Software Engineering for Computational Science

Journal paper: A. Johanson, W. Hasselbring:
"Software Engineering for Computational Science: Past, Present, Future",
In: Computing in Science & Engineering, pp. 90-109, March/April 2018.
https://doi.org/10.1109/MCSE.2018.108162940

Arne Johanson

Wilhelm Hasselbring

Christian-Albrechts-Universität zu Kiel

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

ozean der zukunft
DIE KIELER MEERESWISSENSCHAFTEN

GEOMAR

HOSST
TRANSATLANTIC RESEARCH SCHOOL

Agenda

1. Software Engineering vs. Computational Science
2. Software Engineering for Computational Science
3. Sprat: Domain-specific SE for Ecology
4. Reproducibility
5. Modularity
6. 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]

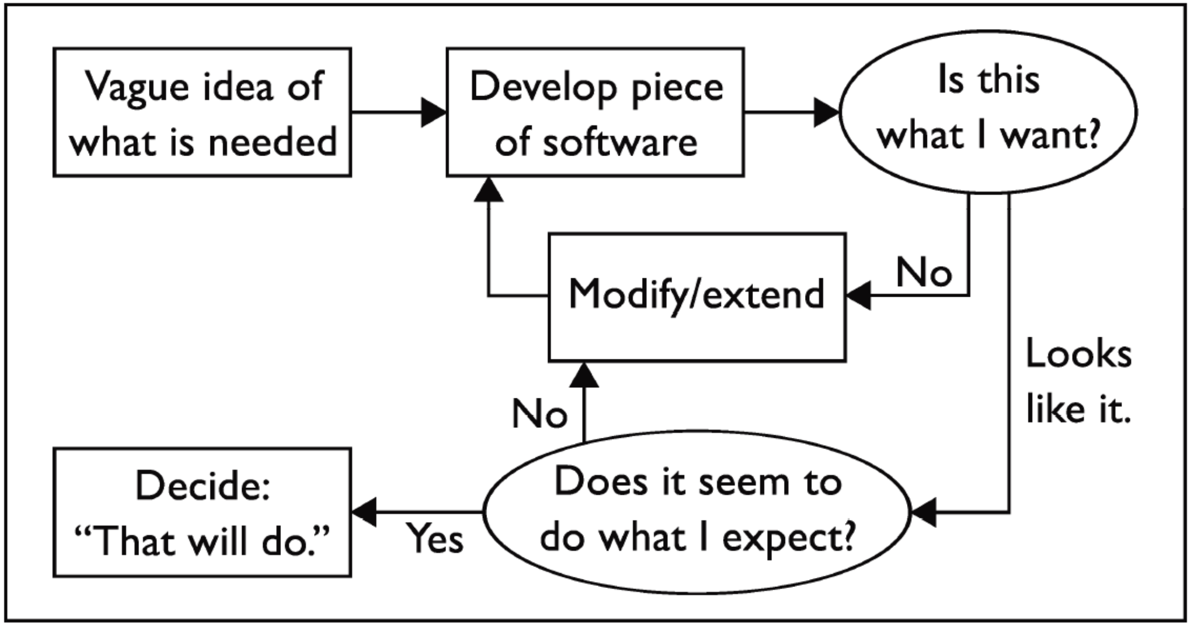
Mutual Ignorance: Software Engineering

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[22] 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

Mutual Ignorance: Computational Science

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 <u>for</u> Computational Science**
3. Sprat: Domain-specific SE for Ecology
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

software carpentry

Teaching basic lab skills
for research computing

https://software-carpentry.org/

SE for Computational Science

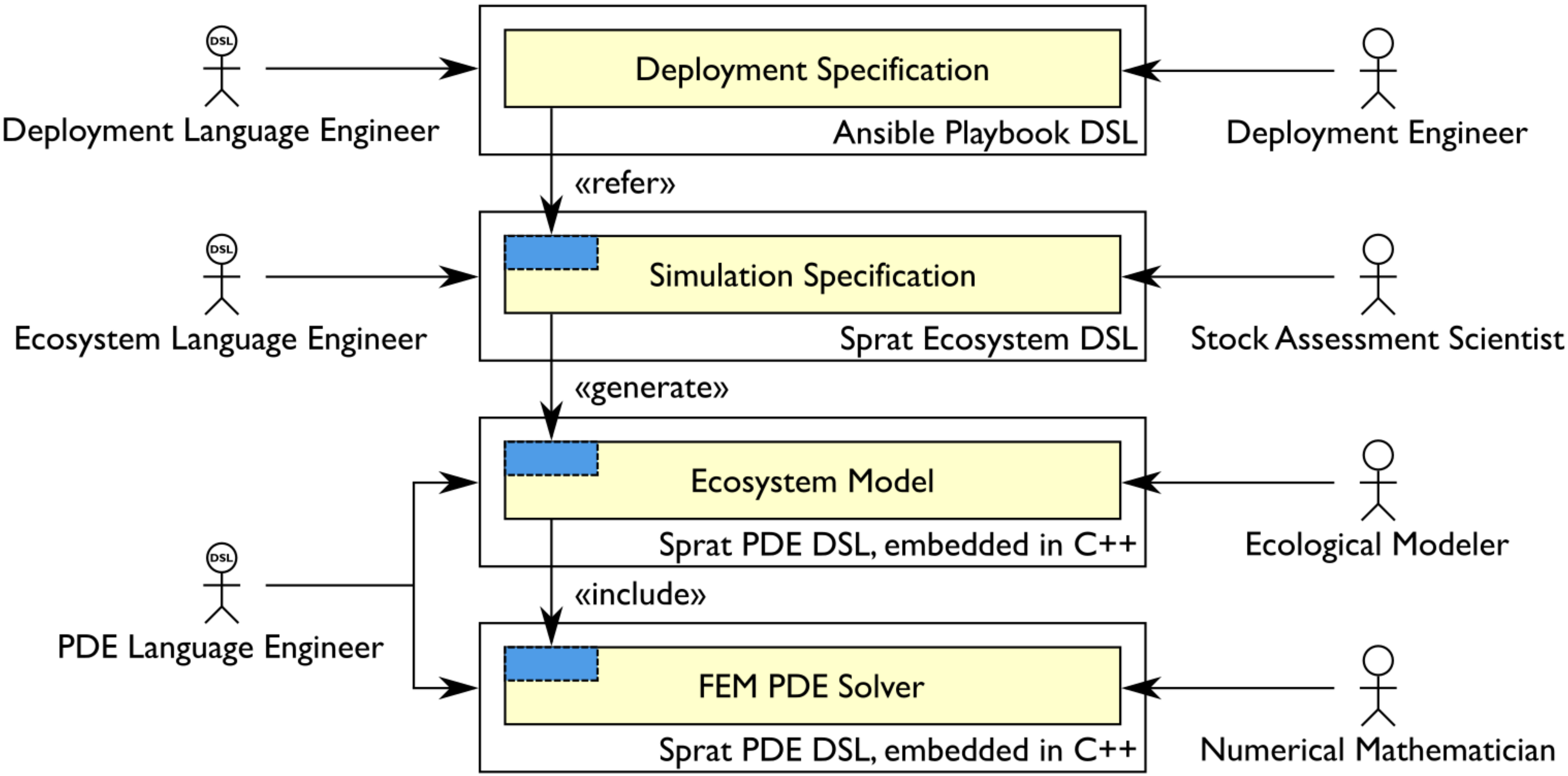
[Johanson & Hasselbring 2018]:

