CAS 741 (Development of Scientific Computing Software)

Winter 2025

05 Program Families

Dr. Spencer Smith

Faculty of Engineering, McMaster University

January 21, 2025

McMaster University

Program Families

- Administrative details
- Motivating a document driven design process
- Finish up SRS Example
- Requirements specification qualities
- Motivation for families
- 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

- Not all projects approved (Repos.csv)
- All rubrics available on Avenue (please verify you can see them)

Administrative Details: Report Deadlines

Problem Statement	Week 02	Jan 17
System Req. Spec. (SRS)	Week 04	Jan 31
System VnV Plan	Week 06	Feb 14
MG + MIS	Week 09	Mar 14
Drasil Code	Week 09	Mar 14
Final Documentation	Week 13	Apr 11

- 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
Syst. VnV	Week 06
POC Demo	Week 07
MG + MIS	Week 09
Drasil	Week 11
Unit VnV/Implement	Week 12

- 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

- SRS Present (15 min)
  - ▶ **Jan 24: Uriel, Yasmin, Christopher, Baptiste, Kiran**
  - ▶ **Jan 28: Bo, Joe, Junwei, Aliyah, Ziyang**
  - ▶ **Jan 31: Yuanqi, Qianlin, Phillip, Hussein, Yinying**
  - ▶ **Feb 4: Ananda, Alaap, Yuchen**
- Syst V&V Plan Present (L11, L12) (20 min)
  - ▶ Feb 11: Yasmin, Qianlin, Junwei, Joe
  - ▶ Feb 14: Yinying, Bo, Hussein, Volunteer?
- Proof of Concept Demonstrations (L14) (20 min)
  - ▶ Feb 28: Ananda, Uriel, Yuchen, Christopher
- MG+MIS Present (L17, L18) (20 minutes)
  - ▶ Mar 11: Ziyang, Aliyah, Yuanqi, Alaap
  - ▶ Mar 14: Phillip, Baptiste, Kiran, Volunteer?

Presentation Sched Cont'd

- Implementation Present (L22 – L25) (15 min each)
    - ▶ Mar 28: Yasmin, Aliyah, Uriel, Ziyang, Yuanqi
    - ▶ Apr 1: Ananda, Christopher, Yuchen, Bo, Joe
    - ▶ Apr 4: Junwei, Hussein, Kiran, Alaap, Qianlin
    - ▶ Apr 8: Yinying, Baptiste, Phillip

Presentation Schedule

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

Administrative Details: Presentations

- Presentation length depends (15-20 min)
- Informal
- HDMI connection in room
- To help you prepare your written document
- Questions from audience
- Grading out of 3
  - Generate discussion, evidence of thought, organized 3/3
  - Any element missing from above 2/3
  - Any two elements missing from above 1/3
  - No presentation 0/3

SRS Presentations

- Project name
- Example of calculation
- 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 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/total slides on it (so the moderator can judge whether you are going too fast or slow
- Presentation slides should be under version control
- Presentation slides should be pushed well before class time
- The instructor will warn you when you have 5 minutes left

GitHub Issues

- Reviewers GitHub issues
  - ▶ **Please limit the number of issues related to formatting :-)**
  - ▶ 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
  - ▶ screenshots, links, full description
- Pull request of Repos.csv for suggested domain reviewer
  - ▶ You can discuss with colleagues and find your own domain reviewer
  - ▶ Otherwise one will be assigned
  - ▶ Due Jan 29

GitHub Reviews

- Project owners
  - ▶ "Domain Expert" and "Secondary Reviewer"
  - ▶ Find your reviewers in Repos.csv
  - ▶ Add your reviewers as collaborators
  - ▶ Assign review issues to myself and your reviewers
  - ▶ **separate** issue for each reviewer
- Reviewers
  - ▶ Provide at least 5 issues on the document
  - ▶ Grading
    - ▶ Not enough issues, or poor issues 0/2
    - ▶ Enough issues, but shallow 1/2
    - ▶ Enough issues and deep (not surface) 2/2
  - ▶ Issues are due 2 days after being assigned
- GitHub conventions (Writing Checklist (last sect.))
- How to give (and take) constructive criticism

