CAS 741 (Development of Scientific Computing Software)

Winter 2025

02 Getting Started

Dr. Spencer Smith

Faculty of Engineering, McMaster University

January 10, 2025

Getting Started

- Administrative details
- Complete course outline
- Problem statements
- More information on Drasil
- Project choice discussion
- Software tools
  - Git, GitLab and GitHub
  - Continuous integration
  - LaTeX
  - Make
- Software engineering for scientific computing literature
- Start projectile example

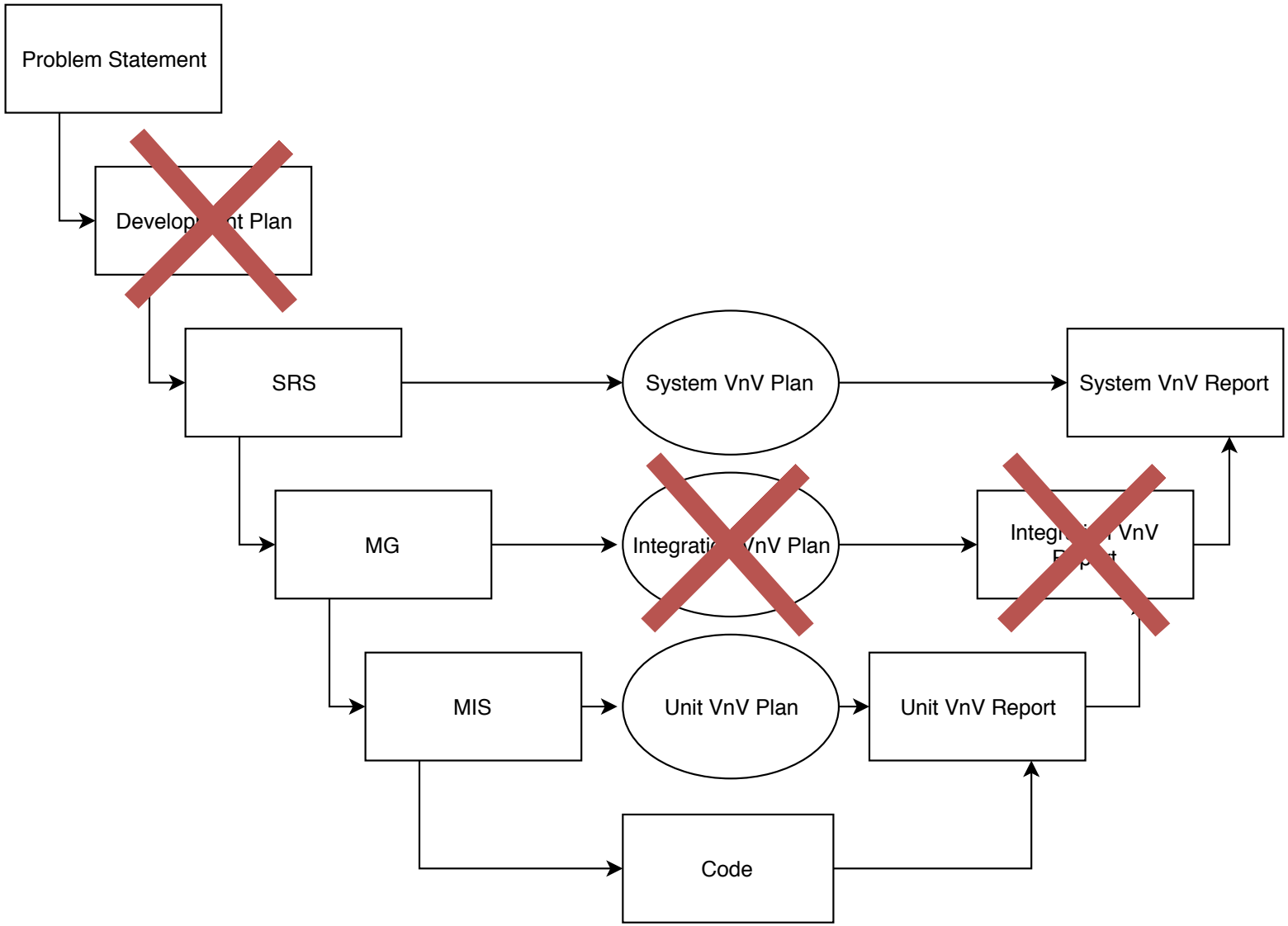
Administrative Details

- Teams channel created
- Can everyone access our course repo on GitLab?
- Create a GitHub account if you don't already have one
- Use the GitHub template to create a new repo
- Add `smiths` to your GitHub repo
- Create a fork (on GitLab) and a merge request to modify Repos.csv with your project details, research level, extra etc.
- Problem statement
  - Problem statement due Fri, Jan 17 by 11:59 pm
  - Assign an issue to instructor to review
- Feel free to add me to you Linked-In network
- Participation grades will be posted before the end of the term, providing an opportunity to improve

Administrative Details

- Domain Expert
  - Create issues for their partner's written deliverables
  - Asks questions during their partner's presentations
- Final documentation grading
  - Most of the grade is decided with the final documentation (60%)
  - Significant weighting toward responding to feedback
  - Use the issue tracker and commits to make tracking changes easy
- Required to use Continuous Integration (CI)

Administrative Details: Our Deliverables



Administrative Details: Report Deadlines

Problem Statement	Week 02	Jan ?19?
System Req. Spec. (SRS)	Week 04	?Feb 2?
System VnV Plan	Week 06	Feb ?16?
MG + MIS	Week 09	Mar ?15?
Drasil Code	Week 09	Mar ?15?
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

SRS	Week 03/04	**Week of Jan 23, 30**?
Syst. VnV	Week 06	Week of Feb 12?
POC Demo	Week 07	Week of Feb 26?
