Article Type: Software Engineering Track

Multi-Dimensional Categorization of Research Software with Examples

Wilhelm Hasselbring, *Software Engineering, Kiel University, Kiel, 24098, Germany*
Stephan Druskat, *German Aerospace Center (DLR), Berlin, 12489, Germany*
Jan Bernoth, *University of Potsdam, Potsdam, 14476, Germany*
Philine Betker, *Department for Epidemiology, Helmholtz Centre for Infection Research, Brunswick, Germany*
Michael Felderer, *German Aerospace Center (DLR) & University of Cologne, Cologne, 51147 , Germany*
Stephan Ferenz, *Department of Computer Science, Carl von Ossietzky Universität Oldenburg, 26129 Oldenburg*
Ben Hermann, *Secure Software Engineering Group, TU Dortmund, 44227 Dortmund*
Anna-Lena Lamprecht, *University of Potsdam, Potsdam, 14476, Germany*
Jan Linxweiler, *TU Braunschweig, Braunschweig, 38106, Germany*
Arnau Prat, *German Aerospace Center (DLR), Braunschweig, 38108, Germany*
Bernhard Rumpe, *Software Engineering, RWTH Aachen University, Germany*
Katrin Schoening-Stierand, *Hub of Computing and Data Science, University of Hamburg, 22761 Hamburg*
Shinhyung Yang, *Software Engineering, Kiel University, Kiel, 24098, Germany*

Abstract—Research software has been categorized in different contexts to serve different goals. We start with a look at what research software is, before we discuss the purpose of research software categories. We propose a multi-dimensional categorization of research software. We present a template for characterizing such categories. As selected dimensions, we present our proposed role-based, readiness-based, developer-based, and dissemination-based categories. Since our work has been inspired by various previous efforts to categorize research software, we discuss them as related works. We characterize all these categories via the previously introduced template, to enable a systematic comparison. We report on the multi-dimensional categorization of selected research software examples.

***Keywords**: Research Software, Software Categorization, Technology Readiness Level, Open Source Software*

Research software is software that is designed and developed to support research activities. 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]. However, research software is gaining appreciation and endorsement for research and as a research result itself [2], [3].

Research Software Engineering (RSE) 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. Research software engineers specialize in developing and maintaining software for research purposes.

In this paper, we propose a multi-dimensional cat-

Digital Object Identifier 10.1109/XXX.0000.0000000

egorization of research software, along the dimensions of roles, readiness, developer, and dissemination. We start with a look at what research software is before we discuss the purpose of research software categories. We present a template for characterizing such categories. Subsequently, our proposed role-based, readiness-based, developer-based, and dissemination-based categories are presented. Our work has been inspired by various previous efforts to categorize research software, which we discuss as related works. We characterize all these categories via the previously introduced template, and conclude with an outlook to future work.

Research Software

For the purposes of this paper, we follow the FAIR for Research Software (FAIR4RS) Working Group in their definition of research software, as software that was created during the research process or for a research purpose [4], [5]. This *prescriptive* definition distinguishes "research software" and "software in research," which includes general purpose software. The software components (e.g., operating systems, programming languages, libraries, etc.) that are used for research but were *not* created during research or with a clear research intent should be considered "software in research" and not "research software." In the present paper, we categorize research software.

A *descriptive* definition of research software could instead include all the software used in research, as for instance done in [6] for analyzing GitHub repositories. While such a descriptive definition may be useful in analyzing research processes, and therefore may be useful for RSE research [7], [8], the prescriptive definition defines a clearer focus for the work presented here, and enables a better disambiguation of properties specific to research software. The Research Data Alliance also adopted the prescriptive distinction between *research software* and *software in research* [5], as we do. Figure 1 shows the resulting segmentation of software.

Purpose of Research Software Categories

We envision the following benefits from using categories for research software, which may serve

- as a basis of institutional guidelines and checklists for research software development;
- to better understand the different types of research software and their specific quality requirements;
- to recommend appropriate software engineering methods for the individual categories;
- to design appropriate teaching / education programs for the individual categories;
- to give stakeholders (especially research software engineers and their management) a better understanding of what kind of software they develop;
- for a better assessment of existing software when deciding to reuse it;
- for research funding agencies, to define appropriate funding schemes;
- to define appropriate metadata labels for FAIR research software [9], [10];
- in RSE Research [7], [8] to provide a framework for classifying research software artifacts.

This list is not exhaustive.

Characterization of Research Software Categories

Categorizations can be described through their scope, purpose, context, properties, consequences for creation and use, and their inter-categorial relations. Table 1 provides a template for systematically describing the characteristics of research software categorizations, which we will use later to characterize some individual categorizations in the subsequent sections.

Role-Based Categorization of Research Software

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 [11], and acts as a proof-of-concept

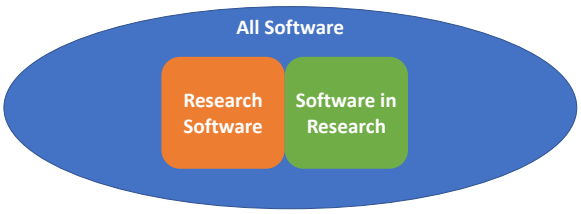

FIGURE 1. Segmentation of all software, research software, and software in research. In the present paper, we further categorize the orange box, i.e., research software.

Criterion	Explanation
Scope	What is the scope of the categorization?
Purpose	What is the purpose of the categorization?
Context	In which contexts are specific categories developed and used?
Properties	What are specific properties of the different categories?
Consequences for Creation	How is and should software of a specific category be developed?
Consequences for Use	How and why is software of a specific category used? What are the differences between the categories in terms of use and reuse, including, e.g., in software publication & citation?
Inter-categorial relations	What are the relations between different categories?

TABLE 1. Template for describing criteria of research software categorizations.

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.

Thus, research software may take various *roles* in the research process [12], [13]. This is similar to software engineering teams, which involve a range of roles that contribute to the development, maintenance, and improvement of software systems. Some common roles in software engineering are software architect, programmer, and tester. Each role may be taken by several persons, and one person may take several roles. These role assignments may also change during a software project.

We propose a similar role-based categorization of research software, with an emphasis on varying quality requirements for the different *roles* that software may take in research. Accordingly, a research software may take several roles, which may also change during the life cycle of the software.

Research software mainly falls into one of the following three top-level role categories (and sometimes combinations):

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) [14]. In different contexts, a software may also be in multiple categories at the same time.

We further refine Category 1 research software for modeling, simulation, and data analytics with several subcategories:

1.1) Modeling and simulation (e.g., numerical modeling, agent-based modeling)
1.2) Data analytics, on observation and simulation data, with statistical analysis and machine learning as methods
1.3) Software analytics (static, dynamic, evolution, repository mining)
1.4) Integrative analysis (data assimilation and decision analysis)
1.5) Scientific visualization

Category 2) for technology research software is used in structural sciences (mathematics and computer science) and in engineering sciences (software, electrical, mechanical, and civil engineering). Technology research software may be related to target contexts:

2.1) Hardware (usually as embedded software)
2.2) Software (e.g., as part of an operating system)
2.3) Human (with a user interface)
2.4) Process (e.g., as part of a business, development or production processes)

Again, one research software may be in multiple categories. In the next section, we will additionally relate this category to technology readiness levels as secondary sub roles.

We further refine Category 3 for research infrastructure software with several subcategories:

3.1) Control and monitoring software for complex experiments and instruments. This includes embedded control software, as well as native and web-based monitoring software.
3.2) Data collection and generation (survey software, sensor-based data collection, synthetic data generation, etc.).
3.3) Pipelines and tools.
3.4) Libraries, for instance for high performance computing.
3.5) Laboratory notebooks.
3.6) Data management.
3.7) Software management.
3.8) Collaboration and publication.

These categories have varying requirements on their software development. For instance, dedicated requirements engineering may be relevant for Category 3),

but not for Category 1). As another example, safety analysis may be relevant for Category 3.1), but not for Categories 1) and 2).

Figure 2, left, shows our resulting role-based categorization.

Table 2 characterizes our multi-dimensional categorization in terms of the template in Table 1. The readiness-based, developer-based, and dissemination-based categorizations are introduced in the following three sections, before we discuss some related categorizations.

Readiness-Based Categorization of Research Software

Technology is the application of conceptual knowledge for achieving practical goals, especially in a reproducible way. The word *technology* can also mean the products resulting from such efforts, including both tangible tools such as utensils or machines, and intangible ones such as software.

