The Kieker Observability Framework Version 2

Shinhyung Yang
shinhyung.yang@email.uni-kiel.de
Kiel University
Kiel, Schleswig-Holstein, Germany

David Georg Reichelt
d.g.reichelt@lancaster.ac.uk
Lancaster University Leipzig
& Universität Leipzig
Leipzig, Saxony, Germany

Reiner Jung
reiner.jung@email.uni-kiel.de
Kiel University
Kiel, Schleswig-Holstein, Germany

Marcel Hansson
marcel.hansson@uni-hamburg.de
University of Hamburg
Hamburg, Germany

Wilhelm Hasselbring
hasselbring@email.uni-kiel.de
Kiel University
Kiel, Schleswig-Holstein, Germany

Abstract

Observability of a software system aims at allowing its engineers and operators to keep the system robust and highly available. With this paper, we present the Kieker Observability Framework Version 2, the successor of the Kieker Monitoring Framework.

In this tool artifact paper, we do not just present the Kieker framework, but also a demonstration of its application to the TeaStore benchmark, integrated with the visual analytics tool ExplorViz. This demo is provided both as an online service and as an artifact to deploy it yourself.

CCS Concepts

• **Software and its engineering → Software reverse engineering**; **Traceability**; **Interoperability**; **Software performance**.

Keywords

Observability Engineering, Monitoring, Tracing, Microservices, Visual Analysis, Dynamic Analysis, Research Software Sustainability

ACM Reference Format:
Shinhyung Yang, David Georg Reichelt, Reiner Jung, Marcel Hansson, and Wilhelm Hasselbring. 2025. The Kieker Observability Framework Version 2. In *Companion of the 16th ACM/SPEC International Conference on Performance Engineering (ICPE Companion '25), May 5–9, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3680256.3721972

1 Introduction

Observability is a fundamental property of a software system that should be considered during system design [18]. Using telemetry data allows various stakeholders to understand the system and answer key questions. Monitoring and testing are integral parts of observability, and both benefit from and contribute to it. Observability of a system has become significant as the architecture of software systems is increasingly containerized with distributed microservices. These requirements have conceived observability tools that provide insights into the internal behavior of software systems using traces, metrics, and logs [7].

In this tool artifact paper, we introduce the Kieker Observability Framework Version 2, the successor of the Kieker Monitoring Framework. We demonstrate Kieker's observability, analysis, and visualization with our TeaStore-Kieker-ExplorViz demo [26], which includes our initial effort for the interoperability between Kieker and OpenTelemetry [8].

The new contribution of this tool artifact paper is not just the Kieker framework, but also a demonstration of its application to the TeaStore benchmark [40], integrated with the visual analytics tool ExplorViz [9]. This demo is provided both as an online service and as an artifact to deploy it yourself.

Section 2 summarizes Kieker's impact, so far. Observability Engineering is introduced in Section 3, before instrumentation and data collection for observability with Kieker is presented in Section 4. The tool artifact is described in Section 5. Section 6 discusses some related work, before the paper is summarized in Section 7 with an outlook to future work.

2 Impact of Kieker

The Kieker Observability Framework started as a monitoring framework in 2006 [11], supporting application performance monitoring and analysis. Kieker enables observing software systems with Java agents, which produce metrics, logs, and trace data. Using the trace data, Kieker allows for reverse engineering and visualization of the software architecture. The Kieker developers have been continuously monitoring its performance overhead with the MooBench monitoring overhead microbenchmark [29, 42].

Kieker allows for the analysis of observed data. The new Kieker Version 2 incorporates the TeeTime pipe-and-filter framework [44], which has restructured the analysis pipeline. It provides an intuitive way for Kieker users to build new analysis applications.

Research areas in which Kieker was successfully employed are performance analysis [4, 27, 28, 31, 48], benchmarking streaming engines [46], anomaly detection [6, 22], online capacity management [36, 41], analysis of software structure and metrics [1, 2, 15, 19, 24, 32], software architecture reconstruction [5], test case prioritization [3], and extraction of load profiles [37, 39]. Martinez Saucedo et al. [21] recently identified Kieker as the most cited tool to assist with the migration of monolithic systems to microservices.

