Software Impacts 5 (2020) 100019

Contents lists available at ScienceDirect

Software Impacts

journal homepage: www.journals.elsevier.com/software-impacts

Original software publication

Kieker: A monitoring framework for software engineering research

Wilhelm Hasselbring [a,*], André van Hoorn [b]

[a] Software Engineering Group, Kiel University, D-24098 Kiel, Germany
[b] Reliable Software Systems Group, University of Stuttgart, D-70569 Stuttgart, Germany

ARTICLE INFO

Keywords:
Software engineering research
Monitoring
Dynamic analysis
Software performance
Reverse engineering

ABSTRACT

Application-level monitoring and dynamic analysis of software systems are a basis for various tasks in software engineering research, such as performance evaluation and reverse engineering. The Kieker framework provides monitoring, analysis, and visualization support for these purposes. It commenced in 2006, and grew toward a high-quality open-source software that has been employed in a variety of software engineering research projects over the last decade. Several research groups constitute the open-source community to advance the Kieker framework. In this paper, we review Kieker's history, development, and impact both in research and technology transfer with industry.

Code metadata

Current code version	1.14
Permanent link to code/repository used for this code version	https://github.com/SoftwareImpacts/SIMPAC-2020-18
Legal Code License	Apache License, Version 2.0
Code versioning system used	Git
Software code languages, tools, and services used	Java
Compilation requirements, operating environments & dependencies	Java 7 or higher
If available Link to developer documentation/manual	http://kieker-monitoring.net/documentation/
Support email for questions	http://kieker-monitoring.net/support/

Software metadata

Current software version	1.14
Permanent link to executables of this version	http://kieker-monitoring.net/download/
Legal Software License	Apache License, Version 2.0
Computing platforms/Operating Systems	Various, including Linux, OS X, Microsoft Windows, Unix-like
Installation requirements & dependencies	Java 7 or higher, and possibly other technologies depending on the use case
Link to user manual	http://kieker-monitoring.net/documentation/
Support email for questions	http://kieker-monitoring.net/support/

1. Application-level monitoring

The Kieker application-level monitoring framework is structured into a monitoring and an analysis part. On the monitoring side, monitoring probes collect measurements of instrumented software systems represented as monitoring records, which a monitoring writer passes to a configured monitoring log or stream. On the analysis side, monitoring readers import monitoring records of interest from the monitoring log/stream and pass them to a configurable pipe-and-filter architecture of analysis plugins. Fig. 1 illustrates a typical dynamic analysis workflow with Kieker. Focusing on application-level monitoring, Kieker includes monitoring probes for collecting timing and trace information from distributed executions of software systems. Additionally, probes for sampling hardware measures, e.g., CPU utilization and memory usage, are included.

* Corresponding author.
E-mail addresses: hasselbring@email.uni-kiel.de (W. Hasselbring), van.hoorn@informatik.uni-stuttgart.de (A. van Hoorn).

https://doi.org/10.1016/j.simpa.2020.100019
Received 29 May 2020; Received in revised form 3 June 2020; Accepted 4 June 2020

