# Research Software for Petri Net Research

Wilhelm Hasselbring[1]

[1]Software Engineering Group, Kiel University, 24107 Kiel, Germany

**Abstract**

Research software is software that is designed and developed to support research activities. It can be used to collect, process, analyze, and visualize data, as well as to model and simulate complex phenomena and control sophisticated experiments. In different contexts, research software has been categorized to serve different goals. Depending on the context in which Petri net research software is used, it may play different roles. If it is used to model and simulate processes subject to research, then it is modeling and simulation research software. If it is used to model, simulate, or mine real-life workflows, such as business processes in an enterprise, then it is technology research software. This paper will examine several examples of research software in this context, emphasize their modular software architectures, and report on the synergies obtained by combining research software activities.

**Keywords**

Research software, Petri nets, software categorization, modular software architectures

## 1. Introduction

Research software can be used to collect, process, analyze, and visualize data, as well as to model complex phenomena and run sophisticated simulations. Research software is also developed to control and monitor lab experiments and environmental observations. In engineering research, research software constitutes a new paradigm of scientific inquiry next to theory and experiment, and acts as a proof-of-concept to invent and evaluate new technological artifacts, including algorithms, methods, systems, tools, and other computer-based technologies. Research software also provides the infrastructure to manage, publish, and archive research data and software.

Research software is developed by researchers themselves or by software engineers working closely with researchers. Research software is typically developed to meet specific research needs, and often has unique requirements that are different from standard commercial software [1]. Research Software Engineering and the related role of Research Software Engineer emerged as a job profile to address the resulting challenges. Thus, Research Software Engineering is a specialized field that applies software engineering principles to address the unique challenges posed by developing software for research, with the goal of enhancing the efficiency, reproducibility, and impact of research outcomes. For good scientific practice, research software should be FAIR [2] and open [3].

Section 2 introduces categories of research software, which are illustrated with concrete categorizations of research software examples in Section 3. The Petri net research software Renew is introduced and categorized in Section 4. Section 5 discusses modularity for research software. Section 6 concludes the paper with an outlook.

## 2. Research Software Categories

When studying the research software ecosystem, categorizations of research software [4] provide a framework for classifying research objects, supporting software corpus analyses, and improving our understanding of the different types of research software and their properties. Of particular interest here is the role that software plays in the research process. Examples of these roles include simulation and

*PNSE'26, International Workshop on Petri Nets and Software Engineering, 2026*

hasselbring@email.uni-kiel.de (W. Hasselbring)
https://www.cau-se.de/ (W. Hasselbring)
0000-0001-6625-4335 (W. Hasselbring)

data analytics software, research infrastructure software, and technology research software [5]. Such structured approaches help organize and interpret the vast landscape of research software, contributing to advancements in research software engineering methods and practices.

Thus, research software may take various *roles* in the research process. Research software mainly falls into one of the following three top-level role categories (and sometimes combinations) [4]:

1. *Modeling, Simulation, and Data Analytics* of, e.g., physical, chemical, social, linguistic, or biological processes in spatio-temporal contexts.
2. *Technology Research Software* in science and engineering research.
3. *Research Infrastructure Software*, such as research data and software management systems.

The assignment of research software to categories may evolve over time. For instance, software specifically developed for a research question (usually Categories 1 & 2) can later turn into infrastructure software (Category 3). In different contexts, a software may also be in multiple categories at the same time.

Figure 1, left, shows our role-based categorization [4]. Technology readiness levels constitute a secondary sub-role for technology research software [5]. Additional categories describe the developer structures and the form of dissemination of the research software.

## 3. Research Software Examples

To illustrate the categorization presented in the previous section, we present the concrete categorization of research software examples (based on [6]):

- SPRAT provides a spatially-explicit marine ecosystem model based on population balance equations [7]. SPRAT was empirically evaluated with ocean modelers and research software engineers researching marine ecosystem simulations [8]. Table 1 presents the multi-dimensional categorization of SPRAT.
- OceanTEA supports research on ocean observation data analytics [9]. To achieve this, OceanTEA provides an online service for data analytics and exploration. OceanTEA was successfully employed for studying polyp activity in cold-water corals using machine learning techniques to analyze high-resolution time series data and photographs obtained from an autonomous lander cluster [10]. Table 2 presents the multi-dimensional categorization of OceanTEA.
- The Kieker observability and monitoring framework has been employed in various software engineering research projects [11, 12, 13]. The Kieker development started in 2006 as a tool for monitoring response times of Java software operations [14]. Research areas where Kieker was successfully employed for software engineering research include performance analysis and software architecture reconstruction. As reported in [12, 15], Kieker was also employed in several industrial collaborations and technology transfer projects. Table 3 presents the multi-dimensional categorization of Kieker.
- The MooBench micro-benchmark can be used to quantify the performance overhead caused by observability framework components. MooBench has been originally developed to examine the performance overhead of Kieker in Java [16, 17] and was extended as a general overhead measurement microbenchmark for various observability tools [18]. Table 4 presents the multi-dimensional categorization of MooBench.
- ExplorViz supports research on software visualization, software comprehension tasks and software collaboration [19, 20, 21, 22]. To achieve this, ExplorViz uses dynamic analysis techniques to provide live trace visualization of the communication in large software landscapes. It targets software system and program comprehension in those landscapes while still providing details on the communication within an application. The ExplorViz development started in 2012 [19]. Table 5 presents the multi-dimensional categorization of ExplorViz.

