

# CAS 741 (Development of Scientific Computing Software)

Winter 2023

## 05 Program Families

Dr. Spencer Smith

Faculty of Engineering, McMaster University

January 17, 2023



# Program Families

- Record lecture
- Administrative details
- Questions?
- Finish up on SRS
- Specification Qualities
- Motivation
- Proposed Family Methods
- Family of Linear Solvers
- Other examples (covered in “bonus” slides)
  - ▶ Family of Mesh Generators (in other set of slides)
  - ▶ Family of Material Behaviour Models (in other set of slides)

# Administrative Details

- Did anyone install a VM following these [instructions](#)?
- [Checklist for Problem Statement](#)
- [Repos.xlsx](#)
- [Example of the value of commit numbers and diffs](#)
- Primary reviewers (“Domain experts”) and secondary reviewers
  - ▶ Ask at least one question after presentation
  - ▶ By two days after each major deliverable, create at least 5 GitHub issues
- Creating issues as a reviewer
  - ▶ **No more issues related to document formatting please :-)**
  - ▶ Issues that are questions are fine
  - ▶ [Example of question label](#)
  - ▶ [Example of “at”, and traceability](#)
  - ▶ One issue, per issue
  - ▶ [Connection to commit and pull request](#)

# Administrative Details: Presentations

- Presentations are about 20 minutes
- Informal
- To help you prepare your written document
- Questions from audience
- Grading out of 3
  - ▶ Generate discussion, evidence of prior thought, organized 3/3
  - ▶ Any element missing from above 2/3
  - ▶ Any two elements missing from above 1/3
  - ▶ No presentation 0/3

# Administrative Details: Report Deadlines

## **SRS** Week 06 Oct 8

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

# Administrative Details: Presentations

## **SRS Present**   Week 05   Week of Sept 28

- Informal presentations with the goal of improving everyone's written deliverables
- Primary (domain expert) and secondary reviewers (and others) will ask questions

# Administrative Details: Presentation Schedule

- SRS Present
  - ▶ **Sep 28: Mohamed, Andrea, Tiago, John, Salah**
  - ▶ **Oct 1: Liz, Xingzhi, Leila, Shayan, Naveen**
  - ▶ **Oct 5: Sid, Gaby, Parsa, Ting-Yu, Xuanming**
- Primary and secondary reviewers given in [Repos.xlsx](#) file
- If you cannot make a presentation, please make an arrangement to swap with a colleague

# SRS Presentations

- Project name
- Draft Goal statements
- Draft Assumptions
- Draft Input and output variables (data constraints)
- Draft General Definitions and Theoretical models
- Draft Instanced models
- Questions (from reviewers, instructor, anyone)



# Presentation Guidelines

- Always start by reminding class of your project
  - ▶ State your project's title
  - ▶ List your goal statement(s)
- Don't get distracted by details, most of your audience will not know enough on the domain to question the details
- Presentation slides should be under version control
- Presentation slides should be pushed well before class time
- Show slide num along with total num of slides (x of y)
- The instructor will warn you when you have 5 minutes left

# Presentation Guidelines Continued

- Start all presentation with your project title and goal statements (not everyone in the audience will remember what you are doing)
- Time will go fast
  - ▶ You can gloss over domain details
  - ▶ Focus on the important/interesting things
  - ▶ Focus on questions that you have
  - ▶ You don't have to start at the “beginning” of any documents, get to the good stuff
  - ▶ Every slides should have  $x/\text{total slides}$  on it (so the moderator can judge whether you are going too fast or slow)
- If virtual, try to limit frequency of slide changes and movement

# Examples

- Double Pendulum
- SpectrumImageAnalysisPy
- Conformer Searching using Evolutionary Computation
- Medical Image Segmentation
- Fourier Series
- Library of Lighting Models
- Scanning Transmission Electron Microscopy
- Chemical Speciation

# Questions?

- Questions about SRS?
- Any questions on the [SRS Template](#)?
- Any questions on the [Writing Checklist](#)?
- Any questions on the [SRS Checklist](#)?
- Is  $a = \frac{dv}{dt}$  a TM or a DD?

# Software Requirements Activities

- A software requirement is a description of how the system should behave, or of a system property or attribute
- Requirements should be abstract, unambiguous, complete, consistent, modifiable, verifiable and traceable
- Requirements should express “What” not “How”
- Formal versus informal specification
- Functional versus nonfunctional requirements
- Software requirements specification (SRS)
- Requirements template

# Specification Qualities

- What are the important qualities for a specification?  
What makes a specification a good specification?

