

**CAS 741 (Development of Scientific Computing  
Software)**

**Winter 2023**

**A Holistic Approach to Pain Relief/Digging  
Deeper**

Dr. Spencer Smith

Faculty of Engineering, McMaster University

March 28, 2023



# A Holistic Approach to Pain Relief/Digging Deeper

- Administrative details
- A Holistic Approach to Pain Relief for Research Software Developers
- Digging Deeper Into the State of the Practice for Domain Specific Research Software

# Administrative Details: Report Deadlines

## Final Documentation    Week 13    Apr 12

- The written deliverables will be graded based on the repo contents as of 11:59 pm of the due date
- If you need an extension for a **written** doc, please ask
- When ready, assign issues to your primary and secondary reviewers
- GitHub issues due two days after assignment deadlines
- From Drasil Code onward, Drasil projects no longer need to maintain traditional SRS

# Administrative Details: Presentations

Drasil                                      Week 11      Week of Mar 27

Unit VnV/Implement      Week 12      Week of Apr 3

- Specific schedule depends on final class registration
- Informal presentations with the goal of improving everyone's written deliverables
- Domain experts and secondary reviewers (and others) will ask questions



# Presentation Schedule

- Drasil Project Present (25 min each)
  - ▶ **Mar 30: Karen, Sam, Jason**

# Presentation Schedule

- Test or Impl. Present (25 min each)
  - ▶ Apr 5: Lesley, Deesha, Volunteer?
  - ▶ Apr 6: Mina, Joachim, Maryam
- 4 presentations each (please check)
- If you will miss a presentation, please trade with someone else
- Implementation presentation could be used to run a code review, or code walkthrough

# Course Evaluations

- Course experience surveys will close on Wednesday, April 12, 2023 11:59 PM
- <https://mcmaster.bluera.com/mcmaster/>

# Questions?

- Questions on administrative details?
- Questions on final documentation?
- Questions on reflection document?
- Questions on final implementation?
- Questions on VnV report?
- Other questions?

# A Holistic Approach to Pain Relief for Research Software Developers

Dr. Spencer Smith

Faculty of Engineering, McMaster University

March 28, 2023



# Outline

- Sustainable and Reproducible Research Software
- Pain Points
- Treatment Options
  - ▶ Literate Programming
  - ▶ Code Generation
  - ▶ Holistic Approach
- Concluding Remarks



# Health Goals

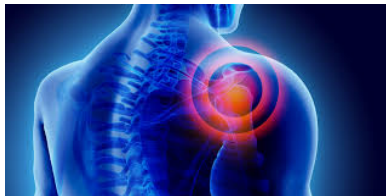
- **Sustainable** software satisfies, for a reasonable amount of effort, the software *requirements* for the present (like *correctness*), while also being maintainable, reusable, and *reproducible* for the future.
- **Reproducible** research includes all data, code, and documentation so that the computations can be *repeated in the future with identical results*.

Requires design, documentation, and verification (testing)

# Problems with Achieving Goals: Pain Points

From Developer Interviews:

- Lack of time
- Lack of software development experience
- Lack of technology experience
- Frequency of change