ICPE Companion '25, Toronto, ON, Canada

ACM ISBN 979-8-4007-1130-5/2025/05
https://doi.org/10.1145/3680256.3721972

Several Kieker-related research projects have been and are being conducted over the years. Examples are diagnoseIT (Expert-Guided Automatic Diagnosis of Performance Problems) [12], DynaMod (Dynamic Analysis for Model-Driven Software Modernization) [35], and OceanDSL (Architecture analysis, interactive visualization) [17]. As reported by Hasselbring and van Hoorn [10], Kieker was also employed in several industrial collaborations and technology transfer projects. More information on Kieker's impact may be found in Hasselbring and van Hoorn [11].

Kieker is included in the SPEC Research Group's repository of peer-reviewed tools for quantitative system evaluation and analysis.[1] The review process and the final acceptance of Kieker for this tool repository triggered manifold activities in the Kieker project and helped to further improve Kieker's product quality (both code and documentation); Kieker's visibility increased considerably, with **480** citations to the ICPE 2012 Kieker tool demo paper [38], so far.[2]

3 Observability Engineering

The term observability has been introduced in the context of control theory [18]. In this context, observability is a measure of how well the internal states of a system can be inferred from knowledge of its external outputs. For software systems, this means how well the internal state of a software system can be inferred from its monitoring data. Observability Engineering [20] focuses on designing, building, and maintaining systems that enable us to understand the internal state of software systems based on monitoring data. It is a crucial part of software engineering, especially in the context of complex distributed systems such as microservices and cloud-native architectures. Figure 1 (introduced by Peter Bourgon[3]) presents the three pillars of observability: metrics, logging, and tracing [7]:

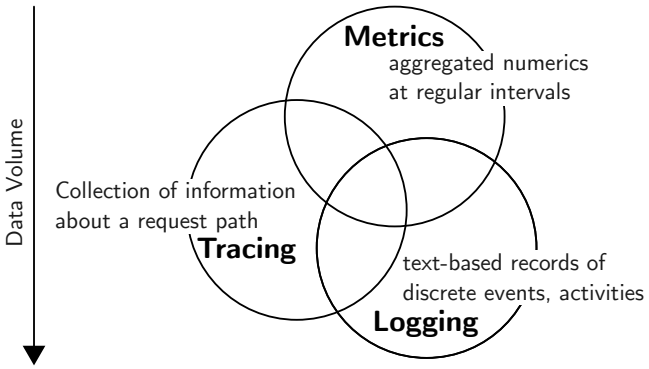

Figure 1: Visual illustration of metrics, logging, and tracing. They are ordered vertically with a higher data volume at the bottom.

- Metrics are aggregated numerical data representing the performance and behavior of systems (e.g., CPU usage, response time, error rates), which are collected at regular intervals.
- Logging is the collection of text-based records of discrete events or activities within a system (e.g., status and debugging messages).
- Tracing is the collection of information about the path a request takes as it moves through a distributed system. Traces are aggregations of highly structured log events, which are called *spans* [8].

Observability is not a substitute for monitoring, nor does it obviate the need for monitoring; they are complementary. Observability is a fundamental property of a software system that must be intentionally considered during system design [20]. It enables different stakeholders to understand the system and answer key questions by utilizing telemetry data. Monitoring and testing are integral parts of observability, and both benefit from and contribute to it.

4 Instrumentation and Data Collection with Kieker

The three pillars of observability [7] are addressed with Kieker as follows:

- Metrics are sampled using Kieker's periodic sampler. It allows for the incorporation of different samplers to collect metrics. Kieker currently supports sampling the JVM garbage collector and sampling CPU and memory information.
- Logs are supported by Kieker as a generic way to handle event-centric messages.
- Traces are recorded by Kieker's tracing agents. Kieker features distributed tracing that records the internal behaviors of the target system in the scopes of components in the deployed microservices. Kieker traces consist of records that are spans.

Kieker enables observability of a software system by collecting data with Kieker agents. Furthermore, Kieker is designed to analyze and visualize the observed software system. Below, we take a look at instrumenting software applications with Kieker (Section 4.1), and on benchmarking the incurred overhead (Section 4.2).

