Microservices as Architectural Style for Research Software

Wilhelm (Willi) Hasselbring

*Software Engineering*

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

VSS, August 28\(^{th}\) 2019, Vienna
Software Engineering for Computational Science: Past, Present, Future

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.

3 Present: Establishment of Domain-Specific Architectures and Focus on Quality Attributes

As discussed in the previous section, the past emphasis was mainly on generic architectural styles such as pipe-and-filter architectures. However, the Domain-Specific Software Architecture (DSSA) engineering process was introduced early in the 1990s to promote a clear distinction between domain and application requirements [57]. A Domain-Specific Software Architecture consists of a domain model and a reference architecture. DSSA was promoted for domains such as avionics.

Meanwhile, various architectures are established for many domains and applications. Where total architectural solutions do not yet exist, partial ones certainly do in the form of catalogs of architectural patterns that help solve many problems and achieve various quality attributes. Various domain-specific architectures emerged, particularly from industrial practice. Examples are data warehouse architectures [60] for business analytics and, more recently, microservice architectures [46] for Internet services. Exemplary, we will take a closer look at recent microservice architectures in Sect. 3.1. Many present architecture approaches focus on quality requirements, which are discussed in Sect. 3.2.

3.1 Example: Microservice Architectures

4 Future: Proper Integration of Architecture Work into Agile Software Development

[Johanson & Hasselbring 2018]
Agenda

1. Research Software
2. Microservices in Commercial Software
3. Microservices in Research Software
4. Migrating Legacy Research Software toward Microservices
5. Domain-Specific SE with Microservices
6. Summary & Outlook
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.
SOFTWARE ENGINEERING

Report on a conference sponsored by the
NATO SCIENCE COMMITTEE
Garmisch, Germany, 7th to 11th October 1968

Chairman: Professor Dr. F. L. Bauer
Co-chairmen: Professor L. Bolliet, Dr. H. J. Helms
Editors: Peter Naur and Brian Randell

January 1969

HIGHLIGHTS

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.

http://homepages.cs.ncl.ac.uk/brian.randell/NATO/index.html

[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.”
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
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].

- Within the scientific community, Climategate initiated a debate about the **reproducibility of computational results**.
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 of 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.”
Research Software Publishing Practices

- 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 (ACM)

Lifespan of Github repositories cited in computational science publications (arXiv)

[Hasselbring et al. 2019]
So, SE for Computational Science

[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
2. Microservices in Commercial Software
3. Microservices in Research Software
4. Migrating Legacy Research Software toward Microservices
5. Domain-Specific SE with Microservices
6. Summary & Outlook
Modular Internet Business Software

Example: otto.de

Modular Internet Business Software

Example: otto.de

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

1. Research Software
2. Microservices in Commercial Software
3. Microservices in Research Software
4. Migrating Legacy Research Software toward Microservices
5. Domain-Specific SE with Microservices
6. 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,\textsuperscript{a} Sascha Flögel,\textsuperscript{b} Wolf-Christian Dullo,\textsuperscript{b} Peter Linke,\textsuperscript{a} Wilhelm Hasselbring\textsuperscript{a}

\textsuperscript{a}Software Engineering Group, Kiel University, Germany
\textsuperscript{b}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 Střemands, 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 cur...
Modular Scientific Software

OceanTEA: Microservice-based Architecture

OceanTEA: [Johanson et al. 2016a, Johanson et al. 2017b]
Microservices in Research Software
Agenda

1. Research Software
2. Microservices in Commercial Software
3. Microservices in Research Software
4. Migrating Legacy Research Software toward Microservices
5. Domain-Specific SE with Microservices
6. Summary & Outlook
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
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
3D Application Visualization

© 2013 - 2019 by the ExplorViz project
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
- GWT
- 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

Server

GWT

Analysis

Visualization

Feature

H2

Filesystem

Client

TCP

HTTP
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. Microservices in Commercial Software
3. Microservices in Research Software
4. Migrating Legacy Research Software toward Microservices
5. Domain-Specific SE with Microservices
6. Summary & Outlook
The Sprat Approach: Hierarchies of DSLs

Deployment Language Engineer

Deployment Specification
Ansible Playbook DSL

Simulation Specification
Sprat Ecosystem DSL

Simulation Specification
Sprat PDE DSL, embedded in C++

Ecosystem Model

Ecosystem Model
Sprat PDE DSL, embedded in C++

FEM PDE Solver
Sprat PDE DSL, embedded in C++

Ecosystem Language Engineer

PDE Language Engineer

Stock Assessment Scientist

Ecological Modeler

Numerical Mathematician

[Johanson & Hasselbring 2014a,b, 2016b]
The Sprat Ecosystem DSL

 Deployment Specification
    Ansible Playbook DSL

 Simulation Specification
    Sprat Ecosystem DSL

 Ecosystem Model
    Sprat PDE DSL, embedded in C++

 FEM PDE Solver
    Sprat PDE DSL, embedded in C++
Evaluation of the Sprat **Ecosystem** DSL

(a) Level of abstraction  
(b) Simplicity of use  
(c) Ease of comprehension

(d) Absence of technicalities  
(e) Maintainability of solutions

[Johanson & Hasselbring 2017]
The Sprat PDE DSL

Evaluation:

- Expert interviews with domain experts and professional DSL developers from industry
- Micro- and macro-benchmarks for performance evaluation [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
Summary & Outlook

- Microservices are established for Internet-scale commercial systems
- 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.
- However, domain-specific software engineering approaches are required for computational science,
  - to avoid mutual ignorance
- Open Science also for Computer Science / Software Engineering research itself
  - “Eat your own dog food”
  - Follow the FAIR principles [Hasselbring et al. 2019]
References


References


References