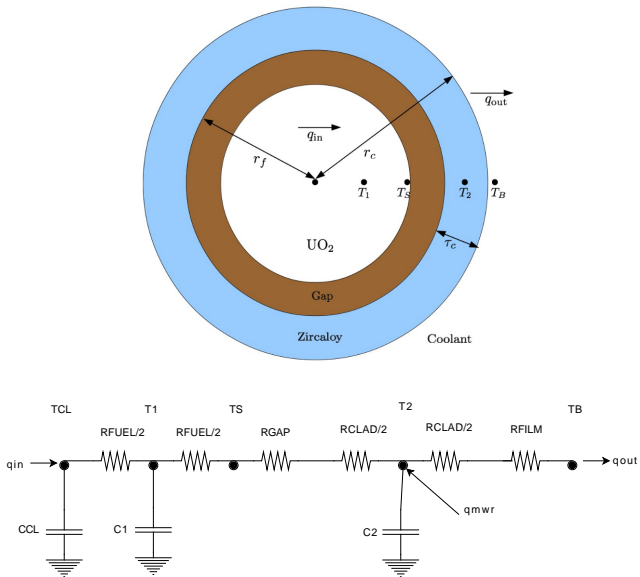




# Treatment 1: Literate Programming

- “instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do” [[11](#), pg. 99]
- Interconnected “web” of pieces of code, or *chunks*
- Tangle extracts code
- Weave extracts docs (as LaTeX, html, pdf, text, etc.)
- CWEB, Sweave (R), Jupyter, emacs org mode, Maple worksheets, etc.





## Example

### B.6.1 Computing $q'_N$ , $T_2$ and $k_c$

The input relative fuel power ( $q'_{NFRAC}$ ) is changed to linear element power ( $q'_N$ ) by multiplying it with the initial linear element rating ( $q'_{N_{max}}$ ) as given by DD25 of the SRS.

$$q'_N = q'_{NFRAC} q'_{N_{max}}; \quad (B.8)$$

This  $q'_N$  is used to determine the relevant temperatures for the fuelpin. We evaluate linear element power as

17     $\langle \text{Calculation of } q'_N \text{ 17} \rangle \equiv$   
           $*q\_N = *q\_NFRAC * (*q\_Nmax);$

This code is used in chunks 15 and 57

# LP Treatment Evaluation

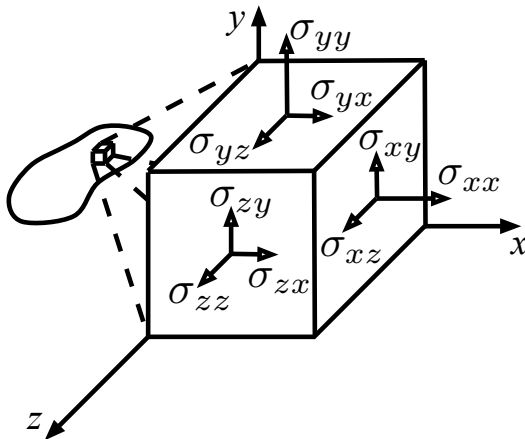
- Uncovered 27 issues with previous docs
- Documentation improves reproducibility
- Pain point score:
  - ▶ Lack of time: ✓
  - ▶ Lack of dev exp: —
  - ▶ Lack of technology exp: ✗
  - ▶ Freq of change: ✓
- Problems with literate programming
  - ▶ Does not scale well (best for small examples, lessons)
  - ▶ Difficult to refactor
  - ▶ *Manually* repeat information in text and code
  - ▶ *Manually* create traceability and structure

## Treatment 2: Code Generation



# A Virtual Material Testing Laboratory

Given the deformation history of a material particle, determine the internal stress within the material particle.



# Calculations

Given  $F, Q, \kappa, \varphi, \gamma$  calculate:

$$\mathbf{K} = \int_V \mathbf{B}^T \mathbf{D}^{vp} \mathbf{B} dV; \mathbf{F} = \mathbf{R}_i - \int_V \mathbf{B}^T \boldsymbol{\sigma}_i dV + \int_V \mathbf{B}^T \Delta \boldsymbol{\sigma}^{vp} dV \quad (1)$$

with

$$\mathbf{D}_{vp} = \mathbf{D} \left[ \mathbf{I} - \Delta t C_1 \lambda' \frac{\partial Q}{\partial \boldsymbol{\sigma}} \left( \frac{\partial F}{\partial \boldsymbol{\sigma}} \right)^T \mathbf{D} \right], \lambda' = \frac{d\lambda}{dF} \quad (2)$$

$$\Delta \boldsymbol{\sigma}^{vp} = \Delta t C_1 \lambda \mathbf{D} \frac{\partial Q}{\partial \boldsymbol{\sigma}} \quad (3)$$

$$C_1 = [1 + \lambda' \Delta t (H_e + H_p)]^{-1} \quad (4)$$

$$H_e = \left( \frac{\partial F}{\partial \boldsymbol{\sigma}} \right)^T \mathbf{D} \left( \frac{\partial Q}{\partial \boldsymbol{\sigma}} \right); H_p = -\frac{\partial F}{\partial \kappa} \left( \frac{\partial \kappa}{\partial \boldsymbol{\epsilon}^{vp}} \right)^T \frac{\partial Q}{\partial \boldsymbol{\sigma}} \quad (5)$$

