp. 20–22.

[14] S. Yang et al. "The Kieker Observability Framework Version 2". In: *ICPE '25 Companion*. 2025. DOI: 10.1145/3680256.3721972.