Motivation: Solar Water Heating System

- Given code for Solar Water Heating System
- Asked to
  - ▶ Verify that the code is correct
  - ▶ Supervise a team maintaining the code
  - ▶ Add thermal cooling from the tank
  - ▶ Create a new code for the thermal analysis of reactor fuel pins

Accompanying "Specification"

A Journal Paper

Collector: The Hottel-Whillier-Bliss equation is used to model a flat plate collector (Duffie and Beckman, 1991). The outlet temperature of the collector is given by:

$$T_o = T_{in} + \frac{F_R A_c}{\dot{m} C_p} ((\tau\alpha)G - U_L(T_{in} - T_{amb})) \tag{1}$$

where T_o is the outlet temperature of the collector (=inlet temperature to the storage tank), T_{in} is the inlet temperature to the collector (=outlet temperature from the storage tank), F_R is the heat removal factor which is a function of the collector mass flow rate, T_{amb} is the ambient temperature and $(\tau\alpha)$ is the transmittance absorbance product and it is taken to be a constant of 0.8 (Mather et al., 2002).

Examples, Checklist and Template

- Projectile Example
- GlassBR Example
- SWHS Example (Let's go through this one)
- Blank SRS from Template (Let's look at the advice/hints)
- Checklist

Questions?

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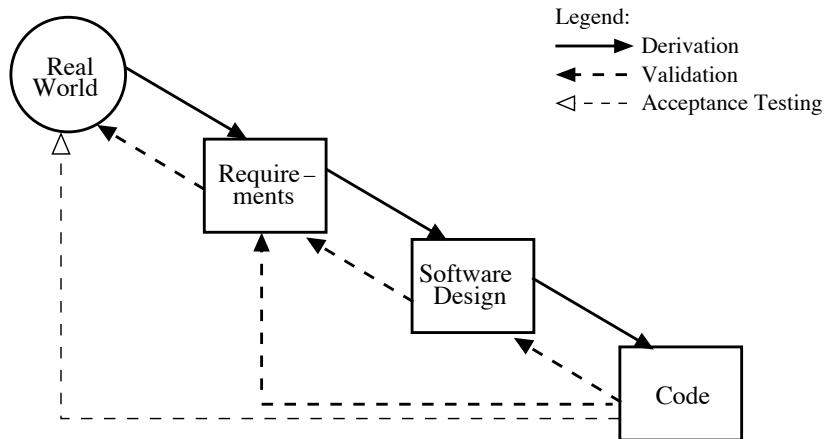- Is $a = \frac{dv}{dt}$ a TM or a DD?

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)
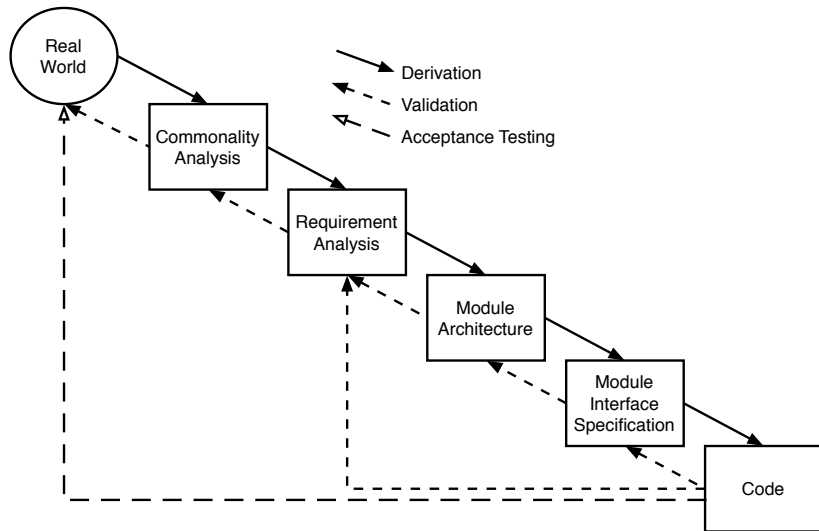- Requirements template