- 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

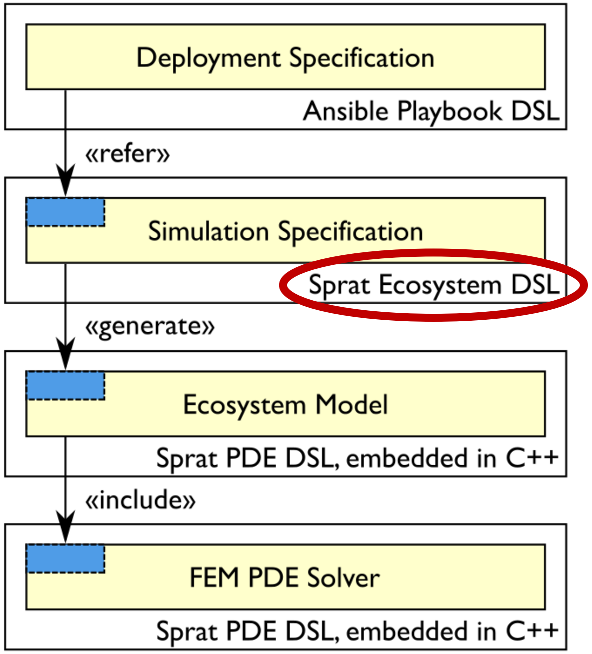
1. Software Engineering vs. Computational Science
2. Software Engineering for Computational Science
3. **Sprat: Domain-specific SE for Ecology**
4. Reproducibility
5. Modularity
6. Summary & Outlook

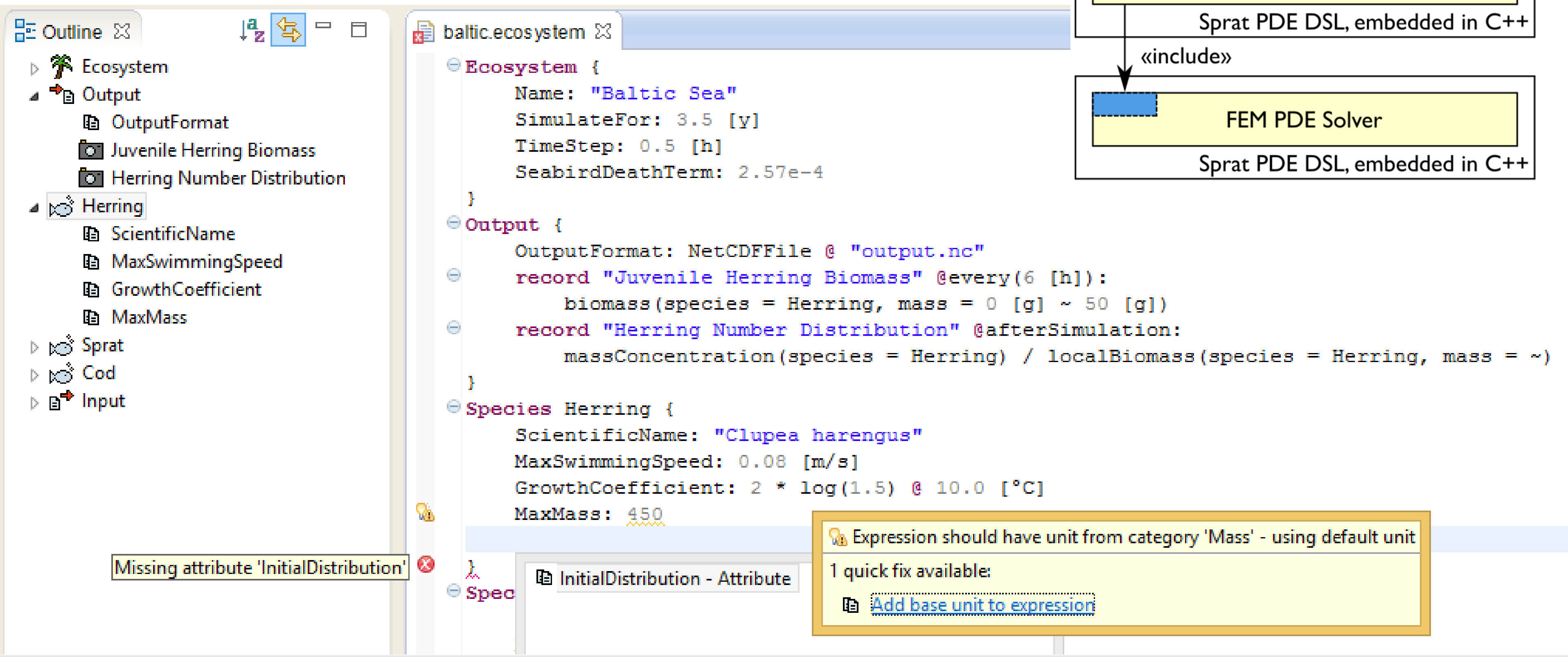
The Sprat Approach: Hierarchies of DSLs



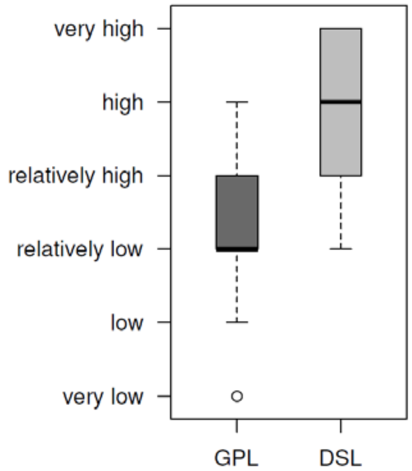
[Johanson & Hasselbring 2014a,b, 2016b]