MG + MIS	Week 09	Week of Mar 11?
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
- Time for presentation includes time for questions
- We will have to be strict with the schedule
- Presentations WILL be interrupted with questions/criticism; please do not take it personally
- Any concerns, let the instructor know

Presentation Schedule

Presentation Sched Cont'd

Presentation Schedule

- 3 or 4 presentations each
  - SRS everyone
  - VnV and POC subset of class
  - Design subset of class
  - Implementation everyone
- If you will miss a presentation, please trade with someone
- Implementation presentation could be used to run a code review, or code walkthrough

Challenge Level and Extra Task

- Projects may be designed as "research" projects
- Research projects involve
  - graduate-level domain knowledge
  - a graduate-level implementation challenge
- Non-research projects require an extra task, selected from:
  - Usability testing
  - User manual
  - Rigorous code walkthroughs
  - Formal proof
  - Other (approved by instructor)
- Extras can be used as bonus grades for advanced projects

Problem Statement

- Modify the Problem statement from the template repo
- Written in LaTeX (or other text-based file format)
- Due electronically (on GitHub) by deadline
- Generated files should NOT be under source control (except pdf)
- By default CI builds pdfs when tex files in doc folder change
- Comments used to give advice, you can use for your own reviews
- Remove comments via `\newif\ifcomments\commentsfalse`
- Although not required, you should look at the Development Plan

Problem Statement Cont'd

- Abstractly state the problem to be solved
  - **What** problem
  - **Not how** to solve
- Characterize the problem in terms of inputs and the outputs
- State why the problem is important
- Give context
  - Stakeholders?
  - Environment for the software?
  - A page description should be sufficient
- Research and extra
- Expected implementation decisions