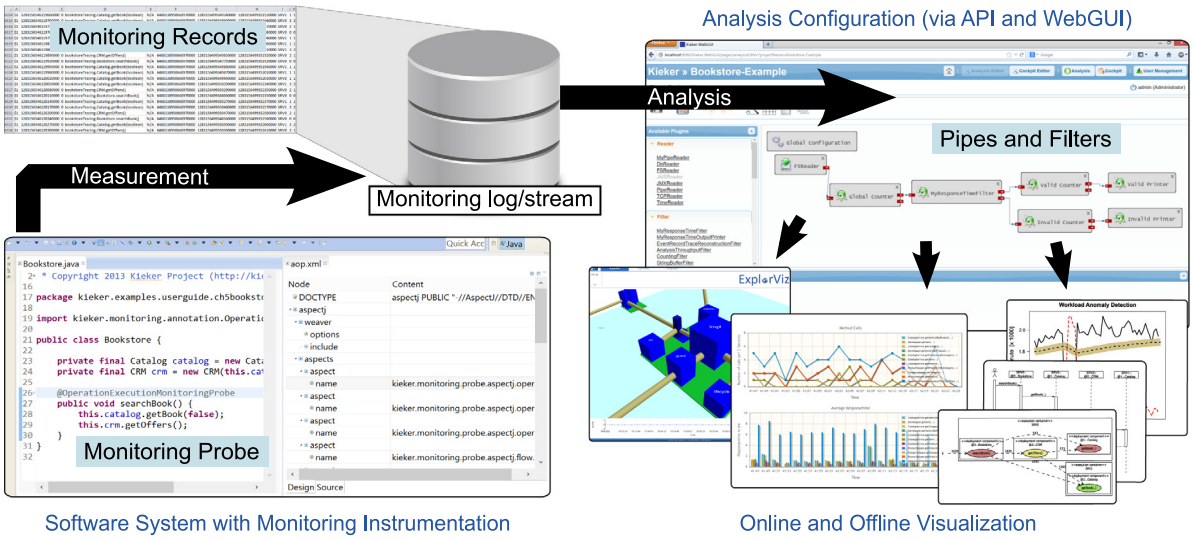

Fig. 1. Illustration of a typical dynamic analysis workflow with Kieker [1].

The core research topics in Kieker are *Application Performance Monitoring* and *Architecture Discovery*, as well as the combination of both topics:

Application Performance Monitoring is concerned with continuously observing a software system's performance-specific runtime behavior, including analyses like assessing service-level compliance or detecting and diagnosing performance problems. Kieker provides Application Performance Monitoring support for both collecting and analyzing monitoring data

Architecture Discovery is concerned with extracting architectural information from an existing software system, including both structural and behavioral aspects like identifying architectural entities (e.g., components and classes) and their interactions (e.g., local or remote procedure calls). Kieker provides Architecture Discovery support by monitoring and analyzing information about distributed execution traces and about consumed software and hardware resources.

2. Development history

Kieker development started in 2006 as a tool for monitoring response times of Java software operations [2]. In 2009, we considerably restructured Kieker toward a generic and extensible framework architecture [3]. In 2011, we significantly improved the quality management with continuous integration, issue tracking, and static code analysis. This process was mainly driven by the successful application process for acceptance in the SPEC RG's repository of peer-reviewed tools for quantitative system evaluation and analysis.[1] The review process and the final acceptance for the tool repository have been a great success for Kieker for several reasons, e.g., the thorough reviews from an external perspective were very useful as they triggered a lot of activities in the Kieker project and helped to further improve Kieker's product quality (both code and documentation); Kieker's visibility was increased considerably [1]. Recently, the Kieker analysis part was re-implemented with the high-throughput pipe-and-filter framework TeeTime [4–6].

[1] The web site of the SPEC Research Group's repository of peer-reviewed tools for quantitative system evaluation and analysis is available at http://research.spec.org/tools/. Similar to the peer-reviewing process for scientific publications, submitted software tools are thoroughly evaluated by at least three reviewers based on the following criteria: *(i.)* relevance to the system evaluation community, *(ii.)* overall utility, *(iii.)* originality or novelty, *(iv.)* tool maturity/user base, *(v.)* ease-of-use and quality of documentation.

3. Research impact

Research areas where Kieker was successfully employed for software engineering research include analysis of software structure and metrics [7–14], performance analysis [15–19], timing behavior anomaly detection [20–23], online failure prediction [24], fault localization [25], online capacity management [26–28], self-adaptive monitoring [22,29,30], software architecture reconstruction [31,32], regression benchmarking [33,34], test case prioritization [35,36], and extraction of load profiles and usage models [37,38].

A number of Kieker-related research projects have been and are being conducted over the years. Examples are Declare (Declarative Performance Engineering) [39], diagnoseIT (Expert-Guided Automatic Diagnosis of Performance Problems) [40], ContinuITy (Automated Load Testing in Continuous Software Engineering) [41], Orcas (Efficient Resilience Benchmarking of Microservice Architectures) [42], MooBench (Monitoring Overhead Benchmark) [33,43,44], iObserve (Integrated Observation and Modeling Techniques to Support Adaptation and Evolution of Software Systems) [45,46] and PubFlow (Workflows for Research Data Publication) [47].