The Sprat Ecosystem DSL

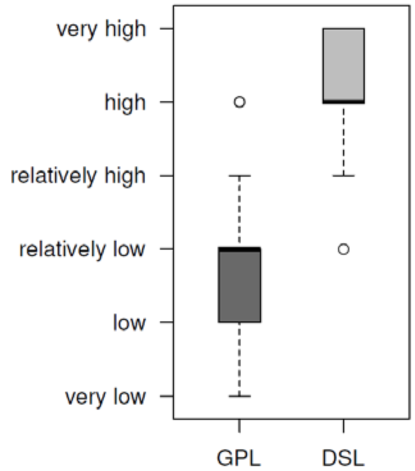




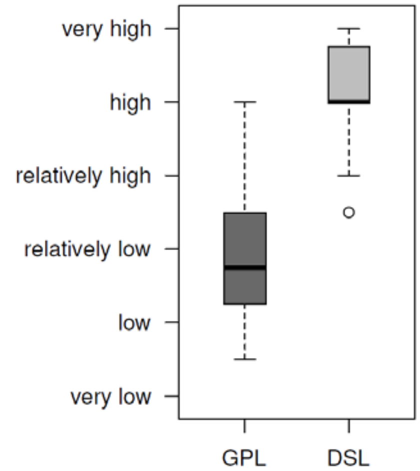
Evaluation of the Sprat <u>Ecosystem</u> DSL



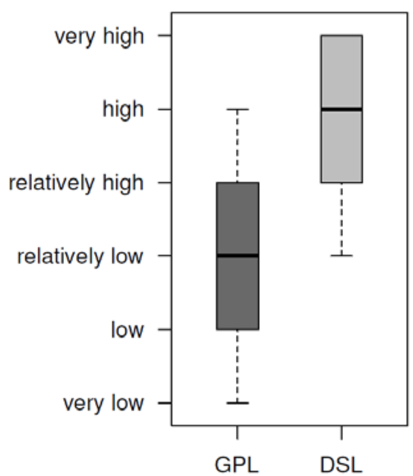
(a) Level of abstraction



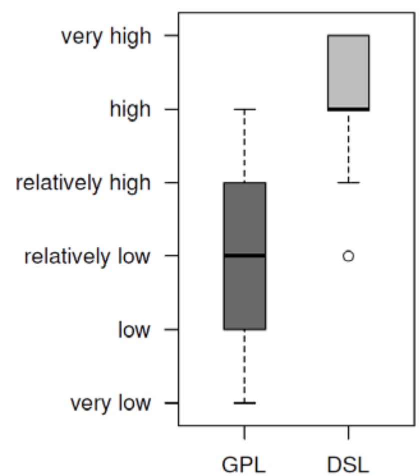
(b) Simplicity of use



(c) Ease of comprehension



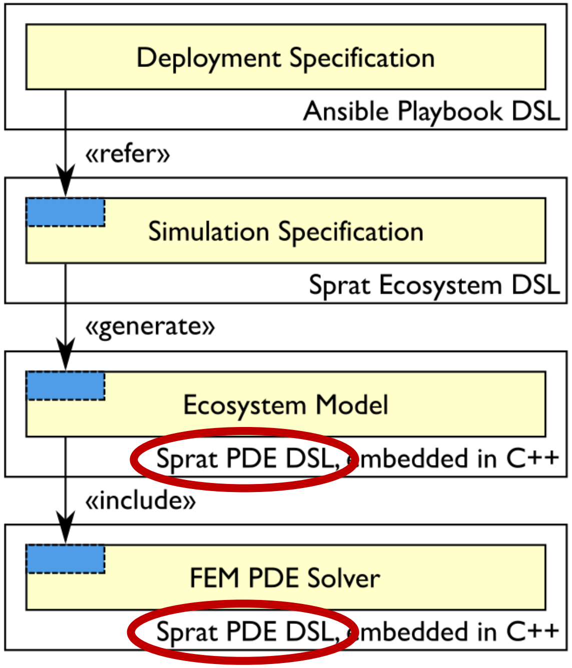
(d) Absence of technicalities



(e) Maintainability of solutions

[Johanson & Hasselbring 2017]

The Sprat PDE DSL



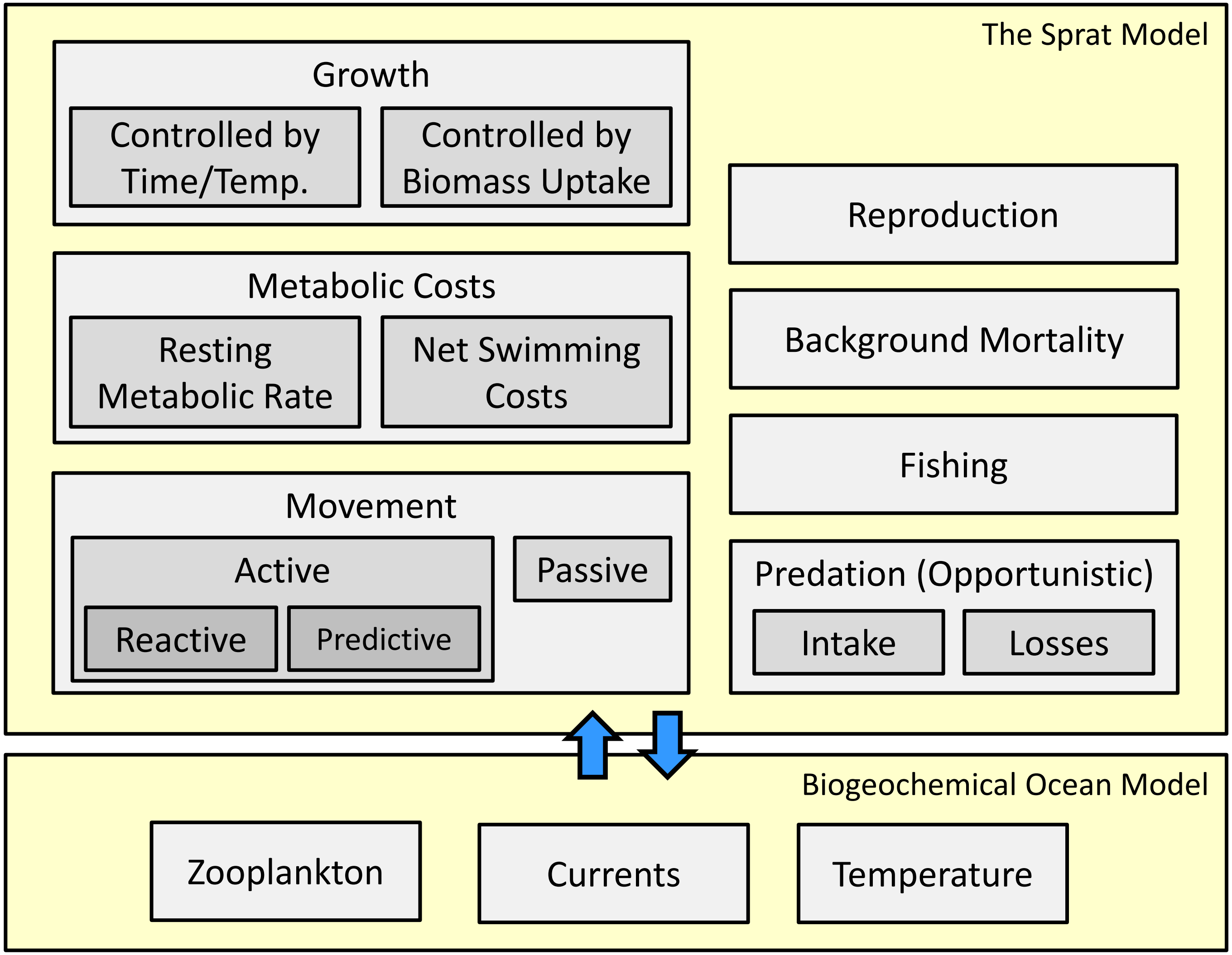

```
DistributedVector  u, q;
ElementVectorArray F_L;
ElementMatrixArray C;
ElementMatrix      D;

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();
```