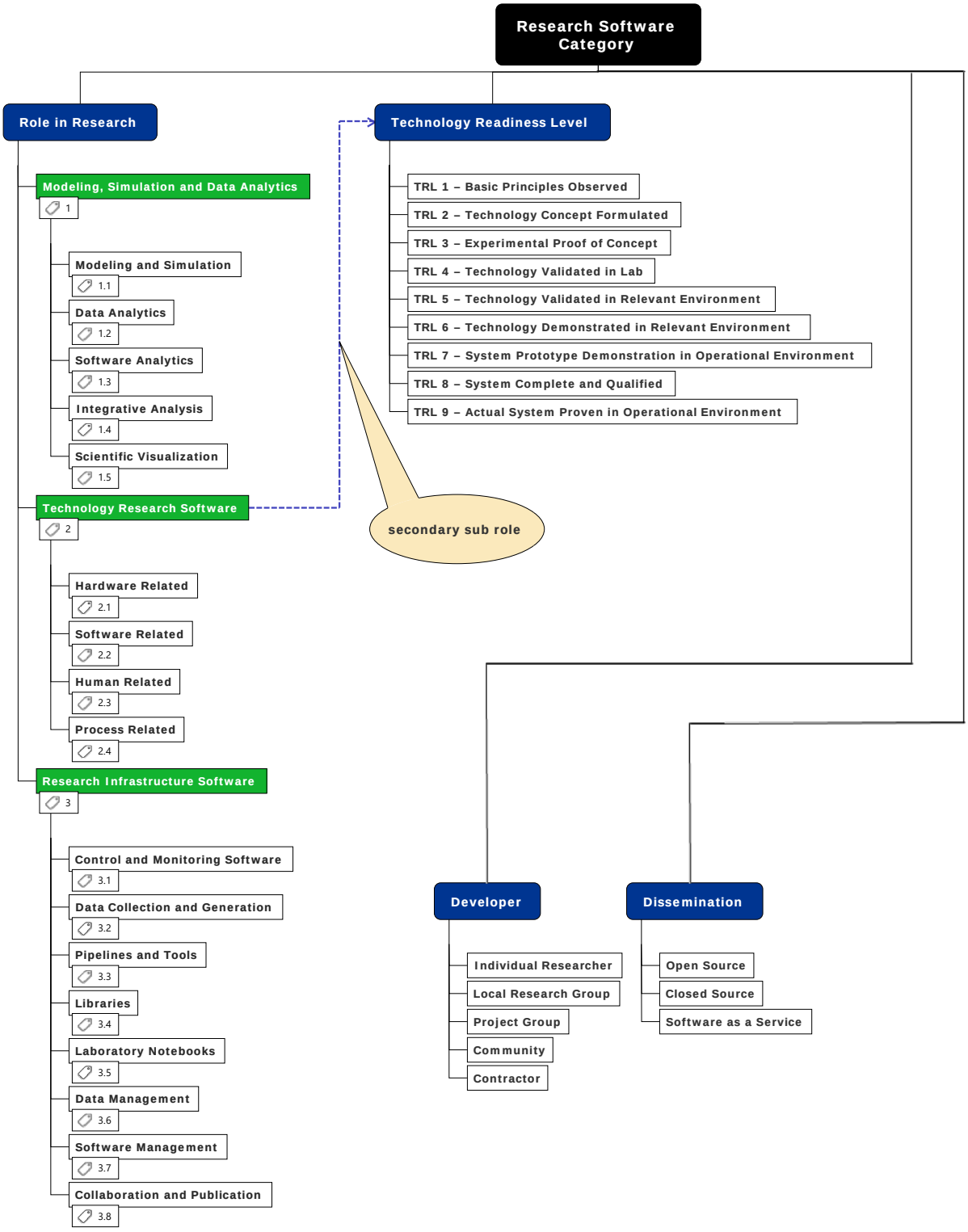


**Figure 1:** Our multi-dimensional categorization of research software, along the dimensions of roles, readiness, developers, and dissemination [4].

| Role | Readiness | Developer | Dissemination |
|---|---|---|---|
| 1.1 Modeling and Simulation<br>1.5 Scientific Visualization | TRL 5 [8] | Individual Researcher | Open Source |

**Table 1**
Multi-dimensional categorization of the **SPRAT** marine ecosystem model simulation approach [7].

| Role | Readiness | Developer | Dissemination |
|---|---|---|---|
| 1.2 Data Analytics<br>1.5 Scientific Visualization | TRL 4 [10] | Local Research Group | Open Source Software as a Service |

**Table 2**
Multi-dimensional categorization of the **OceanTEA** ocean observation data analytics tool [9].

| Role | Readiness | Developer | Dissemination |
|---|---|---|---|
| 1.3 Software Analytics<br>2.2 Software Related | TRL 4 [30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43]<br>TRL 5 [44]<br>TRL 6 [45] | Community | Open Source |

**Table 3**
Multi-dimensional categorization of the **Kieker** observability and monitoring framework [11, 12, 13, 14].

| Role | Readiness | Developer | Dissemination |
|---|---|---|---|
| 2.2 Software Related<br>3.3 Pipelines and Tools | TRL 3 [17]<br>TRL 4 [46, 47, 48, 49, 50]<br>TRL 5 [18] | Community | Open Source |

**Table 4**
Multi-dimensional categorization of the **MooBench** observability overhead benchmark [16, 17, 18].

- Theodolite is a framework for benchmarking the horizontal and vertical scalability of cloud-native applications [23, 24]. Research areas where Theodolite was successfully employed for software engineering research include benchmarking stream processing engines deployed in the cloud [25, 26]. Table 6 presents the multi-dimensional categorization of Theodolite.
- ARCHES is a framework for developing digital twins based on the Robot Operating System (ROS) [27, 28]. Research areas, where ARCHES was successfully employed, are several digital twins of ocean observation systems, including a demo mission [29]. Table 7 presents the multi-dimensional categorization of the ARCHES digital twin framework.