Technology readiness levels (TRLs) are a method for estimating the maturity of technologies. TRLs enable consistent and uniform discussions of technical maturity across different types of technology. Figure 2, right, shows the resulting readiness-based categorization with the titles of the European TRL 1 to TRL 9 [15].

These TRLs may be applied to all types of research software, thus, the category dimensions are *orthogonal*: every research software may be classified independently in each dimension.

In addition, for technology research software, these TRL titles can be read as *secondary* sub roles. Examples are:

TRL 3 : The technology research software takes the role as an "Experimental Proof of Concept" within some research project.

TRL 4 : The technology research software takes the role as a "Technology Validated in Lab" within some research project.

Thus, the TRLs constitute *sub roles* of technology research software.

One specific technology research software may take several such sub roles over its lifecyle, with increasing "readiness". It may also take several roles at the same time, within different contexts: In one project, it may serve as experimental proof of concept (TRL 3); in another project, it may already serve as a technology validated in a lab (TRL 4). Eventually, a technology research software may even become an "Actual System Proven in Operational Environment" (TRL 9).

Engineering research (a.k.a. Design Science) is research that invents and evaluates technological artifacts.[1] Thus, the refinement via TRLs should be appropriate.

"Readiness" is top-level in the mindmap, thus it is it own dimension. If we had put "Readiness" directly below "Technology Research Software", it would not be its own dimension, thus we added the cross-link from "Technology Research Software" to illustrate the additional, secondary sub-role relationship.

The difference between the categories "Modeling and Simulation" and "Technology Research Software" (without consideration of the TRL sub roles) may be illustrated, for instance, with control engineering research:

- As a control engineering researcher, you may build a *simulation* of a control system.
- As a control engineering researcher, you may also build an *actual* control system as a new software system. In an automation lab, this researcher may then experiment with this system (not with the simulation of the system). If this system (which is a technology research software) matures, it may reach higher TRLs.

Here, both, the simulation and the actual control system are research software.

Another difference between "Modeling and Simulation" and "Technology Research Software" is that for "Technology Research Software" the TRLs may denote sub roles, as explained above. For "Modeling and Simulation" and "Infrastructure" research software, the TRLs may describe the maturity, but not sub roles.

Developer-Based Categorization of Research Software

For the developer dimension, we see the following stages for research software:

1) Individual Researcher, such as PhD student, Post-Doc, or Research Software Engineer.
2) Local Research Group.
3) Project Group, in which several research groups may collaborate.
4) Community on a specific research topic.
5) Contractor (professional software company developing the software on behalf of researchers).

[1] https://github.com/acmsigsoft/EmpiricalStandards/blob/master/docs/standards/EngineeringResearch.md

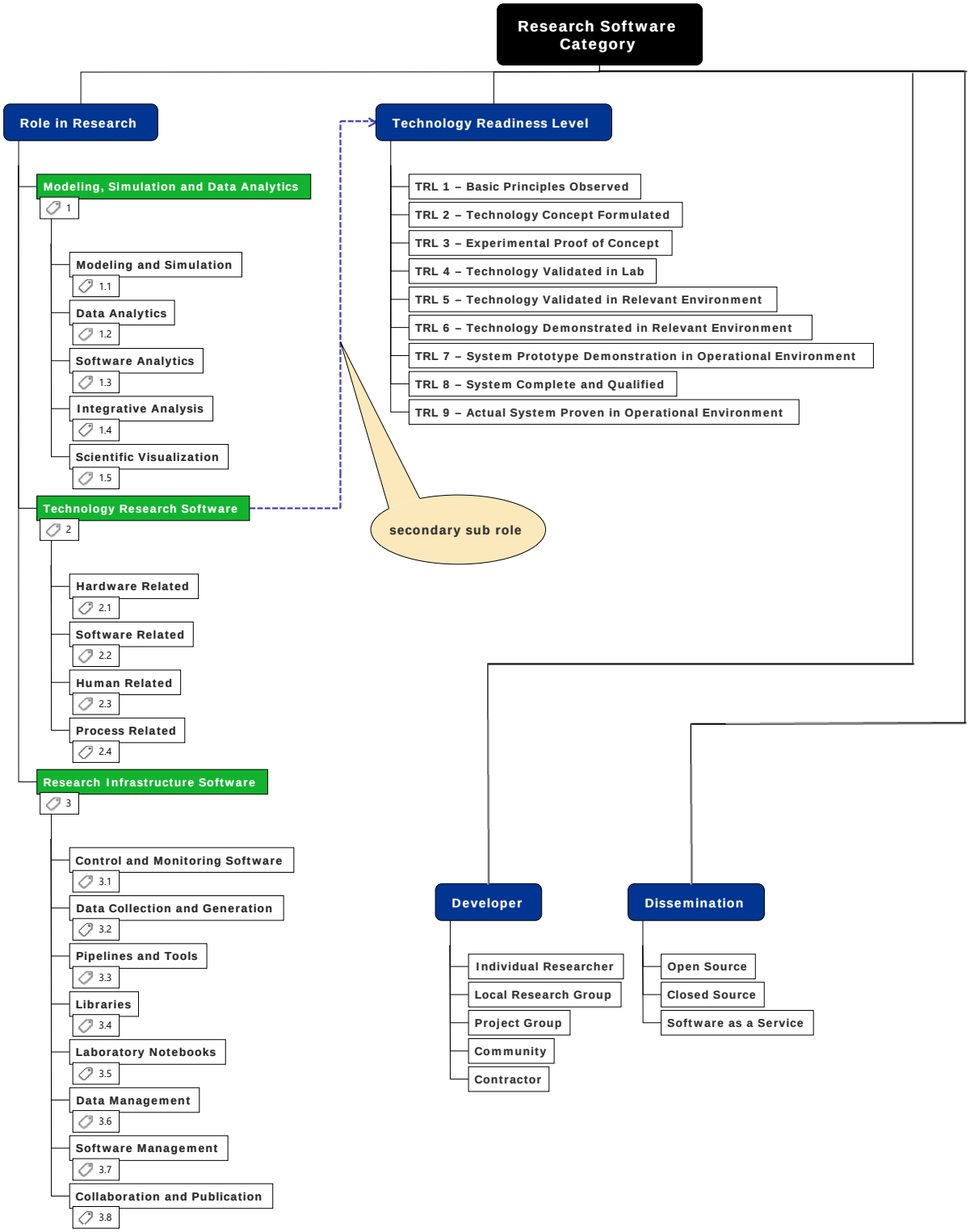

FIGURE 2. Our multi-dimensional categorization of research software, along the dimensions of roles, readiness, developers, and dissemination.

Criterion	Explanation
Scope	This categorization covers the dimensions of roles, readiness, developers, and dissemination.
Purpose	The categorization aims to enable a better understanding of the different types of research software and their specific quality requirements.
Context	The categorization has been produced in the context of a task force of the special interest group on Research Software Engineering, within the German Association of Computer Science (GI e.V.) and the German Society for Research Software (de-RSE e.V.). It is meant to serve different purposes, in particular RSE research [7], [8].
Properties	The categories follow different relevant dimensions, and are defined collaboratively among software engineering researchers and research software engineers.
Consequences for Creation	Depending on its category, software is expected to meet different quality requirements and follow different development processes.
Consequences for Use	Perceive that there are many different types of research software, fulfilling many different roles and functions.
Inter-categorial relations	Individual research software may change its category within one or more dimensions.

TABLE 2. Characteristics of our multi-dimensional categorization for research software.

Dissemination-Based Categorization of Research Software

A community or contractor may develop the software open-source, closed-source, or it may provide research software as an online service.

Figure 2, bottom, shows our developer-based and dissemination-based categorizations.

Related Research Software Categories

Research software has been categorized in different contexts to serve different aims. Some of them are discussed here as related works, as they a) represent a good starting point for a discussion on research software categorization, b) provided significant input to our work, and c) may be used to compare and assess our categorization. We characterize these categories via the previously introduced template in the appendix (supplement).