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

SPRAT

Available online:

- DSL implementations
- Sprat Model source code

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

- Experimental data and analysis scripts

zenodo

http://dx.doi.org/10.5281/zenodo.61373

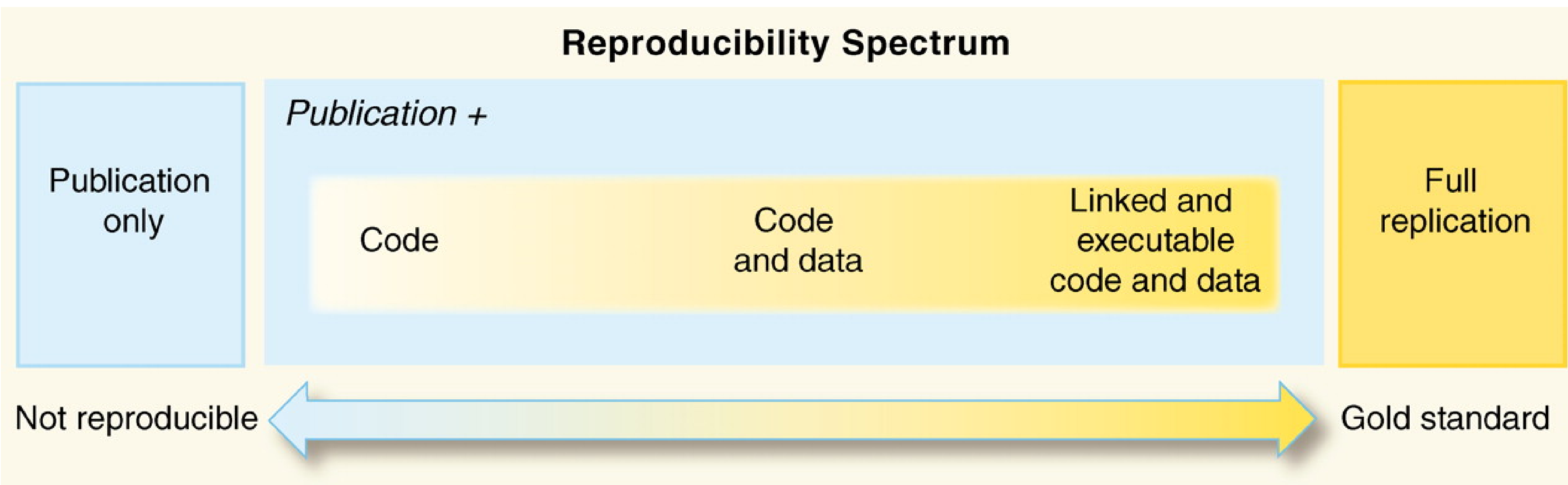
Agenda

1. Software Engineering vs. Computational Science
2. Software Engineering for Computational Science
3. Sprat: Domain-specific SE for Ecology
4. **Reproducibility**
5. Modularity
6. Summary & Outlook

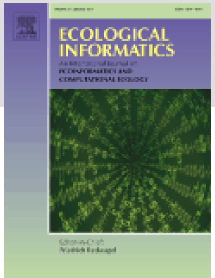
Reproducible Research in Computational Science

Science

"Replication is the ultimate standard by which scientific claims are judged."



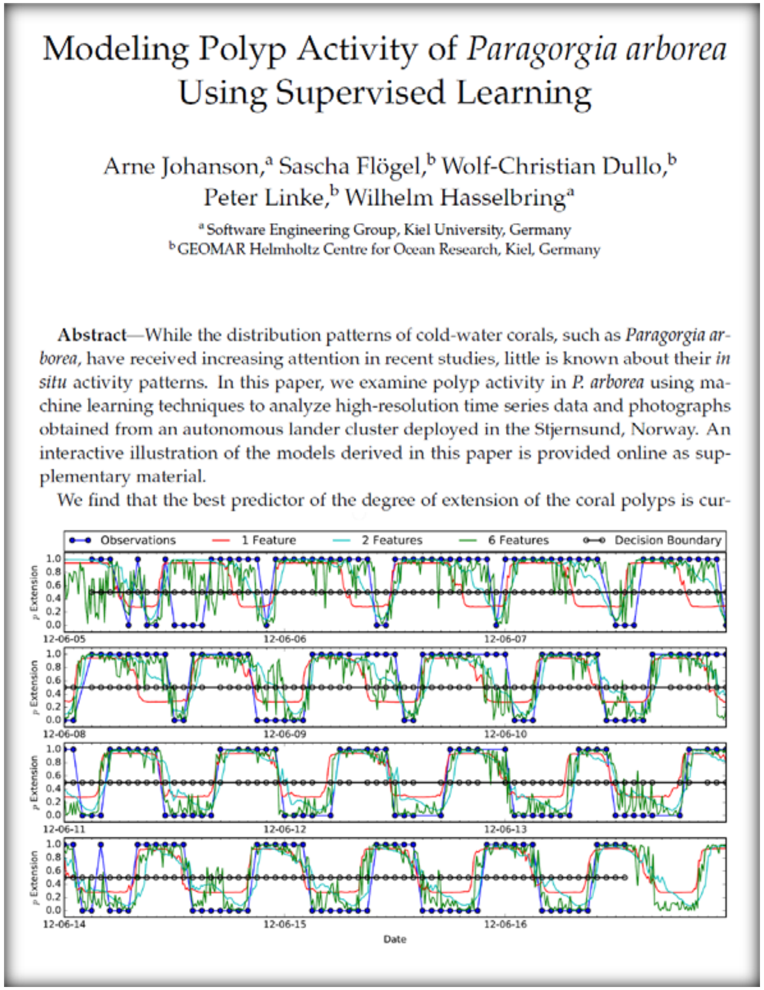
[Peng 2011]