Sample Project Statements

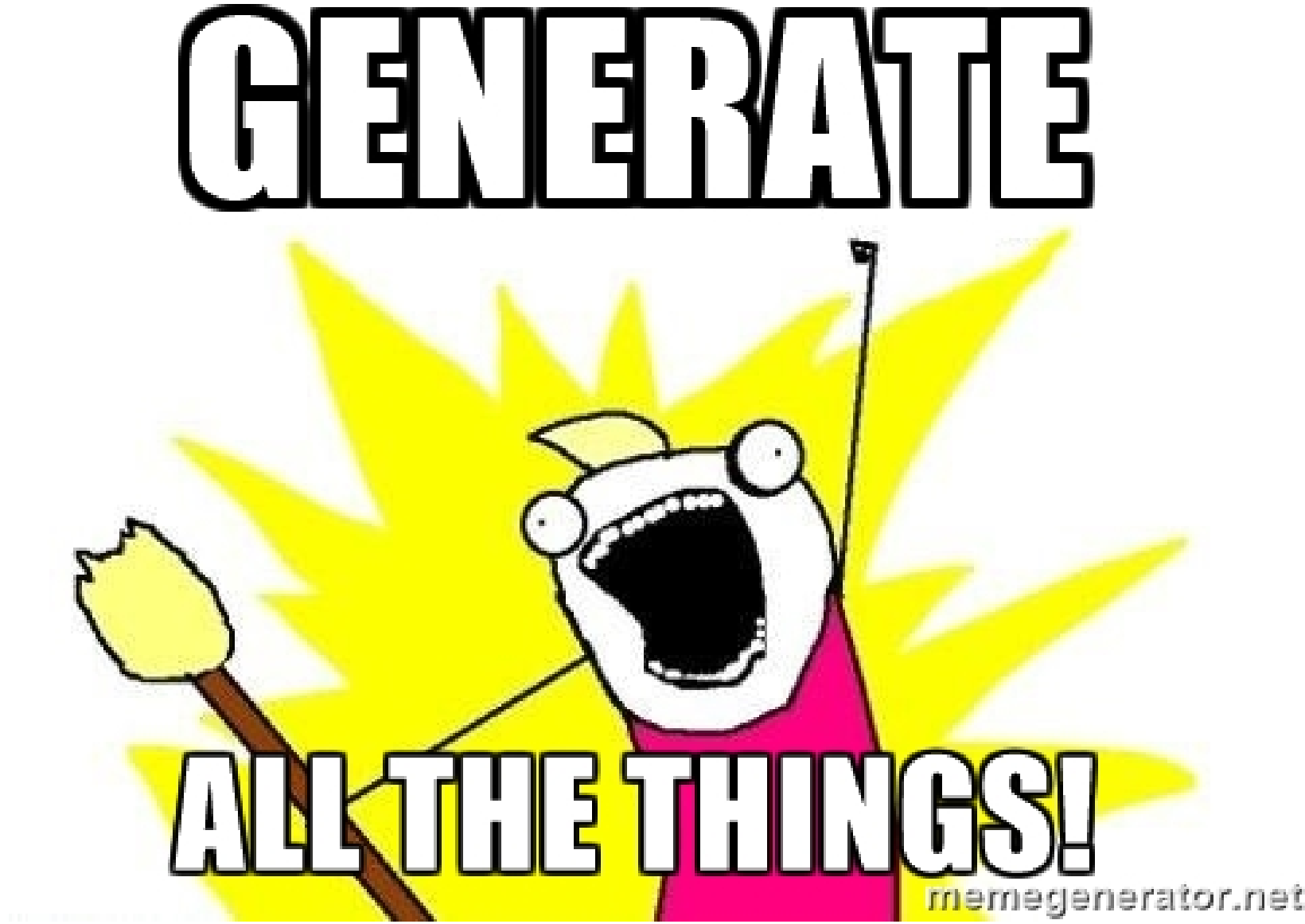
- Solar Cooker
- SpectrumImageAnalysisPy
- Aqueous Speciation Diagram Generator
- FloppyFish
- PyERT - For GPS trip data analysis
- EMA (watch to monitor older adults with lumbar spinal disorders)
- MTO Bridge
- IMU-based Attitude Estimation (2024)
- Bridge Corrosion (2024)
- Synthetic Eddy (2024)

Goals

- Refine problem statement into high level goals
- Selling points for your project (could include you learning new skills)
- Goals should be measurable
- Usually around 5 goals
- Explain goals that are not obvious
- Include goals and stretch goals

