Research software engineering and the importance of scientific models

Benoit Combemale, Jeff Gray, Bernhard Rumpe

Software and Systems Modeling (2023) 22:1081–1083
https://doi.org/10.1007/s10270-023-01119-z

EDITORIAL

Research software engineering and the importance of scientific models

Benoit Combemale[1] · Jeff Gray[2] · Bernhard Rumpe[3]

Published online: 29 July 2023

Research Software Engineering is the use of Software Engineering practices to support the software needs when conducting research [Wikipedia, July 2023].

This is the definition of a new and emerging field of software engineering. Similar to the original term "software engineering," the creation of the subfield "Research Software Engineering" (RSE) was necessary because there is an ongoing crisis in the development of software that supports research activities. The crisis is not coming from constraints on computational resources or complexity of scientific problems to solve, per se, but from the effort needed to develop research software correctly, on time, reliable, reproducible, and all of the other desirable properties of Software Engineering.

Software Engineering was made prominent when F. L. Bauer hosted the first conference on Software Engineering in 1968 to address the general software crisis that was identified in the 1960s. One of the emerging results was that the early phases of software development (i.e., requirements, architecture, design, and testing) could be assisted using model-based techniques. Emerging from the formal methods community, a number of modeling languages have been proposed, such as E/R, Statecharts, and Petri nets, which later emerged into other forms supporting variations of concept modeling, UML, and SysML.

Since 2010, communities for RSE have emerged where conferences were organized to facilitate the exchange of ideas on how to overcome the challenges of research software development, i.e., how to overcome the research software crisis.

✉ Bernhard Rumpe
bernhard.rumpe@sosym.org

Benoit Combemale
benoit.combemale@sosym.org

Jeff Gray
jeff.gray@sosym.org

[1] University of Rennes, Rennes, France

[2] University of Alabama, Tuscaloosa, AL, USA

[3] RWTH Aachen University, Aachen, Germany

What differentiates the challenges of RSE from other forms of SE?

We recognize that there is no uniform representation of what is represented by the term "research software." Research software mainly falls into one of the following categories, often with concept overlap:

- Embedded control software for complex physical or chemical experiments, respectively, their machinery, including many forms of sensor-based data collections.
- Simulation of physical, chemical, social, or biological processes in geometrically distributed spaces.
- Data processing and aggregation of large experimental results.
- Symbolic manipulation systems, such as computer algebra systems or theorem provers.
- Demonstrators and prototypes of various forms and with a large variety of goals and hypotheses.
- Medical device software that is being used to improve intensive care in hospitals.

Much can be said about the characteristics of research software and their differences to traditional, industrial software. We highlight the following main distinctions:

- The research activity of a domain researcher (e.g., in physics, biology, or chemistry) is deeply intertwined with the requirements elicitation for the research software.
- While the original focus may have been to develop efficient software, it now becomes clear that the efficiency of developers must be given more precedence (without ignoring the original focus).
- Testing of research software can be challenging, because while the software is complicated, at development time only one data set may exist on which it needs to be executed.
- Scientific standards currently evolve, including the need for enforcing reproducibility and long-term availability of the software, similar to legal regulations for banking software or factual availability for long-lasting products, such as cars or airplanes.

As a consequence, reuse (beyond copy–paste), modularity, design for long-lasting systems, evolutionary development processes, and especially the understanding of the developed code become much more relevant. The explicit use of models can offer significant benefits because research software typically relies on scientific models.

Scientific models, however, are typically mathematical or physical laws that are defined in scientific papers or general textbooks. Meanwhile, several tools, such as Modelica, can demonstrate how to automatically operationalize these laws, but in practice the efficiency of the code is just "good enough" to prevent developers from optimizing these solutions. The ability to model data structures and architectures explicitly (e.g., with UML) and to map those with target-specific code generators to parallel computation structures is not a strategy that is often used in RSE. Furthermore, test settings are not very common, and in particular not model-based automated test settings.

