CAS 741, CES 741 (Development of Scientific Computing Software)

Fall 2019

02 Getting Started

Dr. Spencer Smith

Faculty of Engineering, McMaster University

September 4, 2019

Getting Started

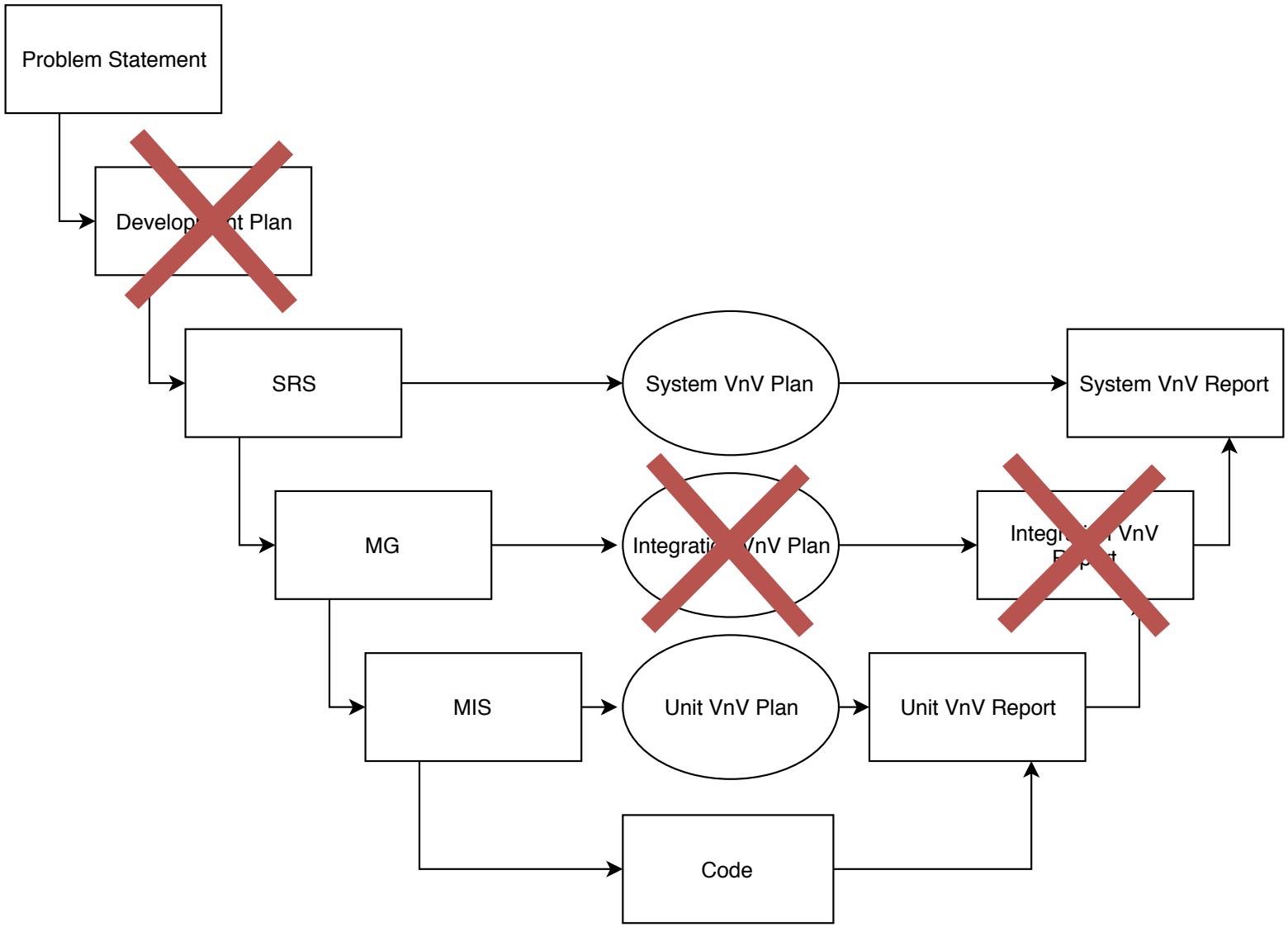
- Administrative details
- Any more introductions?
- Project choices
- Software tools
- Questions on suggested reading?
- Software Engineering for Scientific Computing literature