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/

ozean der zukunft
DIE KIELER MEERESWISSENSCHAFTEN



Modeling Polyp Activity of *Paragorgia arborea* Using Supervised Learning

Arne Johanson,[a] Sascha Flögel,[b] Wolf-Christian Dullo,[b] Peter Linke,[b] Wilhelm Hasselbring[a]

[a] Software Engineering Group, Kiel University, Germany
[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 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 cur-

[Johanson et al. 2017b]

http://oceantea.uni-kiel.de/

COMMUNICATIONS OF THE ACM | MARCH 2015 | VOL. 58 | NO. 3

DOI:10.1145/2658987

Shriram Krishnamurthi and Jan Vitek

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*

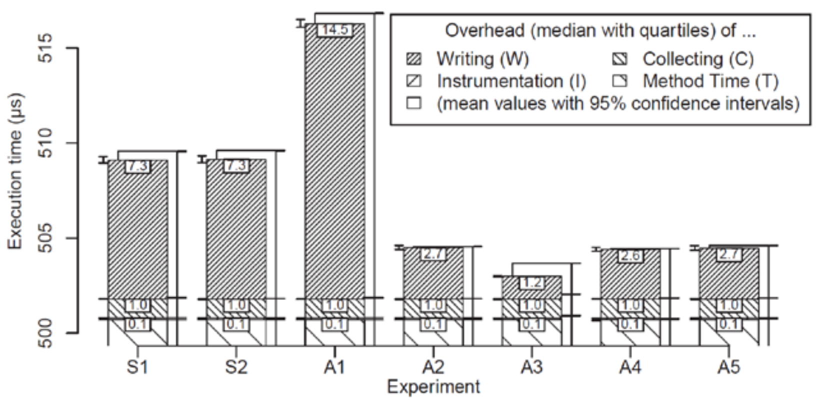
Several ACM SIGMOD, SIGPLAN, and SIGSOFT conferences have initiated **artifact evaluation** processes.

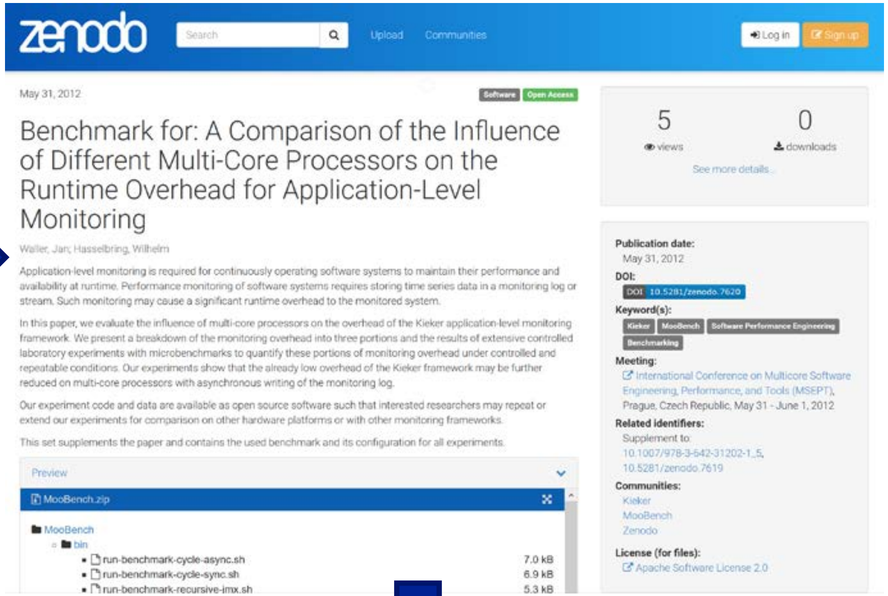
Example Experimental “Reproducibility Data” in Software Engineering

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}@sse.uni-hildesheim.de, sassai@uni-hildesheim.de
University of Hildesheim, Software Systems Engineering, 31141 Hildesheim, Germany

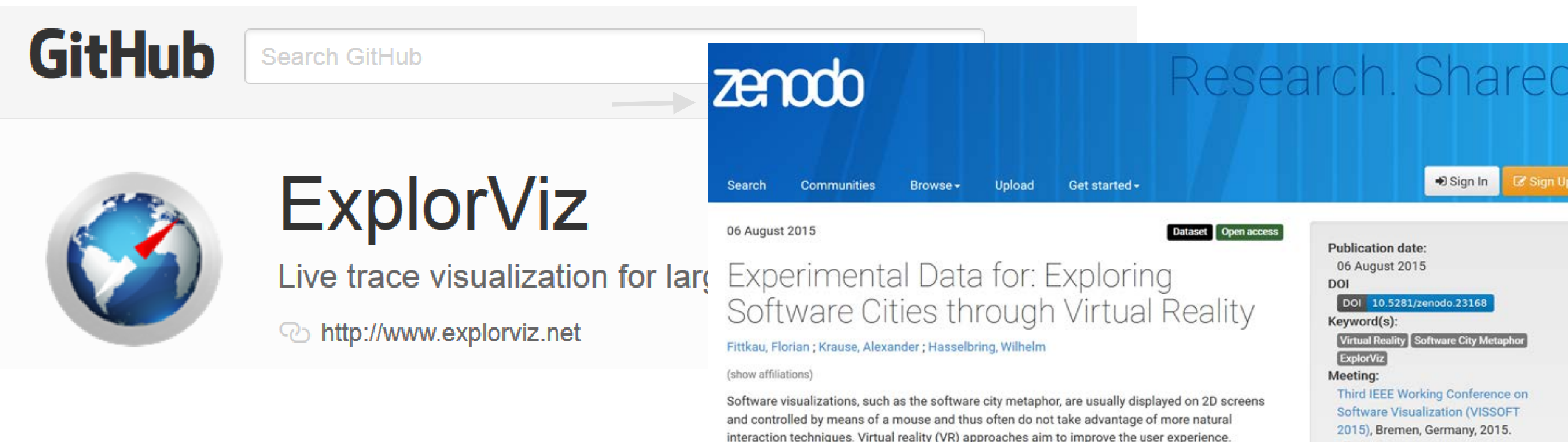
[Waller and Hasselbring 2012]