In the next section, we present a similar categorization of research software for Petri net research.

## 4. Petri Net Research Software

Petri nets are a mathematical and graphical modeling tool used to describe and analyze systems where multiple activities happen concurrently and interact with each other. Petri net are, for instance, used for modeling workflows — business processes, approval chains, manufacturing steps. Petri nets may be used to formally verify properties of a system, such as deadlock freedom and liveness. Deadlock freedom means the system can always continue executing; it never reaches a state where no transitions are enabled. Liveness means that no transition can become permanently disabled; every transition

| Role | Readiness | Developer | Dissemination |
|---|---|---|---|
| 1.3 Software Analytics<br>1.5 Scientific Visualization<br>2.2 Software Related | TRL 4 [51, 52, 53, 54, 55, 56, 57, 58, 59, 60]<br>TRL 5 [61] | Local Research Group | Open Source Software as a Service [62] |

**Table 5**
Multi-dimensional categorization of the **ExplorViz** software visualization tool [19, 20, 21, 22].

| Role | Readiness | Developer | Dissemination |
|---|---|---|---|
| 2.2 Software Related<br>3.3 Pipelines and Tools | TRL 4 [63]<br>TRL 5 [25, 26]<br>TRL 6 [64] | Project Group | Open Source |

**Table 6**
Multi-dimensional categorization of the **Theodolite** framework for benchmarking the scalability of cloud-native applications [23, 24].

| Role | Readiness | Developer | Dissemination |
|---|---|---|---|
| 2.1 Hardware Related<br>3.1 Control and Monitoring Software | TRL 4 [65]<br>TRL 5 [66]<br>TRL 7 [29] | Project Group | Open Source |

**Table 7**
Multi-dimensional categorization of the **ARCHES** digital twin framework [27].

| Role | Readiness | Developer | Dissemination |
|---|---|---|---|
| 1.1 Modeling and Simulation<br>2.4 Process Related | TRL 4 [69, 71, 72, 73]<br>TRL 5 [70, 74] | Local Research Group | Open Source |

**Table 8**
Multi-dimensional categorization of the **Renew** modeling and simulation tool for Petri net formalisms [67, 68].

remains potentially executable in the future, which is generally a stronger property than deadlock freedom.

As an example Petri net research software, we take a look at Renew, which is a Java-based tool for modeling and simulating various Petri net formalisms [67, 68]. As research software, Renew is used in many contexts: in lectures and exercises to teach the concepts of Petri nets, in practical courses for hands-on software development, for long-running simulations for analysis aspects and as a runtime environment for Petri net-based applications.

Depending on the context, in which Renew is used, it may take different roles:

- In case, Renew is used for modeling and simulating some processes that are subject to research, it falls into Category 1.1 of modeling and simulation research software. For instance, if you are conducting research on multi-agent systems, you may use Renew for modeling and simulating these multi-agent systems [69, 70].
- In case, Renew is used for modeling, simulating, or mining [71] real-life workflows, such as business processes in an enterprise, it is technology research software. In such a context, it falls into Category 2.4 of process-related technology research software.

Table 8 presents the multi-dimensional categorization of the Renew modeling and simulation tool for Petri net formalisms.

Renew is used as a runtime environment for Petri net-based applications. For this purpose, it would benefit from monitoring the simulation component. Figure 2 shows a dynamic call tree for Renew, that was generated with the Kieker observability framework. Kieker was introduced in the previous

section as an example software for software analytics (Category 1.3). This call tree serves as a basis for performance evaluation of Renew [75].