On top of the Kieker framework, ExplorViz [48–50] and SynchroVis [51] emerged as tools for 3D visualization of Kieker monitoring logs. In these contexts, Kieker supports research on program comprehension [52,53], including 3D printing [54] and virtual reality [55]. A jQAssistant plugin provides graph-based visualizations of Kieker traces [56].

4. Industry impact

During the past years, Kieker was employed in several industrial collaborations and technology transfer projects. This includes the funded technology-transfer projects DynaMod (Dynamic Analysis for Model-Driven Software Modernization) [57], MENGES (Model-Driven Engineering of Rail Control Centers) [58], and diagnoseIT (Expert-guided automatic diagnosis of performance problems in enterprise applications) [40]. Examples for such industrial collaborations with impact on Kieker's development include the following:

- In collaboration with EWE TEL GmbH – one of the largest regional telecommunication providers in the north of Germany – their web-based Customer Self Service system was instrumented with Kieker for improved server capacity management.
- In a collaboration with CEWE COLOR AG &Co. OHG – Europe's largest digital photo service provider – we instrumented their web portal providing services for ordering photo products.

- We collaborated with XING AG, Hamburg—a social network for business contacts. XING's core system served as a case study to evaluate Kieker's approach for automatic performance anomaly detection.
- The Kieker monitoring adaptors for Visual Basic 6 and .NET have been developed in collaboration with Dataport AöR and HSH Nordbank AG as part of the DynaMod project to analyze the case study systems AIDA-SH and Nordic Analytics, respectively.
- Additional case studies were conducted for dynamic analysis of legacy COBOL [59] and Perl [16] systems.
- Since 2012, Kieker is integrated in b+m Informatik's generative software development platform b+m gear [60].
- Since 2013, we are collaborating with Novatec Consulting GmbH on various topics around application performance management (APM) [61], including works on APM tool interoperability [62], as part of the mentioned diagnoseIT and ContinuITy projects.

These industrial collaborations and case studies also serve as evaluation of the Kieker approach, influence the development of Kieker, e.g., by feature requests, feedback, code contributions, and provide access to real-world data.

5. Community building

In November 2012, we welcomed 50 participants from academia and industry to our first so-called Kieker Days for developers and users of Kieker. As follow-up events, we organize the annual "Symposium on Software Performance" since 2013 as joint performance community meetings with the related research groups Descartes [63] and Palladio [64]. The 2019 edition of the symposium took place in Würzburg in November 2019 and the 2020 edition will be held in Leipzig.[2]

6. Lessons learned and success factors

Looking back, a crucial success factor for establishing Kieker was the early deployment in industrial production systems. Such deployment environments put significant demands on the quality (in our case particularly for performance and reliability) on the software and its development process. Only with sufficient quality, software may be reused by other researchers and employed successfully in technology transfer projects. Another boost came from the rigorous review process by the SPEC Research Group, which implied significant extensions to the quality assurance in our continuous integration setting.

Kieker's architecture is designed as a component-based system for extensibility to allow for custom extensions. Such a modular architecture significantly improves the collaboration in open source research software projects [65]. The development of features is mainly driven by requests from research, technology transfer projects, and industry projects, rather than market research. This way, we keep the architecture lean and extensible.

Kieker is licensed under the Apache License, Version 2.0, such that it may be exploited commercially without any restrictions. Such a license is a good legal framework for technology transfer. The "business model" of the contributing research groups is not based on envisioned revenue via licensing, instead we follow an open source business model based on *impact* of the software. More frequent use of the software means more impact, in this case. Such impact is a great foundation for follow-up projects. Besides research and technology transfer, we provide professional coaching and training for Kieker. We also use Kieker as example software system for software engineering education.

[2] The web site of the annual Symposium on Software Performance is available at http://www.performance-symposium.org/.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Many colleagues contributed to Kieker in different ways and intensities.[3] Note that we do not only acknowledge contributions to source code. The group of contributors can be divided into researchers and students affiliated with the involved universities, as well as externals, i.e., members from other academic or industrial institutions. The contributing researchers are usually involved because they are working on a Kieker-related research project (including PhD theses). Students usually contribute to Kieker as part of their work on Kieker-related study theses or their employment as student assistants.