4.1 Instrumentation

Observability data can be obtained either using instrumentation of code or sampling of existing data collection interfaces, e.g., JMX beans or tracepoints of the Linux kernel. Kieker focuses on instrumentation of application code. This can be done either automatically or via manual source code instrumentation. With Java, the automated instrumentation is performed by the "`-javaagent`" interface of the JVM, which allows developers to change the bytecode during application loading. Kieker supports instrumentation via AspectJ, ByteBuddy, DiSL, and Javassist [25]. Kieker also supports automated instrumentation of existing source code [30], if the Java source code is available. Additionally, Kieker supports automated injection into servlets and Spring applications, and sampling CPU metrics using various OSHI (Operating System and Hardware Information library) samplers. Besides its Java instrumentation, Kieker allows to instrument C, Fortran and Python code [16, 33].

[1] https://research.spec.org/tools/overview/
[2] https://scholar.google.com/scholar?cluster=11724904481060384258 (Jan. 15, 2025)
[3] https://peter.bourgon.org/blog/2017/02/21/metrics-tracing-and-logging.html

4.2 Benchmarking Observability Overhead

The instrumentation for observability imposes overhead, which should be minimized in production environments. Kieker's overhead has been continuously measured since 2015[4] using the MooBench microbenchmark [42]. It measures both the overall overhead of Kieker tracing and the overhead that is incurred by different factors, which are the instrumentation itself, the measurement, and the final serialization of data. Since the measurement of overhead in the JVM requires coping with its non-determinism, the measurement requires the repetition of the measured workload. Therefore, MooBench repeats a method call for a specified iteration count. This method recursively calls itself for a given recursion depth d, and waits for a specified duration t in the leaf node.

MooBench also measures the overhead of the tracing tools OpenTelemetry and inspectIT. Studies show that the overhead incurred by these tools is significantly higher than the overhead of Kieker's instrumentation [29]. This is partially achieved by the complete decoupling of the application thread and the log writer thread. The application thread executes asynchronously immediately after a Kieker record has been written into the queue [34, 43].

5 The Tool Artifact

The architecture of our tool artifact is illustrated in Figure 2. Our tool artifact consists of four software systems (top-level components), which are available as an online services (non-installation) and an offline docker compose file.

The TeaStore software systems consists of seven microservices, each deployed in its own Docker container. The TeaStore microservices, except the database (MariaDB), are instrumented via Kieker agents. The ExplorViz software system consists of eleven microservices, again each deployed in its own Docker container. The connection between the TeaStore system and the ExplorViz system is accomplished via the embedded Kieker agents that send the monitoring data to the Kieker OpenTelemetry Transformer, which forwards it after transformation to the OpenTelemetry Collector.[5] The OpenTelemetry Collector is part of the ExplorViz system, as one of its microservices.

First, JMeter load tests the TeaStore microservices via a configurable number of requests. The Kieker agents in the TeaStore microservices then send the traces to the Kieker OpenTelemetry Transformer. The Kieker OpenTelemetry Transformer consists of several TeeTime stages [44], which translate the received Kieker traces into OpenTelemetry spans. The ExplorViz tool receives the translated spans and renders the dynamic visualization of the TeaStore architecture.

The online service has two URLs, one for the TeaStore Web UI,[6] and another URL for the ExplorViz Web UI.[7] Kiel University hosts the online service, which is available on the SustainKieker homepage.

Installation. Our tool artifact runs with a Docker compose script file. We use it to (1) retrieve Docker images from DockerHub, or (2) build the Docker images locally for archival purposes. The script simplifies the deployment of the TeaStore and ExplorViz software systems, JMeter, and the Kieker OpenTelemetry Transformer. We tested the out-of-the-box experience on all major platforms, Linux, macOS, and Windows. For the open-source access, we use a GitHub repository[8] to provide our tool artifact. A Zenodo upload is available to download and run the current snapshot of the tool artifact.[9]

The tool artifact requires installation of Docker and Git (optional) on all platforms. The installation and launching of our tool artifact follows Listing 1.

```
git clone https://github.com/kieker-monitoring/tool-artifact
cd tool-artifact
docker compose up -d
```

Listing 1: Installing and launching the tool artifact

After the tool artifact launches, two web servers become available: the TeaStore WebUI on the TCP port 8080 (Figure 2.(1)) and the ExplorViz frontend on the TCP port 8082 (Figure 2.(2)).