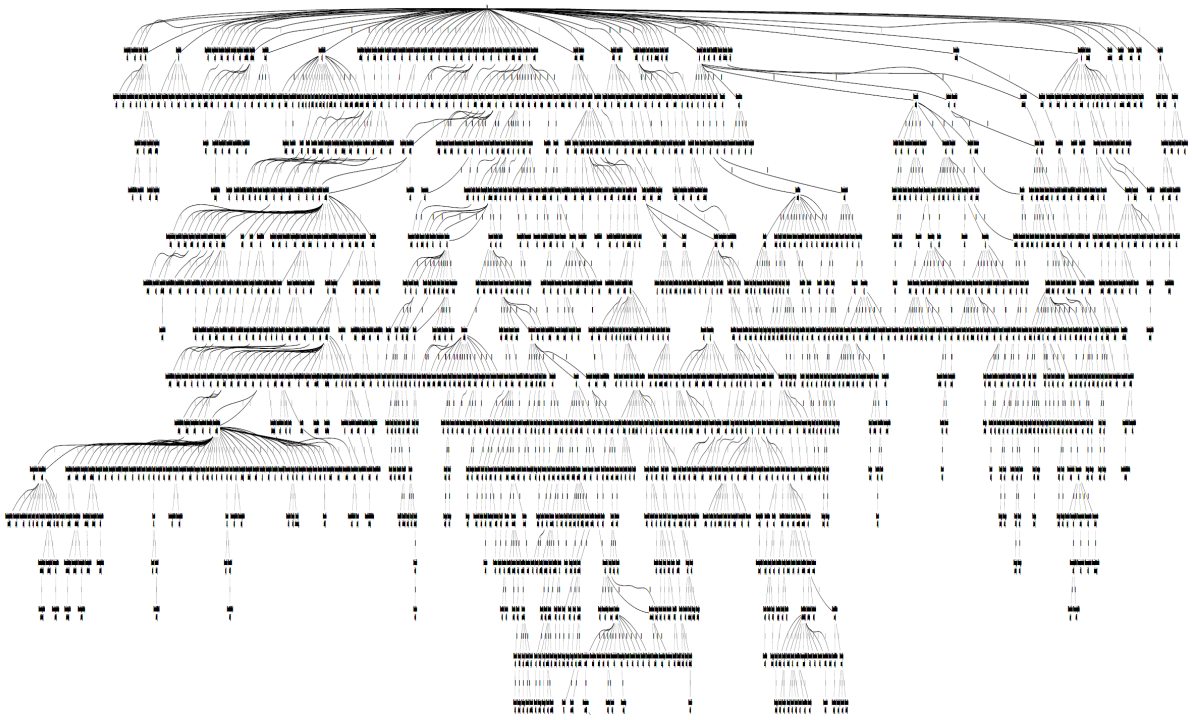

**Figure 2:** Dynamic Call Tree for Renew, generated with Kieker (see also [75]).

Although Renew provides breakpoints for interactive debugging, these mechanisms interrupt execution and are therefore not suitable for observing long-running simulations. Hansson et al. [76] plan to introduce WatchPoints into Renew using the Kieker observability framework to provide a non-intrusive observation mechanism for Petri net simulations. WatchPoints can be attached to net elements and are triggered during simulation without stopping execution. This enables modelers to observe firing behavior, execution frequencies and interaction patterns while preserving the dynamics of the modeled system. As such, WatchPoints support the exploratory modeling, debugging and analysis of Petri nets beyond traditional stepwise execution. In Renew, WatchPoints are attached to places or transitions and are triggered by execution events. They do not interrupt the simulation, but are observed and logged. WatchPoints enable the observability of agents and the monitoring of protocol execution and performance profiling. Figure 3 shows a Petri net in Renew with WatchPoints at transitions T1 to T3. The colors constitute a heatmap ranging from blue for low temperatures to red for high temperatures. The temperature corresponds to the number of times the net element occurs in the WatchPoint log. Petri Nets may also be used for prototyping concurrent application [77], whereby WatchPoints may be a useful feature for explorative prototyping [78].

## 5. Modular Research Software

In the research software engineering community, it is generally accepted that better software leads to better research [79]. In software engineering, it is known that clear architectural foundations have a strong impact to better software [80, 81]. Building on these principles, it is recommended to design modular and maintainable software architectures in research contexts [82], recognizing their critical role in enhancing the quality and impact of research outcomes [83].

The Java Platform Module System (JPMS), introduced with Java 9, provides a framework for structuring Java applications into modules with explicit declarations of dependencies, exported packages, and

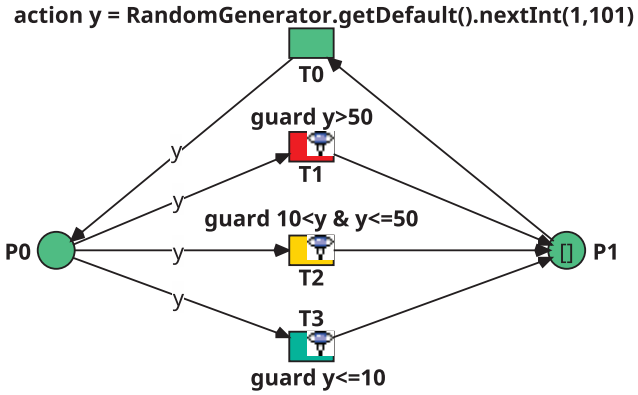


**Figure 3:** Introducing WatchPoints in Renew using the Kieker observability framework to provide a non-intrusive observation mechanism for Petri net simulations, with heatmap coloring based on the number of observed transitions [76].

service interactions [84]. This enhances encapsulation, maintainability, and runtime integrity. Renew uses the JPMS to obtain a modular software architecture, in particular for plugins.

To investigate how Kieker can be used for Java applications with dynamically loaded modules of the Java Platform Module System (JPMS), Hansson [75] conducted experiments with Renew. Renew allows the execution of Java code during transition firing. Monitoring Renew's simulator was challenging because Renew loads core components as plugins in dynamic JPMS module layers and can reach high simulation speeds. Based on this experience, we are currently introducing JPMS into Kieker to further improve its modular software architecture.

## 6. Conclusions and Future Research

Research software can play a variety of roles in the research process. This paper illustrates the mutual benefits of combining research software systems:

- The Renew modeling and simulation tool for Petri net formalisms is instrumented with the Kieker observability framework for performance evaluation [75] and to introduce WatchPoints [76].
- The Kieker observability framework is currently incorporating the Java Platform Module System (JPMS) to enhance its modular software architecture further, building on Renew's experience with the JPMS.

Such collaborations may stimulate and boost significant progress for the research groups involved.

This work also helps to improve the development of research software. The aim of Research Software Engineering Research (RSE Research) is to understand and improve the development of software for research purposes [85]. By enhancing the scholarly understanding of the development and maintenance of research software, RSE Research helps to develop methods, techniques and tools to improve research software engineering practices. Therefore, RSE Research must address a number of research questions. One such question is "Which categories of research software require which software architecture structures?" The impact of software architecture on research software could be investigated to answer this question [83]. Studying Research Software Engineering project lifecycle structures [86] is another domain within RSE Research to achieve a managed software evolution [87].

## Acknowledgements

This research is funded by the Deutsche Forschungsgemeinschaft (grant no. 528713834, project SustainKieker), and UK Research and Innovation (UKRI) through the UKRI Metascience Research Grants program (Reference S26368, project RSE Metascience).

## Declaration on Generative AI

The authors have not employed any Generative AI tools.

