Open Science for Computational Science and for Computer Science

Wilhelm (Willi) Hasselbring

WAIS Seminar, May 8th 2019, Southampton

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

CAU
Kiel University
Christian-Albrechts-Universität zu Kiel
1. Software Engineering vs. Computational Science

2. Software Engineering for Computational Science
   - Domain-specific Software Engineering
   - Modular Software Architectures

3. Open Science
   - For Computational Science
   - For Computer Science / Software Engineering

4. Summary & Outlook
The Origins of the Chasm

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

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

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.

[Randell 2018]
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 Scientific 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 Scientific 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.

• Scientific 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**.
Agenda

1. Software Engineering vs. Computational Science

2. **Software Engineering for Computational Science**
   - Domain-specific Software Engineering
   - Modular Software Architectures

3. Open Science
   - For Computational Science
   - For Computer Science / Software Engineering

4. Summary & Outlook

Journal paper: A. Johanson, W. Hasselbring:
“Software Engineering for Computational Science: Past, Present, Future”,
https://doi.org/10.1109/MCSE.2018.108162940
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/
Among the methods and techniques that software engineering can offer to computational science are:

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

This way, computational science may achieve **maintainable**, long-living software [Goltz et al., 2015], in particular for community software.
Agenda

1. Software Engineering vs. Computational Science
2. Software Engineering for Computational Science
   - Domain-specific Software Engineering
   - Modular Software Architectures
3. Open Science
   - For Computational Science
   - For Computer Science / Software Engineering
4. Summary & Outlook
The Sprat Approach: DSLs for Ecology

- Hierarchies of DSLs for ecology research [Johanson & Hasselbring 2014a,b, 2016b]
- Evaluation of the Sprat Ecosystem DSL via controlled experiments [Johanson & Hasselbring 2017]
- Evaluation of the Sprat PDE DSL with domain experts and professional DSL developers from industry, and benchmarks for performance evaluation [Johanson et al. 2016b]
The Sprat Marine Ecosystem Model

Original scientific contributions to Ecological Modeling [Johanson et al. 2017a]

Growth
- Controlled by Time/Temp.
- Controlled by Biomass Uptake

Metabolic Costs
- Resting Metabolic Rate
- Net Swimming Costs

Movement
- Active
  - Reactive
  - Predictive
- Passive
- Predation (Opportunistic)
  - Intake
  - Losses

Reproduction

Background Mortality

Fishing

Biogeochemical Ocean Model
- Zooplankton
- Currents
- Temperature

New project funded by DFG (German Science Foundation, € 571.400): OceanDSL – Domain-Specific Languages for Ocean Modeling and Simulation
Agenda

1. Software Engineering vs. Computational Science
2. Software Engineering for Computational Science
   – Domain-specific Software Engineering
   – Modular Software Architectures
3. Open Science
   – For Computational Science
   – For Computer Science / Software Engineering
4. Summary & Outlook
Modular Internet Business Software

Example: otto.de

Modular Internet Business Software

Example: otto.de

Scalability, Agility and Reliability [Hasselbring & Steinacker 2017]
Modular Scientific Software

OceanTEA: Microservice-based Architecture

OceanTEA: [Johanson et al. 2016a, Johanson et al. 2017b]
Generic Research Data Infrastructure

http://www.gerdi-project.de/ [Tavares de Sousa et al. 2018]
Agenda

1. Software Engineering vs. Computational Science
2. Software Engineering for Computational Science
   - Domain-specific Software Engineering
   - Modular Software Architectures
3. Open Science
   - For Computational Science
   - For Computer Science / Software Engineering
4. Summary & Outlook
Open Science: FAIR Principles

Implementing FAIR Data Infrastructures

Seminar November 18–21, 2018 – http://www.dagstuhl.de/18472

1. Findable
   - The first step in (re)using data and software is to find it.

2. Accessible
   - Once the user finds the required data and software, she/he needs to know how can they be accessed, possibly including authentication and authorization.