Outline of Discussion of Requirements

- System Requirements Specification and template for beam analysis software
  - ▶ Provides guidelines
  - ▶ Eases transition from general to specific
  - ▶ Catalyses early consideration of design
  - ▶ Reduces ambiguity
  - ▶ Identifies range of model applicability
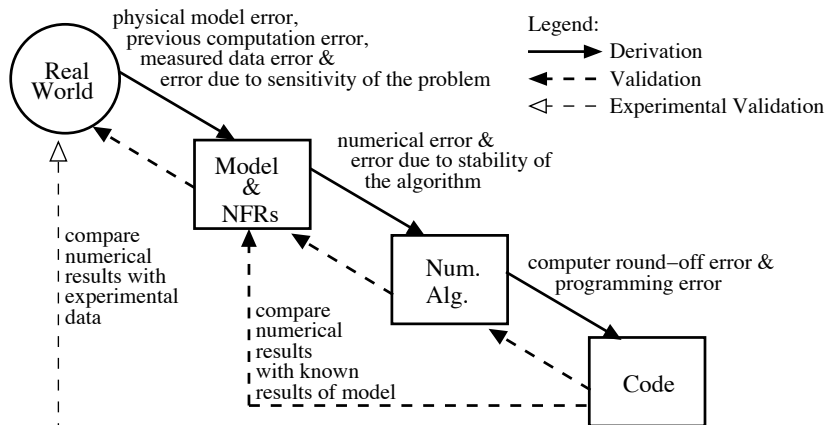 - ▶ Clear documentation of assumptions

A Rational Design Process

Legend:
→ Derivation
◄- - - Validation
◁ - - - Acceptance Testing

Sometimes Include Commonality Analysis

Real World → Commonality Analysis → Requirement Analysis → Module Architecture → Module Interface Specification → Code

Derivation (solid arrow)
Validation (dashed arrow)
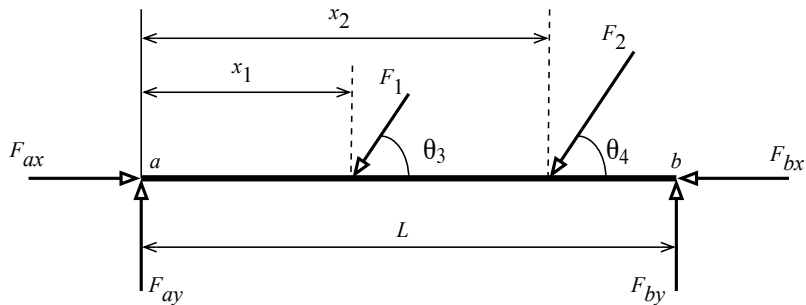Acceptance Testing (dashed open arrow)

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

Why Requirements Analysis?

Legend:
→ Derivation
◄- - - Validation
◁ - - Experimental Validation

Real World

physical model error,
previous computation error,
measured data error &
error due to sensitivity of the problem

Model & NFRs

numerical error &
error due to stability of
the algorithm

Num. Alg.

computer round−off error &
programming error

Code

compare numerical results with experimental data

compare numerical results with known results of model

Beam Analysis Software

Proposed Template

From [20]

1. Reference Material: a) Table of Symbols ...

2. Introduction: a) Purpose of the Document; b) Scope of the Software Product; c) Organization of the Document.

3. General System Description: a) System Context; b) User Characteristics; c) System Constraints.

4. Specific System Description:

   4.1 Problem Description: i) Background Overview ...
   4.2 Solution specification: i) Assumptions; ii) Theoretical Models; ...
   4.3 Non-functional Requirements: i) Accuracy of Input Data; ii) Sensitivity ...