[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



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

Impact of Artifact Evaluation

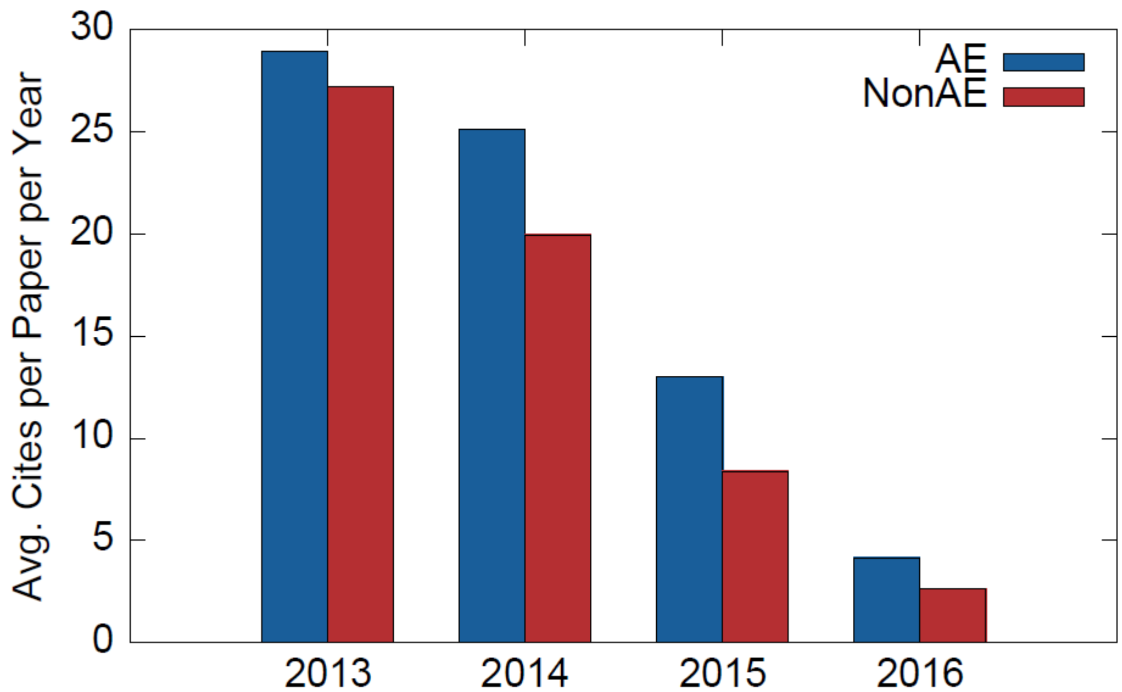


Fig. 1. Average citation counts of AE and non-AE papers for conferences that used AE in 2013 to 2016 (conferences: VISSOFT, PPoPP, POPL, PLDI, PACT, OOPSLA, ISSTA, FSE, ECRTS, ECOOP, CGO, CAV).

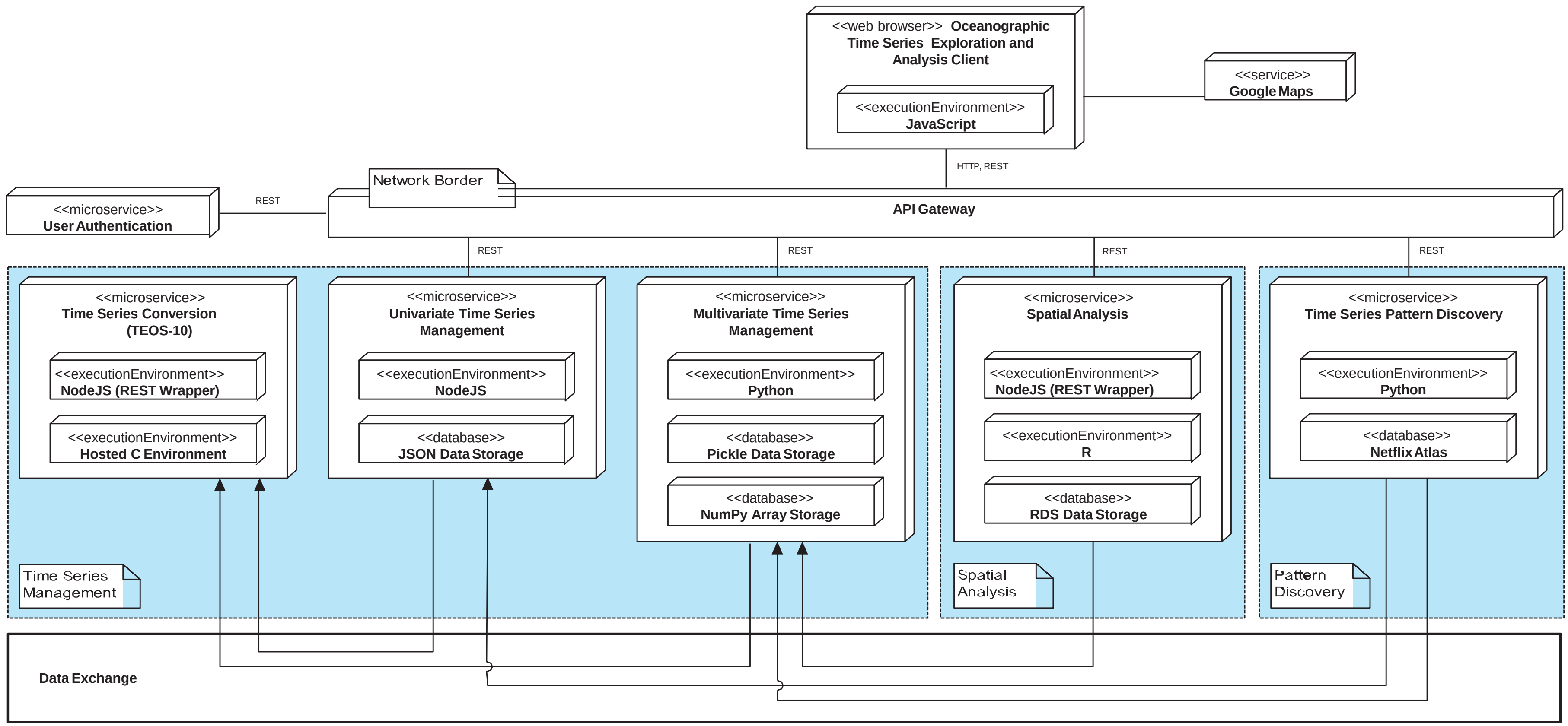
[Childers & Chrysanthis 2017]

