Research Software Engineering

Wilhelm (Willi) Hasselbring

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

July 2nd, 2020, Kiel
Agenda

1. Research Software
2. Research Software Engineering
   – Testing Research Software
   – Modular commercial software
   – Modular research software
   – Domain-specific software engineering
3. Research Software Publishing
4. Summary & Outlook
• **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.

• **Computer science research** develops research software for prototypes, simulations, experiments, analytics, etc.
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

![Flowchart showing the process of research software development]

[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 / ignorance of most modern software engineering methods and tools
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.’ ”
To the Rescue: Software Carpentry

• 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)
• Test automation, static analysis, etc

https://software-carpentry.org/
More 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.
Agenda

1. Research Software
2. Research Software Engineering
   – Testing Research Software
   – Modular commercial software
   – Modular research software
   – Domain-specific software engineering
3. Research Software Publishing
4. Summary & Outlook
Testing Research Software: Challenges

• One chief challenge is the lack of the test oracle.
• Reasons (see also [Peng et al. 2020])
  – Most research software is written to find answers that are previously unknown.
  – It is difficult to determine the correct output for software written to test scientific theory that involves complex calculations,
    • e.g., the large, complex simulations are developed to understand climate change.
    It is difficult to determine the correct output for software written to test scientific theory that involves complex calculations, e.g., the large, complex simulations are developed to understand climate change.
  – Due to the inherent uncertainties in models, some scientific programs do not give a single correct answer for a given set of inputs.
  – Requirements are unclear or uncertain up-front due to the exploratory nature of the software.
  – Choosing suitable tolerances for an oracle when testing numerical programs is difficult due to the involvement of complex floating point computations.
Testing Research Software: Solutions

Approaches to solve the oracle problem [Peng et al. 2020]:

• With **pseudo oracles** an independently developed program that fulfills the same specification as the program under test.

• A **statistical oracle** is based on statistical tests, it does not check the actual output but only some statistical characteristics of it.

• With **metamorphic testing** correctness is not determined by checking an individual concrete output, but by applying a transformation to a test input and observing how the program output “morphs” into a different one as a result.

**Modular software architectures** with proper test interfaces may support testability [Garousi et al. 2019] :

• Small software components with clear responsibilities and test interfaces alleviate the oracle problem.
Agenda

1. Research Software
2. Research Software Engineering
   - Testing Research Software
   - Modular commercial software
   - Modular research software
   - Domain-specific software engineering
3. Research Software Publishing
4. Summary & Outlook
Modular Internet Commercial Software

Example: otto.de

Modular Internet Commercial Software

Example: otto.de

Scalability, Agility and Reliability [Hasselbring & Steinacker 2017]
1. Research Software

2. Research Software Engineering
   - Testing Research Software
   - Modular commercial software
   - Modular research software
   - Domain-specific software engineering

3. Research Software Publishing

4. Summary & Outlook
Publishing Ocean Observation Data & Analytics

- Paper: http://dx.doi.org/10.1016/j.ecoinf.2017.02.007
- Code: https://github.com/cau-se/oceantea/
- Software service with data: http://oceantea.uni-kiel.de/

---

Modeling Polyp Activity of *Paragorgia arborea* Using Supervised Learning

Arne Johanson, a Sascha Flögel, Wolf-Christian Dullo, Peter Linke, Wilhelm Hasselbring

a Software Engineering Group, Kiel University, Germany
b GEOMAR Helmholtz Centre for Ocean Research, Kiel, Germany

correspondence: peter.linke@uni-kiel.de

**Abstract** — While the distribution patterns of cold-water corals, such as *Paragorgia arborea*, have received increasing attention in recent studies, little is known about their in situ activity patterns. In this paper, we examine polyp activity in *P. arborea* using machine learning techniques to analyze high-resolution time series data and photographs obtained from an autonomous lander cluster deployed in the Sjøfjorden, Norway. An interactive illustration of the models derived in this paper is provided online as supplementary material.

We find that the best predictor of the degree of extension of the coral polyps is current temperature.

