Software Engineering for Computational Science: Tests, Modules, Domain-Specific Languages, Flows

Wilhelm (Willi) Hasselbring

*Software Engineering*

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

CRC 1404 FONDA, April 27th, 2021
Agenda

1. Research Software
2. Research Software Engineering
   – Automated testing
   – Modular Software
     • Modular commercial software
     • Modular research software
   – Domain-specific software engineering
   – Flow-based programming
3. 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

![Diagram showing the process of developing software for research](image)
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
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]
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.’”
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)]

https://software-carpentry.org/
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.

Software Engineering for Computational Science:

Past, Present, Future

Despite the increasing importance of in silico experiments to the scientific discovery process, state-of-the-art software engineering practices are rarely adopted in computational science. To understand the underlying causes for this situation and to identify ways to improve it, we conducted a literature survey on software engineering practices in computational science. We identified 13 recurring key characteristics of scientific software development that are the result of the nature of scientific challenges, the limitations of computers, and the cultural environment of scientific software development. Our findings allow us to point out shortcomings of existing approaches for bridging the gap between software engineering and computational science and to provide an outlook on promising research directions that could contribute to improving the current situation.
Agenda

1. Research Software

2. Research Software Engineering
   - Automated testing
   - Modular Software
     • Modular commercial software
     • Modular research software
   - Domain-specific software engineering
   - Flow-based programming

3. Summary & Outlook
Testing the Untestable: Test Oracles?

Stimulus and observations:

- $S$ is anything that can change the observable behavior of the SUT $f$;
- $R$ is anything that can be observed about the system’s behavior;
- $I$ includes $f$’s explicit inputs;
- $O$ is its explicit outputs;
- everything not in $S \cup R$ neither affects nor is affected by $f$.

[Scientific calculations]

[Artificial intelligence]

[Simulation and modelling]

[Kanewala and Bieman 2014]
Metamorphic Testing

• The nature of research software is exploratory.
• Output is usually unknown and cost-intensive to compute.
• Hence it is challenging to validate using conventional testing methodology.
• Metamorphic Testing provides an approach for testing software without test oracles.
  – Validating software by comparing outputs of multiple runs with varying (morphed) input data.
  – The central element of metamorphic testing is the metamorphic relation.
    • The input data is morphed based on this property.
  – If the output is in accordance of the applied morphing to the input data, the test is asserted.

[Segura et al. 2020]
Metamorphic testing may be defined as

\[ f(g(\tilde{x})) = h(f(\tilde{x})) \]

- function under test \( f : X \rightarrow Y \)

Our Goal: To generate metamorphic test cases and metamorphic relations automatically via machine learning for verifying Ocean System Model applications [Hiremath et al. 2021]
1. Research Software

2. Research Software Engineering
   - Automated testing
   - Modular Software
     • Modular commercial software
     • Modular research software
   - Domain-specific software engineering
   - Flow-based programming

3. Summary & Outlook
Modular Commercial Software

Example: otto.de

Modular Commercial Software

Example: otto.de

Scalability, Agility and Reliability [Hasselbring & Steinacker 2017]
Agenda

1. Research Software
2. Research Software Engineering
   – Automated testing
   – Modular Software
     • Modular commercial software
     • Modular research software
   – Domain-specific software engineering
   – Flow-based programming
3. 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.
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.”
Publishing Ocean Observation Data & Analytics

- Paper: [Johanson et al. 2017b]
- Code: https://github.com/cau-se/oceantea/
- Software service with data: https://oceantea.uni-kiel.de/

Modeling Polyp Activity of Paragorgia arborea Using Supervised Learning

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

Software Engineering Group, Kiel University
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 analyze 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 Stjernsund, 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...
Modular Scientific Software

OceanTEA: Microservice-based Architecture

OceanTEA: [Johanson et al. 2016a]
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 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, Krause et al. 2020]
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

https://www.explorviz.net
https://github.com/ExplorViz
3D Application Visualization

ExplorViz Visualization Discovery

© 2013 - 2019 by the ExplorViz project

Modular Software
Some VR Extensions

[HMD Visualization]  [Leap Motion Sensor]

[HMD Visualization]  [Vive Controllers]

User 1  User 2

VR Controller  VR Controller

Controller Ray  Controller Ray

Landscape-Level

Application-Level

Modular Software
ExplorViz

Legacy Layered Architecture

Monitored Server

Application
Monitoring

TCP

Server

GWT

Analysis
Visualization
Feature
H2

Filesystem

HTTP

Client

Modular Software
New Modular Architecture

More details in [Zirkelbach et al. 2019, 2020]
Please go to https://menti.com
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
   - Automated testing
   - Modular Software
     • Modular commercial software
     • Modular research software
   - Domain-specific software engineering
   - Flow-based programming
3. 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
- Provide an infrastructure for building **modular and testable** ocean modeling and simulation software
- Initial focus on configuration and parametrization DSLs [Jung et al. 2021]
1. Research Software
2. Research Software Engineering
   - Automated testing
   - Modular Software
     • Modular commercial software
     • Modular research software
   - Domain-specific software engineering
   - Flow-based programming
3. Summary & Outlook
Control-flow patterns UML activity diagram:
(a) Sequence; (b) Parallel Split; (c) Synchronisation; (d) Exclusive Choice; (e) Simple Merge;
(f) Multi Merge; (g) Arbitrary Cycle; (h) Multiple Instance with a prior design-time knowledge; (i) Multiple Instance with a prior run-time knowledge; and (j) Milestone.

[Butt and Fitch 2021]
Scientific Workflows

From the D-Grid project WISENT on e-Science for Energy Meteorology [Hasselbring et al. 2006]

From the control-flow patterns, only Parallel Split and Synchronisation (aka Fork/Join). No Exclusive Choices or Loops.
Data Analysis Workflows in FONDA

Ulf Leser @ GIBU 2021
Control Flow Versus Data Flow in Distributed Systems Integration: Revival of Flow-Based Programming for the Industrial Internet of Things

Wilhelm Hasselbring, Kiel University, 24118 Kiel, Germany
Maik Wojcieszak, CTO Wobe-Systems GmbH, 24145 Kiel, Germany
Schahram Dustdar, TU Wien, 1040 Vienna, Austria

[Hasselbring et al. 2021], see also https://www.industrial-devops.org/
Goals and measures for analyzing power consumption data in manufacturing enterprises

Sören Henning¹ - Wilhelm Hasselbring¹ - Heinz Burmester² - Armin Möbius³ - Maik Wojcieszak⁴

[Henning et al. 2021]
Developing Analysis Workflows in FONDA

• Like software in the 70ties!
  – No standardized architectural components
  – No established abstractions with common APIs
• Programs tightly tied to software infrastructure
• Low productivity – “Software crisis”

• FONDA’s overall goal

How can we increase human productivity in the creation, maintenance, and execution of DAWs for large-scale scientific data analysis?

How can we increase portability, adaptability, and dependability of DAWs and DAW infrastructures?

Ulf Leser @ GIBU 2021
• Modularity is essential for maintainability, scalability and agility
  – also for *reusability*
  – also for *testability*
  – So, *microservices* could be a beneficial architectural style for research software, too.

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

• When researching data analysis workflows in FONDA,
  – I suggest to emphasize data flow over control flow [Hasselbring et al. 2021]

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


References


References


References