Agenda

1. Software Engineering vs. Computational Science
2. Software Engineering for Computational Science
3. Sprat: Domain-specific SE for Ecology
4. Reproducibility
5. **Modularity**
6. Summary & Outlook

Modular Scientific Software

OceanTEA: Microservice-based Architecture



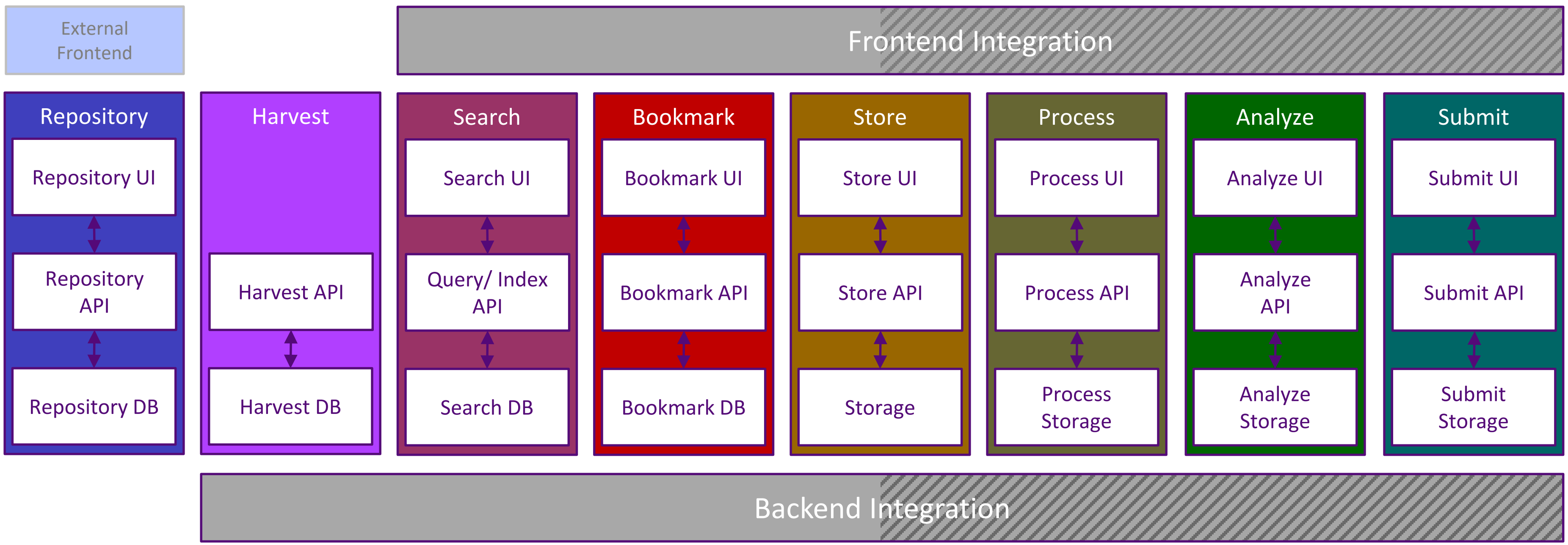
OceanTEA: [Johanson et al. 2016a, Johanson et al. 2017b]

Microservices: [Hasselbring 2016, 2018, Hasselbring & Steinacker 2017, Knoche & Hasselbring 2018, 2019]

Generic Research Data Infrastructure

GeRDI

DFG



ZBW Leibniz-Informationszentrum Wirtschaft Leibniz Information Centre for Economics

CAU Christian-Albrechts-Universität zu Kiel

TECHNISCHE UNIVERSITÄT DRESDEN

lrz

DFN

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

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 - Conference for Research Software Engineers in Germany
https://www.de-rse.org/en/conf2019/

RSE de-RSE | deRSE19 Conference | Call for Contributions | Contact | For delegates ▼ | Sponsorshi

deRSE19 - Call for Contributions

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

The organising committee welcomes submissions for workshops, talks, and posters for the deRSE19 conference, as well
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

Summary & Outlook

Slides: http://eprints.uni-kiel.de/45769/
Preprint: http://eprints.uni-kiel.de/41442/

- 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

MarDATA HELMHOLTZ SCHOOL FOR MARINE DATA SCIENCE

References

[Carver et al. 2007] J.C. Carver et al., “Software Development Environments for Scientific and Engineering Software: A Series of Case Studies,” Proc. 29th Int'l Conf. Software Eng. (ICSE 07), 2007, pp. 550–559.

[Childers & Chrysanthis 2017] B.R. Childers and P.K. Chrysanthis, "Artifact Evaluation: Is It a Real Incentive?," 2017 IEEE 13th International Conference on e-Science, 2017, pp. 488-489. http://doi.org/10.1109/eScience.2017.79

[Eichelberger et al. 2016] H. Eichelberger et al., “From reproducibility problems to improvements: A journey,” Softwaretechnik-Trends: Proceedings of the Symposium on Software Performance (SSP'16). Vol. 36. No. 4. 2016.

[Fittkau et al. 2013] F. Fittkau, J. Waller, C. Wulf, W. Hasselbring: “Live Trace Visualization for Comprehending Large Software Landscapes: The ExplorViz Approach“, In: 1st IEEE International Working Conference on Software Visualization (VISSOFT 2013).

[Fittkau et al. 2015a] F. Fittkau, S. Roth, W. Hasselbring: “ExplorViz: Visual Runtime Behavior Analysis of Enterprise Application Landscapes“, In: 23rd European Conference on Information Systems (ECIS 2015).