5. Traceability Matrix

6. List of Possible Changes in the Requirements

7. Values of Auxiliary Constants

Provides Guidance

- Details will not be overlooked, facilitates multidisciplinary collaboration
- Encourages a systematic process
- Acts as a checklist
- Separation of concerns
  - ▶ Discuss purpose separately from organization
  - ▶ Functional requirements separate from non-functional
- Labels for cross-referencing
  - ▶ Sections, physical system description, goal statements, assumptions, etc.
  - ▶ PS1.a "the shape of the beam is long and thin"

Eases Transition from General to Specific

- "Big picture" first followed by details
- Facilitates reuse
- "Introduction" to "General System Description" to "Specific System Description"
- Refinement of abstract goals to theoretical model to instanced model
  - ▶ **G1**. Solve for the unknown external forces applied to the beam
  - ▶ **T1** $\sum F_{xi} = 0$, $\sum F_{yi} = 0$, $\sum M_i = 0$
  - ▶ **M1** $F_{ax} - F_1 \cdot \cos\theta_3 - F_2 \cdot \cos\theta_4 - F_{bx} = 0$

Ensures Special Cases are Considered

H_2

$S_{unkF} \notin \mathbb{P}_3$	-
$S_{unkF} = \{@F_{ax}, @F_{bx}, @F_{ay}\}$	-
$S_{unkF} = \{@F_{ax}, @F_{ay}, @F_1\}$	$x_1 \neq 0 \wedge \theta_3 \neq 0 \wedge \theta_3 \neq 180$
	otherwise

H_1 / G

	$S_{GET} = S_{sym} - S_{unkF}$	$S_{GET} \neq (S_{sym} - S_{unkF})$
	$(ErrorMsg' = InvalidUnknown) \wedge ChangeOnly(ErrorMsg)$	
	$ErrorMsg' = NoSolution \wedge ChangeOnly(ErrorMsg)$	$FALSE$
	$F'_{ax} = \frac{-\cos\theta_3 F_2 x_2 \sin\theta_4 + \cos\theta_3 F_{by} L + F_2 \cos\theta_4 x_1 \sin\theta_3 + F_{bx} x_1 \sin\theta_3}{x_1 \sin\theta_3}$ $\wedge$ $F'_{ay} = -\frac{F_2 x_2 \sin\theta_4 - F_{by} L - F_2 \sin\theta_4 x_1 + F_{by} x_1}{x_1}$ $\wedge F'_1 = \frac{-F_2 x_2 \sin\theta_4 + F_{by} L}{x_1 \sin\theta_3} \wedge ChangeOnly(S_{unkF})$	
	$(ErrorMsg' = Indeterminant) \wedge ChangeOnly(ErrorMsg)$	

Catalyses Early Consideration of Design

- Identification of significant issues early will improve the design
- Section for considering sensitivity
  - ▶ Conditioning?
  - ▶ Buckling of beam
- Non-functional requirements
  - ▶ Tradeoffs in design
  - ▶ Speed efficiency versus accuracy
- Tolerance allowed for solution: $|\sum F_{xi}|/\sqrt{\sum F_{xi}^2} \leq \epsilon$
- Solution validation strategies (now in a separate document)
- List of possible changes in requirements

Reduces Ambiguity

- Unambiguous requirements allow communication between experts, requirements review, designers do not have to make arbitrary decisions
- Tabular expressions allow automatic verification of completeness
- Table of symbols
- Abbreviations and acronyms
- Scope of software product and system context
- User characteristics
- Terminology definition and data definition
- Ends arguments about the relative merits of different designs