---

http://oceantea.uni-kiel.de/

---

[Johanson et al. 2017b]
Modular Scientific Software

OceanTEA: Microservice-based Architecture

OceanTEA: [Johanson et al. 2016a, Johanson et al. 2017b]
Using Microservices for Legacy Software Modernization

Holger Knoche and Wilhelm Hasselbring, Kiel University

Microservices promise high maintainability, making them an interesting option for software modernization. This article presents a migration process to decompose an application into microservices, and presents experiences from applying this process in a legacy modernization project. //

reduce coordination effort and improve team productivity.

It is therefore not surprising that companies are considering microservice adoption as a viable option for modernizing their existing software assets. Although some companies have succeeded in a complete rewrite of their applications, incremental approaches are commonly preferred that gradually decompose the existing application into microservices. Other approaches to modernization—e.g., restructuring and refactoring of existing legacy applications—are also valid options. However, decomposing a large, complex application is far from trivial. Even seemingly easy questions like "Where should I start?" or "What services do I need?" can actually be very hard to answer.

In this article, we present a process to modernize a large existing software solution using microservice principles, and report on experiences from implementing it in an ongoing industrial modernization project. We particularly focus on the process of actually decomposing the

[Knoche & Hasselbring 2018]
Live Trace Visualization Tool

- 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
- [Fittkau et al. 2013, 2015a-d, 2017; Krause et al. 2018; Zirkelbach et al. 2019]

https://www.explorviz.net
https://github.com/ExplorViz
Migrating Legacy Research Software toward Microservices
Some VR Extensions

[HMD Visualization]  [Leap Motion Sensor]

[HMD Visualization]  [Vive Controllers]

Migrating Legacy Research Software toward Microservices
Legacy Layered Architecture

Monitored Server
- Application
- Monitoring

Server
- Analysis
- Visualization
- Feature
- H2
- Filesystem

TCP

Client

HTTP
Collaborative Development

With ExplorViz Legacy

(Student) Collaboration > New Feature > New Git Branch > No extension mechanism

Merging Problematic Single Codebase

(icons from www.flaticon.com)
More details in [Zirkelbach et al. 2019] (Best paper award)
Legacy Layered Architecture

Monitored Server

- Application
- Monitoring

Server

- GWT
- Analysis
- Visualization
- Feature
- H2
- Filesystem

Client

TCP

HTTP
Lazy Refactoring, guiding principles [Adorf 2019]:

- **Modularity**, the core design principle naturally leads to the development of tools with clearly delimited scopes and well-defined Application Programming Interfaces (APIs).
- **Public release** of software source code. The need to validate scientific results and the need to integrate disparate software tools make the ability to access source code valuable.
Lazy Refactoring

- **Initial** development should always lead to single-use code,
  - but this code is **refactored** into a reusable solution as soon as two further uses for it are found.
- This method requires that some **quality controls** be imposed on even initial prototypes to ensure that refactoring remains possible.
- The lazy refactoring approach is designed to **balance sustainably** improving the scientific software landscape with making immediate **scientific progress**.
- It does this by advocating for individual researchers to
  - evaluate existing software for its **reuse potential** prior to any code development,
  - **adapt existing code** bases for the problem at hand,
  - **refactor existing code** bases into proper packages whenever there are more than two use cases, and
  - develop rapidly evolving **prototype** code strictly focused on solving the problem at hand in all other cases.
A Comment on Jupyter Notebooks

• Jupyter is a free, open-source, interactive web tool, which researchers can use to combine (literate programming)
  – software code,
  – computational output,
  – explanatory text and
  – multimedia resources in a single document.
• You may also use them in lectures and labs.
• Various cloud services such as BinderHub and Code Ocean emerge to provide Jupyter notebooks as a service (SaaS).
• However, Jupyter notebooks also encourage poor coding practice [Perkel 2018],
  – by making it difficult to organize code into reusable modules and
  – develop tests to ensure the code is working properly.
• Notebooks do require discipline from programmers!
  – With great power comes great responsibility.
Migrating Computational Science Models?

The software architecture of climate models
[Alexander & Easterbrook 2015]

Figure 1. Architecture diagram for CESM1-BGC.