# Specification Qualities

- Clear, unambiguous, understandable
- Consistent
- Complete
  - ▶ Internal completeness
  - ▶ External completeness
- Incremental
- Validatable
- Abstract
- Traceable

Summarized in [15, p. 406]

# Clear, Unambiguous, Understandable

- Specification fragment for a word-processor
  - ▶ Selecting is the process of designating areas of the document that you want to work on. Most editing and formatting actions require two steps: first you select what you want to work on, such as text or graphics; then you initiate the appropriate action.
- What are the potential problems with this specification?



# Clear, Unambiguous, Understandable

- Specification fragment for a word-processor
  - ▶ Selecting is the process of designating areas of the document that you want to work on. Most editing and formatting actions require two steps: first you select what you want to work on, such as text or graphics; then you initiate the appropriate action.
- What are the potential problems with this specification?
  - ▶ Can an area be scattered?
  - ▶ Can both text and graphics be selected?

# Clear, Unambiguous, Understandable

- Specification fragment from a real safety-critical system
  - ▶ The message must be triplicated. The three copies must be forwarded through three different physical channels. The receiver accepts the message on the basis of a two-out-of-three voting policy.
- What is a potential problems with this specification?

# Clear, Unambiguous, Understandable

- Specification fragment from a real safety-critical system
  - ▶ The message must be triplicated. The three copies must be forwarded through three different physical channels. The receiver accepts the message on the basis of a two-out-of-three voting policy.
- What is a potential problems with this specification?
  - ▶ Can a message be accepted as soon as we receive 2 out of 3 identical copies, or do we need to wait for receipt of the 3rd

# Unambiguous, Validatable

- Specification fragment for an end-user program
  - ▶ The program shall be user friendly.
- What is a potential problems with this specification?

# Unambiguous, Validatable

- Specification fragment for an end-user program
  - ▶ The program shall be user friendly.
- What is a potential problems with this specification?
  - ▶ What does it mean to be user friendly?
  - ▶ Who is a typical user?
  - ▶ How would you measure success or failure in meeting this requirement?

# Unambiguous, Validatable

- Specification fragment for a linear solver
  - ▶ Given  $A$  and  $b$ , solve the linear system  $Ax = b$  for  $x$ , such that the error in any entry of  $x$  is less than 5 %.
- What is a potential problems with this specification?

# Unambiguous, Validatable

- Specification fragment for a linear solver
  - ▶ Given  $A$  and  $b$ , solve the linear system  $Ax = b$  for  $x$ , such that the error in any entry of  $x$  is less than 5 %.
- What is a potential problems with this specification?
  - ▶ Is  $A$  constrained to be square?
  - ▶ Can  $A$  be singular?
  - ▶ Even if the problem is made completely unambiguous, the requirement cannot be validated.

# Consistent

- Specification fragment for a word-processor
  - ▶ The whole text should be kept in lines of equal length. The length is specified by the user. Unless the user gives an explicit hyphenation command, a carriage return should occur only at the end of a word.
- What is a potential problems with this specification?



# Consistent

- Specification fragment for a word-processor
  - ▶ The whole text should be kept in lines of equal length. The length is specified by the user. Unless the user gives an explicit hyphenation command, a carriage return should occur only at the end of a word.
- What is a potential problems with this specification?
  - ▶ What if the length of a word exceeds the length of the line?

# Same Symbol/Term Different Meaning

- Can you think of some symbols/terms that have different meanings depending on the context?

# Consistent

- Language and terminology must be consistent within the specification
- Potential problem with homonyms, for instance consider the symbol  $\sigma$ 
  - ▶ Represents standard deviation
  - ▶ Represents stress
  - ▶ Represents the Stefan-Boltzmann constant (for radiative heat transfer)
- Changing the symbol may be necessary for consistency, but it could adversely effect understandability
- Potential problem with synonyms
  - ▶ Externally funded graduate students, versus eligible graduate students, versus non-VISA students
  - ▶ Material behaviour model versus constitutive equation

# Complete

- Internal completeness
  - ▶ The specification must define any new concept or terminology that it uses
    - ▶ A glossary is helpful for this purpose
- External completeness
  - ▶ The specification must document all the needed requirements
    - ▶ Difficulty: when should one stop?

