FAIR and Open Research Software


Wilhelm (Willi) Hasselbring

Software Engineering
http://se.informatik.uni-kiel.de

FAU Erlangen-Nürnberg, November 11th 2022
Research Software

- Research software is software
  - that is employed in the scientific discovery process or
  - a research object itself.

- Computational science (also scientific computing) involves the development of research software
  - for model simulations and
  - data analytics
  to understand natural systems answering questions that neither theory nor experiment alone are equipped to answer.
Agenda

1. Research Software
   • Characteristics
   • Mutual ignorance
2. Research Software Publishing
3. Open Science
   • For Computational Science
   • For Computer Science / Software Engineering: Artifact Evaluation
4. FAIR and Open Research Software
5. Summary & Outlook
Characteristics of Research Software

- **Functional Requirements** are not known up front
  - And often hard to comprehend without some PhD in science

- **Verification** and validation are difficult,
  - and strictly scientific

- Overly formal software **processes** restrict research

[Diagram]

Johanson & Hasselbring 2018
Characteristics of Research Software

• Software **quality requirements**
  • Jeffrey Carver and colleagues found that scientific software developers rank the following characteristics as the most important, in descending order [Carver et al. 2007]:
    1. functional (scientific) correctness,
    2. performance,
    3. portability, and
    4. maintainability.

• Research software in itself has **no value**
  • Not really true for community software

• Few scientists are **trained** in software engineering
  • Disregard of most modern software engineering methods and tools
Although much of the discussions were of a detailed technical nature, the report also contains sections reporting on discussions which will be of interest to a much wider audience. This holds for subjects like:

- the problems of achieving sufficient reliability in the data systems which are becoming increasingly integrated into the central activities of modern society
- the difficulties of meeting schedules and specifications on large software projects
- the education of software (or data systems) engineers
- the highly controversial question of whether software should be priced separately from hardware.

Thus, while the report is of particular concern to the immediate users of computers and to computer manufacturers, many points may serve to enlighten and warn policy makers at all levels. Readers from the wider audience should note, however, that the conference was concentrating on the basic issues and key problems in the critical areas of software engineering. It therefore did not attempt to provide a balanced review of the total state of software, and tends to understress the achievements of the field.

In fact, a tremendously excited and enthusiastic atmosphere developed at the conference as participants came to realize the degree of common concern about what some were even willing to term the “software crisis”, and general agreement arose about the importance of trying to convince not just other colleagues, but also policy makers at all levels, of the seriousness of the problems that were being discussed.

[Randell 2018]
Mutual Ignorance: Software Engineering

Software Engineering and Computer Science for Generality [Randell 2018]:

• “That NATO was the sponsor of this conference marks the relative distance of software engineering from computation in the academic context.

• The perception was that while errors in scientific data processing applications might be a ‘hassle,’ they are all in all tolerable.

• In contrast, failures in mission-critical military systems might cost lives and substantial amounts of money.

• Based on this attitude, software engineering—like computer science as a whole—aimed for generality in its methods, techniques, and processes and focused almost exclusively on business and embedded software.

• Because of this ideal of generality, the question of how specifically computational scientists should develop their software in a well-engineered way would probably have perplexed a software engineer, whose answer might have been:
  • ‘Well, just like any other application software.’”
Mutual Ignorance: Computational Science

The **Productivity Crisis** in Computational Science

- As early scientific software was developed by small teams of scientists primarily for their own research, **modularity**, **maintainability**, and team coordination could often be neglected without a large impact.

The **Credibility Crisis** in Computational Science:

- **Climategate.** The scandal erupted after hackers leaked the email correspondence of scientists just before the 2009 United Nations Climate Change Conference.

- While the accusations that data was forged for this conference turned out to be unfounded, the emails uncovered a **lack of programming skills** among the researchers and exposed to a large public audience the widely applied practice in climate science of **not releasing simulation code and data** together with corresponding publications [Merali 2010].

- This in itself was, of course, enough to **undermine the scientists’ work**, as the predictive capabilities of simulations are only as good as their code quality and their code was not even available for peer review—not to mention public review [Fuller and Millett 2011].
Modular Scientific Code

Highlights

• Ch4-project is a fluid dynamics code used in academia for the study of fundamental problems in fluid mechanics.

• It has contributed to the understanding of global scaling laws in non-ideal turbulent thermal convection.

• It has been used for the characterisation of statistical properties of bubbles and particles in developed turbulence.