ExplorViz [9] is an open-source research visualization tools, which uses dynamic analysis techniques to provide a live trace visualization of software landscapes, in our case the microservices of the TeaStore. ExplorViz targets system and program comprehension for software landscapes or single applications while still providing details on the communication within an application. It utilizes the 3D city metaphor combined with an interactive concept of showing only details that are in focus of the analysis.

The JMeter load driver initially launches a TestPlan that generates 6400 requests on the TeaStore. This way, ExplorViz can visualize those parts of the TeaStore that were called for this load. Afterwards, only the interaction of a user with the TeaStore Web interface is observed and visualized with ExplorViz.

Documentation. Comprehensive documentation, including an introduction and a quick start guide for Kieker Version 2 is available online.[10] In addition, a (JavaDoc) API documentation[11] provides insights for developers to further expand or work with the Kieker source files.

6 Related Work

Related work exists in the field of *observability tools* and *software visualization*.

Concerning *observability tools*, a variety of alternatives to Kieker exists [14]. Example proprietary tools are DataDog[12] and the DynaTrace One Agent.[13] Example open source tools, are Jaeger[14] and Elastic APM.[15] For these observability tools integrations with OpenTelemetry exist, as presented in our work. Therefore, they could also be used for TeaStore instrumentation and the date export to ExplorViz; however, there is currently no available implementation.

Yang et al. [45] present Cloudprofiler, a tool for profiling of systems that are processing streaming workloads. They focus on the synchronization of multiple virtual machines and compress the logs to reduce I/O usage. They benchmark Cloudprofiler using the

[4] https://kieker-monitoring.net/performance-benchmarks/
[5] https://opentelemetry.io/docs/collector/
[6] https://teastore.sustainkieker.kieker-monitoring.net/
[7] https://explorviz.sustainkieker.kieker-monitoring.net/
[8] https://github.com/kieker-monitoring/tool-artifact
[9] https://doi.org/10.5281/zenodo.14989908
[10] https://kieker-monitoring.readthedocs.io/en/latest/index.html
[11] https://api.kieker-monitoring.org/2.0.2/
[12] https://docs.datadoghq.com/agent
[13] https://www.dynatrace.com/platform/oneagent/
[14] https://www.jaegertracing.io/
[15] https://www.elastic.co/observability/