3. Interoperable
   - The data and software usually need to be integrated with other data and software.

4. Reusable
   - To achieve this, metadata, data and software should be well-described so that they can be replicated and/or combined in different settings.
1. Software Engineering vs. Computational Science
2. Software Engineering for Computational Science
   – Domain-specific Software Engineering
   – Modular Software Architectures
3. Open Science
   – For Computational Science
   – For Computer Science / Software Engineering
4. Summary & Outlook
“Replication is the ultimate standard by which scientific claims are judged.”

[Peng 2011]
Sprat: Open Science

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://www.sprat.uni-kiel.de/

http://dx.doi.org/10.5281/zenodo.61373
Publishing Ocean Observation Data & Analysis

- 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, 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 St. Kilda Archipelago, Scotland. 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

[Graphics of small graphs showing trends over time]

http://oceantea.uni-kiel.de/
1. Software Engineering vs. Computational Science

2. Software Engineering for Computational Science
   - Domain-specific Software Engineering
   - Modular Software Architectures

3. Open Science
   - For Computational Science
   - For Computer Science / Software Engineering

4. 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.
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

Overhead (median with quartiles) of ...
- Writing (W)
- Collecting (C)
- Instrumentation (I)
- Method Time (T)
(mean values with 95% confidence intervals)

From Reproducibility Problems to Improvements: A journey

Holger Eichelberger, Aike Sass, Klaus Schmid
{eichelberger, schmid}@esse.uni-hildesheim.de, sassi@uni-hildesheim.de
University of Hildesheim, Software Systems Engineering, 31141 Hildesheim, Germany

[Waller and Hasselbring 2012] [Eichelberger et al. 2016]
Example Empirical “Reproducibility Data” with Artifact Evaluation

Hierarchical Software Landscape Visualization for System Comprehension: A Controlled Experiment

Florian Fittkau, Alexander Krause, and Wilhelm Hasselbring
Software Engineering Group, Kiel University, Kiel, Germany
Email: {ffi, akr, wha}@informatik.uni-kiel.de

[GitHub] [Zenodo]

ExplorViz
Live trace visualization for large systems
http://www.explorviz.net

Experimental Data for: Exploring Software Cities through Virtual Reality
Fittkau, Florian; Krause, Alexander; Hasselbring, Wilhelm

Software visualizations, such as the software city metaphor, are usually displayed on 2D screens and controlled by means of a mouse and thus often do not take advantage of more natural interaction techniques. Virtual reality (VR) approaches aim to improve the user experience.

[Open Science for Computer Science] [Fittkau et al. 2013, 2015a-d, 2017]
Artifact Review and Badging:

A variety of research communities have embraced the goal of reproducibility in experimental science. [more information]

**Artifacts Evaluated – Functional**
The artifacts associated with the research are found to be documented, consistent, complete, exercisable, and include appropriate evidence of verification and validation.

**Artifacts Evaluated – Reusable**
The artifacts associated with the paper are of a quality that significantly exceeds minimal functionality.

**Artifacts Available**
Author-created artifacts relevant to this paper have been placed on a publically accessible archival repository.

**Results Replicated**
The main results of the paper have been obtained in a subsequent study by a person or team other than the authors, using, in part, artifacts provided by the author.

**Results Reproduced**
The main results of the paper have been independently obtained in a subsequent study by a person or team other than the authors, without the use of author-supplied artifacts.
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?

If I have seen further, it is by standing on the shoulders of giants.

- Isaac Newton

“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

![Bar chart 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).]

Fig. 1. 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]
Artifact Evaluation in Subdisciplines

• As mentioned, several ACM SIGMOD, SIGPLAN, and SIGSOFT conferences have initiated artifact evaluation processes.
  – SIGMOD calls it reproducibility evaluation and offers the “Most Reproducible Paper Award”
• ACM SIGIR, for instance, is currently introducing:
  – SIGIR Initiative to Implement ACM Artifact Review and Badging [Ferro & Kelly 2018]
  – CENTRE reproducibility evaluation of CLEF/NTCIR/TREC tasks http://www.centre-eval.org/
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
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.

Modularity is essential for maintainability, scalability and agility
   — but also for reusability
   — Reproducibility is essential for good scientific practice.

Open Science also for Computer Science / Software Engineering
   — “Eat your own dog food”
   — Follow the FAIR principles

We are recruiting (first deadline May 15th next week, more calls to come):
https://mardata.de
References


References


References