References

[1] A. van Hoorn, J. Waller, W. Hasselbring, Kieker: A framework for application performance monitoring and dynamic software analysis, in: Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, ICPE '12, ACM, 2012, pp. 247–248.
[2] M. Rohr, A. van Hoorn, J. Matevska, N. Sommer, L. Stöver, S. Giesecke, W. Hasselbring, Kieker: Continuous monitoring and on demand visualization of Java software behavior, in: Proceedings of the IASTED International Conference on Software Engineering 2008, SE '08, ACTA Press, 2008, pp. 80–85.
[3] A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, D. Kieselhorst, Continuous Monitoring of Software Services: Design and Application of the Kieker Framework, Tech. Rep. TR-0921, Department of Computer Science, Kiel University, Germany, 2009.
[4] C. Wulf, C.C. Wiechmann, W. Hasselbring, Increasing the throughput of pipe-and-filter architectures by integrating the task farm parallelization pattern, in: Proceedings of the 19th International Symposium on Component-Based Software Engineering, CBSE 2016, 2016, pp. 13–22.
[5] C. Wulf, W. Hasselbring, J. Ohlemacher, Parallel and generic pipe-and-filter architectures with TeeTime, in: 2017 IEEE International Conference on Software Architecture Workshops, ICSAW, 2017, pp. 290–293.
[6] C. Wulf, Efficient Engineering and Execution of Pipe-and-Filter Architectures (Ph.D. thesis), Kiel University, 2019.
[7] Q. Zheng, Z. Ou, L. Liu, T. Liu, A novel method on software structure evaluation, in: Proceedings of the 2nd IEEE International Conference on Software Engineering and Service, ICSESS '11, IEEE, 2011, pp. 251–254.
[8] Y. Qu, Q. Zheng, T. Liu, J. Li, X. Guan, In-depth measurement and analysis on densification power law of software execution, in: Proceedings of the 5th International Workshop on Emerging Trends in Software Metrics, ACM, 2014, pp. 55–58.
[9] Y. Qu, X. Guan, Q. Zheng, T. Liu, J. Zhou, J. Li, Calling network: A new method for modeling software runtime behaviors, SIGSOFT Softw. Eng. Notes 40 (1) (2015) 1–8.
[10] Y. Qu, X. Guan, Q. Zheng, T. Liu, L. Wang, Y. Hou, Z. Yang, Exploring community structure of software call graph and its applications in class cohesion measurement, J. Syst. Softw. 108 (2015) 193–210.
[11] W. Jin, T. Liu, Y. Qu, J. Chi, D. Cui, Q. Zheng, Dynamic cohesion measurement for distributed system, in: Proceedings of the 1st International Workshop on Specification, Comprehension, Testing, and Debugging of Concurrent Programs, SCTDCP 2016, ACM, 2016, pp. 20–26.
[12] W. Jin, T. Liu, Y. Qu, Q. Zheng, D. Cui, J. Chi, Dynamic structure measurement for distributed software, Softw. Qual. J. 26 (3) (2018) 1119–1145.
[13] H. Schnoor, W. Hasselbring, Comparing static and dynamic weighted software coupling metrics, in: Proceedings of the 25th International Conference on Information and Software Technologies, ICIST 2019, Springer, Cham, 2019, pp. 285–298.
[14] H. Schnoor, W. Hasselbring, Comparing static and dynamic weighted software coupling metrics, Computers 9 (2) (2020) 1–21.
[15] M. Rohr, A. van Hoorn, S. Giesecke, J. Matevska, W. Hasselbring, S. Alekseev, Trace-context sensitive performance profiling for enterprise software applications, in: Performance Evaluation - Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop, SIPEW '08, in: Lecture Notes in Computer Science (LNCS), vol. 5119, Springer-Verlag, Heidelberg, 2008, pp. 283–302.

[3] A list of Kieker contributors is available at http://kieker-monitoring.net/framework/.

[16] C. Zirkelbach, W. Hasselbring, L. Carr, Combining Kieker with Gephi for performance analysis and interactive trace visualization, in: Symposium on Software Performance 2015, Softw.tech. - Trends 35 (3) (2015) 26–28.

[17] J. Cito, P. Leitner, C. Bosshard, M. Knecht, G. Mazlami, H. Gall, PerformanceHat: augmenting source code with runtime performance traces in the IDE, in: Companion Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, ACM, 2018, pp. 41–44.