## References


[1] A. Johanson, W. Hasselbring, Software Engineering for Computational Science: Past, Present, Future, Computing in Science & Engineering 20 (2018) 90–109. doi:10.1109/MCSE.2018.021651343.

[2] W. Hasselbring, L. Carr, S. Hettrick, H. Packer, T. Tiropanis, From FAIR research data toward FAIR and open research software, it - Information Technology 62 (2020) 39–47. doi:10.1515/itit-2019-0040.

[3] W. Hasselbring, L. Carr, S. Hettrick, H. Packer, T. Tiropanis, Open source research software, Computer 53 (2020) 84–88. doi:10.1109/mc.2020.2998235.

[4] W. Hasselbring, S. Druskat, J. Bernoth, P. Betker, M. Felderer, S. Ferenz, B. Hermann, A.-L. Lamprecht, J. Linxweiler, A. Prat, B. Rumpe, K. Schoening-Stierand, S. Yang, Multidimensional research software categorization, Computing in Science & Engineering 27 (2025) 59–68. doi:10.1109/mcse.2025.3555023.

[5] W. Hasselbring, D. S. Katz, R. van Nieuwpoort, Technology research software: An often overlooked category of research software, Computing in Science & Engineering 28 (2026) 94–99. doi:10.1109/mcse.2026.3662117.

[6] W. Hasselbring, S. Druskat, J. Bernoth, P. Betker, M. Felderer, S. Ferenz, B. Hermann, A.-L. Lamprecht, J. Linxweiler, A. Prat, B. Rumpe, K. Schoening-Stierand, S. Yang, Multi-dimensional categorization of research software with examples, Zenodo (2025). doi:10.5281/zenodo.14082553.

[7] A. N. Johanson, A. Oschlies, W. Hasselbring, B. Worm, SPRAT: a spatially-explicit marine ecosystem model based on population balance equations, Ecological Modelling 349 (2017) 11–25. doi:10.1016/j.ecolmodel.2017.01.020.

[8] A. Johanson, W. Hasselbring, Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment, Empirical Software Engineering 22 (2017) 2206–2236. doi:10.1007/s10664-016-9483-z.

[9] A. Johanson, S. Flögel, C. Dullo, W. Hasselbring, OceanTEA: Exploring ocean-derived climate data using microservices, in: Proceedings of the Sixth International Workshop on Climate Informatics (CI 2016), 2016, pp. 25–28. doi:10.5065/D6K072N6, Technical Note NCAR/TN-529+PROC.

[10] A. Johanson, S. Flögel, C. Dullo, P. Linke, W. Hasselbring, Modeling polyp activity of Paragorgia arborea using supervised learning, Ecological Informatics 39 (2017) 109–118. doi:10.1016/j.ecoinf.2017.02.007.

[11] 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 2012), 2012, pp. 247–248. doi:10.1145/2188286.2188326.

[12] W. Hasselbring, A. van Hoorn, Kieker: A monitoring framework for software engineering research, Software Impacts 5 (2020) 100019. doi:10.1016/j.simpa.2020.100019.

[13] S. Yang, D. G. Reichelt, R. Jung, M. Hansson, W. Hasselbring, The Kieker Observability Framework Version 2, in: Companion of the 16th ACM/SPEC International Conference on Performance Engineering (ICPE 2025), ACM, 2025, pp. 11–15. doi:10.1145/3680256.3721972.

[14] M. Rohr, A. van Hoorn, J. Matevska, N. Sommer, L. Stoever, 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), 2008, pp. 80–85. doi:10.5555/1722603.1722619.

[15] W. Hasselbring, A. van Hoorn, Open-Source Software as Catalyzer for Technology Transfer:

Kieker's Development and Lessons Learned, Technical Report TR-1508, Department of Computer Science, Kiel University, 2015. URL: https://nbn-resolving.org/urn:nbn:de:gbv:8:1-zs-00000275-a1.
[16] J. Waller, W. Hasselbring, A benchmark engineering methodology to measure the overhead of application-level monitoring, in: Proc. Symposium on Software Performance: Joint Kieker/Palladio Days 2013, CEUR, 2013. URL: https://ceur-ws.org/Vol-1083/paper7.pdf.
[17] J. Waller, N. C. Ehmke, W. Hasselbring, Including performance benchmarks into continuous integration to enable DevOps, SIGSOFT Softw. Eng. Notes 40 (2015) 1–4. doi:10.1145/2735399.2735416.
[18] D. G. Reichelt, S. Yang, M. Hanson, W. Hasselbring, Benchmarking the overhead of distributed tracing agents, in: Proceedings of the 17th ACM/SPEC International Conference on Performance Engineering, ACM, 2026, pp. 147–161. doi:10.1145/3777884.3797004.
[19] F. Fittkau, J. Waller, C. Wulf, W. Hasselbring, Live trace visualization for comprehending large software landscapes: The ExplorViz approach, in: Proc. IEEE Int. Working Conference on Software Visualization (VISSOFT 2013), 2013, pp. 1–4. doi:10.1109/VISSOFT.2013.6650536.
[20] F. Fittkau, S. Roth, W. Hasselbring, ExplorViz: Visual runtime behavior analysis of enterprise application landscapes, in: Proc. European Conference on Information Systems (ECIS 2015 Completed Research Papers), AIS Electronic Library, 2015, pp. 1–13. doi:10.18151/7217313.
[21] F. Fittkau, A. Krause, W. Hasselbring, Software Landscape and Application Visualization for System Comprehension with ExplorViz, Information and Software Technology 87 (2017) 259–277. doi:10.1016/j.infsof.2016.07.004.
[22] W. Hasselbring, A. Krause, C. Zirkelbach, ExplorViz: Research on software visualization, comprehension and collaboration, Software Impacts 6 (2020). doi:10.1016/j.simpa.2020.100034.
[23] S. Henning, W. Hasselbring, Theodolite: Scalability benchmarking of distributed stream processing engines in microservice architectures, Big Data Research 25 (2021) 1–17. doi:10.1016/j.bdr.2021.100209.
[24] S. Henning, W. Hasselbring, A configurable method for benchmarking scalability of cloud-native applications, Empirical Software Engineering 27 (2022) 1–42. doi:10.1007/s10664-022-10162-1.
[25] S. Henning, A. Vogel, M. Leichtfried, O. Ertl, R. Rabiser, Shufflebench: A benchmark for large-scale data shuffling operations with distributed stream processing frameworks, in: Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering, 2024, pp. 2–13. doi:10.1145/3629526.3645036.
[26] S. Henning, W. Hasselbring, Benchmarking scalability of stream processing frameworks deployed as microservices in the cloud, Journal of Systems and Software 208 (2024) 111879. doi:10.1016/j.jss.2023.111879.
[27] A. Barbie, W. Hasselbring, ARCHES PiCar-X: Software for Digital Twin Research, Journal of Open Source Software 9 (2024). doi:10.21105/joss.07179.
[28] A. Barbie, W. Hasselbring, From digital twins to digital twin prototypes: Concepts, formalization, and applications, IEEE Access 12 (2024) 75337–75365. doi:10.1109/access.2024.3406510.
[29] A. Barbie, N. Pech, W. Hasselbring, S. Flögel, F. Wenzhöfer, M. Walter, E. Shchekinova, M. Busse, M. Türk, M. Hofbauer, S. Sommer, Developing an Underwater Network of Ocean Observation Systems with Digital Twin Prototypes – A Field Report from the Baltic Sea, IEEE Internet Computing 26 (2022) 33–42. doi:10.1109/mic.2021.3065245.
[30] J. Cito, P. Leitner, C. Bosshard, M. Knecht, G. Mazlami, H. Gall, PerformanceHat: augmenting source code with runtime performance traces in the IDE, in: Proc. 40th Int. Conference on Software Engineering, 2018, pp. 41–44. doi:10.1145/3183440.3183481.
[31] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, Q. Zheng, Service candidate identification from monolithic systems based on execution traces, IEEE Transactions on Software Engineering 47 (2021) 987–1007. doi:10.1109/tse.2019.2910531.
[32] Q. Zheng, Z. Ou, L. Liu, T. Liu, A novel method on software structure evaluation, in: Proc. 2nd IEEE Int. Conference on Software Engineering and Service (ICSESS '11), IEEE, 2011, pp. 251–254. doi:10.1109/ICSESS.2011.5982301.
[33] R. Dabrowski, On architecture warehouses and software intelligence, in: Proc. 4th Int. Mega-

Conference on Future Generation Information Technology (FGIT 2012), volume 7709 of *LNCS*, Springer, 2012, pp. 251–262. doi:10.1007/978-3-642-35585-1_35.

[34] Y. Qu, Q. Zheng, T. Liu, J. Li, X. Guan, In-depth measurement and analysis on densification power law of software execution, in: Proc. 5th Int. Workshop on Emerging Trends in Software Metrics, ACM, 2014, pp. 55–58. doi:10.1145/2593868.2593878.

[35] 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 (2015) 1–8. doi:10.1145/2693208.2693223.

[36] 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, Journal of Systems and Software 108 (2015) 193–210. doi:10.1016/j.jss.2015.06.015.

[37] W. Jin, T. Liu, Y. Qu, J. Chi, D. Cui, Q. Zheng, Dynamic cohesion measurement for distributed system, in: Proc. 1st Int. Workshop on Specification, Comprehension, Testing, and Debugging of Concurrent Programs, ACM, 2016, pp. 20–26. doi:10.1145/2975954.2975956.

[38] W. Jin, T. Liu, Y. Qu, Q. Zheng, D. Cui, J. Chi, Dynamic structure measurement for distributed software, Software Quality Journal 26 (2018). doi:10.1007/s11219-017-9369r3.

[39] S. J. J. Leemans, D. Fahland, W. M. P. van der Aalst, Scalable process discovery and conformance checking, Software & Systems Modeling 17 (2018) 599–631. doi:10.1007/s10270-016-0545-x.

[40] H. Schnoor, W. Hasselbring, Comparing Static and Dynamic Weighted Software Coupling Metrics, Computers 9 (2020) 1–21. doi:10.3390/computers9020024.

[41] 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), volume 5119 of *Lecture Notes in Computer Science (LNCS)*, Springer, 2008, pp. 283–302.

[42] B. Andrade, S. Santos, A. R. Silva, A comparison of static and dynamic analysis to identify microservices in monolith systems, in: Software Architecture, Springer, 2023, pp. 354–361. doi:10.1007/978-3-031-42592-9_25.

[43] W. Hasselbring, R. Jung, H. Schnoor, Software architecture evaluation of earth system models, Journal of Software Engineering and Applications 18 (2025) 113–138. doi:10.4236/jsea.2025.183008.

[44] S. Yang, Y. Jeong, C. Hong, H. Jun, B. Burgstaller, Scalability and State: A Critical Assessment of Throughput Obtainable on Big Data Streaming Frameworks for Applications With and Without State Information, Springer, 2018, pp. 141–152. doi:10.1007/978-3-319-75178-8_12.