Role-Based Categorization

Van Nieuwpoort and Katz [12] present a role-based categorization. They categorize research software as an integral component of instruments used in research, as the instrument itself, for analyzing research data, for presenting research results, for assembling or integrating existing components, as infrastructure or an underlying tool, and for facilitating research-oriented collaboration. This categorization inspired our work. Based on discussions with the authors of the present paper, van Nieuwpoort and Katz extended their categorization with our *Technology Research Software* category [13]

Maturity-Based Categorization

In their National Agenda for Research Software [16], the Australian Research Data Commons – an Australian research data infrastructure facility – argue for research software to be recognized as a first-class output of research. They describe a three-level maturity categorization of research software that is related to our readiness dimension:

1) *Research Data Processes* captured as software. The result is analysis code that captures research processes and methodology: the steps taken for tasks like data generation, preparation, analysis, and visualization.
2) *Novel Methods and Models* captured as software. The results are prototype tools that demonstrate a new idea, method, or model for research.
3) *Accepted Methods and Models* captured as software. The result can become research software infrastructure that captures more broadly accepted and used ideas, methods, and models for research.

Each category faces specific challenges with regard to recognition, from making research practice transparent, to creating impact through quality software and safeguarding longer-term maintenance.

Application classes in institutional software engineering guidelines

Institutional guidelines typically define so-called application classes for research software, which require appropriate quality properties, and, thus software engineering methods [17], [18]:

- For software in Application Class 0, the focus is on personal use in conjunction with a small scope.
- For software in Application Class 1, it should be possible, for those not involved in the develop-

ment, to use it to the extent specified and to continue its development.
- For software in Application Class 2, it is intended to ensure long-term development and maintainability. It is the basis for a transition to product status.
- For software in Application Class 3, it is essential to avoid errors and to reduce risks. This applies in particular to critical software.

The application classes relate to our readiness domain and to some extent to our developer-based categorization.

EOSC Research Software Lifecycle

The European Open Science Cloud (EOSC) aims to create a virtual environment for sharing and accessing research data across borders and scientific disciplines. The SubGroup 1 "On the Software Lifecycle" of the EOSC Task Force "Infrastructure for quality research software" provides a categorization for software in the research lifecycle [19]:

1) Individual creating research software for own use (e.g. a PhD student).
2) A research team creating an application or workflow for use within the team.
3) A team / community developing (possibly broadly applicable) open source research software.
4) A team or community creating a research service.

This categorization is covered by our developer-based categorization.

Computational research in the earth system sciences

Döll et al. [20] provide recommendations for sustainable research software for high-quality computational research in the Earth System Sciences, and categorize this research software as follows:

- Simulation of Earth system processes by Earth system models.
- Design, processing and analysis of Earth observation and lab experiment data.
- Integrative analysis of simulation models, large data bases, and stakeholder knowledge.

These categories correspond to our role-based categories 1.1), 1.2), and 1.4), respectively.

Categorizing the Software Stack

Another dimension is the research software stack, from non-scientific infrastructure, scientific infrastructure, discipline-specific software, up to project-specific software [21]. This dimension could be the basis for another branch in our multi-dimensional categorization.

Qualitative Evaluation

As a pre-review study, we conducted a multi-dimensional categorization of selected research software examples, to check whether we can categorize selected research software in multiple dimensions. The selection is mainly based on in-depth knowledge of the respective research software by the authors, such that we are able to confidently categorize these research software examples, in particular the readiness level.

We categorize the following research software, in alphabetical order:

ARCHES: ARCHES is a framework for developing digital twins based on the Robot Operating System (ROS) [22], [23]. Research areas, where ARCHES was successfully employed, are several digital twins of ocean observation systems, including a demo mission [24]. Table 3 presents the multi-dimensional categorization of the ARCHES digital twin framework.

ExplorViz: ExplorViz supports research on software visualization, software comprehension tasks and software collaboration [27], [28], [29], [30]. 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 [27]. Table 4 presents the multi-dimensional categorization of ExplorViz.

Hexatomic: Hexatomic is an extensible, OS-independent platform for deep multi-layer linguistic annotation of corpora [39], [40]. It constitutes a technology research software demonstrator for interoperability within a software ecosystem for multi-layer linguistic corpus workflows, *corpus-tools.org* [41]. Hexatomic is applied in corpus linguistics for manual and semi-automated annotation. Table 5 presents the multi-dimensional categorization of Hexatomic.

Role	Readiness	Developer	Dissemination
2.1 Hardware Related	TRL 4 [25] TRL 5 [26] TRL 7 [24]	Project Group	Open Source

TABLE 3. Multi-dimensional categorization of the **ARCHES** digital twin framework [22].

Role	Readiness	Developer	Dissemination
1.3 Software Analytics 1.5 Scientific Visualization 2.2 Software Related	TRL 4 [31], [32], [33], [34], [35], [36] TRL 5 [37]	Local Research Group	Open Source Software as a Service [38]

TABLE 4. Multi-dimensional categorization of the **ExplorViz** software visualization tool [27], [28], [29], [30].

Role	Readiness	Developer	Dissemination
1.2 Data Analytics 1.4 Integrative Analysis 2.2 Software Related 3.2 Data Collection and Generation 3.3 Pipelines and Tools	TRL 4	Local Research Group	Open Source

TABLE 5. Multi-dimensional categorization of **Hexatomic** for deep multi-layer linguistic annotation [39], [40].

Role	Readiness	Developer	Dissemination
1.3 Software Analytics 2.2 Software Related	TRL 4 [42], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53] TRL 5 [54] TRL 6 [55]	Community	Open Source

TABLE 6. Multi-dimensional categorization of the **Kieker** observability and monitoring framework [56], [57].

Role	Readiness	Developer	Dissemination
1.1 Modeling and Simulation 2.2 Software Related	TRL 9 [58], [59]	Contractor	Closed Source

TABLE 7. Multi-dimensional categorization of the **MATLAB** programming language [60].

Role	Readiness	Developer	Dissemination
1.1 Modeling and simulation 2.2 Software Related 3.3 Pipelines and Tools	TRL 4-8 [61], [62], [63], [64]	Community	Open Source Software as a Service

TABLE 8. Multi-dimensional categorization of the **MontiCore** framework for the development of software languages [65].

Role	Readiness	Developer	Dissemination
2.2 Software Related 3.3 Pipelines and Tools	TRL 6	Local Research Group	Open Source

TABLE 9. Multi-dimensional categorization of the **mosaik** for co-simulating energy systems [66].

Role	Readiness	Developer	Dissemination
1.2 Data Analytics 1.5 Scientific Visualization	TRL 4 [67]	Local Research Group	Open Source Software as a Service

TABLE 10. Multi-dimensional categorization of the **OceanTEA** ocean observation data analytics tool [68].

Kieker: The Kieker observability and monitoring framework has been employed in various software engineering research projects [56], [57]. The Kieker development started in 2006 as a tool for monitoring response times of Java software operations [69]. Research areas where Kieker was successfully employed for software engineering research include performance analysis and software architecture reconstruction. As reported in [57], Kieker was also employed in several industrial collaborations and technology transfer projects. Table 6 presents the multi-dimensional categorization of Kieker.

MATLAB: MATLAB is a programming language to design mathematical models to solve problems in science and industry [60]. It is a commercial software developed by MathWorks. The software fits to a case where a technology research software is closed-source and has a strong user base not only in academia, but also in industry. Table 7 presents the multi-dimensional categorization of MATLAB.

MontiCore: The MontiCore [70], [71], [72], [73] framework for the development of domain specific languages started in 2004 as proof-of-concept for researching design methods for software languages with large complexity, such as the UML-P [74], [75] or SysML [76]. Currently, MontiCore is used for developing code and test generators, domain specific languages, and model analysis tools [61], [62], [77], [78], [79]. Thus it turned into a research infrastructure. Still, original research on developing domain specific languages is done in the MontiCore context and many possible and useful extensions are asked for. Table 8 presents the multi-dimensional categorization of MontiCore.

mosaik: mosaik is a co-simulation framework for simulations in the energy domain [66]. The software is used as technology research software for the developers (computer scientists) while it is an infrastructure software for the main users (energy researchers). The multi-dimensional categorization of mosaik can be found in Table 9.

OceanTEA: OceanTEA supports research on ocean observation data analytics [68]. 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 [67]. Table 10 presents the multi-dimensional categorization of OceanTEA.

PIA: An example for a contractor-developed research software is PIA, the Prospective Monitoring and Management App, which has been developed by professional software companies as open-source software, on behalf of the Helmholtz Centre for Infection Research to conduct observational epidemiological studies by facilitating longitudinal data collection and cohort management [81]. Table 11 presents the multi-dimensional categorization of PIA.