Identifies Range of Model Applicability

- Clear documentation as to when model applies
- Can make the design specific to the problem
- Input data constraints are identified
  - Physically meaningful: $0 \leq x_1 \leq L$
  - Maintain physical description: PS1.a, $0 < h \leq 0.1L$
  - Reasonable requirements: $0 \leq \theta_3 \leq 180$
- The constraints for each variable are documented by tables, which are later composed together
- $(min_f \leq |F_{ax}| \leq max_f) \wedge (|F_{ax}| \neq 0) \Rightarrow$
  $\forall(FF | @FF \in S_F \cdot FF \neq 0 \wedge \frac{max\{|F_{ax}|, |FF|\}}{min\{|F_{ax}|, |FF|\}} \leq 10^{r_f})$

Summary of Variables

Var	Type	Physical Constraints	System Constraints	Prop
x	Real	$x \geq 0 \wedge x \leq L$	$min_d \leq x \leq max_d$	NIV
x_1	Real	$x_1 \geq 0 \wedge x_1 \leq L$	$min_d \leq x_1 \leq max_d$	IN
x_2	Real	$x_2 \geq 0 \wedge x_2 \leq L$	$min_d \leq x_2 \leq max_d$	IN
e	Real	$e > 0 \wedge e \leq h$	$min_e \leq e \leq max_e$	IN
h	Real	$h > 0 \wedge h \leq 0.1L$	$min_h \leq h \leq max_h$	IN
L	Real	$L > 0$	$min_d \leq L \leq max_d$	IN
E	Real	$E > 0$	$min_E \leq E \leq max_E$	IN
θ_3	Real	$-\infty < \theta_3 < +\infty$	$0 \leq \theta_3 \leq 180$	IN
θ_4	Real	$-\infty < \theta_4 < +\infty$	$0 \leq \theta_4 \leq 180$	IN
V	Real	$-\infty < V < +\infty$	-	OUT
M	Real	$-\infty < M < +\infty$	-	OUT
y	Real	$-\infty < y < +\infty$	-	OUT
...	...	...	...	...

Clear Documentation of Assumptions

Phy. Sys. /Goal	Data /Model	Assumption										Model	
		A1	A2	...	A4	...	A8	A9	A10	...	A14	**M1**	...
G1	**T1**	√		...		...	√	√		...		√	...
G2	**T2**	√		...		...	√	√		...			...
G3	**T3**	√		...		...		√	√	...			...
	M1		√	...		...				...		√	...
PS1.a	*L*			...		...			√	...		...	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...