# Incremental

- Referring to the specification process
  - ▶ Start from a sketchy document and progressively add details
  - ▶ A document template can help with this
- Referring to the specification document
  - ▶ Document is structured and can be understood in increments
  - ▶ Again a document template can help with this

# Traceable

- Explicit links
  - ▶ Within document
  - ▶ Between documents
- Use labels, cross-references, traceability matrices
- Common sense suggests traceability improves maintainability
- Shows consequence of change
- Minimizes cost of recertification
- Additional advantages
  - ▶ Program comprehension
  - ▶ Impact analysis
  - ▶ Reuse
- Why is traceability important?

# Accuracy Versus Precision



A



B



C



D

What is the distinction between accuracy and precision?

# Program Family Examples





# Program Families

- Can think of general purpose (or multi-purpose) SC software as a program family
- Some examples of physical models are also appropriate for consideration as a family
- A program family is a set of programs where it makes more sense to develop them together as opposed to separately
- Analogous to families in other domains
  - ▶ Automobiles
  - ▶ Computers
  - ▶ ...
- Need to identify the commonalities
- Need to identify the variabilities
- Discussed in general in [5, 11]

# Background

- Program family idea since the 1970s (Dijkstra, Parnas, Weiss, Pohl, ...) - variabilities are often from a finite set of simple options [9, 10, 7]
- Families of algorithms and code generation in SC (Carette, ATLAS, Blitz++, ...) - not much emphasis on requirements [4, 24, 20, 3]
- Work on requirements for SC
  - ▶ Template for a single physical model [17, 16]
  - ▶ Template for a family of multi-purpose tool [12, 14, 13]
  - ▶ Template for a family of physical models [19, 18, 8]

# Motivation

- Requirements documentation
  - ▶ Allows judgement of quality
  - ▶ Improves communication
    - ▶ Between domain experts
    - ▶ Between domain experts and programmers
    - ▶ Explicit assumptions
    - ▶ Range of applicability
- A family approach, potentially including a DSL to allow generation of specialized programs
  - ▶ Improves efficiency of product and process
  - ▶ Facilitates reuse of requirements and design, which improves reliability
  - ▶ Improves usability and learnability
  - ▶ Clarifies the state of the art

# Advantages of Program Families to SC?

- Usual benefits
  - ▶ Reduced development time
  - ▶ Improved quality
  - ▶ Reduced maintenance effort
  - ▶ Increased ability to cope with complexity
- Reusability
  - ▶ Underused potential for reuse in SC
  - ▶ Reuse commonalities
  - ▶ Systematically handle variabilities
- Usability
  - ▶ Documentation often lacking in SC
  - ▶ Documentation part of program family methodology
  - ▶ Create family members that are only as general purpose as necessary
- Improved performance

# Is SC Suited to a Program Family Approach?

Based on criteria from Weiss [2, 22, 23, 6, 21]

- The redevelopment hypothesis
  - ▶ A significant portion of requirements, design and code should be common between family members
  - ▶ Common model of software development in SC is to rework an existing program
  - ▶ Progress is made by removing assumptions
- The oracle hypothesis
  - ▶ Likely changes should be predictable
  - ▶ Literature on SC, example systems, mathematics
- The organizational hypothesis
  - ▶ Design so that predicted changes can be made independently
  - ▶ Tight coupling between data structures and algorithms
  - ▶ Need a suitable abstraction

# Challenges

## 1. Validatable

- ▶ Requirements can be complete, consistent, traceable and unambiguous, but still not validatable
- ▶ Input and outputs are continuously valued variables
- ▶ Correct solution is unknown a priori
- ▶ Given  $dy/dt = f(t, y)$  and  $y(t_0) = y_0$ , find  $y(t_n)$

## 2. Abstract

- ▶ If too abstract, then difficult to meet NFRs for accuracy and speed
- ▶ Assumptions can help restrict scope, but possibly as much work as solving the original problem
  - ▶  $Ax = b$
  - ▶  $x^T Ax > 0, \forall x$
- ▶ Algorithm selection should occur at the design stage

# Challenges (Continued)

## 3. Nonfunctional requirements

- ▶ Proving accuracy requirements with a priori error analysis is a difficult mathematical exercise that generally leads to weak error bounds
- ▶ Context sensitive tradeoffs between NFRs can be difficult to specify
- ▶ Absolute quantitative requirements are often unrealistic

## 4. Capture and Reuse Existing Knowledge

- ▶ Cannot ignore the enormous wealth of information that currently exists
- ▶ A good design will often involve integrating existing software libraries
- ▶ Reuse software and the requirements documentation

# Goal Statements for a Family of Linear Solvers?

What would be a good goal statement for a library of linear solvers?