Quantum Optics Control Software: This closed-source software is used to control ultracold atom experiments in microgravity environments such as sounding rockets or the International Space Station (ISS) [82], [83]. It also provides several Domain-Specific Languages (DSLs) for driver code and experiment sequence generation. Research areas where it has been successfully used are the MAIUS-1 mission [92], the first to create a Bose-Einstein Condensate (BEC) in space, and the MAIUS-2 mission. Future missions that will use the software include BECCAL [93] and CARIOQA-PMP [94] among others. Table 12 shows the multidimensional categorization of this software.

SPRAT: SPRAT provides a spatially-explicit marine ecosystem model based on population balance equations [85]. OceanTEA was empirically evaluated with ocean modelers and research software engineers researching marine ecosystem simulations [84]. Table 13 presents the multi-dimensional categorization of SPRAT.

Theodolite: Theodolite is a framework for benchmarking the horizontal and vertical scalability of cloud-native applications [89], [90]. Research areas where Theodolite was successfully employed for software engineering research include benchmarking stream processing engines deployed in the cloud [87], [88]. Table 14 presents the multi-dimensional categorization of Theodolite.

VirtualFluids: VirtualFluids is a Computational Fluid Dynamics (CFD) framework based on the Lattice Boltzmann method [95], [96], [97], [98]. The software demonstrated its use in operational environments in projects with industrial partners. Table 15 presents the multi-dimensional categorization of VirtualFluids.

Role	Readiness	Developer	Dissemination
3.2 Data Collection and Generation	TRL 9 [80]	Contractor	Open Source

TABLE 11. Multi-dimensional categorization of the **PIA** prospective monitoring and management app [81].

Role	Readiness	Developer	Dissemination
2.2 Software Related 3.1 Control and Monitoring Software 3.2 Data Collection and Generation	TRL 9	Project Group	Closed Source

TABLE 12. Multi-dimensional categorization of a **Quantum Optics Control Software** [82], [83].

Role	Readiness	Developer	Dissemination
1.1 Modeling and Simulation 1.5 Scientific Visualization	TRL 5 [84]	Individual Researcher	Open Source

TABLE 13. Multi-dimensional categorization of the **SPRAT** ocean observation data analytics tool [85].

Role	Readiness	Developer	Dissemination
2.2 Software Related 3.3 Pipelines and Tools	TRL 4 [86] TRL 5 [87], [88]	Project Group	Open Source

TABLE 14. Multi-dimensional categorization of the **Theodolite** framework for benchmarking the scalability of cloud-native applications [89], [90].

Our qualitative evaluation shows that it is possible to categorize different research software along multiple categories. In particular, it shows that our categorization is applicable to research software independently of a single dimension: we successfully categorized software at different maturity levels, developed by different actors, and disseminated through different means. We expect that our categorization can significantly contribute to categorizing research software. It increases coverage over existing approaches to categorization by adding the dissemination category and integrating:

- role-based categorization [13], [20] in our role categories;
- maturity-based categorization [16], [17] in our readiness categories;
- lifecycle-based categorization [19] in our developer categories;

In our evaluation, example research software has been categorized with 1–5 roles. This shows a high precision to cover different roles research software can take in different contexts, while manifesting that research software roles are not exclusive. While *PIA*, for example, serves a single purpose within a single context, *Hexatomic* can be used for different subtasks in different data-centric application contexts. As infrastructure software that can be used to integrate tools into a pipeline, Hexatomic combines research-related tasks such as data generation and integration with research tasks such as data editing and analysis. Simultaneously, it is technology research software whose target system is an existing ecosystem of software tools for linguistic research. The Hexatomic example reveals a property of research software that is central to our argument, i.e., that different contexts and perspectives put software into different roles, which makes a multi-dimensional categorization necessary.

As future work, we intend to ask more members of the RSE community to categorize their research software. In particular, the (self-) assessment of the readiness levels requires a profound knowledge of the software and its use.

As future work, we also intend to conduct more in-depth quantitative research into our categorization to assess and improve its granularity and precision. Based on this, we intend to analyze relations and correlations between categorical dimensions. We also plan to widen the corpus of categorized research software by asking more members of the RSE community to categorize their own research software. In particular, the assessment of the readiness levels requires a profound knowledge of the software and its use. To quantitatively evaluate our categorization scheme, we intend to apply more systematic and replicable research via a systematic literature review of published research software [99].

Role	Readiness	Developer	Dissemination
1.1 Modeling and Simulation	TRL 7	Local Research Group	Open Source [91]

TABLE 15. Multi-dimensional categorization of **VirtualFluids** for simulating fluid flow systems.

Conclusion

We categorize research software along various dimensions, contributing to fostering effective development, recognition, and utilization of research software within the research community. One essential use case of this categorization is its incorporation into forthcoming guidelines for research software development. As we classify research software, we enable tailoring guidelines to specific classes, offering developers a structured framework that aligns with each category's unique requirements and challenges. The evaluation via a systematic mapping study for our role-based categorization, and the multi-dimensional categorization of selected research software examples stimulated the refinement and strengthening of our categorization.

Moreover, the categorization is intended to be a valuable tool for stakeholders, especially research software engineers and their group, chair, department, or institute leaders. The categorization may provide these individuals with a better understanding of the software they are developing, offering insights into its nature, purpose, and potential impact. This knowledge is essential for informed decision-making, adequate resource allocation, and strategic planning within research institutions.

Recognition for research software engineers is another outcome we anticipate from categorizing research software. By delineating different types of software and acknowledging the diverse skill sets required for their development and maintenance, our categorization aims to contribute to elevating the status of research software engineers. We hope this recognition motivates individuals and fosters a culture that values and appreciates the crucial role played by software in advancing research efforts.

Categorizations may also help assess external software when considering its use. We envision that it contributes to a standardized framework for evaluating software's relevance, applicability, and quality, facilitating informed decisions in adopting tools from different sources.

The categorization may become particularly valuable in allocating project-based or permanent funding. It can help researchers and developers clearly articulate their software's significance in a funding proposal. We envision this classification providing a framework that helps researchers and funding agencies.

Additionally, the categorization may help to emphasize which software is critical, highlighting the importance of its maintenance and continued development for its continued functionality. By highlighting this importance, we seek to contribute to an enhanced awareness of the ongoing support and resources required to ensure the longevity and sustainability of research software.

In the realm of Research Software Engineering (RSE) research [7], [8], we hope that the categorization provides a framework for classifying research objects, supporting software corpus analyses, and enhancing our understanding of the different types of research software and their properties. This structured approach may aid in organizing and interpreting the vast landscape of research software, contributing to advancements in RSE methodologies and practices.

We propose a multi-dimensional categorization of research software, along the dimensions of roles, readiness, developers, and dissemination. The various dimensions of the categorization are not completely independent of each other. Looking at the dependency between the dimension and identifying constraints on combinations of the dimensions is the subject of future work. Additional dimensions could be the reuse scenarios (such as single-use/single-purpose, extensibility, reusability), the users (such as scientists, humans as research subjects, and citizens), the research software stack [21], and the criticality (for instance, mission-critical software). Such extensions and refinements are subject to future work.

To evaluate the *breath* of our categorization scheme, we conducted a multi-dimensional categorization of selected research software examples. As future work, we plan to conduct a systematic literature study [99] to evaluate the *depth* of our categorization.