A10. The deflection of the beam is caused by bending moment only, the shear does not contribute.
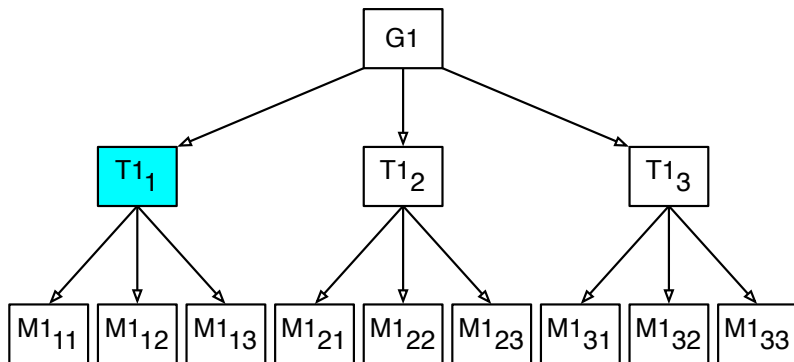
Refinement from Abstract to Concrete
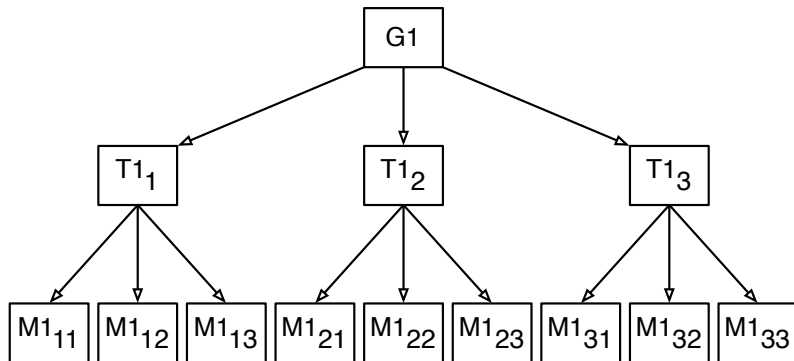
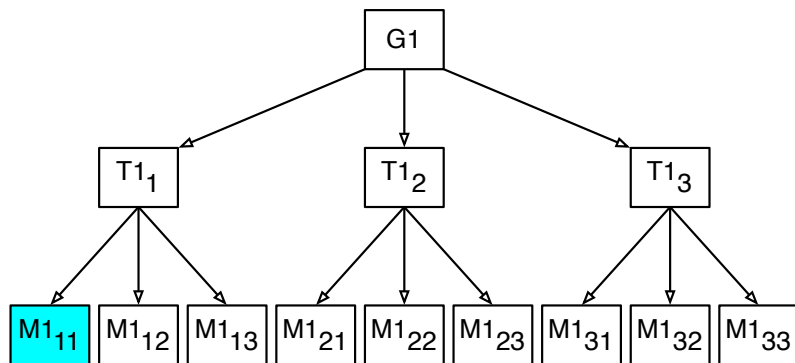# Refinement from Abstract to Concrete
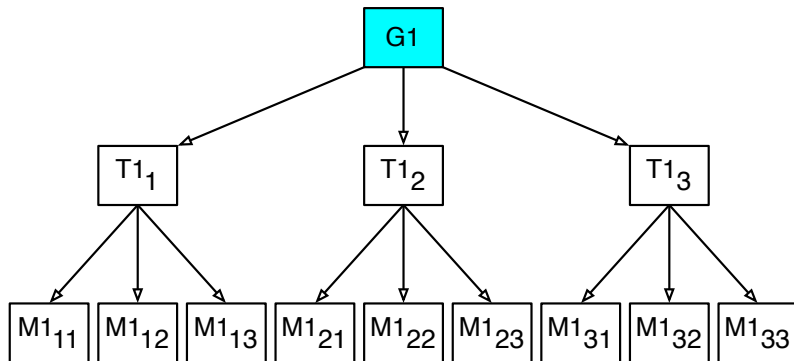
G1: Solve for unknown forces
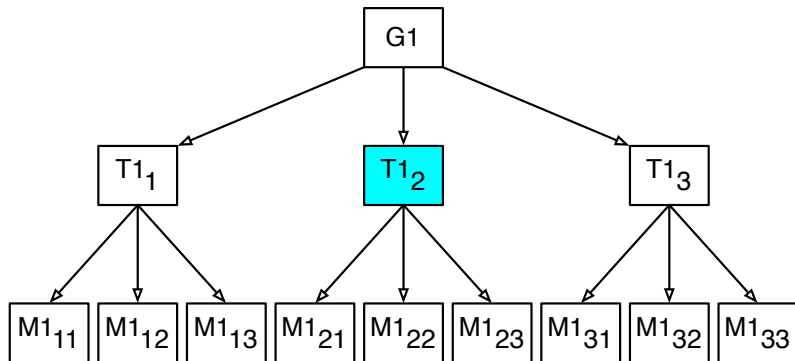
Refinement from Abstract to Concrete

$$(\mathbf{T1}_1) \begin{cases} \sum F_{xi} = 0 \\ \sum F_{yi} = 0 \\ \sum M_i = 0 \end{cases}$$

Refinement from Abstract to Concrete

$$\textbf{(M1)} \begin{cases} F_{ax} - F_1 \cdot \cos\theta_3 - F_2 \cdot \cos\theta_4 - F_{bx} = 0 \\ F_{ay} - F_1 \cdot \sin\theta_3 - F_2 \cdot \sin\theta_4 + F_{by} = 0 \\ -F_1 \cdot x_1 \sin\theta_3 - F_2 \cdot x_2 \sin\theta_4 + F_{by} \cdot L = 0 \end{cases}$$