• It is currently employed for a variety for research projects on inertial particle dynamics and convective melting.

• Its modular code structure allows for a low learning threshold and to easily implement new features.
Modular Scientific Code

[Calzavarini 2019]:
“A dream for principal investigators in this field is to not have to deal with different (and soon mutually incompatible) code versions for each project and junior researcher in his/her own group.

• In this respect an object-oriented modular code structure would be the ideal one,
  • but this makes the code less prone to modifications by the less experienced users.

• The choice made here is to rely on a systematic use of C language preprocessing directives and on a hierarchical naming convention in order to configure the desired simulation setting in a module-like fashion at compiling time.”
So, SE for Computational Science

[Johanson & Hasselbring 2018]:

• Among the methods and techniques that software engineering can offer to computational science are
  • testing without test oracles,
  • modular software architectures, and
  • model-driven software engineering with domain-specific languages.

• This way, computational science may achieve maintainable, long-living software [Goltz et al., 2015; Reussner et al. 2019],
  • in particular for community software.
• Programming / Coding
  • Fortran, C++, Python, R, etc
  • Using compilers, interpreters, editors, etc
• Using version control (git etc)
• Team coordination (GitHub, Gitlab, etc)
• Continuous integration (Jenkins, etc)
• https://software-carpentry.org/
Agenda

1. Research Software
   • Characteristics
   • Mutual ignorance

2. Research Software Publishing

3. Open Science
   • For Computational Science
   • For Computer Science / Software Engineering: Artifact Evaluation

4. FAIR and Open Research Software

5. Summary & Outlook
Open Source Research Software

Wilhelm Hasselbring, Kiel University
Leslie Carr, University of Southampton
Simon Hettrick, Software Sustainability Institute and University of Southampton
Heather Packer and Thanassis Tiropoulos, University of Southampton

For good scientific practice, research software should be open source. It should be both archived for reproducibility and actively maintained for reusability.

Software Engineering for Sustainable Research Software
Research Software Publishing

• Relating research software to research publications:

Research software is identified either by
• research publications that cite software repositories or
• software repositories that cite research publications.
Research Software Publishing Practices

(a) Research areas of publications cited from Github repositories

(b) Research areas of ACM computer science publications citing GitHub repositories

[Hasselbring et al. 2020a]
Research Software Publishing Practices

(c) Research areas of arXiv publications citing GitHub repositories
(d) Computer science publications in arXiv from Figure 2c refined into sub-areas

[Hasselbring et al. 2020a]
Covered Research Areas

A first interesting observation is that our three data sets cover quite different research areas:

• The GitHub research software set is drawn mainly from the **computational sciences**, particularly the life sciences.

• The ACM research software set is dominated by **software engineering**, information systems, social and professional topics and human-centered computing.

• The arXiv research software set is dominated by computer science topics,
  • which is mainly composed of **AI topics** (computer vision, machine learning, computational linguistics, Figure 2d).
Sustainability of Research Software

- Research software publishing practices in computer science and in computational science show significant differences:
  - computational science emphasizes **reproducibility**,
  - computer science emphasizes **reuse**.

Lifespan of Github repositories cited in computer science publications

Lifespan of Github repositories cited in computational science publications

[Hasselbring et al. 2020a]
Sustainability of Research Software

• The **computer science** software repositories’ lifespan is distributed with a median of 5 years.
  • Our hypothesis is that in computer science research, often commercial open-source software frameworks are employed.
  • These software frameworks are maintained over long times by employees of the associated companies.

• The **computational science** software repositories’ lifespan has a distribution with a median lifespan of 15 days. A third of these repositories are live for less than 1 day.
  • Our hypothesis is that in computational science research, often the research software is only published when the corresponding paper has been published. The software is then not further maintained at GitHub, but at some private place as before (if it is further maintained at all).

• The arXiv repositories are somewhere in between with a median of 8 months lifespan. Furthermore, 75% of the arXiv repositories are live.
  • Our hypothesis is that the attitude of publishing as early as possible in parts of the artificial intelligence community also motivates the researchers to develop their research software openly from the start of research projects.
Categories of Research Software

We observe different categories and relationships between research publications and research software:

- Software as an output of research, **collaboratively constructed and maintained through an active open source community**.
- Software as an output of research, **privately developed but published openly and abandoned after publication**.
- Software itself as an **object** of study or analysis.
- Software that then leads to a **fork** (in GitHub) that is independently developed as a research output and published openly (if successful, it may be fed back into the original project via pull requests).
- Software used as a **tool or framework** to do the research.