REFERENCES

1. A. Johanson and W. Hasselbring, "Software engineering for computational science: Past, present, future," *Computing in Science & Engineering*, vol. 20, no. 2, pp. 90–109, Mar. 2018. doi: 10.1109/MCSE.2018.021651343
2. C. Jay, R. Haines, and D. S. Katz, "Software Must be Recognised as an Important Output of Scholarly Research," *International Journal of Digital Curation*, vol. 16, no. 1, p. 6, Apr. 2021. doi: 10.2218/ijdc.v16i1.745
3. H. Anzt, F. Bach, S. Druskat, F. Löffler, A. Loewe, B. Y. Renard *et al.*, "An environment for sustainable research software in Germany and beyond: Current state, open challenges, and call for action," *F1000Research*, vol. 9, p. 295, Jan. 2021. doi: 10.12688/f1000research.23224.2
4. M. Gruenpeter, D. S. Katz, A.-L. Lamprecht, T. Honeyman, D. Garijo, A. Struck *et al.*, "Defining Research Software: A controversial discussion," *Zenodo*, Sep. 2021. doi: 10.5281/zenodo.5504016
5. N. P. Chue Hong, D. S. Katz, M. Barker, A.-L. Lamprecht, C. Martinez, F. E. Psomopoulos *et al.*, "FAIR Principles for Research Software (FAIR4RS Principles)," *Research Data Alliance*, May 2022. doi: 10.15497/RDA00068
6. W. Hasselbring, L. Carr, S. Hettrick, H. Packer, and T. Tiropanis, "Open source research software," *Computer*, vol. 53, no. 8, pp. 84–88, Aug. 2020. doi: 10.1109/mc.2020.2998235
7. M. Felderer, M. Goedicke, L. Grunske, W. Hasselbring, A.-L. Lamprecht, and B. Rumpe, "Toward research software engineering research," *Zenodo*, 2023. doi: 10.5281/zenodo.8020525
8. ——, "Investigating research software engineering: Toward RSE Research," *Communications of the ACM*, vol. 68, no. 2, Feb. 2025. doi: https://doi.org/10.1145/3685265
9. A.-L. Lamprecht, L. Garcia, M. Kuzak, C. Martinez, R. Arcila, E. M. D. Pico *et al.*, "Towards FAIR principles for research software," *Data Science*, vol. 3, no. 1, pp. 37–59, Jun. 2020. doi: 10.3233/ds-190026
10. W. Hasselbring, L. Carr, S. Hettrick, H. Packer, and T. Tiropanis, "From FAIR research data toward FAIR and open research software," *it - Information Technology*, vol. 62, no. 1, pp. 39–47, Feb. 2020. doi: 10.1515/itit-2019-0040
11. T. Hey, S. Tansley, K. Tolle, and J. Gray, *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, Oct. 2009. ISBN 978-0-9825442-0-4. [Online]. Available: https://www.microsoft.com/en-us/research/publication/fourth-paradigm-data-intensive-scientific-discovery/
12. R. van Nieuwpoort and D. S. Katz, "Defining the roles of research software," *Upstream*, Jun. 2023. doi: 10.54900/9akm9y5-5ject5y
13. ——, "Defining the roles of research software (Version 2)," *Upstream*, Jul. 2024. doi: 10.54900/xdh2x-kj281
14. D. S. Katz, "Incentives and frictions in community software projects," *Zenodo*, Jun. 2022. doi: 10.5281/zenodo.6677821
15. A. D. Rose, M. Buna, C. Strazza, N. Olivieri, T. Stevens, L. Peeters, and D. Tawil-Jamault, "Technology readiness level: guidance principles for renewable energy technologies," *European Commission, Directorate General for Research and Innovation*, 2017. doi: 10.2777/577767
16. Australian Research Data Commons, "A National Agenda for Research Software," *Zenodo*, Mar. 2022. doi: 10.5281/zenodo.6378082
17. T. Schlauch, M. Meinel, and C. Haupt, "DLR Software Engineering Guidelines," *Zenodo*, Aug. 2018. doi: 10.5281/zenodo.1344612
18. O. Bertuch, D. Oliveira, U. Schelhaas, and A. Storm, "Guidelines for the development and distribution of software at Forschungszentrum Jülich," Forschungszentrum Jülich, Tech. Rep., 2022. [Online]. Available: http://hdl.handle.net/2128/33259
19. G. Courbebaisse, B. Flemisch, K. Graf, U. Konrad, J. Maassen, and R. Ritz, "Research software lifecycle," *Zenodo*, Sep. 2023. doi: 10.5281/zenodo.8324828
20. P. Döll, M. Sester, U. Feuerhake, H. Frahm, B. Fritzsch, D. C. Hezel *et al.*, "Sustainable research software for high-quality computational research in the Earth system sciences: Recommendations for universities, funders and the scientific community in Germany," *FID GEO*, Feb. 2023. doi: 10.23689/fidgeo-5805
21. K. Hinsen, "Dealing With Software Collapse," *Computing in Science Engineering*, vol. 21, no. 3, pp. 104–108, May 2019. doi: 10.1109/MCSE.2019.2900945
22. A. Barbie and W. Hasselbring, "ARCHES PiCar-X: Software for Digital Twin Research," *Journal of Open Source Software*, vol. 9, no. 102, Oct. 2024. doi: 10.21105/joss.07179
23. ——, "From digital twins to digital twin prototypes: Concepts, formalization, and applications," *IEEE Access*, vol. 12, pp. 75 337–75 365, 2024. doi: 10.1109/access.2024.3406510
24. A. Barbie, N. Pech, W. Hasselbring, S. Flögel, F. Wenzhöfer, M. Walter, E. Shchekinova, M. Busse,

M. Türk, M. Hofbauer, and S. Sommer, "Developing an Underwater Network of Ocean Observation Systems with Digital Twin Prototypes – A Field Report from the Baltic Sea," *IEEE Internet Computing*, vol. 26, no. 3, pp. 33–42, 2022. doi: 10.1109/mic.2021.3065245

25. A. Barbie and W. Hasselbring, "Toward reproducibility of digital twin research: Exemplified with the PiCar-X," *arXiv*, 2024. doi: 10.48550/ARXIV.2408.13866
26. A. Barbie, W. Hasselbring, and M. Hansen, "Digital twin prototypes for supporting automated integration testing of smart farming applications," *Symmetry*, vol. 16, no. 2, p. 221, Feb. 2024. doi: 10.3390/sym16020221
27. F. Fittkau, J. Waller, C. Wulf, and 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
28. F. Fittkau, S. Roth, and 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
29. F. Fittkau, A. Krause, and W. Hasselbring, "Software Landscape and Application Visualization for System Comprehension with ExplorViz," *Information and Software Technology*, vol. 87, pp. 259–277, Jul. 2017. doi: 10.1016/j.infsof.2016.07.004
30. W. Hasselbring, A. Krause, and C. Zirkelbach, "ExplorViz: Research on software visualization, comprehension and collaboration," *Software Impacts*, vol. 6, Nov. 2020. doi: 10.1016/j.simpa.2020.100034
31. F. Fittkau, A. Krause, and 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
32. F. Fittkau, S. Finke, W. Hasselbring, and 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
33. F. Fittkau, A. Krause, and W. Hasselbring, "Hierarchical software landscape visualization for system comprehension: A controlled experiment," in *Proc. 3rd IEEE Working Conference on Software Visualization (VISSOFT 2015)*. IEEE, Sep. 2015, pp. 36–45. doi: 10.1109/VISSOFT.2015.7332413
34. A. Krause-Glau, M. Hansen, and W. Hasselbring, "Collaborative program comprehension via software visualization in extended reality," *Information and Software Technology*, vol. 151, p. 107007, Nov. 2022. doi: 10.1016/j.infsof.2022.107007
35. A. Krause, M. Hansen, and W. Hasselbring, "Live visualization of dynamic software cities with heat map overlays," in *2021 Working Conference on Software Visualization (VISSOFT)*. IEEE, Sep. 2021, pp. 125–129. doi: 10.1109/vissoft52517.2021.00024
36. A. Krause-Glau and W. Hasselbring, "Collaborative, code-proximal dynamic software visualization within code editors," in *2023 IEEE Working Conference on Software Visualization (VISSOFT)*, Oct. 2023, pp. 50–61. doi: 10.1109/vissoft60811.2023.00016
37. A. Krause, C. Zirkelbach, W. Hasselbring, S. Lenga, and 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
38. A. Krause-Glau and 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, Sep. 2022, pp. 182–187. doi: 10.1109/ic2e55432.2022.00026
39. S. Druskat, T. Krause, C. Lachenmaier, and B. Bunzeck, "Hexatomic: An extensible, OS-independent platform fordeep multi-layer linguistic annotation of corpora," *Journal of Open Source Software*, vol. 8, no. 86, p. 4825, Jun. 2023. doi: 10.21105/joss.04825
40. ——, "Hexatomic (v1.4.5)," *Zenodo*, Oct. 2024. doi: 10.5281/zenodo.13959844
41. S. Druskat, V. Gast, T. Krause, and F. Zipser, "Corpus-tools.org: An Interoperable Generic Software Tool Set for Multi-layer Linguistic Corpora," in *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. Portorož, Slovenia: European Language Resources Association (ELRA), May 2016, pp. 4492–4499. [Online]. Available: https://aclanthology.org/L16-1711
42. J. Cito, P. Leitner, C. Bosshard, M. Knecht, G. Mazlami, and 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
43. W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, and Q. Zheng, "Service candidate identification from monolithic systems based on execution traces," *IEEE Transactions on Software Engineering*, vol. 47, no. 5, pp. 987–1007, May 2021. doi: 10.1109/tse.2019.2910531