# Code Generation

- Specify variabilities:  $F, Q, \kappa, \varphi, \gamma$
- Symbolically calculate terms, including  $\frac{\partial Q}{\partial \sigma}, \frac{\partial F}{\partial \sigma}$ , etc.
- Symbolic processing avoids tedious and error-prone hand calculations
  - ▶ Reduces workload
  - ▶ Allows non-experts to deal with new problems
  - ▶ Increases reliability
- Use Maple Computer Algebra System



# Knowledge Capture and Code Generation

Code generation works by codifying additional knowledge:

- Maple – symbolic math
- org mode – simple document structure
- lex and yacc – regular expressions and grammars
- ATLAS – hardware knowledge [26]
- Spiral – FFT knowledge [15]
- Dolphin – Finite elem variational forms [12]
- Doxygen – API information

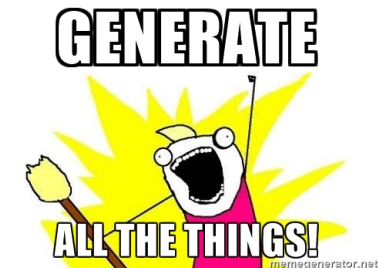
# Treatment and side effects

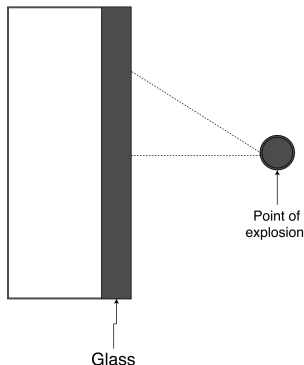
- Domain level programming
- Pain point scores:
  - ▶ Lack of time: ✓
  - ▶ Lack of dev exp: ✓
  - ▶ Lack of technology exp: ✗
  - ▶ Freq of change: ✓
- Problems
  - ▶ Focus is generally only on the code
  - ▶ Code generation does not help with reproducibility

# Holistic Approach



- Combine
  - ▶ Lit programming emphasis on documentation
  - ▶ Code gen, but for everything
- Codify more knowledge
  - ▶ Physics knowledge
  - ▶ Computing knowledge
  - ▶ Document knowledge
  - ▶ Design knowledge
  - ▶ Traceability knowledge
  - ▶ Technology knowledge





Given

- dimensions of plane
- glass type
- explosion characteristics
- tolerable breakage probability

Predict whether the glass will withstand the explosion

## Drasil Inputs:

- Program Name: GlassBR
- Authors: Nikitha K and Spencer S
- Symbols: tolerable load ( $\hat{q}_{tol}$ ), Risk of failure ( $B$ ), ...
- Assumptions: Load duration factor constant,
- Data definitions: relation for  $B$ , ...
- Design decisions:
  - Modularity (input module),
  - Implementation Type (Program),
  - Logging (Yes),
  - Input Structure (Bundled),
  - Constant Structure (Inlined),
  - Constant Rep (Constants),
  - Real Number Rep (Double),
  - ...

Drasil Source for software to predict whether a plate of glass will break

- Program Name: GlassBR
- Authors: Nikitha K and Spencer S
- Symbols: tolerable load ( $q_{tol}$ ), Risk of failure ( $B$ ), ...
- Assumptions: Load distrib. fact. constant,
- Data definitions: relation for  $B$ , ...
- Design decisions:
  - Modularity (input module),
  - Implementation Type (Program),
  - Logging (Yes),
  - Input Structure (Bundled),
  - Constant Structure (Inlined),
  - Constant Rep (Constants),
  - Real Number Rep ((Double)), ...