# Goal Statements for a Family of Linear Solvers

- G1 Given a system of  $n$  linear equations represented by matrix  $A$  and column vector  $b$ , return  $x$  such that  $Ax = b$ , if possible

# Theoretical Model for a Family of Linear Solvers?

- Is the theoretical model a commonality or a variability?
- What is the theoretical model for a family of linear solvers?

# Theoretical Model for a Family of Linear Solvers

Given a square matrix  $A$  and column vector  $b$ , the possible solutions for  $x$  are as follows:

1. A unique solution  $x = A^{-1}b$ , if  $A$  is nonsingular
2. An infinite number of solutions if  $A$  is singular and  $b \in \text{span}(A)$
3. No solution if  $A$  is singular and  $b \notin \text{span}(A)$

[1]

# Instance Model for a Family of Linear Solvers?

- Is there an instance model for a family of linear solvers?

# Symbols and Terminology for a Family of Linear Solvers?

- What symbols and terminology will you need to define?

# Sample Symbols and Terminology

$n : \mathbb{N}$	number of linear equations/number of unknowns
$A : \mathbb{R}^{n \times n}$	$n \times n$ real matrix
$x : \mathbb{R}^{n \times 1}$	$n \times 1$ real column vector
$b : \mathbb{R}^{n \times 1}$	$n \times 1$ real column vector
$I : \mathbb{R}^{n \times n}$	an $n \times n$ matrix where all entries are 0, except for the diagonal entries, which are 1
$\ v\ $	the norm (estimate of magnitude) of vector $v$
$A^{-1} : \mathbb{R}^{n \times n}$	the inverse matrix, with the property that $A^{-1}A = I$
singular	matrix $A$ is singular if $A^{-1}$ does not exist
residual	$\ b - Ax\ $

# What Would be the Most General Binding Time?

- What would be the most general binding time for the variabilities?

# What Are Some Potential Input Variabilities?

- What are some potential input variabilities? What are the associated parameters of variation?



<b>Variability</b>	<b>Parameter of Variation</b>
Allowed structure of $A$	Set of { full, sparse, banded, tridiagonal, block triangular, block structured, diagonal, upper triangular, lower triangular, Hessenberg }
Allowed definiteness for $A$	Set of { not definite, positive definite, positive semi-definite, negative definite, negative semi-definite }
Allowed class of $A$	Set of { diagonally dominant, Toeplitz, Vandermonde }
Symmetric?	boolean
Values for $n$	set of $\mathbb{N}$
Entries in $A$	set of $\mathbb{R}$
Entries in $b$	set of $\mathbb{R}$

<b>Variability</b>	<b>Parameter of Variation</b>
Source of input	Set of { from a file, through the user interface, passed in memory }
Encoding of input	Set of {binary, text }
Format of input $A$	Set of {arbitrary, by row, by column, by diagonal }
Format of input $b$	Set of {arbitrary, ordered }

# What Are Some Potential Output Variabilities?

- What are some potential output variabilities? What are the associated parameters of variation?

# Output Variabilities

<b>Variability</b>	<b>Parameter of Variation</b>
Destination for output $x$	Set of { to a file, to the screen, to memory }
Encoding of output $x$	Set of {binary, text }
Format of output $x$	Set of {arbitrary, ordered }
Output residual	boolean (true if the program returns the residual)
Possible entries in $x$	set of $\mathbb{R}$

# What Are Some Potential Calculation Variabilities?

- What are some potential calculation variabilities? What are the associated parameters of variation?

# Calculation Variabilities

<b>Variability</b>	<b>Parameter of Variation</b>
Check input?	boolean (false if the input is assumed to satisfy the input assumptions)
Exceptions generated?	boolean (false if the goal is non-stop arithmetic)
Norm used for residual	Set of {1-norm, 2-norm, $\infty$ -norm }

# References I



Howard Anton.

*Elementary Linear Algebra.*

Wiley, fifth edition, 1987.



Mark Ardis and David M. Weiss.