44. Q. Zheng, Z. Ou, L. Liu, and 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
45. R. Dabrowski, "On architecture warehouses and software intelligence," in *Proc. 4th Int. Mega-Conference on Future Generation Information Technology (FGIT 2012)*, ser. LNCS, vol. 7709. Springer, 2012, pp. 251–262. doi: 10.1007/978-3-642-35585-1_35
46. Y. Qu, Q. Zheng, T. Liu, J. Li, and 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
47. Y. Qu, X. Guan, Q. Zheng, T. Liu, J. Zhou, and J. Li, "Calling network: A new method for modeling software runtime behaviors," *SIGSOFT Softw. Eng. Notes*, vol. 40, no. 1, pp. 1–8, Jan. 2015. doi: 10.1145/2693208.2693223
48. Y. Qu, X. Guan, Q. Zheng, T. Liu, L. Wang, Y. Hou, and Z. Yang, "Exploring community structure of software call graph and its applications in class cohesion measurement," *Journal of Systems and Software*, vol. 108, pp. 193–210, 2015. doi: 10.1016/j.jss.2015.06.015
49. W. Jin, T. Liu, Y. Qu, J. Chi, D. Cui, and 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
50. W. Jin, T. Liu, Y. Qu, Q. Zheng, D. Cui, and J. Chi, "Dynamic structure measurement for distributed software," *Software Quality Journal*, vol. 26, no. 3, 2018. doi: 10.1007/s11219-017-9369r3
51. S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Scalable process discovery and conformance checking," *Software & Systems Modeling*, vol. 17, no. 2, pp. 599–631, 2018. doi: 10.1007/s10270-016-0545-x
52. H. Schnoor and W. Hasselbring, "Comparing Static and Dynamic Weighted Software Coupling Metrics," *Computers*, vol. 9, no. 2, pp. 1–21, Mar. 2020. doi: 10.3390/computers9020024
53. B. Andrade, S. Santos, and 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
54. S. Yang, Y. Jeong, C. Hong, H. Jun, and 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
55. A. van Hoorn, S. Frey, W. Goerigk, W. Hasselbring, H. Knoche, S. Köster, H. Krause, M. Porembski, T. Stahl, M. Steinkamp, and N. Wittmüss, "Dynamod project: Dynamic analysis for model-driven software modernization," in *Proc. Int. Workshop on Model-Driven Software Migration (MDSM) 2011*, vol. 708. CEUR, 2011, pp. 12–13. [Online]. Available: https://ceur-ws.org/Vol-708/
56. A. van Hoorn, J. Waller, and 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)*, Apr. 2012, pp. 247–248. doi: 10.1145/2188286.2188326
57. W. Hasselbring and A. van Hoorn, "Kieker: A monitoring framework for software engineering research," *Software Impacts*, vol. 5, p. 100019, Aug. 2020. doi: 10.1016/j.simpa.2020.100019
58. S. S. P. Kode, Y. Shtessel, A. Levant, J. Rakoczy, M. Hannan, and J. Orr, "Development of relative degree-based aerospace sliding mode control matlab toolbox with case studies," *IEEE Aerospace and Electronic Systems Magazine*, vol. 39, no. 9, pp. 72–96, 2024. doi: 10.1109/MAES.2022.3177576
59. L. R. Ribeiro and N. M. F. Oliveira, "Uav autopilot controllers test platform using matlab/simulink and x-plane," in *2010 IEEE Frontiers in Education Conference (FIE)*, 2010. doi: 10.1109/FIE.2010.5673378
60. The MathWorks Inc., "Matlab," Natick, Massachusetts, United States, 2024. [Online]. Available: https://www.mathworks.com/products/matlab.html
61. A. Butting, K. Hölldobler, B. Rumpe, and A. Wortmann, "Compositional Modelling Languages with Analytics and Construction Infrastructures Based on Object-Oriented Techniques - The MontiCore Approach," in *Composing Model-Based Analysis Tools*. Springer, July 2021, pp. 217–234. doi: 10.1007/978-3-030-81915-6_10
62. I. Blundell, J. M. Eppler, A. Morrison, K. Perun, D. Plotnikov, B. Rumpe, and G. Trensch, "Reengineering NestML with Python and MontiCore," *Zenodo*, July 2018. doi: 10.5281/zenodo.1319653
63. J. C. Kirchhof, B. Rumpe, D. Schmalzing, and A. Wortmann, "MontiThings: Model-driven Development and Deployment of Reliable IoT Applications," *Journal of Systems and Software*, vol. 183, pp. 1–21, Jan. 2022. doi: 10.1016/j.jss.2021.111087
64. A. Gerasimov, P. Letmathe, J. Michael, L. Netz, and B. Rumpe, "Modeling Financial, Project and

Staff Management: A Case Report from the MaCoCo Project," *Enterprise Modelling and Information Systems Architectures - International Journal of Conceptual Modeling*, vol. 19, Feb. 2024. doi: 10.18417/emisa.19.3

65. H. Krahn, B. Rumpe, and S. Völkel, "Monticore: a framework for compositional development of domain specific languages," *International Journal on Software Tools for Technology Transfer*, vol. 12, no. 5, 2010. doi: 10.1007/s10009-010-0142-1
66. C. Steinbrink, M. Blank-Babazadeh, A. El-Ama, S. Holly, B. Lüers, M. Nebel-Wenner, R. P. Ramírez Acosta, T. Raub, J. S. Schwarz, S. Stark, A. Nieße, and S. Lehnhoff, "CPES testing with mosaik: Co-simulation planning, execution and analysis," *Applied Sciences*, vol. 9, no. 5, 2019. doi: 10.3390/app9050923
67. A. Johanson, S. Flögel, C. Dullo, P. Linke, and W. Hasselbring, "Modeling polyp activity of Paragorgia arborea using supervised learning," *Ecological Informatics*, vol. 39, pp. 109–118, May 2017. doi: 10.1016/j.ecoinf.2017.02.007
68. A. Johanson, S. Flögel, C. Dullo, and W. Hasselbring, "OceanTEA: Exploring ocean-derived climate data using microservices," in *Proceedings of the Sixth International Workshop on Climate Informatics (CI 2016)*, Sep. 2016, pp. 25–28, Technical Note NCAR/TN-529+PROC. doi: 10.5065/D6K072N6
69. M. Rohr, A. van Hoorn, J. Matevska, N. Sommer, L. Stoever, S. Giesecke, and 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)*, Feb. 2008, pp. 80–85. doi: 10.5555/1722603.1722619
70. K. Hölldobler, O. Kautz, and B. Rumpe, *MontiCore Language Workbench and Library Handbook: Edition 2021*, ser. Aachener Informatik-Berichte, Software Engineering, Band 48. Shaker Verlag, May 2021. ISBN 978-3-8440-8010-0
71. K. Hölldobler and B. Rumpe, *MontiCore 5 Language Workbench Edition 2017*, ser. Aachener Informatik-Berichte, Software Engineering, Band 32. Shaker Verlag, December 2017. ISBN 978-3-8440-5713-3
72. N. Jansen and B. Rumpe, "Compositional Modeling Languages in Action: Engineering and Application of Heterogeneous Languages with MontiCore," in *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Tutorials*, I. Malavolta and J. Michael, Eds., Västerås, Sweden, October 2023.
73. A. Butting, R. Gupta, N. Jansen, N. Regnat, and B. Rumpe, "Towards Modular Development of Reusable Language Components for Domain-Specific Modeling Languages in the MagicDraw and MontiCore Ecosystems," *Journal of Object Technology (JOT)*, vol. 22, no. 1, pp. 1:1–21, September 2023. doi: 10.5381/jot.2023.22.1.a4
74. B. Rumpe, *Modeling with UML: Language, Concepts, Methods*. Springer International, July 2016. [Online]. Available: https://mbse.se-rwth.de/
75. ——, *Agile Modeling with UML: Code Generation, Testing, Refactoring*. Springer International, May 2017. [Online]. Available: https://mbse.se-rwth.de/
76. H. Kausch, M. Pfeiffer, D. Raco, B. Rumpe, and A. Schweiger, "Model-driven Development for Functional Correctness of Avionics Systems: A Verification Framework for SysML Specifications," *CEAS Aeronautical Journal*, vol. 15, no. 4, October 2024. doi: 10.1007/s13272-024-00762-6
77. A. Haber, K. Hölldobler, C. Kolassa, M. Look, K. Müller, B. Rumpe, and I. Schaefer, "Engineering Delta Modeling Languages," in *Software Product Line Conference (SPLC'13)*. ACM, 2013, pp. 22–31. doi: 10.1145/2491627.2491632
78. B. Rumpe, M. Schoenmakers, A. Radermacher, and A. Schürr, "UML + ROOM as a Standard ADL?" in *Engineering of Complex Computer Systems, ICECCS'99 Proceedings*, ser. IEEE Computer Society, 1999. doi: 10.1109/ICECCS.1999.802849
79. T. Clark, M. v. d. Brand, B. Combemale, and B. Rumpe, "Conceptual Model of the Globalization for Domain-Specific Languages," in *Globalizing Domain-Specific Languages*, ser. LNCS 9400. Springer, 2015, pp. 7–20. doi: 10.1007/978-3-319-26172-0_2
80. J.-K. Heise, R. Dey, M. Emmerich, Y. Kemmling, S. Sistig, G. Krause, and S. Castell, "Putting digital epidemiology into practice: PIA – prospective monitoring and management application," *Informatics in Medicine Unlocked*, vol. 30, p. 100931, 2022. doi: 10.1016/j.imu.2022.100931
81. S. Castell, J.-K. Heise, L. Weber, L. Dietsch, and T. Wangler, "eResearch System: Prospective Monitoring and Management – App (PIA)," *Zenodo*, Feb. 2024. doi: 10.5281/zenodo.10635668
82. B. Weps, D. Lüdtke, T. Franz, O. Maibaum, T. Wendrich, H. Müntinga, and A. Gerndt, "A model-driven software architecture for ultra-cold gas experiments in space," in *Proceedings of the International Astronautical Congress, IAC*, Oct. 2018. [Online]. Available: https://elib.dlr.de/126145/
83. A. Prat, J. Sommer, A. M. Nepal, T. Franz, H. Müntinga, A. Gerndt, and D. Lüdtke, "The BECCAL Experiment Design and Control Software," in *2021 IEEE Aerospace*