[45] 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: Proc. Int. Workshop on Model-Driven Software Migration (MDSM) 2011, volume 708, CEUR, 2011, pp. 12–13. URL: https://ceur-ws.org/Vol-708/.

[46] 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), volume 7303 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 42–53. doi:10.1007/978-3-642-31202-1_5.

[47] D. G. Reichelt, S. Kühne, W. Hasselbring, Overhead Comparison of OpenTelemetry, inspectIT and Kieker, in: Proc. 12th Symposium on Software Performance, CEUR, 2021. URL: https://ceur-ws.org/Vol-3043/short3.pdf.

[48] D. G. Reichelt, S. Kühne, W. Hasselbring, Towards solving the challenge of minimal overhead monitoring, in: Companion of the 2023 ACM/SPEC International Conference on Performance Engineering, 2023, pp. 381–388. doi:10.1145/3578245.3584851.

[49] D. G. Reichelt, R. Jung, A. van Hoorn, Overhead measurement noise in different runtime environments, Softwaretechnik-Trends 44 (2024) 12–14. URL: https://dl.gi.de/handle/20.500.12116/45533, (14th Symposium on Software Performance, November 2024, Linz).

[50] D. G. Reichelt, L. Bulej, R. Jung, A. Van Hoorn, Overhead comparison of instrumentation frame-

works, in: Companion of the 15th ACM/SPEC International Conference on Performance Engineering, 2024, pp. 249–256. doi:10.1145/3629527.3652269.

[51] F. Fittkau, A. Krause, W. Hasselbring, Exploring software cities in virtual reality, in: Proc. 3rd IEEE Int. Working Conference on Software Visualization (VISSOFT 2015), IEEE, 2015, pp. 130–134. doi:10.1109/VISSOFT.2015.7332423.

[52] F. Fittkau, S. Finke, W. Hasselbring, J. Waller, Comparing trace visualizations for program comprehension through controlled experiments, in: Proc. IEEE Int. Conference on Program Comprehension (ICPC 2015), IEEE, 2015, pp. 266–276. doi:10.1109/ICPC.2015.37.

[53] F. Fittkau, A. Krause, W. Hasselbring, Hierarchical software landscape visualization for system comprehension: A controlled experiment, in: Proc. 3rd IEEE Working Conference on Software Visualization (VISSOFT 2015), IEEE, 2015, pp. 36–45. doi:10.1109/VISSOFT.2015.7332413.

[54] A. Krause-Glau, M. Hansen, W. Hasselbring, Collaborative program comprehension via software visualization in extended reality, Information and Software Technology 151 (2022) 107007. doi:10.1016/j.infsof.2022.107007.

[55] A. Krause, M. Hansen, W. Hasselbring, Live visualization of dynamic software cities with heat map overlays, in: 2021 Working Conference on Software Visualization (VISSOFT), IEEE, 2021, pp. 125–129. doi:10.1109/vissoft52517.2021.00024.

[56] A. Krause-Glau, W. Hasselbring, Collaborative, code-proximal dynamic software visualization within code editors, in: 2023 IEEE Working Conference on Software Visualization (VISSOFT), 2023, pp. 50–61. doi:10.1109/vissoft60811.2023.00016.

[57] L. Damerau, M. Hansen, W. Hasselbring, Flexible snapshot creation in debugging sessions for code cities, in: 2025 IEEE Working Conference on Software Visualization (VISSOFT), 2025, pp. 147–148. doi:10.1109/vissoft67405.2025.00031.

[58] M. Hansen, J. Bamberg, N. Baumann, W. Hasselbring, Semantic zoom and mini-maps for software cities, in: 2025 IEEE Working Conference on Software Visualization (VISSOFT), 2025, pp. 47–57. doi:10.1109/vissoft67405.2025.00013.

[59] M. Hansen, L. Damerau, D. König, W. Hasselbring, Visualization of the linux kernel with ExplorViz, in: 2025 IEEE Working Conference on Software Visualization (VISSOFT), 2025, pp. 117–120. doi:10.1109/vissoft67405.2025.00023.

[60] M. Hansen, D. Moreno-Lumbreras, W. Hasselbring, HTML Structure Exploration in 3D Software Cities, in: 2025 IEEE Working Conference on Software Visualization (VISSOFT), IEEE, 2025, pp. 89–93. doi:10.1109/vissoft67405.2025.00020.

[61] A. Krause, C. Zirkelbach, W. Hasselbring, S. Lenga, D. Kröger, Microservice Decomposition via Static and Dynamic Analysis of the Monolith, in: Proceedings of the IEEE International Conference on Software Architecture Companion (ICSA-C), 2020, pp. 9–16. doi:10.1109/ICSA-C50368.2020.00011.

[62] A. Krause-Glau, W. Hasselbring, Scalable collaborative software visualization as a service: Short industry and experience paper, in: 10th IEEE International Conference on Cloud Engineering (IC2E 2022), Pacific Grove, California, USA, 2022, pp. 182–187. doi:10.1109/ic2e55432.2022.00026.

[63] G. Kp, G. Pierre, R. Rouvoy, Studying the energy consumption of stream processing engines in the cloud, in: 2023 IEEE International Conference on Cloud Engineering (IC2E), 2023, pp. 99–106. doi:10.1109/ic2e59103.2023.00019.

[64] S. Henning, A. Vogel, M. Leichtfried, O. Ertl, R. Rabiser, ShuffleBench: A benchmark for large-scale data shuffling operations with distributed stream processing frameworks, in: Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering, 2024, pp. 2–13. doi:10.1145/3629526.3645036.

[65] A. Barbie, W. Hasselbring, Toward reproducibility of digital twin research: Exemplified with the PiCar-X, arXiv (2024). doi:10.48550/ARXIV.2408.13866.