Besides these relationships, software is cited as related work, background, or example.
Agenda

1. Research Software
   • Characteristics
   • Mutual ignorance

2. Research Software Publishing

3. Open Science
   • For Computational Science
   • For Computer Science / Software Engineering: Artifact Evaluation

4. FAIR and Open Research Software

5. Summary & Outlook
Reproducible Research in Computational Science

“Replication is the ultimate standard by which scientific claims are judged.”

[Peng 2011]
Agenda

1. Research Software
   • Characteristics
   • Mutual ignorance

2. Research Software Publishing

3. Open Science
   • For Computational Science
   • For Computer Science / Software Engineering: Artifact Evaluation

4. FAIR and Open Research Software

5. Summary & Outlook
Viewpoint
The Real Software Crisis: Repeatability as a Core Value
Sharing experiences running artifact evaluation committees for five major conferences.

“Science advances faster when we can build on existing results, and when new ideas can easily be measured against the state of the art.”

Repeatability, replicability & reproducibility
Several ACM SIGMOD, SIGPLAN, and SIGSOFT conferences have initiated artifact evaluation processes.
Example Experimental “Reproducibility Data” in Software Engineering

A Comparison of the Influence of Different Multi-Core Processors on the Runtime Overhead for Application-Level Monitoring

Jan Waller¹ and Wilhelm Hasselbring¹,²

¹ Software Engineering Group, Christian-Albrechts-University Kiel, Germany
² SPEC Research Group, Steering Committee, Gainesville, VA, USA

From Reproducibility Problems to Improvements: A journey

Holger Eichelberger, Aike Sass, Klaus Schmid

eichelberger, schmid)@se.uni-hildesheim.de, sass@uni-hildesheim.de

University of Hildesheim, Software Systems Engineering, 31141 Hildesheim, Germany

[Waller and Hasselbring 2012] [Eichelberger et al. 2016]
Report: Artifact Evaluation Track

Wilhelm (Willi) Hasselbring  
Kiel University, Germany

Petr Tuma
Charles University, Czech Republic
Some numbers for ICPE 2018

• 59 submitted full research papers
• 14 accepted full research papers
• 6 submitted artifacts
• 2 accepted artifacts, evaluated as functional
• 0 accepted artifacts, evaluated as reusable

• It seems that repeatability and reproducibility of performance research results brings specific challenges
• However, it is also of particular importance to this field
• Is it worth making the effort?
“Science advances faster when we can build on existing results, and when new ideas can easily be measured against the state of the art.”

[Krishnamurthi & Vitek 2015]
Impact of Artifact Evaluation

![Graph showing average citation counts of AE and non-AE papers for conferences that used AE in 2013 to 2016 (conferences: VISSOFT, PPoPP, POPL, PLDI, PACT, OOPSLA, ISSTA, FSE, ECRTS, ECOOP, CGO, CAV).]

[Childers & Chrysanthis 2017]
Agenda

1. Research Software
   • Characteristics
   • Mutual ignorance
2. Research Software Publishing
3. Open Science
   • For Computational Science
   • For Computer Science / Software Engineering: Artifact Evaluation
4. FAIR and Open Research Software
5. Summary & Outlook
Open Science for Research Software

1. Findable
   • Software citation
   • Domain-specific Metadata

2. Accessible
   • GitHub etc. for use and involvement
   • Zenodo etc. for archival

3. Interoperable
   • Obey to standards.
   • Proper interfaces in modular software

4. Reusable
   • Artifact evaluations support this.
   • Domain-specific languages may help with comprehensibility
   • Modular software architecture allow for reusing parts
## Recommendations for FAIR Research Software [Hasselbring et al. 2020b]

