Software Engineering for Computational Science

Journal paper: A. Johanson, W. Hasselbring:
“Software Engineering for Computational Science: Past, Present, Future”,
https://doi.org/10.1109/MCSE.2018.108162940

Arne Johanson
Wilhelm Hasselbring

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

1. Software Engineering vs. Computational Science
2. Software Engineering for Computational Science
4. Reproducibility
5. Modularity
6. Summary & Outlook
The Origins of the Chasm

Part 1: Software Engineering vs. Computational Science

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 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

• **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
4. Reproducibility
5. Modularity
6. Summary & Outlook
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.
1. Software Engineering vs. Computational Science
2. Software Engineering for Computational Science
4. Reproducibility
5. Modularity
6. Summary & Outlook
The Sprat Approach: Hierarchies of DSLs

1. Deployment Language Engineer
   - Ansible Playbook DSL
   - «refer» Simulation Specification

2. Ecosystem Language Engineer
   - Sprat Ecosystem DSL
   - «generate» Ecosystem Model
   - Sprat PDE DSL, embedded in C++
   - «include» FEM PDE Solver
   - Sprat PDE DSL, embedded in C++

3. PDE Language Engineer
   - Ecosystem Model
   - Sprat PDE DSL, embedded in C++

4. Deployment Engineer
5. Stock Assessment Scientist
6. Ecological Modeler
7. 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

- Expert interviews with domain experts and professional DSL developers from industry
- Micro- and macro-benchmarks for performance evaluation [Johanson et al. 2016b]

```cpp
DistributedVector u, q;
ElementVectorArray F_L;
ElementMatrixArray C;
ElementMatrix D;

void foreach_omp(tau, Elements(mesh), private(D), {
  foreach(i, ElementDoF(tau), {
    foreach(j, ElementDoF(tau), {
      D(i, j) = max(i.globalIndex(), j.globalIndex());
    }
  }
  F_L[tau] = C[tau]*q + D*u;
}
})
u *= u.dotProduct(q);
u.exchangeData();
```
The Sprat Marine Ecosystem Model

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

Metabolic Costs
- Resting Metabolic Rate
- Net Swimming Costs

Movement
- Active
  - Reactive
  - Predictive
- Passive

Reproduction

Background Mortality

Fishing

Predation (Opportunistic)
- Intake
- Losses

Biogeochemical Ocean Model
- Zooplankton
- Currents
- Temperature

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

1. Software Engineering vs. Computational Science
2. Software Engineering for Computational Science
4. Reproducibility
5. Modularity
6. Summary & Outlook
“Replication is the ultimate standard by which scientific claims are judged.”

Reproducibility Spectrum

- Publication only
  - Code
- Publication +
  - Code and data
  - Linked and executable code and data
- Full replication
  - Gold standard

[Peng 2011]
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 Sñffsmann, 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.

[Johanson et al. 2017b]
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 Waller$^1$ and Wilhelm Hasselbring$^{1,2}$

$^1$ Software Engineering Group, Christian-Albrechts-University Kiel, Germany
$^2$ SPEC Research Group, Steering Committee, Gainesville, VA, USA

From Reproducibility Problems to Improvements: A journey

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

[Waller and Hasselbring 2012] [Eichelberger et al. 2016]
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: {ffk, akr, wha}@informatik.uni-kiel.de

[GitHub] [Zenodo] [ExplorViz]

Experimental Data for: Exploring Software Cities through Virtual Reality

06 August 2015

06 August 2015
DOI

06.5.28/unzennod:Z33356

Publication date:

Keyword(s):

Virtual Reality
Software City Metaphor
ExplorViz

Meeting:

Third IEEE Working Conference on Software Visualization (VISSOFT 2015), Breman, Germany. 2015.

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.

[Fittkau et al. 2013, 2015a-d, 2017]
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]
Agenda

1. Software Engineering vs. Computational Science
2. Software Engineering for Computational Science
4. Reproducibility
5. Modularity
6. Summary & Outlook
OceanTEA: [Johanson et al. 2016a, Johanson et al. 2017b]
Generic Research Data Infrastructure

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

Part 5: Modularity
Jupyter Computational Notebooks

- Jupyter is a free, open-source, web tool, which researchers can use to combine
  - software code,
  - computational output,
  - explanatory text and
  - multimedia resources in a single document.

- Besides exploration, the result may be a “computational narrative”
  - A document that allows researchers to supplement their code and data with analysis, hypotheses and conjecture.
  - You may also use notebooks to create tutorials or interactive manuals for your software.

- Some features:
  - Provenance tools.
  - Back-end ‘kernels’ run the code (on HPC servers) and return the results.
  - JupyterLab offers an enhanced, IDE-line interface, which can be extended through extensions.
  - JupyterHub allows to provide Jupyter notebooks as a service (SaaS).
  - Various cloud services such as BinderHub and Code Ocean exist.

- 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.
deRSE19 - Call for Contributions

Following the success of the first three international Conferences of Research Software Engineers in the UK, deRSE19, research software and the people behind it within the German research landscape will be held at the Albert Einstein Science Park.

The organising committee welcomes submissions for workshops, talks, and posters for the deRSE19 conference, as well as the presentation of research projects and tools.

The aim is to reflect the diverse community of research software engineers by seeking input from all levels of experience, genders, and ethnicities.

Timeline

- 20 December 2018 - We are open for submissions
- 28 February 2019 - Deadline for submissions
- 22 March 2019 - Notification of acceptance
- 04-06 June 2019 - deRSE19 conference
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 by reviewing published literature (the *present*).

We found that scientific software development’s unique characteristics *prevent* scientists from using state-of-the-art software engineering tools and methods.

- This situation created a *chasm* between software engineering and computational science, which resulted in productivity and credibility crises of the latter discipline.

We examined attempts to bridge the gap in order to reveal the shortcomings of existing solutions and to point out further research directions,

- such as the use of DLSs and testing techniques without predefined oracles (the *possible future*).

**Reproducibility** is essential for good scientific practice.

**Modularity** is essential for maintainability, scalability and agility

We are recruiting (deadline in April):
http://mardata.de
References


References


References