[66] A. Barbie, W. Hasselbring, M. Hansen, Digital twin prototypes for supporting automated integration testing of smart farming applications, Symmetry 16 (2024) 221. doi:10.3390/sym16020221.

[67] O. Kummer, F. Wienberg, M. Duvigneau, J. Schumacher, M. Köhler, D. Moldt, H. Rölke, R. Valk, An extensible editor and simulation engine for petri nets: Renew, in: Applications and Theory of

Petri Nets 2004, Springer, 2004, pp. 484–493. doi:10.1007/978-3-540-27793-4_29.

[68] D. Moldt, J. Johnsen, R. Streckenbach, L.-O. Clasen, M. Haustermann, A. Heinze, M. Hansson, M. Feldmann, K. Ihlenfeldt, RENEW: modularized architecture and new features, in: Application and Theory of Petri Nets and Concurrency, Springe, 2023, pp. 217–228. doi:10.1007/978-3-031-33620-1_12.

[69] M. Duvigneau, D. Moldt, H. Rölke, Concurrent architecture for a multi-agent platform, in: Agent-Oriented Software Engineering III, Springer, 2003, pp. 59–72. doi:10.1007/3-540-36540-0_5.

[70] L. Capra, M. Köhler-Bußmeier, H. Rölke, J. Sudeikat, Petri nets as run-time models for self-adaptive cyber-physical systems, in: Proc. Petri Nets and Software Engineering (PNSE'24), CEUR, 2024, pp. 164–181. URL: https://ceur-ws.org/Vol-3730/paper10.pdf.

[71] H. Rölke, Towards process mining for nets within nets, in: Proc. Petri Nets and Software Engineering (PNSE'25), CEUR, 2025, pp. 81–96. URL: https://ceur-ws.org/Vol-3998/paper05.pdf.

[72] N. Fernandes, J. Barros, R. Campos Rebelo, Leveraging high-level Petri nets for cyber-physical systems development, in: Proc. Petri Nets and Software Engineering (PNSE'24), CEUR, 2024, pp. 155–163. URL: https://ceur-ws.org/Vol-3730/paper09.pdf.

[73] M. Köhler-Bußmeier, L. Capra, Analysing probabilistic hornets, in: Application and Theory of Petri Nets and Concurrency, Springer, 2025, pp. 287–309. doi:10.1007/978-3-031-94634-9_14.

[74] S. Hustiu, J. Ezpeleta, C. Mahulea, M. Kloetzer, Multi-robot motion planning based on nets-within-nets modeling and simulation, Robotics and Autonomous Systems 197 (2026) 105287. doi:10.1016/j.robot.2025.105287.

[75] M. Hansson, D. Moldt, First Steps for Performance Monitoring of Petri Net Simulator Renew with Kieker, Softwaretechnik-Trends 44 (2025) 77–79. URL: https://dl.gi.de/handle/20.500.12116/47958, (15th Symposium on Software Performance, November 2025, Kiel).

[76] M. Hansson, D. Moldt, K. Wittenburg, C. Bracker, WatchPoints for Renew, in: International Workshop on Petri Nets and Software Engineering (PNSE'26), CEUR, 2026.

[77] W. Hasselbring, Programming languages and systems for prototyping concurrent applications, ACM Computing Surveys 31 (2000) 43–79. doi:10.1145/349194.349199.

[78] C. Floyd, A systematic look at prototyping, in: Approaches to Prototyping, Springer, 1984, pp. 1–18. doi:10.1007/978-3-642-69796-8_1.

[79] C. Goble, Better software, better research, IEEE Internet Computing 18 (2014) 4–8. doi:10.1109/mic.2014.88.

[80] W. Hasselbring, Software-Architektur – Das aktuelle Schlagwort, Informatik-Spektrum 29 (2006) 48–52. doi:10.1007/s00287-005-0049-5.

[81] W. Hasselbring, Software architecture: Past, present, future, in: The Essence of Software Engineering, Springer, Cham, 2018, pp. 169–184. doi:10.1007/978-3-319-73897-0_10.

[82] C. Zirkelbach, A. Krause, W. Hasselbring, The Collaborative Modularization and Reengineering Approach CORAL for Open Source Research Software, International Journal on Advances in Software 13 (2020) 34–49. URL: http://www.iariajournals.org/software/tocv13n12.html.

[83] S. Druskat, N. U. Eisty, R. Chisholm, N. Chue Hong, R. C. Cocking, M. B. Cohen, M. Felderer, L. Grunske, S. A. Harris, W. Hasselbring, T. Krause, J. Linxweiler, C. C. Venters, Better architecture, better software, better research, Computing in Science & Engineering 27 (2025) 45–57. doi:10.1109/mcse.2025.3573887.

[84] N. Parlog, The Java Module System, Manning, 2019.

[85] M. Felderer, M. Goedicke, L. Grunske, W. Hasselbring, A.-L. Lamprecht, B. Rumpe, Investigating research software engineering: Toward RSE Research, Communications of the ACM 68 (2025). doi:10.1145/3685265.

[86] Y. Yehudi, M. Cashman, M. Goedicke, W. Hasselbring, D. S. Katz, S. Müller, C. Goble, C. Jay, Research software lifecycles and stages, Electronic Communications of the EASST 85 (2025). doi:10.14279/ECEASST.V85.2693.

[87] R. Reussner, M. Goedicke, W. Hasselbring, B. Vogel-Heuser, J. Keim, L. Märtin, Managed Software Evolution, Springer, Cham, 2019. doi:10.1007/978-3-030-13499-0.