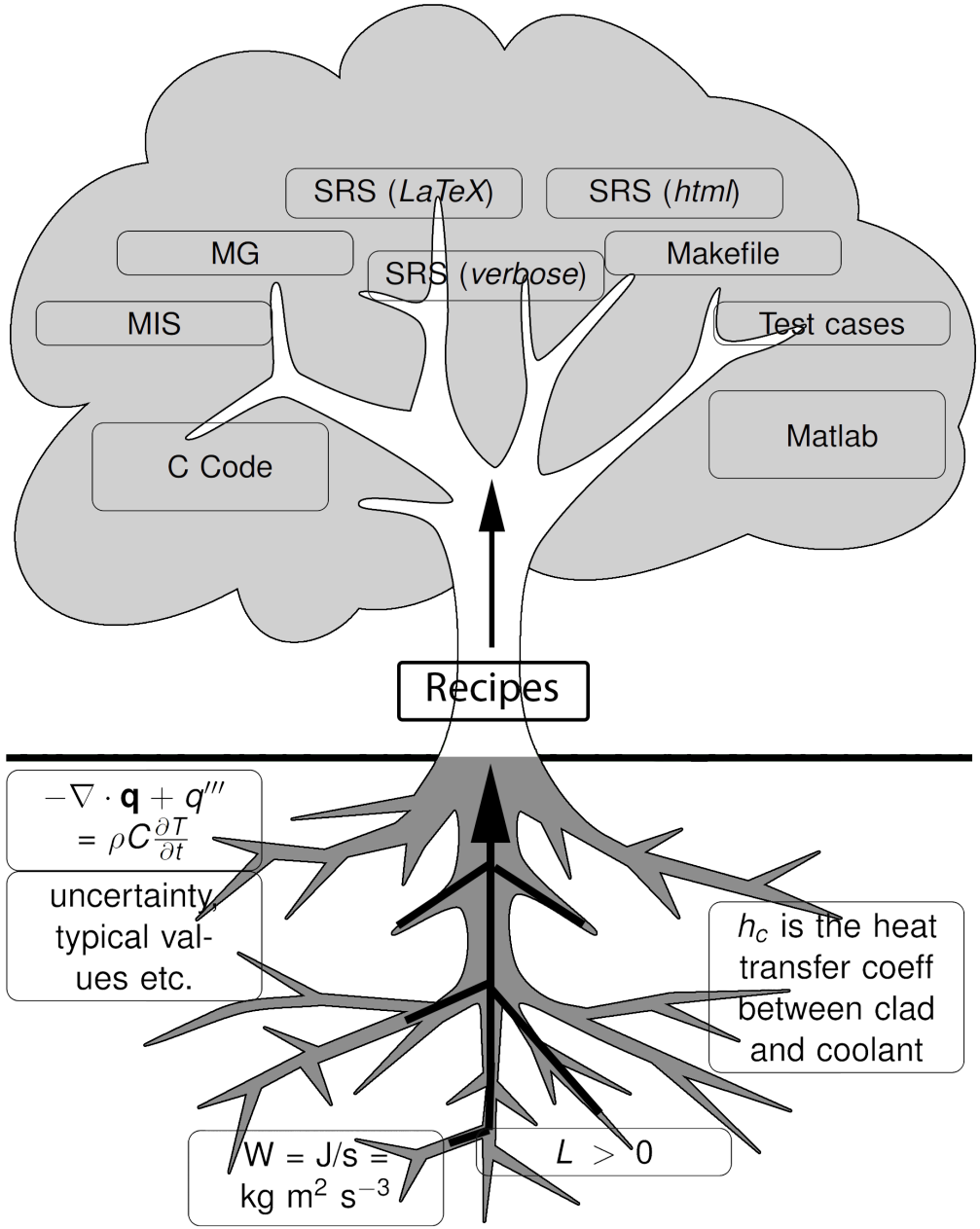
Sample Goals

- Skeleton Key
- Hot Mesh Solutions
- Smart Farm Solutions


GENERATE
ALL THE THINGS!
memegenerator.net

Knowledge Capture

Drasil


SRS (*LaTeX*)
SRS (*html*)
MG
SRS (*verbose*)
Makefile
MIS
Test cases
C Code
Matlab
Recipes
$-\nabla \cdot \mathbf{q} + q''' = \rho C \frac{\partial T}{\partial t}$
uncertainty, typical values etc.
h_c is the heat transfer coeff between clad and coolant
W = J/s = kg m^2 s^{-3}
$L > 0$