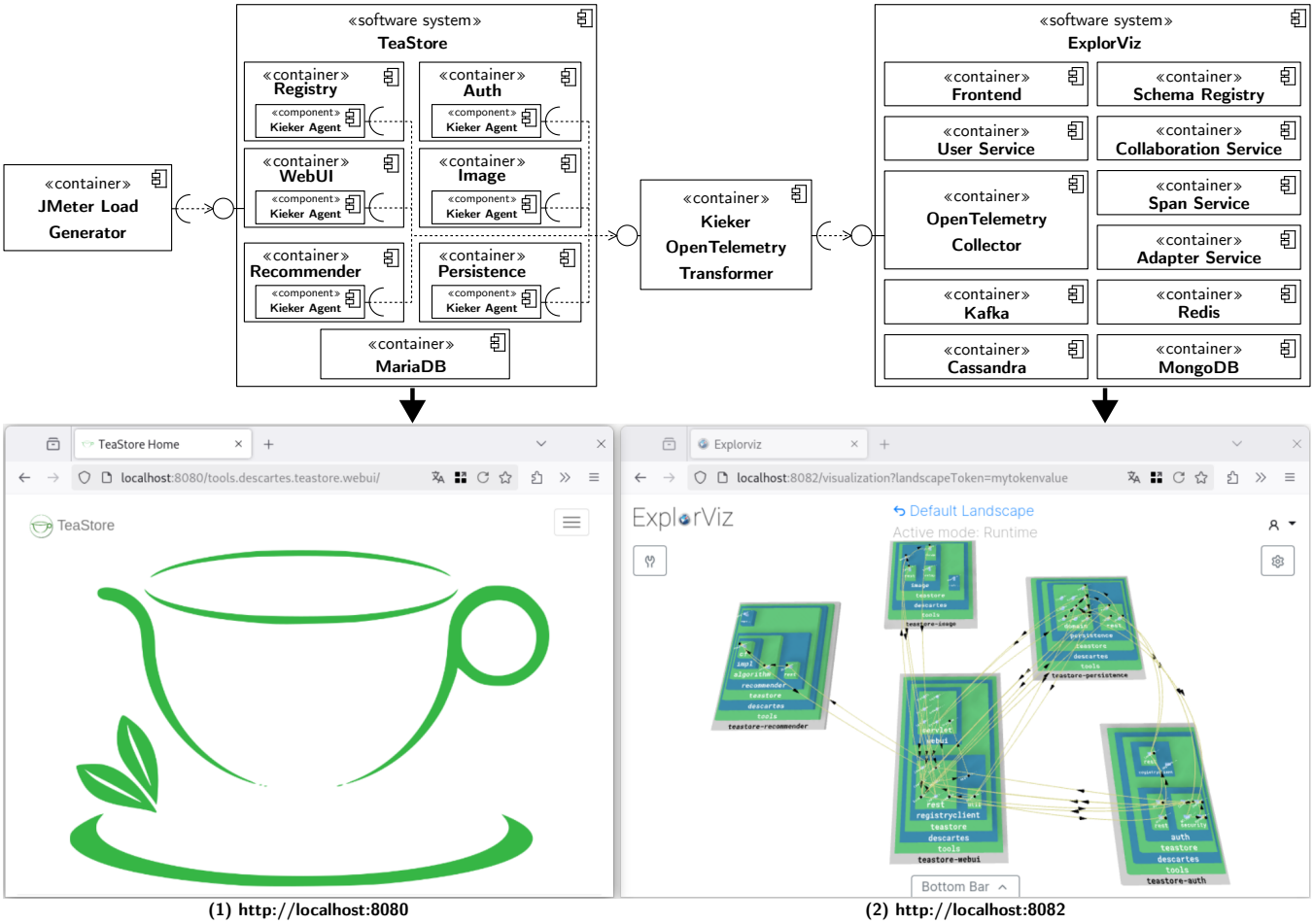

Figure 2: Architecture of the tool artifact. The provided Docker compose file launches all the containers in the component diagram, and it is accessible with two web servers on the Ports 8080 (1) and 8082 (2).

Yahoo streaming benchmark and measure an overhead of 2.2 %. The MooBench benchmark with Cloudprofiler finds that the overhead of Cloudprofiler is close to the overhead of Kieker [47]. An integration of Cloudprofiler and the OpenTelemetry standard does not exist currently, therefore, it could not be used to instrument the TeaStore and export observability data to ExplorViz.

Concerning *visualization*, different visualizations of Kieker traces exist. Müller and Fischer [23] provide an alternative visualization of Kieker traces based on jQAssist. Although they also visualize call trees, their visualization is less flexible than the ExplorViz UI. Alternative approaches use static software analysis data [13]. While they can also visualize the component structure of an application, they cannot present runtime information such as the number of calls between the components, since these data need to be obtained by instrumentation.

7 Summary and Outlook

The application of the Kieker observability framework to the TeaStore benchmark, integrated with the visual analytics tool ExplorViz is presented in this paper. This tool artifact is provided both as an online service and as an artifact to deploy it yourself.

With our current SustainKieker project,[16] we intend to further sustain Kieker as a reusable, high-quality observability framework that will be employed in a wider variety of research areas and used by a larger community. The online version of the tool artifact, as presented in this paper, also serves as a demo application to use Kieker for observability.

Several demo applications to be developed in SustainKieker will serve as basis for new interactive tutorial examples for Kieker. Besides the online manual and step-by-step instructions in tutorials, we aim to produce screencasts based on the demos illustrating core features of Kieker. We will also empirically evaluate the effect of our new tutorials with the embedded online demos.

Acknowledgment

This research is funded by the Deutsche Forschungsgemeinschaft (DFG – German Research Foundation), grant no. 528713834.