Administrative Details

- Classes are 1.5 hours long :-)
- Use folder structure given in repo (will be updating)
- Post any questions as issues in our repo
- Problem statement due Friday, Sept 14 by 11:59 pm

Benches and Glassboards

Administrative Details: Our Deliverables



Administrative Details: Grade Assessment

1. Presentations and class discussion 10%
2. Quality of GitHub issues provided to classmates 5%
3. Problem Statement 0%
4. System Requirements Specification (SRS) 20%
5. Verification and Validation (VnV) Plan 15%
   - 5.1 System VnV Plan 10%
   - 5.2 Unit VnV Plan 5%
6. Module Guide (MG) 5%
7. Module Interface Specification (MIS) 15%
8. Final Documentation 30%
   - 8.1 Problem Statement, SRS, System VnV Plan, MG, MIS, Unit VnV Plan (Revised)
   - 8.2 Code
   - 8.3 System VnV Report
   - 8.4 Unit VnV Report

Administrative Details: Report Deadlines

Problem Statement	Week 02	Sept 14
System Requirements Specification (SRS)	Week 05	Oct 4
System VnV Plan	Week 07	Oct 22
Module Guide (MG)	Week 09	Nov 5
Module Interface Specification (MIS)	Week 11	Nov 19
Unit VnV Plan	Week 13	Dec 3
Final Documentation	Week 14	Dec 10

- 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, please ask
- Two days after each major deliverable, your GitHub issues will be due

Administrative Details: Presentations

SRS Present	Week 04	Week of Sept 24
Syst. VnV Present	Week 06	Week of Oct 15
MG Present	Week 08	Week of Oct 29
MIS Present	Week 10	Week of Nov 12
Unit VnV or Implement Present	Week 12	Week of Nov 26

- Tentative dates
- Specific schedule depends on final class registration and need
- Informal presentations with the goal of improving everyone's written deliverables
- You will be assigned to ask questions of your colleagues

Introductions

- Your name
- Degree program
- Academic background
- Experience with:
  - Scientific computing
  - Continuous math
  - Discrete math
  - Software engineering
  - Software development technology
    - Git
    - GitHub or GitLab
    - LaTeX
    - Make etc.
- What do you hope to get out of this course?

Project Selection: Desired Qualities

- Related to scientific computing
- Simple, but not trivial
- If feasible, select a project related to your research
- Possibly re-implement existing software
- Each student project needs to be unique
- Possibly a specific physical problem
- Possibly a (family of) general purpose tool(s)
- Some examples follow, the links are just places to get started

Project Selection: Specific Physical Problem

- Heated rod
- Heated plate
- Double pendulum
- Rigid body dynamics
- Column buckling
- Damped harmonic oscillator
- Stoichiometric calculations (chemical balance)
- Predator prey dynamics
- Imaging: filters, edge detection etc.
- Medical Imaging
- etc.

Project Selection: Family of General Purpose Tools

- Solution of ODEs
- Solution of $Ax = b$
- Regression
- Interpolation
- Numerical integration
- FFT
- Mesh generation
- Finite element method
- Any chapter from a standard numerical methods textbook
- etc.

Tool Tutorials

- Best way to learn is by doing
- Some getting started information and exercises in the ToolTutorials folder, modified from undergrad classes
- Many other resources on-line
- Your colleagues can help too

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 [24]
  - 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 [25]

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
- Create an issue

LaTeX

- A typesetting language
- Some initial information in course repo
- Start from an example
  - The lectures notes
  - The Blank Project Template
  - The problem statement

Make

- The Blank Project Template

Suggested Reading Questions?

- Smith2016 [19]
- SmithEtAl2007 [21]
- ParnasAndClements1986 [14]
- Solar Water Heating System Example

SE For SC Literature

- CAS 741 process is document driven, adapted from the waterfall model [6, 23]
- 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, favorably [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 document driven is not used, does not mean it will not work
- 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

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 [22]
- 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 [20]

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

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*, chapter Section I – Examples of the Application of Traditional Software Engineering Practices to Science, pages 33–63. Taylor & Francis, 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.

References IX

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.

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.

References X

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.

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.