Refinement from Abstract to Concrete

The virtual work done by all the external forces and couples acting on the system is zero for each independent virtual displacement of the system, or mathematically $\delta U = 0$

Other goals and models

- **G2**: Solve for the functions of shear force and bending moment along the beam
- **G3**: Solve for the function of deflection along the beam
- **T3$_1$**: $\frac{d^2y}{dx^2} = \frac{M}{EI}$, $y(0) = y(L) = 0$
- **T3$_2$**: y determined by moment area method
- **T3$_3$**: y determined using Castigliano's theorem
- **M3$_{11}$**: $y = \frac{12 \int_0^L (\int_0^L M dx) dx}{Eeh^3}$, $y(0) = y(L) = 0$

Kreyman and Parnas Five Variable Model

- See [8]
- An alternative approach
- Unfortunately the numerical algorithm is not hidden in the requirements specification
- The analogy with real-time systems leads to some confusion

Links to Papers

- SmithAndChen2004 [17] Commonality Analysis overview
- SmithAndChen2004b [16] CA example MG
- SmithAndLai2005 [19] New requirements template
- Smith2006 [14] General purpose tool template
- SmithEtAl2007 [20] Template for SC audience
- SmithAndKoothoor2016 [18] Nuclear fuelpin example
- SmithEtAl2019_arXiv [13] Debunk upfront myth
- Smith2016 [15] Overview of artifacts
- KreymanAndParnas2002 [8] 4 variable method

Refined Theories Version of SRS

- Shortcomings of the template
  - ▶ Tedious to write and maintain (addressed by Drasil)
  - ▶ Three levels of refinement is arbitrary
  - ▶ Sloppy for notation (T should be $T(t)$ etc.)
  - ▶ Assumptions are not systematically propagated
- Future refined theories version
  - ▶ Arbitrary levels of refinement
  - ▶ Pre and post conditions for theories
  - ▶ Assumptions are inherited
  - ▶ Proof (rationale) obligations for final theories
  - ▶ Example for noPCM

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 [18, 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 σ
  - ▶ 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, 12]

Background

- Program family idea since the 1970s (Dijkstra, Parnas, Weiss, Pohl, ...) – variabilities are often from a finite set of simple options [10, 11, 7]
- Families of algorithms and code generation in SC (Carette, ATLAS, Blitz++, ...) – not much emphasis on requirements [4, 27, 23, 3]
- Work on requirements for SC
  - ▶ Template for a single physical model [20, 19]
  - ▶ Template for a family of multi-purpose tool [14, 17, 16]
  - ▶ Template for a family of physical models [22, 21, 9]

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, 25, 26, 6, 24]

- 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 $b \in span(A)$

3. No solution if A is singular and $b \notin span(A)$

[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?

Variability	Parameter of Variation
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 $\mathbb{R}$

Variability	Parameter of Variation
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

Variability	Parameter of Variation
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

Variability	Parameter of Variation
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, ∞-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.

📄 K. Kreyman and D. L. Parnas.
On documenting the requirements for computer programs
based on models of physical phenomena.
SQRL Report 1, Software Quality Research Laboratory,
McMaster University, January 2002.

References IV

John McCutchan.
A generative approach to a virtual material testing laboratory.
Master's thesis, McMaster University, Hamilton, ON, Canada, September 2007.

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, SE-5(2):128–138, March 1979.

References V

K. Pohl, G. Böckle, and F. van der Linden.
Software Product Line Engineering: Foundations, Principles, and Techniques.
Springer-Verlag, 2005.

Spencer Smith, Malavika Srinivasan, and Sumanth Shankar.
Debunking the myth that upfront requirements are infeasible for scientific computing software debunking the myth that upfront requirements are infeasible for scientific computing software.
https://arxiv.org/abs/1906.07812, June 2019.

References VI

📄 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.
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 VII

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

📄 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 VIII

📄 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 IX

📄 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 X

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 XI

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 XII

📄 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 XIII

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.