<table>
<thead>
<tr>
<th>FAIR Principle</th>
<th>Recommended Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Findability</strong></td>
<td>Provide software metadata to improve software retrieval</td>
</tr>
<tr>
<td></td>
<td>Use software citation to allow for proper reference and reward</td>
</tr>
<tr>
<td></td>
<td>Employ research software observatories which may serve as retrieval service</td>
</tr>
<tr>
<td><strong>Accessibility</strong></td>
<td>Use software development platforms such as GitHub for code cloning</td>
</tr>
<tr>
<td></td>
<td>Use repositories such as Zenodo to access archived software versions</td>
</tr>
<tr>
<td></td>
<td>Use research software observatories as dedicated repository services</td>
</tr>
<tr>
<td><strong>Interoperability</strong></td>
<td>Provide proper interface definitions in modular software architectures</td>
</tr>
<tr>
<td></td>
<td>Conform to established software standards</td>
</tr>
<tr>
<td></td>
<td>Use software virtualization techniques for portability</td>
</tr>
<tr>
<td></td>
<td>Participate in artifact evaluation processes to evaluate interoperability</td>
</tr>
<tr>
<td><strong>Reusability</strong></td>
<td>Use software development platforms such as GitHub for active involvement</td>
</tr>
<tr>
<td></td>
<td>Build modular software architectures to allow for reusing parts of research software</td>
</tr>
<tr>
<td></td>
<td>Use domain-specific languages for comprehensibility and modularity of research software</td>
</tr>
<tr>
<td></td>
<td>Follow good software engineering practices to achieve high software quality</td>
</tr>
<tr>
<td></td>
<td>Use software virtualization techniques such as Docker to support reusability across platforms</td>
</tr>
<tr>
<td></td>
<td>Use software-as-a-service platforms such as BinderHub for immediate execution</td>
</tr>
<tr>
<td></td>
<td>Use research software observatories for online analytics</td>
</tr>
<tr>
<td></td>
<td>Participate in artifact evaluation processes to evaluate reusability</td>
</tr>
</tbody>
</table>
Modularization of Earth-system simulation software

How to
• improve maintainability, stability, reusability, reproducibility, ... ?
• enable scalable execution in the Cloud?
• parallelize for high performance and exascale computing?
• test for higher quality?
• achieve higher flexibility?
Live Trace Visualization Tool \textit{ExplorViz}

• Program- and system comprehension for software engineers
• Started as a Ph.D project in 2012
• Open Source from the beginning (Apache License, Version 2.0)
• Continuously extended over the years
• \url{https://ExplorViz.dev}
\url{https://github.com/ExplorViz}
• See also Kieker [Hasselbring et al. 2020d]
\url{https://www.performance-symposium.org/2022/}
3D Application Visualization with ExploreViz

A: opened and pinned
B: pinged
C: highlighted
Summary & Outlook

• On the basis of an examination of the historical development of the relationship between software engineering and computational science (the past),
  • we identified key characteristics of scientific software development (the present).

• We examined attempts to bridge the gap in order to reveal the shortcomings of existing solutions and indicate further research directions (the possible future),
  • such as the use of domain-specific software engineering methods (OceanDSL project).

• Modularity is essential for maintainability, scalability and agility
  • also for reusability
  • also for testability

• Open Science also for Computer Science / Software Engineering research itself
  • “Eat your own dog food”
  • Follow the FAIR principles and publish research software open source
deRSE23 - Conference for Research Software Engineering in Germany
20-21 Feb 2023 Paderborn (Germany)

https://de-rse23.sciencesconf.org/

**Contribution Deadline:** Nov. 21st 2022

Associated to:

https://se-2023.gi.de/
Bionotes

Wilhelm Hasselbring
Christian-Albrechts-Universität zu Kiel,
D-24098 Kiel, Germany
hasselbring@email.uni-kiel.de

Prof. Dr. Wilhelm Hasselbring is a full professor of software engineering in the Department of Computer Science at Kiel University.

Heather Packer
University of Southampton, S017 1TW
Southampton, UK
hp3@ecs.soton.ac.uk

Dr. Heather Packer is a New Frontier fellow in the Department of Electronics and Computer Science at the University of Southampton.

Leslie Carr
University of Southampton, S017 1TW
Southampton, UK
lac@ecs.soton.ac.uk

Prof. Dr. Leslie Carr is a full professor of web science in the Department of Electronics and Computer Science at the University of Southampton.

Thanassis Tiropanis
University of Southampton, S017 1TW
Southampton, UK
t.tiropanis@southampton.ac.uk

Prof. Dr. Thanassis Tiropanis is an associate professor in the Department of Electronics and Computer Science at the University of Southampton.

Simon Hettrick
University of Southampton, S017 1TW
Southampton, UK
sjh@ecs.soton.ac.uk

Prof. Dr. Simon Hettrick is deputy director of UK’s Software Sustainability Institute and a full professor in the Department of Electronics and Computer Science at the University of Southampton.
References


References