[16] https://sustainkieker.kieker-monitoring.net/

References

[1] Bernardo Andrade, Samuel Santos, and Antonio Rito Silva. 2023. A Comparison of Static and Dynamic Analysis to Identify Microservices in Monolith Systems. In *Software Architecture*. 354–361. https://doi.org/10.1007/978-3-031-42592-9_25

[2] Lingli Cao and Cheng Zhang. 2022. Implementation of Domain-oriented Microservices Decomposition based on Node-attributed Network. In *ICSCA 2022 (ICSCA 2022)*. 136–142. https://doi.org/10.1145/3524304.3524325

[3] Jianlei Chi, Yu Qu, Qinghua Zheng, Zijiang Yang, Wuxia Jin, Di Cui, and Ting Liu. 2020. Relation-based test case prioritization for regression testing. *Journal of Systems and Software* 164 (2020), 110539. https://doi.org/10.1016/j.jss.2020.110539

[4] Jürgen Cito, Philipp Leitner, Christian Bosshard, Markus Knecht, Genc Mazlami, and Harald Gall. 2018. PerformanceHat: augmenting source code with runtime performance traces in the IDE. In *ICSE 2018*. 41–44. https://doi.org/10.1145/3183440.3183481

[5] Robert Dabrowski. 2012. On Architecture Warehouses and Software Intelligence. In *FGIT 2012*. 251–262. https://doi.org/10.1007/978-3-642-35585-1_35

[6] Jens Ehlers, André van Hoorn, Jan Waller, and Wilhelm Hasselbring. 2011. Self-Adaptive Software System Monitoring for Performance Anomaly Localization. In *Proc. ICAC 2011*. 197–200. https://doi.org/10.1145/1998582.1998628

[7] Radoslav Gatev. 2021. *Observability: Logs, Metrics, and Traces*. Apress, 233–252. https://doi.org/10.1007/978-1-4842-6998-5_12

[8] Daniel Gomez Blanco. 2023. *Practical OpenTelemetry: Adopting Open Observability Standards Across Your Organization*. Apress. https://doi.org/10.1007/978-1-4842-9075-0

[9] Wilhelm Hasselbring, Alexander Krause, and Christian Zirkelbach. 2020. ExplorViz: Research on software visualization, comprehension and collaboration. *Software Impacts* 6 (Nov. 2020). https://doi.org/10.1016/j.simpa.2020.100034

[10] Wilhelm Hasselbring and Andre van Hoorn. 2015. *Open-Source Software as Catalyzer for Technology Transfer: Kieker's Development and Lessons Learned*. TR-1508. Kiel Univ. https://nbn-resolving.org/urn:nbn:de:gbv:8:1-zs-00000275-a1

[11] Wilhelm Hasselbring and André van Hoorn. 2020. Kieker: A monitoring framework for software engineering research. *Software Impacts* 5 (2020), 100019. https://doi.org/10.1016/j.simpa.2020.100019

[12] Christoph Heger, André van Hoorn, Dusan Okanovic, Stefan Siegl, and Alexander Wert. 2016. Expert-Guided Automatic Diagnosis of Performance Problems in Enterprise Applications. In *EDCC 2016*. https://doi.org/10.1109/EDCC.2016.16

[13] Adrian Hoff, Lea Gerling, and Christoph Seidl. 2022. Utilizing Software Architecture Recovery to Explore Large-Scale Software Systems in Virtual Reality. In *VISSOFT 2022*. 119–130. https://doi.org/10.1109/VISSOFT55257.2022.00020

[14] Andrea Janes, Xiaozhou Li, and Valentina Lenarduzzi. 2023. Open tracing tools: Overview and critical comparison. *JSS* 204 (2023), 111793.

[15] Wuxia Jin, Ting Liu, Yuanfang Cai, Rick Kazman, Ran Mo, and Qinghua Zheng. 2021. Service Candidate Identification from Monolithic Systems Based on Execution Traces. *IEEE Transactions on Software Engineering* 47, 5 (May 2021), 987–1007. https://doi.org/10.1109/TSE.2019.2910531

[16] Reiner Jung, Sven Gundlach, and Wilhelm Hasselbring. 2021. Instrumenting C and Fortran Software with Kieker. In *CEUR workshop proceedings*. CEUR. http://ceur-ws.org/Vol-3043/

[17] Reiner Jung, Sven Gundlach, Serafim Simonov, and Wilhelm Hasselbring. 2021. Developing Domain-Specific Languages for Ocean Modeling. In *CEUR workshop proceedings*. CEUR, 1–12. http://ceur-ws.org/Vol-2814/

[18] Rudolf E. Kálmán. 1960. On the general theory of control systems. *IFAC Proceedings Volumes* 1, 1 (1960), 491–502. https://doi.org/10.1016/S1474-6670(17)70094-8 1st International IFAC Congress on Automatic and Remote Control.