Generate

```
glassbr
Website/GlassBR_SRS.html
Website/GlassBR_SRS.css
/SRS/bibfile.bib
/SRS/Makefile
/SRS/GlassBR_SRS.tex
/SRS/GlassBR_SRS.pdf
/src/python
/src/python/README.md
/src/python/InputParameters.py
/src/python/Calculations.py
/src/python/Makefile
/src/python/doxConfig

...
/src/java/GlassBR/Calculations.java
/src/java/Makefile
/src/java/README.md

...
/src/cpp/GlassBR
/src/cpp/ReadTable.cpp
/src/cpp/InputFormat.hpp
/src/cpp/Calculations.cpp

...
/src/swift/Calculations.swift
...
/src/csharp/Control.cs
```

## Software Requirements Specification for GlassBR

Nikitha K and Spencer S

### Table of Symbols

$q_{tol}$   
 $B$

### Introduction

... The software, herein called GlassBR, ...

### Assumptions

LdConstant: LDF is constant, depends on assumed value of  $t_d$  and  $m$ , ...

### Data Definitions

$$B = \frac{k}{(ab)^{m-1}} (Eh^2)^m LDF e^J$$

$$B = \frac{k}{(ab)^{m-1}} (Eh^2)^m LDF e^J$$

## GlassBR

Authors Nikitha K and Spencer S

**How to Run the Program:** In your terminal command line, enter the same directory as this README file. Then enter the following line  
make run RUNARGS=input.txt  
Configuration Files: SDF.txt, TSD.txt must be in the same directory as the executable to run successfully  
Versioning: Python Version 3.5.1

```
...
build: GlassBR/Control.class
...
GlassBR/Control.class:
GlassBR/Control.java ...
python Control.py javac GlassBR/Control.java
...

run: build
java GlassBR.Control $(RUNARGS)
...
```

```
## \file Calculations.py
## \author Nikitha Krithnan and W. Spencer Smith
## \brief Provides functions for calculating the ...

## \brief Calculates risk of failure
## \param inParams structure holding the input v
## \param J stress distribution factor (Function
## \return risk of failure
def func_B(inParams):
    outfile = open("log.txt", "a")
    print("function func_B called with inputs: ")
    outfile.close()

    return 2.86e-53 / ((inParams.a * inParams.b)
    inParams.h ** 2.0) ** 7.0 * inParams.LDF * math
```

```
package GlassBR;
/** \file Calculations.java
 * \author Nikitha Krithnan and W. Spencer Smith
 * \brief Provides functions for calculating the outputs
 */

public static double func_B(InputParameters inParams, double J) throws IOException {
    PrintWriter outfile;
    outfile = new PrintWriter(new FileWriter(new File("log.txt"), true));
    outfile.println("function func_B called with inputs: {}");
    ...
    outfile.close();

    return 2.86e-53 / Math.pow(inParams.a * inParams.b, 7.0 - 1.0) *
    Math.pow(7.17e10 * Math.pow(inParams.h, 2.0), 7.0) * inParams.LDF
    * Math.exp(J);
}
```

# Holistic Treatment and Side Effects

- Sustainable and reproducible
- Can generate literate documents, if desired
- Pain point scores:
  - ▶ Lack of time: ✓
  - ▶ Lack of dev exp: ✓
  - ▶ Lack of technology exp: ✓
  - ▶ Freq of change: ✓
- Treats all pain points, and no side effects, but expensive medicine!