Introduction to Drasil

- Drasil uses a generative approach
- Knowledge is captured in a Domain Specific Language (DSL)
- Documentation (in tex and html) and code (in Java, C++, C#, Python and Swift) are generated
- Changes are propagated throughout documentation and code
- Consistency and completeness checks
- Reuse throughout document, between documents and between projects

Getting Started with Drasil

- New workspace setup
- Information on git
- Creating your own project in Drasil

Project Selection Ideas/Questions

Let's discuss some of your project ideas

Tool Tutorials

- Best way to learn is by doing
- Some getting started information and exercises in the ToolTutorials folder, modified from undergrad classes
- Tutorials for se 2aa4 and cs 2me3
- Many other resources on-line
- Your colleagues can help too
- Instructions for setting up a Virtual Machines
- Shared Team's Video on git and GitHub with extra material
- Shared Team's Video on Continuous Integration with extra material

Git, GitLab and GitHub

- Git manages changes to documents
  - Tracks changes
  - Keeps history, you can roll back
  - Useful documentation over time
  - Allows people to work simultaneously
- Benefits for SC [25]
  - Not necessary to make a backup copy of everything, stores just enough information to recreate
  - Do not need to come up with names for backup copies - same file name, but with timestamps
  - Enforces changelog discipline
  - Facilitates identifying conflict and merging changes
- The real bottleneck in scientific computing [26]

Git Typical Usage

First either init repo or clone (git init, git clone), then typical workflow is

1. update repo (git pull)
2. create files
3. stage changes to be committed (git status, git add)
4. commit staged changes (git commit -m "message")
5. push to remote, if using one (git push)

- Commit after every separate issue, and when need to stop working
- Always include a meaningful and descriptive commit message for the log
- If a push reveals conflicts, take appropriate action to merge

GitLab and GitHub Issue Tracking

- See brief document in course repo
- See examples
- Tutorials for se 2aa4 and cs 2me3
- Create an issue

Continuous Integration

- Building and testing software on every push to the code repository (see Fowler)
- Requires:
  - A version control system
  - A fully automated build system
  - An automated test system
  - An automated system for other tasks, like code checking (linting), doc building, web-site updating
  - An integration build system
- A good idea for your projects
- A useful skill to have

LaTeX

- A typesetting language
- Some initial information in course repo
- Tutorials for se 2aa4 and cs 2me3
- Start from an example
  - The lectures notes
  - The Project Template
  - The problem statement

Make

- Software Carpentry: Automation and Make
- The Project Template

Suggested Reading Questions?

- Smith2016 [20]
- SmithEtAl2007 [22]
- ParnasAndClements1986 [14]
- Solar Water Heating System Example

SE For SC Literature

- CAS 741 process is document driven, adapted from the waterfall model [6, 24]
- Many say a document driven process is not used by, nor suitable for, scientific software.
  - Scientific developers naturally use an agile philosophy [1, 4, 5, 17],
  - or an amethododical process [9]
  - or a knowledge acquisition driven process [10].
- Scientists do not view rigid, process-heavy approaches, favourably [4]
- Reports for each stage of development are counterproductive [16, p. 373]
- Up-front requirements are impossible [4, 18]
- What are some arguments in favour of a rational document driven process?