[19] Bo Liu, Jingliu Xiong, Qiurong Ren, Shmuel Tyszberowicz, and Zheng Yang. 2022. Log2MS: a framework for automated refactoring monolith into microservices using execution logs. In *ICWS 2022*. https://doi.org/10.1109/icws55610.2022.00065

[20] Charity Majors, Liz Fong-Jones, and George Miranda. 2022. *Observability engineering: achieving production excellence*. O'Reilly.

[21] Ana Martinez Saucedo, Guillermo Rodriguez, Fabio Gomes Rocha, and Rodrigo Pereira dos Santos. 2025. Migration of monolithic systems to microservices: A systematic mapping study. *Information and Software Technology* 177 (Jan. 2025), 107590. https://doi.org/10.1016/j.infsof.2024.107590

[22] Nina S. Marwede, Matthias Rohr, André van Hoorn, and Wilhelm Hasselbring. 2009. Automatic Failure Diagnosis in Distributed Large-Scale Software Systems based on Timing Behavior Anomaly Correlation. In *Proc. CSMR'09*. 47–57. https://doi.org/10.1109/CSMR.2009.15

[23] Richard Müller and Matteo Fischer. 2019. Graph-based analysis and visualization of software traces. *Softwaretechnik-Trends* 39, 4 (2019), 26–28.

[24] Yu Qu, Xiaohong Guan, Qinghua Zheng, Ting Liu, Lidan Wang, Yuqiao Hou, and Zijiang Yang. 2015. Exploring community structure of software call graph and its applications in class cohesion measurement. *Journal of Systems and Software* 108 (2015), 193–210. https://doi.org/10.1016/j.jss.2015.06.015

[25] David Georg Reichelt, Lubomír Bulej, Reiner Jung, and André van Hoorn. 2024. Overhead Comparison of Instrumentation Frameworks. In *Companion of the 15th ACM/SPEC ICPE*. https://doi.org/10.1145/3629527.365226

[26] David Georg Reichelt, Malte Hansen, Shinhyung Yang, and Wilhelm Hasselbring. 2024. Interoperability From Kieker to OpenTelemetry: Demonstrated as Export to ExplorViz. https://arxiv.org/abs/2411.07982

[27] David Georg Reichelt and Stefan Kühne. 2016. Empirical Analysis of Performance Problems on Code Level. In *ICPE 2016*. https://doi.org/10.1145/2851553.2892038

[28] David Georg Reichelt, Stefan Kühne, and Wilhelm Hasselbring. 2019. PeASS: A Tool for Identifying Performance Changes at Code Level. In *Proc. ASE 2019*. ACM, 1146–1149. https://doi.org/10.1109/ASE.2019.00123

[29] David Georg Reichelt, Stefan Kühne, and Wilhelm Hasselbring. 2021. Overhead Comparison of OpenTelemetry, inspectIT and Kieker. In *SSP 2021*. http://ceur-ws.org/Vol-3043/

[30] David Georg Reichelt, Stefan Kühne, and Wilhelm Hasselbring. 2023. Towards solving the challenge of minimal overhead monitoring. In *ICPE '23 Companion*. 381–388. https://doi.org/10.1145/3578245.35848

[31] Matthias Rohr, André van Hoorn, Simon Giesecke, Jasminka Matevska, Wilhelm Hasselbring, and Sergej Alekseev. 2008. Trace-context sensitive performance profiling for enterprise software applications. In *SIPEW2008*. Springer, 283–302. https://doi.org/10.1007/978-3-540-69814-2_18

[32] Henning Schnoor and Wilhelm Hasselbring. 2020. Comparing Static and Dynamic Weighted Software Coupling Metrics. *Computers* 9, 2 (March 2020), 1–21. https://doi.org/10.3390/computers9020024

[33] Serafim Simonov, Thomas Duellmann, Reiner Jung, and Sven Gundlach. 2023. Instrumenting Python with Kieker. *Softwaretechnik-Trends* 43, 1 (2023), 26–28. https://dl.gi.de/handle/20.500.12116/43639

[34] Hannes Strubel and Christian Wulf. 2016. Refactoring Kieker's monitoring component to further reduce the runtime overhead. In *SSP 2016*. https://dl.gi.de/handle/20.500.12116/40624

[35] André van Hoorn et al. 2011. DynaMod Project: Dynamic Analysis for Model-Driven Software Modernization. In *MDSM 2011*. http://ceur-ws.org/Vol-708/.