# Concluding Remarks

- Documentation *does not have to be painful*
- Combine benefits of Literate Programming
  - ▶ Emphasis on documentation, reproducibility
  - ▶ Organize information for a human being
- with benefits of Code Generation
  - ▶ *Capture knowledge only once*
  - ▶ *Generate all things!*
  - ▶ *Refactoring by regenerating*
- *Codify as much knowledge as possible*
- *Domain experts work at domain expert level*
- *Consistent by construction*
- Can address additional pain points
- Can absorb other treatment options, like testing, CI
- Requires additional research and “clinical trials”



# Digging Deeper Into the State of the Practice for Domain Specific Research Software

Dr. Spencer Smith

Faculty of Engineering, McMaster University

March 28, 2023



# Outline

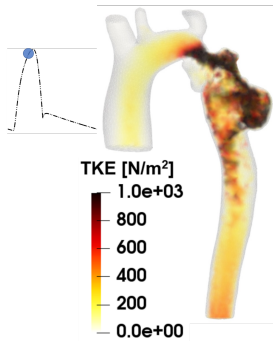
- Motivation and Example
- Overview of SOP Assessment Methodology
- Identify Software (RQ1)
- Rank
  - ▶ By Best Practices (RQ2)
  - ▶ Versus Community Ranking (RQ3)
- Comparison to Other Research Software
  - ▶ Artifacts (RQ4)
  - ▶ Tools (RQ5)
  - ▶ Processes (RQ6)
  - ▶ Pain Points (RQ7, RQ8)
- Threats to Validity
- Concluding Remarks

# Motivation

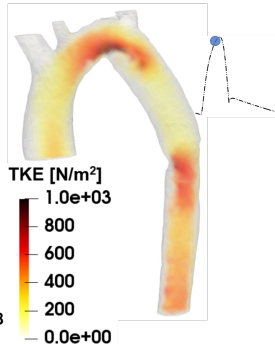
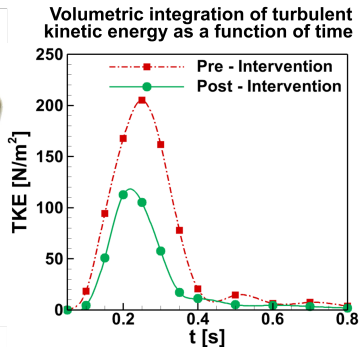
- Previous studies:
  - ▶ Survey developers [6, 14, 17].
  - ▶ Mining [5, 22].
  - ▶ Recruit broadly, or by prog lang [17], or role of developer [13].
  - ▶ Case studies [3, 19].
- We focus on:
  - ▶ Contents of repositories (manual and automated),
  - ▶ Interviews with developers,
  - ▶ One domain at a time.
- Include Domain Expert in the assessment.
- Help users select software.
- Learn what works in a domain, and what doesn't.

# LBM Running Example

## Turbulent Kinetic Energy (TKE)



## Turbulent Kinetic Energy (TKE)



# Methodology

From [21]:

1. Identify the domain of interest.
2. List candidate software packages for the domain.
3. Filter the software package list.
4. Measure using the measurement template.
  - ▶ 10 qualities (installability, correctness, reliability, etc.).
  - ▶ 108 measures.
5. Use AHP to rank the software packages.
6. Interview the developers (Semi-structured).
  - ▶ 20 questions.
  - ▶ How they organize their projects?
  - ▶ Their understanding of the users?
  - ▶ Current and past difficulties?
  - ▶ Solutions the team has found or will try?
  - ▶ Use of documentation?
7. Domain analysis.

# Identify Software (RQ1)

Chosen domain should have the following properties:

1. Well-defined and stable theoretical underpinnings.
2. A community of people studying it.
3. Open source options.
4. Approximately 30 candidate packages.

Find candidates via:

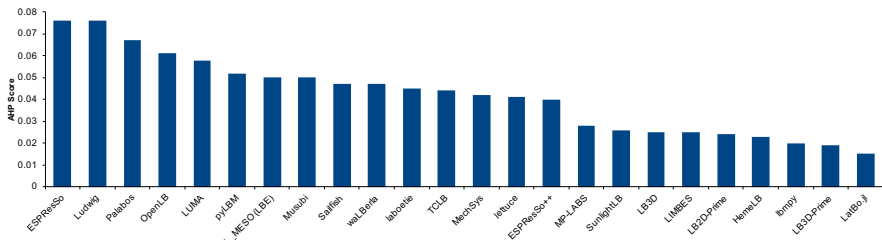
1. Search engine queries.
2. [GitHub](#).
3. [swMATH](#).
4. Scholarly articles.
5. Domain Expert.