We believe that the next step needs to be taken to arrive either at more general "ResearchML" or a collection of domain-specific languages that address various aspects of a research software starting from requirements, through architecture, coding, and especially testing. This could be realized through the use of customized domain-specific languages for individual large pieces of research software. For example, NESTML is a domain-specific language developed for neuron and synapse models that is used in simulations of brain activity in a simulator as part of the European Human Brain Project. In the future, it may even be that the scientific paper, where the original models come from, is directly connected with the code, which could assist with the automated traceability of RSE requirements.

As editors-in-chief of the SoSyM journal, we are interested in success stories that describe the development and application of modeling languages in long-lasting, relevant scientific and industrial projects. Researchers in various domains seek to understand the world through explicit software models that can help them to improve the efficiency and quality of the software in their specific research focus. The benefits of research software include the ability to assist with reproducibility of results, understandability of the evolvable code, and verifiability of the scientific correctness of the results.

This issue's expert voice: This issue also contains an insightful Expert Voice article on the Rubus Component Model by Alessio Bucaioni, Federico Ciccozzi, Amleto Di Salle, and Mikael Sjödin. Their contribution describes the results of a series of European projects that developed a component structure for distributed real-time systems. These projects also produced a set of tools that allow explicit modeling of specific aspects of the systems that are mapped to this component infrastructure. The authors give an overview of the publications around the Rubus Component Model over the last 20 years. The Rubus Component Model represents an example of research software that served as a blueprint for industrial software approaches that emerged into industrially applicable software tooling. In SoSyM, we welcome such overview articles because they also demonstrate how research software starts, emerges within the discussion of scientific communities, and leads to industrial adoption.

1 Content of this Issue

1. **Expert Voice**
   - "From low-level programming to full-fledged industrial model-based development: the story of the Rubus Component Model" by Alessio Bucaioni, Federico Ciccozzi, Amleto Di Salle, and Mikael Sjödin
2. **Theme Section Paper**
   - "On the use of domain knowledge for process model repair" by Kate Cerqueira Revoredo
3. **Regular Papers**
   - "The complexities of the satisfiability checking problems of feature diagram sublanguages" by Oliver Kautz
   - "Real-time collaborative multi-level modeling by conflict-free replicated data types" by Istvan David and Eugene Syriani
   - "CalcGraph: Taming the high costs of deep learning using models" by Joe Lorentz, Thomas Hartmann, Assaad Moawad, Francois Fouquet, Djamila Aouada, and Yves Le Traon
   - "Modeling ecosystems of reference frameworks for assurance: A case on privacy impact assessment regulation and guidelines" by Alejandra Ruiz, Yod-Samuel Martin, Jabier Martinez, Jacobo Quintans, Guillaume Mockly, Amelie Gyrard, and Tommaso Crepax
   - "Flexmi: A generic and modular textual syntax for domain-specific modelling" by Dimitris Kolovos and Alfonso de la Vega
   - "Visual query languages to design complex queries: A systematic literature review" by Edson Silva, Robson Fidalgo, Márcio Ferro, and Natália Franco
   - "MBIPV: A model-based approach for identifying privacy violations from software requirements" by Tong Ye, Yi Zhuang, and Gongzhe Qiao
   - "A graph-based framework for model-driven optimization facilitating impact analysis of mutation operator properties" by Stefan John, Jens Kosiol, Leen Lambers, and Gabriele Taentzer

- "A generic framework for representing and analyzing model concurrency" by Steffen Zschaler, Erwan Bousse, Julien DeAntoni, and Benoit Combemale
- "Improving active participation during enterprise operations modeling with an extended story-card-method and participative modeling software" by Marne de Vries and Petra Opperman
- "Gamifying model-based engineering: The PapyGame experience" by Antonio Bucchiarone, Maxime Savary-Leblanc, Xavier Le Pallec, Antonio Cicchetti, Sebastien Gerard, Simone Bassanelli, Federica Gini, and Annapaola Marconi

Funding Open Access funding enabled and organized by Projekt DEAL.