Conference (50100), Mar. 2021, pp. 1–9. doi: 10.1109/AERO50100.2021.9438129

84. A. Johanson and W. Hasselbring, "Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment," *Empirical Software Engineering*, vol. 22, no. 4, pp. 2206–2236, Aug. 2017. doi: 10.1007/s10664-016-9483-z
85. A. N. Johanson, A. Oschlies, W. Hasselbring, and B. Worm, "SPRAT: a spatially-explicit marine ecosystem model based on population balance equations," *Ecological Modelling*, vol. 349, pp. 11–25, 2017. doi: 10.1016/j.ecolmodel.2017.01.020
86. G. Kp, G. Pierre, and R. Rouvoy, "Studying the energy consumption of stream processing engines in the cloud," in *2023 IEEE International Conference on Cloud Engineering (IC2E)*, Sep. 2023, pp. 99–106. doi: 10.1109/ic2e59103.2023.00019
87. S. Henning, A. Vogel, M. Leichtfried, O. Ertl, and 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*, May 2024. doi: 10.1145/3629526.3645036
88. S. Henning and W. Hasselbring, "Benchmarking scalability of stream processing frameworks deployed as microservices in the cloud," *Journal of Systems and Software*, vol. 208, p. 111879, Feb. 2024. doi: 10.1016/j.jss.2023.111879
89. ——, "Theodolite: Scalability benchmarking of distributed stream processing engines in microservice architectures," *Big Data Research*, vol. 25, no. 100209, pp. 1–17, Jul. 2021. doi: 10.1016/j.bdr.2021.100209
90. ——, "A configurable method for benchmarking scalability of cloud-native applications," *Empirical Software Engineering*, vol. 27, no. 143, pp. 1–42, 2022. doi: 10.1007/s10664-022-10162-1
91. S. Peters, K. Kutscher, M. Schönherr, M. Geier, H. Alihussein, A. Wellmann, and H. Korb, "Virtualfluids," *Zenodo*, 2024. doi: 10.5281/zenodo.10283048
92. D. Becker *et al.*, "Space-borne Bose–Einstein condensation for precision interferometry," *Nature*, vol. 562, no. 7727, pp. 391–395, Oct. 2018. doi: 10.1038/s41586-018-0605-1
93. K. Frye *et al.*, "The Bose-Einstein Condensate and Cold Atom Laboratory," *EPJ Quantum Technology*, vol. 8, no. 1, pp. 1–38, Dec. 2021. doi: 10.1140/epjqt/s40507-020-00090-8
94. M. Schilling *et al.*, "Scientific assessment of the CARIOQA-PMP quantum accelerometer pathfinder," in *XXVIII General Assembly of the International Union of Geodesy and Geophysics (IUGG)*. GFZ German Research Centre for Geosciences, Jul. 2023. doi: 10.57757/IUGG23-4073
95. H. Alihussein, A. Prasannakumar, M. Geier, and M. Krafczyk, "Direct cumulant lattice boltzmann simulations of transitional flow in gyroidal structures including experimental validation," *Computers & Mathematics with Applications*, vol. 157, p. 159–172, Mar. 2024. doi: 10.1016/j.camwa.2023.12.029
96. D. Adekanye, A. Khan, A. Burns, W. McCaffrey, M. Geier, M. Schönherr, and R. Dorrell, "Graphics processing unit accelerated lattice boltzmann method simulations of dilute gravity currents," *Physics of Fluids*, vol. 34, no. 4, Apr. 2022. doi: 10.1063/5.0082959
97. K. Kutscher, M. Geier, and M. Krafczyk, "Multiscale simulation of turbulent flow interacting with porous media based on a massively parallel implementation of the cumulant lattice boltzmann method," *Computers & Fluids*, vol. 193, p. 103733, Oct. 2019. doi: 10.1016/j.compfluid.2018.02.009
98. J. Linxweiler, M. Krafczyk, and J. Tölke, "Highly interactive computational steering for coupled 3d flow problems utilizing multiple gpus," *Computing and Visualization in Science*, pp. 1–16, 2011. doi: 10.1007/s00791-010-0151-3
99. C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, "Systematic literature studies," in *Experimentation in Software Engineering*, 2nd ed. Springer, 2024, pp. 51–63. doi: 10.1007/978-3-662-69306-3_4

Wilhelm Hasselbring is a professor of software engineering at Kiel University, Germany. His research interests include software engineering, distributed systems, and open science. Hasselbring received a PhD in computer science from the University of Dortmund. He is a member of the ACM, IEEE Computer Society, and the German Association for Computer Science, at which he is vice chair of the special interest group on research software engineering. Contact him at hasselbring@email.uni-kiel.de.

Stephan Druskat is a computer science researcher at the German Aerospace Center (DLR), and a Fellow of the Software Sustainability Institute (UK). He co-founded, and served on the inaugural board of, de-RSE – Society for Research Software. His research interests include research software sustainability, software citation and publication, and empirical research software engineering. He is a member of the Society for Research Software Engineering, the German Society for Research Software, and the German Association for Computer Science, where he co-founded the special interest group on research software engineering. Contact him at stephan.druskat@dlr.de.

Jan Bernoth is a researcher at the University of Potsdam, affiliated with the Chair of Complex Multimedia Application Architectures. His main research focus is on designing an architecture of the infrastructure to support Research Data and Software Management within a National Research Data Infrastructure Germany consortium NFDIxCS. Additionally, he investigates in his PhD thesis the creation of a research environment aimed at investigating analytical dashboards for science communication. Contact him at jan.bernoth@uni-potsdam.de.

Philine Betker is a researcher at the Helmholtz Centre for Infection Research, Brunswick, Germany. She is funded as part of a subproject of NAKO by the Federal Ministry of Education and Research (BMBF, project funding reference number 01ER2301/12), and the Helmholtz Association, with additional financial support by the participating universities and the institutes of the Leibniz Association. Contact her at philine.betker@helmholtz-hzi.de.