Name	Dim	PII	Com	Rflx	MFI	Turb	CGE
DL.MESO	2, 3	MPI/OMP	Y	Y	Y	Y	Y
ESPResSo	1, 2, 3	CUDA/MPI	Y	Y	Y	Y	Y
ESPResSo++	1, 2, 3	MPI	Y	Y	Y	Y	Y
HemeLB	3	MPI	Y	Y	Y	Y	Y
laboetie	2, 3	MPI	Y	Y	Y	Y	Y
LatBo.jl	2, 3	-	Y	Y	Y	N	Y
LB2D-Prime	2	MPI	Y	Y	Y	Y	Y
LB3D	3	MPI	N	Y	Y	Y	Y
...							
Sailfish	2, 3	CUDA	Y	Y	Y	Y	Y
SunlightLB	3	-	Y	Y	N	N	Y
TCLB	2, 3	CUDA/MPI	Y	Y	Y	Y	Y
waLBerla	2, 3	MPI	Y	Y	Y	Y	Y





## Ranking By Best Practices (RQ2)



67% of packages rank in top five for at least one quality

Name	Our Rank	Stars	Star Rank	Watches	Watch Rank
ESPResSo	1	145	2	19	2
Ludwig	2	27	8	6	7
Palabos	3	34	6	GitLab	GitLab
OpenLB	4	N/A	N/A	N/A	N/A
LUMA	5	33	7	12	4
pyLBM	6	95	3	10	5
DL_MESO	7	N/A	N/A	N/A	N/A
Musubi	8	N/A	N/A	N/A	N/A
Sailfish	9	186	1	41	1
...	...	...	...	...	...
LatBo.jl	24	17	10	8	6

# Compare Artifacts to Recommendations (RQ4)

- United States Geological Survey [24]
- DLR Software Guidelines [18]
- Scottish COVID-19 Consortium [2]
- Good Enough Practices in Scientific Computing [28]
- xSDK Community Package Policies [20]
- Trilinos Developer's Guide [7]
- EURISE Network Technical Reference [23]
- CLARIAH Task Force [25]
- Software Quality Assurance Baseline Criteria [16]

Artifact	Count	LBM
LICENSE	8	C
README	7	C
CONTRIBUTING	7	U
CITATION	3	U
CHANGELOG	4	U
INSTALL	4	C
Uninstall	1	R
Dep. List	3	C
Authors	3	C
Code Conduct	1	–
Acknowledge	3	R
Style Guide	4	R
Release Info.	3	U
Prod. Roadmap	3	R

Artifact	Count	LBM
Getting started	4	C
User manual	2	U
Tutorials	1	C
FAQ	3	R
Issue Track	6	C
Version Control	8	C
Build Scripts	6	C
Requirements	3	–
Design Doc.	6	U
API Doc.	4	R
Test Plan	2	U
Test Cases	8	U

# Tools (RQ5)

- Summarize tools
  - ▶ Visible in repos
  - ▶ Mentioned in interviews
- Compare to other research software
  - ▶ Version Control
    - ▶ Almost all software guides recommend
    - ▶ 81% of rsch soft [1]
    - ▶ 67% of LBM software use
  - ▶ Continuous Integration
    - ▶ Many software guides recommend
    - ▶ 70% of popular GitHub projects [8]
    - ▶ 12.5% of LBM software use
    - ▶ 17% of medical image analysis [4]
- Observed LBM tools include code editors, verification tools, build tools, domain specific libraries, collaboration tools, document generation tools, etc.