Figure 2. Architecture diagram for GFDL-ESM2M.

Figure 3. Architecture diagram for GISS-E2-R-TCADI.

Figure 4. Architecture diagram for UVic ESCM 2.9.
Agenda

1. Research Software
2. Research Software Engineering
   – Testing Research Software
   – Modular commercial software
   – Modular research software
   – Domain-specific software engineering
3. Research Software Publishing
4. Summary & Outlook
The Sprat Approach: Hierarchies of DSLs

[Johanson & Hasselbring 2014a,b, 2016b]
Evaluation of the Sprat

- Controlled experiments with domain scientists [Johanson & Hasselbring 2017]
- Expert interviews and benchmarks [Johanson et al. 2016b]
- The Sprat Marine Ecosystem Model:
  Original scientific contributions to Ecological Modeling [Johanson et al. 2017a]
Sprat: Summary

The **Sprat Approach**: Model-driven software engineering for computational science

- Concept of DSL Hierarchies
- DSLs for Marine Ecosystem Modeling
- Empirical Evaluation of the Sprat Approach

Available online:

- DSL implementations
- Sprat Model source code
- Experimental data and analysis scripts

http://dx.doi.org/10.5281/zenodo.61373

http://www.sprat.uni-kiel.de/
Outlook: OceanDSL

• OceanDSL – Domain-Specific Languages for Ocean Modeling and Simulation
• New project funded by DFG (German Science Foundation)
• Provide an infrastructure for building **modular** ocean modeling and simulation software
Agenda

1. Research Software
   - Characteristics of the domain computational science
   - Mutual ignorance

2. Research Software Engineering
   - Modular commercial software
   - Modular research software
   - Domain-specific software engineering

3. Research Software Publishing

4. Summary & Outlook
To relate research software to research publications, research software is identified either by

• research publications that cite software repositories or
• software repositories that cite research publications.

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

Research areas of publications cited from GitHub repositories

ACM computer science publications citing GitHub repositories

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

arXiv publications citing GitHub repositories

[Hasselbring et al. 2020b]

Computer science publications in arXiv, extracted from the left
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).

[Hasselbring et al. 2020b]
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 citing computational science publications

[Hasselbring et al. 2020b]
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.

[Hasselbring et al. 2020b]
Summary

• Modularity is essential for maintainability, scalability and agility
  – but also for **reusability**
  – **Reproducibility** is essential for good scientific practice.
  – Lazy **refactoring** may help.

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

• However, **domain-specific** software engineering approaches are required for computational science
  – Implausible to modernize legacy scientific code

• **Open Science** also for Computer Science / Software Engineering research itself
  – “Eat your own dog food”
  – Follow the FAIR principles [Hasselbring et al. 2020a]
Outlook

Dear MarDatas,

• if you are only interested in getting a Ph.D.,
  – this talk was not really of interest to you, sorry.

If you are (also) interested in scientific impact, publish with open access

  – research papers,
  – research data,
  – research software, and
  – do networking with related stakeholders.
References

Computing in Science & Engineering, 21(2), March 2019, S. 66–79. DOI 10.1109/MCSE.2018.2882355


[Calzavarini 2019] E. Calzavarini, „Eulerian–Lagrangian Fluid Dynamics Platform: The Ch4-Project“. Software Impacts, 1,


Software Landscapes: The ExplorViz Approach“, In: 1st IEEE International Working Conference on Software
Visualization (VISSOFT 2013).


Comprehension: A Controlled Experiment”. In: 3rd IEEE Working Conference on Software Visualization, 2015.

Working Conference on Software Visualization, September 2015, Bremen, Germany.

[Fittkau et al. 2015d] F. Fittkau, S. Finke, W. Hasselbring, J. Waller: “Comparing Trace Visualizations for Program
Comprehension through Controlled Experiments”, In: 23rd IEEE International Conference on Program

comprehension with ExplorViz”, In: Information and Software Technology. DOI 10.1016/j.infsof.2016.07.004

April 2019, S. 35–64. DOI 10.1016/j.infsof.2018.12.003
References


References