Counter Arguments

- Just because not used, doesn't mean docs shouldn't be
- Documentation provides many benefits [15]:
  - easier reuse of old designs
  - better communication about requirements
  - more useful design reviews
  - easier integration of separately written modules
  - more effective code inspection
  - more effective testing
  - more efficient corrections and improvements.
- Actually faking a rational design process
- Too complex for up-front requirements sounds like an excuse
  - Laws of physics/science slow to change
  - Often simple design patterns
  - Think program family, not individual member
- Debunking myth against up-front requirements [19]

Literature on SE applied to SCS

- Highlights problems with SE
  - Miller2006 [12]
  - Hatton2007 [7]
  - Sleipner A oil rig collapse [13, p. 38]
  - Patriot missile disaster [13, p. 36]
- Highlights gap/chasm between SE and SC
  - Kelly2007 [11]
  - Storer2017 [23]
- Studies of SE applied to SC
  - CarverEtAl2007 [4]
  - Segal2005 [17]

Literature on SE applied to SCS

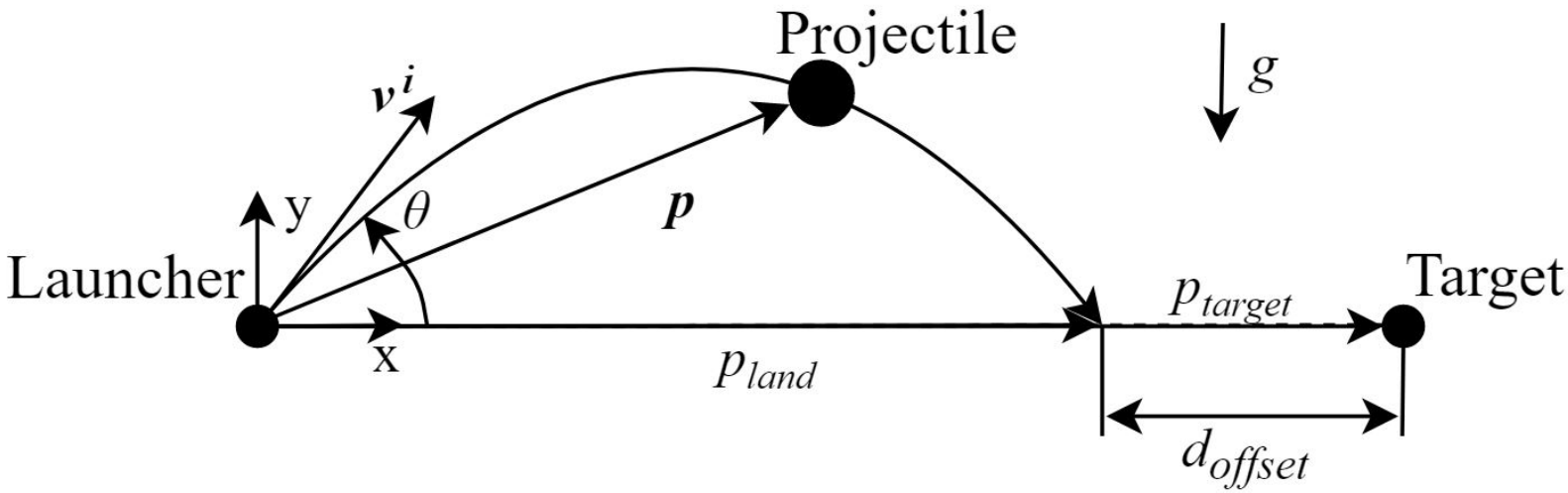
- Reproducibility
  - BaileyEtAl2016 [2]
  - BenureauAndRougier2017 [3]
- Future of SE for SC
  - JohansonAndHasselbring2018 [8]
  - Smith2018 [21]

Recommended Reading Order SRS

- Goal Statement
- Instance Models
- Requirements
- Introduction
- Specific System Description

Probably best to use the same order when doing your requirements presentation

Requirements for Projectile



- Goal(s)?
- Inputs?
- Outputs?
- Simplifying assumptions?
- Kinematic theories for translational motion?
- Refined Theories Projectile SRS

References I

Karen S. Ackroyd, Steve H. Kinder, Geoff R. Mant, Mike C. Miller, Christine A. Ramsdale, and Paul C. Stephenson.
Scientific software development at a research facility.
IEEE Software, 25(4):44–51, July/August 2008.