# Processes (RQ6)

- Literature suggests
  - ▶ Agile philosophy [3, 19]
  - ▶ Amethododical process [10]
  - ▶ Peer review [7, 16, 24]
- LBM
  - ▶ Interviews confirm agile-like
  - ▶ ESPResSO uses peer review

# Pain Points (RQ7, RQ8)

Potential pain points [27, 17, 9]:

- lack of time,
- lack of funding,
- cross-platform compatibility,
- scope bloat,
- lack of user feedback,
- dependency management,
- reproducibility, and
- oracle problem.

Highlights for LBM:

- lack of time,
- lack of funding, and
- difficulty with ensuring correctness.

# SOP for LBM

- Developers are addressing pain points by
  - ▶ designing for change,
  - ▶ circumventing the oracle problem, and
  - ▶ prioritizing documentation and usability.
- For future improvements we suggest:
  - ▶ employing linters,
  - ▶ conducting rigorous peer reviews,
  - ▶ writing and submitting more papers on software,
  - ▶ growing the number of contributors by following current recommendations for open source projects, and
  - ▶ augmenting the theory manuals to include more requirements specification relevant information.



# Threats to Validity

- Reliability
  - ▶ One person measures all packages
  - ▶ Measurements take several months to complete
- Construct Validity
  - ▶ Qualities measured indirectly
  - ▶ Assume high ratio of comments improves maintainability
  - ▶ AHP ranking assumes equal weight between qualities
  - ▶ Approximate popularity by stars and watches
- Internal validity
  - ▶ Not all activities will leave a trace in the repos
  - ▶ Small sample of developers
- External validity
  - ▶ Generalization depends on LBM software development being similar to the development of research software in general

# Concluding Remarks

- Measuring a domain is time-consuming
  - ▶ Requires manual work for some steps
  - ▶ Approx 173 person hours per domain
- Benefits
  - ▶ Looking at what developers are doing, not just what they say they are doing
  - ▶ Help users select software
  - ▶ Customized advice for a given domain
  - ▶ Lessons and warnings for other domains
- Future Work
  - ▶ Deeper measures of usability, performance
  - ▶ Meta-analysis

# References I



Yasmin AlNoamany and John A. Borghi.

Towards computational reproducibility: researcher perspectives on the use and sharing of software.

*PeerJ Computer Science*, 4(e163):1–25, September 2018.



Alys Brett, James Cook, Peter Fox, Ian Hinder, John Nonweiler, Richard Reeve, and Robert Turner.

Scottish covid-19 response consortium.

[https://github.com/ScottishCovidResponse/  
modelling-software-checklist/blob/main/software-checklist.md](https://github.com/ScottishCovidResponse/modelling-software-checklist/blob/main/software-checklist.md),  
August 2021.

# References II



Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post.

Software development environments for scientific and engineering software: A series of case studies.

In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 550–559, Washington, DC, USA, 2007. IEEE Computer Society.



Ao Dong.

Assessing the state of the practice for medical imaging software.

Master's thesis, McMaster University, Hamilton, ON, Canada, September 2021.

# References III



A Grannan, K Sood, B Norris, and A Dubey.

Understanding the landscape of scientific software used on high-performance computing platforms.

*The International Journal of High Performance Computing Applications*, 34(4):465–477, 2020.



Jo Erskine Hannay, Carolyn MacLeod, Janice Singer, Hans Petter Langtangen, Dietmar Pfahl, and Greg Wilson.

How do scientists develop and use scientific software?

In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, SECSE '09, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society.

## References IV



Michael A. Heroux, James M. Bieman, and Robert T. Heaphy.  
Trilinos developers guide part II: ASC software quality  
engineering practices version 2.0.  
[https://faculty.csbsju.edu/mheroux/fall2012\\_csci330/  
TrilinosDevGuide2.pdf](https://faculty.csbsju.edu/mheroux/fall2012_csci330/TrilinosDevGuide2.pdf), April 2008.



Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov,  
and Danny Dig.  
Usage, costs, and benefits of continuous integration in  
open-source projects.  
*In 2016 31st IEEE/ACM International Conference on  
Automated Software Engineering (ASE)*, pages 426–437, 2016.