[18] J. Cito, P. Leitner, M. Rinard, H. Gall, Interactive production performance feedback in the IDE, in: Proceedings of the 41st International Conference on Software Engineering, IEEE Press, 2019, pp. 971–981.

[19] D.G. Reichelt, S. Kühne, W. Hasselbring, PeASS: A tool for identifying performance changes at code level, in: Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, ACM, 2019, pp. 1146–1149.

[20] M. Rohr, S. Giesecke, W. Hasselbring, Timing behavior anomaly detection in enterprise information systems, in: Proceedings of the 9th International Conference on Enterprise Information Systems, ICEIS'07, 2007, pp. 494–497.

[21] N.S. Marwede, M. Rohr, A. van Hoorn, W. Hasselbring, Automatic failure diagnosis in distributed large-scale software systems based on timing behavior anomaly correlation, in: Proceedings of the 13th European Conference on Software Maintenance and Reengineering, CSMR'09, IEEE, 2009, pp. 47–57.

[22] J. Ehlers, A. van Hoorn, J. Waller, W. Hasselbring, Self-adaptive software system monitoring for performance anomaly localization, in: Proceedings of the 8th IEEE/ACM International Conference on Autonomic Computing, ICAC 2011, ACM, 2011, pp. 197–200.

[23] M. Rohr, Workload-Sensitive Timing Behavior Analysis for Fault Localization in Software Systems (dissertation), Department of Computer Science, Kiel University, Kiel, Germany, 2014, Faculty of Engineering, Kiel University.

[24] T. Pitakrat, D. Okanovic, A. van Hoorn, L. Grunske, Hora: Architecture-aware online failure prediction, J. Syst. Softw. 137 (2018) 669–685.

[25] D. Okanović, A. van Hoorn, Z. Konjović, M. Vidaković, SLA-driven adaptive monitoring of distributed applications for performance problem localization, Comput. Sci. Inf. Syst. 10 (10) (2013) 26–51.

[26] A. van Hoorn, M. Rohr, I.A. Gul, W. Hasselbring, An adaptation framework enabling resource-efficient operation of software systems, in: Proc. of the Warm Up Workshop (WUP 2009) for ACM/IEEE ICSE 2010, ACM, 2009, pp. 37–40.

[27] R. von Massow, A. van Hoorn, W. Hasselbring, Performance simulation of runtime reconfigurable component-based software architectures, in: Proceedings of the European Conference on Software Architecture, ECSA 2011, in: Lecture Notes in Computer Science, vol. 6903, Springer-Verlag, 2011, pp. 43–58.

[28] A. van Hoorn, Model-Driven Online Capacity Management for Component-Based Software Systems (dissertation), Department of Computer Science, Kiel University, Kiel, Germany, 2014, Faculty of Engineering, Kiel University.

[29] J. Ehlers, W. Hasselbring, A self-adaptive monitoring framework for component-based software systems, in: Software Architecture (Proceedings ECSA 2011), in: Lecture Notes in Computer Science, vol. 6903, Springer-Verlag, 2011, pp. 278–286.

[30] J. Ehlers, Self-Adaptive Performance Monitoring for Component-Based Software Systems (dissertation), Department of Computer Science, Kiel University, Kiel, Germany, 2012, Faculty of Engineering, Kiel University.

[31] R. Dabrowski, On architecture warehouses and software intelligence, in: Proceedings of the 4th International Mega-Conference on Future Generation Information Technology, FGIT 2012, in: LNCS, vol. 7709, Springer, 2012, pp. 251–262.

[32] V. Markovets, R. Dabrowski, G. Timoszuk, K. Stencel, Know thy source code, in: Proceedings of the 6th Balkan Conference in Informatics, BCI '13, vol. 1036, CEUR-WS.org, 2013, pp. 128–131.

[33] J. Waller, N.C. Ehmke, W. Hasselbring, Including performance benchmarks into continuous integration to enable DevOps, SIGSOFT Softw. Eng. Notes 40 (2) (2015) 1–4.

[34] D.G. Reichelt, S. Kühne, Better early than never: Performance test acceleration by regression test selection, in: Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, ACM, 2018, pp. 127–130.