Defining families: The commonality analysis.

*In Proceedings of the Nineteenth International Conference on Software Engineering*, pages 649–650. ACM, Inc., 1997.



Blitz.

Blitz++, object-oriented scientific computing, Last Accessed in December 2001.

# References II



Jacques Carette.

Gaussian elimination: A case study in efficient genericity with MetaOCaml.

*Science of Computer Programming*, 62(1):3–24, 2006.



Paul Clements and Linda M. Northrop.

*Software product lines: practices and patterns*.

Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.



David A. Cuka and David M. Weiss.

Specifying executable commands: An example of FAST domain engineering.

*Submitted to IEEE Transactions on Software Engineering*, pages 1 – 12, Submitted 1997.



# References III



Edsger W. Dijkstra.

Notes on structured programming.

In O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, editors, *Structure Programming*, pages 1–82. Academic Press Ltd., London, UK, UK, 1972.



John McCutchan.

A generative approach to a virtual material testing laboratory.

Master's thesis, McMaster University, Hamilton, ON, Canada, September 2007.

# References IV



David Parnas.

On the design and development of program families.

*IEEE Transactions on Software Engineering*, SE-2(1):1–9, 1976.



David L. Parnas.

Designing software for ease of extension and contraction.

*IEEE Transactions on Software Engineering*, pages 128–138, March 1979.



K. Pohl, G. Böckle, and F. van der Linden.

*Software Product Line Engineering: Foundations, Principles, and Techniques.*

Springer-Verlag, 2005.

# References V



W. Spencer Smith.

Systematic development of requirements documentation for general purpose scientific computing software.

In *Proceedings of the 14th IEEE International Requirements Engineering Conference, RE 2006*, pages 209–218, Minneapolis / St. Paul, Minnesota, 2006.



W. Spencer Smith and Chien-Hsien Chen.

Commonality analysis for mesh generating systems.

Technical Report CAS-04-10-SS, McMaster University, Department of Computing and Software, 2004.

45 pp.

# References VI



W. Spencer Smith and Chien-Hsien Chen.  
Commonality and requirements analysis for mesh  
generating software.

In F. Maurer and G. Ruhe, editors, *Proceedings of the  
Sixteenth International Conference on Software  
Engineering and Knowledge Engineering (SEKE 2004)*,  
pages 384–387, Banff, Alberta, 2004.



W. Spencer Smith and Nirmitha Koothoor.  
A document-driven method for certifying scientific  
computing software for use in nuclear safety analysis.

*Nuclear Engineering and Technology*, 48(2):404–418, April  
2016.

# References VII



W. Spencer Smith and Lei Lai.

A new requirements template for scientific computing.

In J. Ralyté, P. Ågerfalk, and N. Kraiem, editors,  
*Proceedings of the First International Workshop on  
Situational Requirements Engineering Processes –  
Methods, Techniques and Tools to Support  
Situation-Specific Requirements Engineering Processes,  
SREP'05*, pages 107–121, Paris, France, 2005. In  
conjunction with 13th IEEE International Requirements  
Engineering Conference.

# References VIII



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.



W. Spencer Smith, John McCutchan, and Jacques Carette.

Commonality analysis of families of physical models for use in scientific computing.

In *Proceedings of the First International Workshop on Software Engineering for Computational Science and Engineering (SECSE 2008)*, Leipzig, Germany, May 2008. In conjunction with the 30th International Conference on Software Engineering (ICSE).

# References IX

8 pp.



W. Spencer Smith, John McCutchan, and Jacques Carette.

Commonality analysis for a family of material models.  
Technical Report CAS-17-01-SS, McMaster University,  
Department of Computing and Software, 2017.



Todd. L. Veldhuizen.

Arrays in Blitz++.

*In Proceedings of the 2nd International Scientific  
Computing in Object-Oriented Parallel Environments  
(ISCOPE'98), Lecture Notes in Computer Science.*  
Springer-Verlag, 1998.

# References X



D. Weiss and C.T.R. Lai.

*Software Product Line Engineering: A Family-Based Software Development Process.*

Addison-Wesley, 1999.



David M. Weiss.

Defining families: The commonality analysis.

*Submitted to IEEE Transactions on Software Engineering,*  
1997.



David M. Weiss.

Commonality analysis: A systematic process for defining families.

*Lecture Notes in Computer Science, 1429:214–222, 1998.*



# References XI



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.