[36] André van Hoorn, Matthias Rohr, Imran Asad Gul, and Wilhelm Hasselbring. 2009. An Adaptation Framework Enabling Resource-efficient Operation of Software Systems. In *Proc. WUP 2009*. 37–40. https://doi.org/10.1145/1527033.1527047

[37] Andre van Hoorn, Christian Vögele, Eike Schulz, Wilhelm Hasselbring, and Helmut Krcmar. 2014. Automatic Extraction of Probabilistic Workload Specifications for Load Testing Session-Based Application Systems. In *Proc. ValueTools 2014*. 139–146. https://doi.org/10.4108/icst.valuetools.2014.258171

[38] André van Hoorn, Jan Waller, and Wilhelm Hasselbring. 2012. Kieker: a Framework for Application Performance Monitoring and Dynamic Software Analysis. In *ICPE 2012*. 247–248. https://doi.org/10.1145/2188286.2188326

[39] Christian Vögele, André van Hoorn, Eike Schulz, Wilhelm Hasselbring, and Helmut Krcmar. 2018. WESSBAS: extraction of probabilistic workload specifications for load testing and performance prediction—a model-driven approach for session-based application systems. *Software & Systems Modeling* 17, 2 (May 2018), 443–477. https://doi.org/10.1007/s10270-016-0566-5

[40] Joakim Von Kistowski, Simon Eismann, Norbert Schmitt, Andre Bauer, Johannes Grohmann, and Samuel Kounev. 2018. TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research. In *MASCOTS 2018*. 223–236. https://doi.org/10.1109/MASCOTS.2018.00030

[41] Robert von Massow, André van Hoorn, and Wilhelm Hasselbring. 2011. Performance Simulation of Runtime Reconfigurable Component-Based Software Architectures. In *Proc. ECSA 2011 (LNCS, Vol. 6903)*. Springer, 43–58. https://doi.org/10.1007/978-3-642-23798-0_5

[42] Jan Waller, Nils C. Ehmke, and Wilhelm Hasselbring. 2015. Including Performance Benchmarks into Continuous Integration to Enable DevOps. *SIGSOFT Softw. Eng. Notes* 40, 2 (March 2015). https://doi.org/10.1145/2735399.2735416

[43] Jan Waller and Wilhelm Hasselbring. 2012. A Comparison of the Influence of Different Multi-Core Processors on the Runtime Overhead for Application-Level Monitoring. In *MSEPT 2012*. https://doi.org/10.1007/978-3-642-31202-1_5

[44] Christian Wulf, Wilhelm Hasselbring, and Johannes Ohlemacher. 2017. Parallel and Generic Pipe-and-Filter Architectures with TeeTime. In *ICSAW 2017*. 290–293. https://doi.org/10.1109/ICSAW.2017.20

[45] Shinhyung Yang, Jiun Jeong, Bernhard Scholz, and Bernd Burgstaller. 2022. Cloudprofiler: TSC-based inter-node profiling and high-throughput data ingestion for cloud streaming workloads. https://doi.org/10.48550/arXiv.2205.09325

[46] Shinhyung Yang, Yonguk Jeong, ChangWan Hong, Hyunje Jun, and Bernd Burgstaller. 2018. Scalability and State: A Critical Assessment of Throughput Obtainable on Big Data Streaming Frameworks for Applications With and Without State Information. In *Euro-Par 2017*. Springer, 141–152. https://doi.org/10.1007/978-3-319-75178-8_12

[47] Shinhyung Yang, David Georg Reichelt, and Wilhelm Hasselbring. 2024. Evaluating the Overhead of the Performance Profiler Cloudprofiler With MooBench. *Softwaretechnik-Trends* 44, 4 (2024). https://dl.gi.de/handle/20.500.12116/45534

[48] Christian Zirkelbach, Wilhelm Hasselbring, and Leslie Carr. 2015. Combining Kieker with Gephi for Performance Analysis and Interactive Trace Visualization. *Softwaretechnik-Trends* 35, 3 (2015). https://dl.gi.de/handle/20.500.12116/40767