[Fittkau et al. 2015b] F. Fittkau, A. Krause, W. Hasselbring: “Hierarchical Software Landscape Visualization for System Comprehension: A Controlled Experiment”. In: 3rd IEEE Working Conference on Software Visualization, 2015.

[Fittkau et al. 2015c] F. Fittkau, A. Krause, W. Hasselbring: “Exploring Software Cities in Virtual Reality”, In: 3rd IEEE Working Conference on Software Visualization, September 2015, Bremen, Germany.

[Fittkau et al. 2015d] F. Fittkau, S. Finke, W. Hasselbring, J. Waller: “Comparing Trace Visualizations for Program Comprehension through Controlled Experiments”, In: 23rd IEEE International Conference on Program Comprehension (ICPC 2015), May 2015, Florence.

[Fittkau et al. 2017] F. Fittkau, A. Krause, W. Hasselbring: “Software landscape and application visualization for system comprehension with ExplorViz”, In: Information and Software Technology. DOI 10.1016/j.infsof.2016.07.004

[Fuller and Millett 2011] S.H. Fuller and L.I. Millett, “Computing Performance: Game Over or Next Level?,” Computer, vol. 44, no. 1, 2011, pp. 31–38.

[Goltz et al., 2015] U. Goltz et al., “Design for Future: Managed Software Evolution,” Computer Science - Research and Development, vol. 30, no. 3, 2015, pp. 321–331.

References

[Hasselbring 2016] W. Hasselbring, "Microservices for Scalability (Keynote Presentation)," In: 7th ACM/SPEC International Conference on Performance Engineering (ACM/SPEC ICPE 2016), March 15, 2016 , Delft, NL.

[Hasselbring 2018] W. Hasselbring, "Software Architecture: Past, Present, Future," In: The Essence of Software Engineering. Springer, pp. 169-184. 2018. DOI 10.1007/978-3-319-73897-0_10

[Hasselbring & Steinacker 2017] W. Hasselbring, G. Steinacker: "Microservice Architectures for Scalability, Agility and Reliability in E-Commerce", In: Proceedings of the IEEE International Conference on Software Architecture (ICSA 2017), April 2017, Gothenburg, Sweden.

[Johanson & Hasselbring 2014a] A. Johanson, W. Hasselbring: "Hierarchical Combination of Internal and External Domain-Specific Languages for Scientific Computing". In: International Workshop on DSL Architecting & DSL-Based Architectures (DADA'14), August 2014, Vienna, Austria, pp. 17:1-17:8.

[Johanson & Hasselbring 2014b] A. Johanson, W. Hasselbring: "Sprat: Hierarchies of Domain-Specific Languages for Marine Ecosystem Simulation Engineering". In: Spring Simulation Multi-Conference (SpringSim 2014), April 2014, Tampa, Florida, USA, pp. 187-192.

[Johanson et al. 2016a] A. Johanson, S. Flögel, C. Dullo, W. Hasselbring: "OceanTEA: Exploring Ocean-Derived Climate Data Using Microservices". In: Sixth International Workshop on Climate Informatics (CI 2016), September 2016, Boulder, Colorado.

[Johanson et al. 2016b] A. Johanson, W. Hasselbring, A. Oschlies, B. Worm: "Evaluating Hierarchical Domain-Specific Languages for Computational Science: Applying the Sprat Approach to a Marine Ecosystem Model". In: Software Engineering for Science. CRC Press. 175-200.

[Johanson et al. 2017a] A. Johanson, A. Oschlies, W. Hasselbring, A. Worm: "SPRAT: A spatially-explicit marine ecosystem model based on population balance equations", In: Ecological Modelling, 349, pp. 11-25, 2017.

[Johanson et al. 2017b] A. Johanson, S. Flögel, C. Dullo, P. Linke, W. Hasselbring: "Modeling Polyp Activity of Paragorgia arborea Using Supervised Learning", In: Ecological Informatics, 39, pp. 109-118, 2017.

References

[Johanson & Hasselbring 2017] A. Johanson, W. Hasselbring: “Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment”, In: Empirical Software Engineering 22 (8). pp. 2206-2236, 2017.

[Johanson & Hasselbring 2018] A. Johanson, W. Hasselbring: “Software Engineering for Computational Science: Past, Present, Future”, In: Computing in Science & Engineering, pp. 90-109, March/April 2018.

[Kanewala and Bieman 2014] U. Kanewala and J.M. Bieman, “Testing Scientific Software: A Systematic Literature Review,” Information and Software Technology, vol. 56, no. 10, 2014, pp. 1219–1232.

[Knoche and Hasselbring 2018] H. Knoche and W. Hasselbring, “Using Microservices for Legacy Software Modernization IEEE Software, 35 (3). pp. 44-49. 2018. DOI 10.1109/MS.2018.2141035.

[Knoche and Hasselbring 2019] H. Knoche and W. Hasselbring, “Drivers and Barriers for Microservice Adoption - A Survey among Professionals in Germany,” Enterprise Modelling and Information Systems Architectures (EMISAJ) - International Journal of Conceptual Modeling, 14 (1). pp. 1-35. 2019. DOI https://doi.org/10.18417/emisa.14.1.

[Merali 2010] Z. Merali, “Computational Science: Error, Why Scientific Programming Does Not Compute,” Nature, vol. 467, no. 7317, 2010, pp. 775–777

[Peng 2011] R.D. Peng, “Reproducible Research in Computational Science,” 334(6060), pp. 1226-1227, 2011

[Perkel 2018] J.M. Perkel, “Why Jupyter is data scientists’ computational notebook of choice,” Nature 563:145-146. 2018

[Randell 2018] B. Randell: 50 years of Software Engineering. May 2018, https://arxiv.org/abs/1805.02742

[Tavares de Sousa et al. 2018] N. Tavares de Sousa, W. Hasselbring, T. Weber, D. Kranzlmüller: “Designing a Generic Research Data Infrastructure Architecture with Continuous Software Engineering”, In: 3rd Workshop on Continuous Software Engineering (CSE 2018), March 2018, Ulm, Germany.

[Thew et al. 2009] S. Thew et al., “Requirements Engineering for e-Science: Experiences in Epidemiology,” IEEE Software, vol. 26, no. 1, 2009.

[Waller and Hasselbring 2012] J. Waller and W. Hasselbring, “A Comparison of the Influence of Different Multi-Core Processors on the Runtime Overhead for Application-Level Monitoring,” In: International Conference on Multicore Software Engineering, Performance, and Tools (MSEPT), 2012