David H. Bailey, Jonathan M. Borwein, and Victoria Stodden.
Reproducibility: Principles, Problems, Practices, chapter Facilitating reproducibility in scientific computing: principles and practice, pages 205–232.
John Wiley and Sons, New York, 2016.

References II

F. Benureau and N. Rougier.
Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions.
ArXiv e-prints, August 2017.

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.

References III


Steve M. Easterbrook and Timothy C. Johns.
Engineering the software for understanding climate change.
Comuting in Science & Engineering, 11(6):65–74, November/December 2009.


Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli.
Fundamentals of Software Engineering.
Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.


Les Hatton.
The chimera of software quality.
Computer, 40(8), August 2007.

References IV


Arne N. Johanson and Wilhelm Hasselbring.
Software engineering for computational science: Past, present, future.
Computing in Science & Engineering, Accepted:1–31, 2018.


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.

References V


Diane Kelly.
Scientific software development viewed as knowledge acquisition: Towards understanding the development of risk-averse scientific software.
Journal of Systems and Software, 109:50–61, 2015.


Diane F. Kelly.
A software chasm: Software engineering and scientific computing.
IEEE Software, 24(6):120–119, 2007.


Greg Miller.
SCIENTIFIC PUBLISHING: A Scientist's Nightmare: Software Problem Leads to Five Retractions.
Science, 314(5807):1856–1857, 2006.

References VI


Suely Oliveira and David E. Stewart.
Writing Scientific Software: A Guide to Good Style.
Cambridge University Press, New York, NY, USA, 2006.


David L. Parnas and P.C. Clements.
A rational design process: How and why to fake it.
IEEE Transactions on Software Engineering,
12(2):251–257, February 1986.


David Lorge Parnas.
Precise documentation: The key to better software.
In *The Future of Software Engineering*, pages 125–148,
2010.

References VII


Patrick J. Roache.
Verification and Validation in Computational Science and Engineering.
Hermosa Publishers, Albuquerque, New Mexico, 1998.


Judith Segal.
When software engineers met research scientists: A case study.
Empirical Software Engineering, 10(4):517–536, October 2005.


Judith Segal and Chris Morris.
Developing scientific software.
IEEE Software, 25(4):18–20, July/August 2008.

References VIII

Spencer Smith, Malavika Srinivasan, and Sumanth Shankar.
Debunking the myth that upfront requirements are infeasible for scientific computing software.
In *2019 International Workshop on Software Engineering for Science (held in conjunction with ICSE'19)*, pages 1–8, 2019.

W. Spencer Smith.
A rational document driven design process for scientific computing software.
In Jeffrey C. Carver, Neil Chue Hong, and George Thiruvathukal, editors, *Software Engineering for Science*, Chapman & Hall/CRC Computational Science, chapter Examples of the Application of Traditional Software

References IX

Engineering Practices to Science, pages 33–63. Chapman and Hall/CRC, Boca Raton, FL, 2016.


W. Spencer Smith.
Beyond software carpentry.
In *2018 International Workshop on Software Engineering for Science (held in conjunction with ICSE'18)*, pages 1–8, 2018.


W. Spencer Smith, Lei Lai, and Ridha Khedri.
Requirements analysis for engineering computation: A systematic approach for improving software reliability.
Reliable Computing, Special Issue on Reliable Engineering Computation, 13(1):83–107, February 2007.

References X

Tim Storer.
Bridging the chasm: A survey of software engineering practice in scientific programming.
ACM Comput. Surv., 50(4):47:1–47:32, August 2017.

Hans van Vliet.
Software Engineering (2nd ed.): Principles and Practice.
John Wiley & Sons, Inc., New York, NY, USA, 2000.

Greg Wilson, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, and Tracy K. Teal.
Good enough practices in scientific computing.
CoRR, abs/1609.00037, 2016.

References XI


Gregory V. Wilson.
Where's the real bottleneck in scientific computing? Scientists would do well to pick some tools widely used in the software industry.
American Scientist, 94(1), 2006.