SE in den Naturwissenschaften:
Research Software Engineering and Publishing

Wilhelm (Willi) Hasselbring

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

SE 2020, February 25th, 2020, Innsbruck
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
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.
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
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
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.’ ”
Johanson & Hasselbring 2018: Among the methods and techniques that software engineering can offer to computational science are:

- specific requirements engineering techniques [Thew et al. 2009],
- testing without test oracles [Kanewala and Bieman 2014],
- model-driven software engineering with domain-specific languages, and
- modular software architectures,

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
   – 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
Modular Internet Commercial Software

Example: otto.de

Modular Internet Commercial Software

Example: otto.de

Scalability, Agility and Reliability [Hasselbring & Steinacker 2017]
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
Eulerian-Lagrangian fluid dynamics platform: The ch4-project

Enrico Calzavarini

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.
“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.”
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**

**Author(s):**
- Arne Johanson,
- Sascha Flögel,
- Wolf-Christian Dullo,
- Peter Linke,
- Wilhelm Hasselbring

- **Software Engineering Group, Kiel University, Germany**
- **GEOMAR Helmholtz Centre for Ocean Research, Kiel, Germany**

**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 Storflaten, Norway. An interactive illustration of the models derived in this paper is provided online as supplementary material.

**Key Findings:**
- We find that the best predictor of the degree of extension of the coral polyps is

**Graphs and Tables:**
- Various graphs showing time series data and model predictions.
- Tables indicating model parameters and coefficients.

**References:**
- Johanson et al. 2017b

**[Johanson et al. 2017b]**

**http://oceantea.uni-kiel.de/**
OceanTEA: Microservice-based Architecture

OceanTEA: [Johanson et al. 2016a, Johanson et al. 2017b]
Migrating toward Microservices

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,\(^2\) incremental approaches are commonly preferred that gradually decompose the existing application into microservices.\(^3\) Other approaches to modernization—e.g., restructuring and refactoring of existing legacy applications—are also valid options.\(^4\) 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 application 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
Some VR Extensions

[HMD Visualization]  [Leap Motion Sensor]

[HMD Visualization]  [Vive Controllers]

User 1

VR Controller

Controller Ray

User 2

VR Controller

Controller Ray

Application-Level

Landscape-Level
Legacy Layered Architecture

Monitored Server
- Application
- Monitoring

Server
- Analysis
- Visualization
- Feature
- H2
- Filesystem

Client

TCP

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)
Monitored Server

Application

Monitoring

TCP

Server

GWT

Analysis

Visualization

Feature

H2

Filesystem

HTTP

Client
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
   – 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
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]
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
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.

[Hasselbring et al. 2019]
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. 2019]
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. 2019]
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).

[Hasselbring et al. 2019]
• Research software publishing practices in computer science and in computational science show significant differences:
  – computational science emphasizes **reproducibility**,  
  – computer science emphasizes **reuse**.
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. 2019]
**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.**
  
  - For instance, Caffe is a deep learning framework that has been developed as research software. It has meanwhile been maintained at GitHub for five years with a large user community and commercial forks.

- **Software as an output of research, privately developed but published openly and abandoned after publication.**
  
  - For instance, the software for the genetics study by Hough et al. has been published at GitHub in 2014 in parallel with the paper. This repository is now five years old with a lifespan of one day.

- **Software itself as an object of study or analysis.**
  
  - For instance, Costa et al. studied the performance of the Google Guava core libraries for Java. They did not develop or adapt this software.

- **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).**
  
  - For instance, Bosagh Zadeh et al. extended the Apache Spark analytics engine as a GitHub fork in their research and managed to merge their software extensions back into the master software repository.

- **Software used as a tool or framework to do the research.**
  
  - For instance, O’Donovan et al. used the three.js Javascript 3D library to study 3D modeling approaches.

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

[Hasselbring et al. 2019]
Threads to Validity

We had to make some assumptions for this initial study:

• To make our analysis **tractable** and **repeatable**, we assume that a research publication refers to some GitHub repository for the related research software or that somewhere in a GitHub repository a publication identifier is available.

• Research software is not always cited with a link to a GitHub repository.
  – It could also be published, for instance, in Bitbucket or GitLab repositories.
  – Alternative citations may refer to papers, manuals or books that introduce the software.
  – Our initial analysis does not cover such additional citation links.

• We are well aware that these assumptions restrict the coverage of our analysis, but even with this limited coverage, we already observed interesting differences in software publication behaviors in different research domains.

• To allow for a more comprehensive study of the relationships between research software and research publications, specific **Observatories** for research software could provide appropriate citation links and citation graphs.

[Hasselbring et al. 2019]
Summary

• Modularity is essential for maintainability, scalability and agility
  – but also for **reusability**
  – **Reproducibility** is essential for good scientific practice.
  – So, microservices could be a beneficial architectural style for research software, too.

• 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. 2020]
What is specific for SE in Science?

- **Functional Requirements are not known up front**
  - No requirements specification, which could be handed over from scientists to software engineers
  - No test oracles available
  - “Agile” development

- **Maintainability has low priority**
  - Reproducibility valued higher than reusability
  - Implausible to modernize legacy scientific code

- **Modularity to the rescue**
  - Essential for maintainability, scalability, reusability, reliability, testability, and agility

- **Task sharing unsuitable**
  - Domain-specific software engineering approaches for scientists
  - Software carpentry

[Slides: http://eprints.uni-kiel.de/49003/]
References


References


References