[35] J. Chi, Y. Qu, Q. Zheng, Z. Yang, W. Jin, D. Cui, T. Liu, Test case prioritization based on method call sequences, in: 2018 IEEE 42nd Annual Computer Software and Applications Conference, COMPSAC, vol. 01, 2018, pp. 251–256.

[36] J. Chi, Y. Qu, Q. Zheng, Z. Yang, W. Jin, D. Cui, T. Liu, Relation-based test case prioritization for regression testing, J. Syst. Softw. 163 (2020).

[37] A. van Hoorn, C. Vögele, E. Schulz, W. Hasselbring, H. Krcmar, Automatic extraction of probabilistic workload specifications for load testing session-based application systems, in: Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools, ValueTools 2014, 2014, pp. 139–146.

[38] C. Vögele, A. van Hoorn, E. Schulz, W. Hasselbring, H. Krcmar, WESSBAS: extraction of probabilistic workload specifications for load testing and performance prediction—a model-driven approach for session-based application systems, Softw. Syst. Model. 17 (2) (2018) 443–477.

[39] J. Walter, A. van Hoorn, H. Koziolek, D. Okanovic, S. Kounev, Asking "what"?, automating the "how"?: The vision of declarative performance engineering, in: Proceedings of the 7th ACM/SPEC International Conference on Performance Engineering, ICPE 2016, ACM, 2016, pp. 91–94.

[40] C. Heger, A. van Hoorn, D. Okanovic, S. Siegl, A. Wert, Expert-guided automatic diagnosis of performance problems in enterprise applications, in: Proceedings of the 12th European Dependable Computing Conference, EDCC 2016, IEEE Computer Society, 2016, pp. 185–188.

[41] H. Schulz, T. Angerstein, A. van Hoorn, Towards automating representative load testing in continuous software engineering, in: Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE 2018, Berlin, Germany, April 09–13, 2018, ACM, 2018, pp. 123–126.

[42] A. van Hoorn, A. Aleti, T.F. Düllmann, T. Pitakrat, ORCAS: Efficient resilience benchmarking of microservice architectures, in: Proceedings of the 2018 IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE 2018, IEEE Computer Society, 2018, pp. 146–147.

[43] J. Waller, W. Hasselbring, A comparison of the influence of different multi-core processors on the runtime overhead for application-level monitoring, in: Proceedings of the International Conference on Multicore Software Engineering, Performance, and Tools, MSEPT 2012, in: Lecture Notes in Computer Science, vol. 7303, Springer, Berlin / Heidelberg, 2012, pp. 42–53.

[44] J. Waller, Performance Benchmarking of Application Monitoring Frameworks (dissertation), Department of Computer Science, Kiel University, Kiel, Germany, 2014, Faculty of Engineering, Kiel University.

[45] W. Hasselbring, R. Heinrich, R. Jung, A. Metzger, K. Pohl, R. Reussner, E. Schmieders, IObserve: Integrated Observation and Modeling Techniques To Support Adaptation and Evolution of Software Systems, Technical Report TR-1309, Kiel University, Kiel, Germany, 2013.

[46] R. Heinrich, E. Schmieders, R. Jung, K. Rostami, A. Metzger, W. Hasselbring, R. Reussner, K. Pohl, Integrating run-time observations and design component models for cloud system analysis, in: Proceedings of the 9th Workshop on Models@Run.Time, in: Workshop Proceedings, vol. 270, CEUR, 2014, pp. 41–46.

[47] P.C. Brauer, W. Hasselbring, PubFlow: A scientific data publication framework for marine science, in: Proceedings of the International Conference on Marine Data and Information Systems, IMDIS 2013, vol. 54, 2013, pp. 29–31.

[48] F. Fittkau, J. Waller, C. Wulf, W. Hasselbring, Live trace visualization for comprehending large software landscapes: The ExplorViz approach, in: Proceedings of the IEEE International Working Conference on Software Visualization, VISSOFT 2013, 2013, pp. 1–4.

[49] F. Fittkau, S. Roth, W. Hasselbring, ExplorViz: Visual runtime behavior analysis of enterprise application landscapes, in: Proceedings of the European Conference on Information Systems, ECIS 2015 Completed Research Papers, AIS Electronic Library, 2015, pp. 1–13.