Matthias Katerbow and Georg Feulner.  
Recommendations on the development, use and provision of  
Research Software, March 2018.

# References V



Diane Kelly.

Industrial scientific software: A set of interviews on software development.

*In Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research, CASCON '13*, pages 299–310, Riverton, NJ, USA, 2013. IBM Corp.



D. E. Knuth.

Literate programming.

*The Computer Journal*, 27(2):97–111, 1984.



Anders Logg.

Automating the finite element method.

Technical Report Preprint 2006-01, Finite Element Centre, 2006.

## References VI



Udit Nangia and Daniel S. Katz.

Track 1 Paper: Surveying the U.S. National Postdoctoral Association Regarding Software Use and Training in Research. pages 1–6. Zenodo, June 2017.

This paper was submitted to WSSSPE5.1 -

<http://wssspe.researchcomputing.org.uk/wssspe5-1/> The final accepted version is

<https://doi.org/10.6084/m9.figshare.5328442>.



Luke Nguyen-Hoan, Shayne Flint, and Ramesh Sankaranarayana.

A survey of scientific software development.

In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, pages 12:1–12:10, New York, NY, USA, 2010. ACM.



# References VII



Georg Ofenbeck, Tiark Rompf, and Markus Püschel.  
Staging for generic programming in space and time.  
In *GPCE*, pages 15–28. ACM, 2017.



Pablo Orviz, Álvaro López García, Doina Cristina Duma,  
Giacinto Donvito, Mario David, and Jorge Gomes.  
A set of common software quality assurance baseline criteria  
for research projects, 2017.



Gustavo Pinto, Igor Wiese, and Luis Felipe Dias.  
How do scientists develop and use scientific software? an  
external replication.  
In *Proceedings of 25th IEEE International Conference on  
Software Analysis, Evolution and Reengineering*, pages  
582–591, February 2018.

## References VIII



Tobias Schlauch, Michael Meinel, and Carina Haupt.  
Dlr software engineering guidelines, August 2018.



Judith Segal.

When software engineers met research scientists: A case study.

*Empirical Software Engineering*, 10(4):517–536, October 2005.



Barry Smith, Roscoe Bartlett, and xSDK Developers.  
xsdk community package policies, Dec 2018.



W. Spencer Smith, Jacques Carette, Peter Michalski,  
Ao Dong, and Oluwaseun Owojaiye.

Methodology for assessing the state of the practice for domain  
X.

<https://arxiv.org/abs/2110.11575>, October 2021.

35 pp.

# References IX



Kanika Sood, Anshu Dubey, boyana norris, and Lois Curfman McInnes.

Repository-Analysis of Open-source and Scientific Software Development Projects.

2 2019.



Carsten Thiel.

EURISE network technical reference.

<https://technical-reference.readthedocs.io/en/latest/>, 2020.



USGS.

USGS (united states geological survey) software planning checklist.

[https:](https://www.usgs.gov/media/files/usgs-software-planning-checklist)

[//www.usgs.gov/media/files/usgs-software-planning-checklist](https://www.usgs.gov/media/files/usgs-software-planning-checklist),  
December 2019.

# References X



Maarten van Gompel, Jauco Noordzij, Reinier de Valk, and Andrea Scharnhorst.

Guidelines for software quality, CLARIAH task force 54.100.  
<https://github.com/CLARIAH/software-quality-guidelines/blob/master/softwareguidelines.pdf>, September 2016.



R. C. Whaley, A. Petitet, and J. J. Dongarra.

Automated empirical optimization of software and the ATLAS project.

*Parallel Computing*, 27(1–2):3–35, 2001.



I. S. Wiese, I. Polato, and G. Pinto.

Naming the pain in developing scientific software.

*IEEE Software*, pages 1–1, 2019.

# References XI



Greg Wilson, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, and Tracy K. Teal.

Good enough practices in scientific computing.

*CoRR*, [abs/1609.00037](https://arxiv.org/abs/1609.00037), 2016.