Michael Felderer is the director of the Institute for Software Technology at German Aerospace Center (DLR) and a full professor in the Department of Computer Science at the University of Cologne. His research interests include software quality and security, software engineering for AI, quantum computing and digital twin technologies as well as empirical and research software engineering. Contact him at michael.felderer@dlr.de.

Stephan Ferenz is a senior researcher at the Carl von Ossietzky Universität Oldenburg, Germany. His research interests are research data management, research software engineering, research software metadata. He holds a Master's degree in Electrical Engineering from the Leibniz Universität Hannover, Germany. Stephan Ferenz is a member of the German Association for Computer Science. Contact him at stephan.ferenz@uol.de.

Ben Hermann is a professor for secure software engineering at TU Dortmund University. His research is focused on static program analysis and metascience. Especially, his work on research artifact evaluation (i.e. peer review) have caught much attention. He received his PhD in computer science from TU Darmstadt. Contact him at ben.hermann@cs.tu-dortmund.de.

Anna-Lena Lamprecht is professor of software engineering at the University of Potsdam, Germany. She focuses on research software engineering (RSE) and is particularly known for her work on the FAIR Principles for Research Software (FAIR4RS) and on the automated composition of scientific workflows. Lamprecht is chair of the new special interest group on RSE that has recently been installed as a joint endeavour of the German Association for Computer Science and the German Association for Research Software Engineering. Contact her at anna-lena.lamprecht@uni-potsdam.de.

Jan Linxweiler is the general manager of the Center for Mechanics, Uncertainty and Simulation in Engineering (MUSEN) at TU Braunschweig and head of IT and research-related services at the University library. His research interests include Research Software Engineering, High Performance Computing, Research Data Management, and Open Science. Linxweiler holds a PhD in Engineering and is a founding member of the German RSE association of which he recently became chairman of the board. Contact him at j.linxweiler@tu-braunschweig.de.

Arnau Prat is a researcher at the German Aerospace Center (DLR). He leads the Payload Software research group. His current research interests include model-driven software engineering for space systems and the use of novel programming languages for safety-critical systems. He is currently involved in several space missions, ranging from payloads on board the

ISS to sounding rockets or satellites. Contact him at arnau.pratisala@dlr.de.

Bernhard Rumpe is the Software Engineering chair at RWTH Aachen University and Editor-In-Chief of the SoSyM Journal. His main interests are rigorous and practical software and system development methods based on adequate modeling techniques. This includes agile development methods and model-engineering based on UML/SysML-like notations and domain specific languages. He also helps to apply modeling, e.g., to human brain simulation, contract digitalization, production automation, or cloud. He is author books of several like "Agile Modeling with the UML" and "Engineering Modeling Languages: Turning Domain Knowledge into Tools". Contact him at rumpe@se-rwth.de.

Katrin Schoening-Stierand is a senior digital consultant at the Hub for Computing and Data Science (HCDS) at the University of Hamburg. Her focus is on fostering interdisciplinary collaboration between different fields of science. Her research interests are informatics in the natural sciences and Research Software Engineering. She is involved in teaching and offers a course on Research Software Engineering. Contact her at katrin.schoening-stierand@uni-hamburg.de.

Shinhyung Yang is a postdoctoral researcher in the Software Engineering group at Kiel University with the SustainKieker project, focusing on the sustainability of research software. His research interests include performance engineering of cloud-native applications with a special focus on Java virtual machines and native applications in distributed and parallel architecture. He received a PhD degree from Yonsei University in South Korea, and his research is supported by the Deutsche Forschungsgemeinschaft (DFG – German Research Foundation), grant no. 528713834. Contact him at shinhyung.yang@email.uni-kiel.de.

Appendix
Characterization of Related Research Software Categories

The related research software categories are characterized in terms of the template in Table 1.
Table 16 characterizes the role-based categorization by van Nieuwpoort and Katz.
Table 17 characterizes the ARDC categorization.
Table 18 characterizes the institutional guideline application class categorization.
Table 19 characterizes the EOSC research software lifecycle categorization.
Table 20 characterizes the categories in computational research in the Earth system sciences.
Table 21 characterizes the software stack categorization.

Criterion	Explanation
Scope	Role-based categorization.
Purpose	Funding organizations joined forces to explore how they could effectively contribute to making research software sustainable.
Context	International workshop in 2022 on the future of research software, organized by the Research Software Alliance (ReSA) and the Netherlands eScience Center.
Properties	The roles for research software are defined from the point of view of a researcher, with the goal of making this understandable for funders and policymakers.
Consequences for Creation	Depending on its role category, software is expected to meet different quality requirements and follow different development processes.
Consequences for Use	Perceive that there are many different types of research software, fulfilling many different roles and functions.
Inter-categorial relations	Individual research software may change its role or take multiple roles.

TABLE 16. Characteristics of the role-based categorization by van Nieuwpoort and Katz [12], [13].

Criterion	Explanation
Scope	The categorization in [16] supports a discussion about recognition of software in research, with the aim to increase this recognition.
Purpose	The categorization aims to describe the purpose of the software it categorizes as capturing applied or widely accepted research ideas, methodology, and models, or demonstrating new ones.
Context	The categorization has been produced in the context of ARDC's research software policy.
Properties	The properties of the categories represent different challenges faced by software that fall in the respective category.
Consequences for Creation	Depending on its category, software is expected to meet different requirements. While analysis code should be FAIR [5], prototype tools should exhibit a "high quality", and research software infrastructure must be created for sustainability, which is realized through safeguarding its long-term maintenance.
Consequences for Use	Software use is featured only implicitly in the categorization. We expect that software under the different categories are expected to be used differently: Analysis tools are used for specific research tasks, and are more likely to have a small scope, e.g., are applied only to answer a specific research question. Prototype tools are used to test the methodological hypotheses they implement, but may also be used experimentally to answer specific research questions.
Inter-categorial relations	The categories are related through *evolution* and *transitive value*. One category evolves from another, e.g., analysis code may evolve into a prototype tool, that in turn evolves into research software infrastructure.

TABLE 17. Characteristics of ARDC's research software categorization [16].

Criterion	Explanation
Scope	Guidelines for software engineering at an academic institution.
Purpose	Identify suitable quality requirements.
Context	Institutional policy and practice.
Properties	Criticality, institutional risk, projected use, development timeline, distribution, commercial exploitation.
Consequences for Creation	Increasingly employ established software engineering methods.
Consequences for Use	Increased (critical) use by increasingly large community.
Inter-categorial relations	Transitive requirements, legal requirements.

TABLE 18. Characteristics of institutional guideline application classes [17], [18].

Criterion	Explanation
Scope	Developer- and stakeholder-based categorization.
Purpose	Achieve a common understanding of the current processes in research software engineering, particularly the research software lifecycle.
Context	SubGroup 1 “On the Software Lifecycle” of the EOSC Task Force “Infrastructure for quality research software”.
Properties	Different levels of adopting software engineering practice, different publication requirements and usage scenarios, different stakeholders
Consequences for Creation	Depending on its developer category, software is expected to meet different quality requirements and follow different development processes.
Consequences for Use	Increasing maturity and support for reproducibility
Inter-categorial relations	Not specified

TABLE 19. Characteristics of the EOSC research software lifecycle categorization [19].

Criterion	Explanation
Scope	Recommendations for universities, funders, and the scientific community.
Purpose	Safeguard the quality and efficiency of computational research in Earth System Sciences and make research results that have been generated by research software reproducible.
Context	Ideas of a DFG round table meeting on sustainable research software for high-quality computational research in the Earth System Sciences.
Properties	Research software developed in the Earth System Sciences is characterized by the complexity of the underlying models, multifaceted dependencies, the multi-modality of the data, and the size of the data, which can impose specific hardware and software requirements.
Consequences for Creation	Depending on its role category, software is expected to meet different quality requirements and follow different development processes.
Consequences for Use	Dependency on the research cycle
Inter-categorial relations	Combination, integration

TABLE 20. Characteristics of categories in computational research in the Earth system sciences [20].

Criterion	Explanation
Scope	Describing principles of software collapse.
Purpose	Identify dependent layers of different (academic) specificity to model threat.
Context	Research software sustainability.
Properties	Domain specificity.
Consequences for Creation	Build on stable lower layers, quickly react to threats, accept agility.
Consequences for Use	Decreasing specificity of application domain from top to bottom.
Inter-categorial relations	Dependency, transitive threats.

TABLE 21. Characteristics of categorizing the software stack [21].