[50] F. Fittkau, A. Krause, W. Hasselbring, Software landscape and application visualization for system comprehension with ExplorViz, Inf. Softw. Technol. 87 (2017) 259–277.

[51] J. Waller, C. Wulf, F. Fittkau, P. Döhring, W. Hasselbring, SynchroVis: 3D visualization of monitoring traces in the city metaphor for analyzing concurrency, in: 1st IEEE International Working Conference on Software Visualization, VISSOFT 2013, 2013, pp. 1–4.

[52] F. Fittkau, A. Krause, W. Hasselbring, Hierarchical software landscape visualization for system comprehension: A controlled experiment, in: Proceedings of the 3rd IEEE Working Conference on Software Visualization, VISSOFT 2015, IEEE, 2015, pp. 36–45.

[53] F. Fittkau, S. Finke, W. Hasselbring, J. Waller, Comparing trace visualizations for program comprehension through controlled experiments, in: Proceedings of the IEEE International Conference on Program Comprehension, ICPC 2015, IEEE, 2015, pp. 266–276.

[54] F. Fittkau, E. Koppenhagen, W. Hasselbring, Research perspective on supporting software engineering via physical 3D models, in: Proceedings of the 3rd IEEE International Working Conference on Software Visualization, VISSOFT 2015, IEEE, 2015, pp. 125–129.

[55] F. Fittkau, A. Krause, W. Hasselbring, Exploring software cities in virtual reality, in: Proceedings of the 3rd IEEE International Working Conference on Software Visualization, VISSOFT 2015, IEEE, 2015, pp. 130–134.

[56] R. Müller, M. Fischer, Graph-based analysis and visualization of software traces, in: Proceedings of the 10th Symposium on Software Performance, 2019, pp. 26–28.

[57] A. van Hoorn, S. Frey, W. Goerigk, W. Hasselbring, H. Knoche, S. Köster, H. Krause, M. Porembski, T. Stahl, M. Steinkamp, N. Wittmüss, Dynamod project: Dynamic analysis for model-driven software modernization, in: Proceedings of the International Workshop on Model-Driven Software Migration, MDSM 2011, vol. 708, CEUR, 2011, pp. 12–13.

[58] W. Goerigk, R. von Hanxleden, W. Hasselbring, G. Hennings, R. Jung, H. Neustock, H. Schaefer, C. Schneider, E. Schultz, T. Stahl, S. Weik, S. Zeug, Entwurf einer domänenspezifischen Sprache für elektronische Stellwerke, in: Software Engineering 2012, P-198 of LNI, GI, 2012, pp. 119–130..

[59] H. Knoche, A. van Hoorn, W. Goerigk, W. Hasselbring, Automated source-level instrumentation for dynamic dependency analysis of COBOL systems, in: Proceedings of the 14th Workshop Software-Reengineering, WSR '12, 2012, pp. 33–34.

[60] T. Stahl, M. Völter, Model-Driven Software Development – Technology, Engineering, Management, Wiley & Sons, 2006.

[61] C. Heger, A. van Hoorn, M. Mann, D. Okanovic, Application performance management: State of the art and challenges for the future, in: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, ICPE 2017, ACM, 2017, pp. 429–432.

[62] D. Okanovic, A. van Hoorn, C. Heger, A. Wert, S. Siegl, Towards performance tooling interoperability: An open format for representing execution traces, in: Proceedings of the 13th European Workshop on Performance Engineering, EPEW 2016, in: Lecture Notes in Computer Science, vol. 9951, Springer, 2016, pp. 94–108.

[63] N. Huber, F. Brosig, S. Spinner, S. Kounev, M. Bähr, Model-based self-aware performance and resource management using the Descartes modeling language, IEEE Trans. Softw. Eng. 43 (5) (2017) 432–452.

[64] R. Reussner, S. Becker, J. Happe, R. Heinrich, A. Koziolek, H. Koziolek, M. Kramer, K. Krogmann, Modeling and Simulating Software Architectures: The Palladio Approach, MIT Press, Cambridge, Massachusetts, 2016.

[65] C. Zirkelbach, A. Krause, W. Hasselbring, Modularization of research software for collaborative open source development, in: The Ninth International Conference on Advanced Collaborative Networks, Systems and Applications, COLLA 2019, 2019, pp. 1–7.