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

Fall 2017

05 Program Families

Dr. Spencer Smith

Faculty of Engineering, McMaster University

September 14, 2018

Program Families

- Administrative details
- Questions?
- Finish up on SRS
- Specification Qualities
- Motivation
- Proposed Family Methods
- Family of Mesh Generators
- Family of Linear Solvers
- Family of Material Behaviour Models

Administrative Details

- Problem statement should be clear on input and output
- Presentations
  - VGA by default, ask if need adapter
  - Can use my laptop
- Do NOT reproduce all of the cas 741 repo in your repo, just the blank project template (moved to the top level)
- Use the same names as the original
- Delete example text from templates
- 80 columns in tex files
- Spell check
- Replace "in order to" by "to"
- Use a `.gitignore` file
- Look at work of class mates

Administrative Details: Deadlines

Problem Statement	Week 02	Sept 15
SRS Present	Week 04	Week of Sept 25
SRS	Week 05	Oct 4
V&V Present	Week 06	Week of Oct 16
V&V Plan	Week 07	Oct 25
MG Present	Week 08	Week of Oct 30
MG	Week 09	Nov 8
MIS Present	Week 10	Week of Nov 13
MIS	Week 11	Nov 22
Impl. Present	Week 12	Week of Nov 27
Final Documentation	Week 13	Dec 6

Administrative Details: Presentation Schedule

- SRS Present
  - Tuesday: Paul, Isobel, Keshav
  - Friday: Devi, Shushen, Xiaoye
- V&V Present
  - Tuesday: Steven, Alexandre P., Alexander S.
  - Friday: Geneva, Jason, Yuzhi
- MG Present
  - Tuesday: Xiaoye, Shushen, Devi, Keshav, Alex P, Paul
  - Friday: Yuzhi, Jason, Geneva, Alex S, Isobel, Steven
- MIS Present
  - Tuesday: Isobel, Keshav, Paul
  - Friday: Shushen, Xiaoye, Devi
- Impl. Present
  - Tuesday: Alexander S., Steven, Alexandre P.
  - Friday: Jason, Geneva, Yuzhi

Questions?

- Questions about problem statements?
- Questions about SRS?

More on the Template

- Why a new template?
- The new template
  - Overview of changes from existing templates
  - Goal $\rightarrow$ Theoretical Model $\rightarrow$ Instanced Model hierarchy
  - Traceability matrix
  - System behaviour, including input constraints

Why a New Template?

From [17, 8]

1. One user viewpoint for the physical model
2. Assumptions distinguish models
3. High potential for reuse of functional requirements
4. Characteristic hierarchical nature facilitates change
5. Continuous mathematics presents a challenge

Overview of the New Template

- Reference Material
- Introduction: a) Purpose of the Document b) Scope of the Software Product c) Organization of the Document
- General System Description: a) System Context b) User Characteristics c) System Constraints
- Specific System Description: a) Problem Description b) Solution Characteristics Specification c) Non-functional Requirements
- Other System Issues
- Traceability Matrix
- List of Possible Changes in the Requirements
- Values of Auxiliary Constants
- References

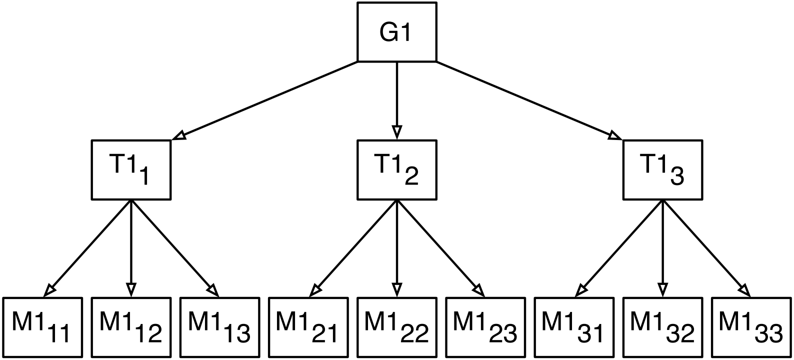
Overview of the New Template

- Reference Material
- Introduction: a) Purpose of the Document b) Scope of the Software Product c) Organization of the Document
- General System Description: a) System Context b) User Characteristics c) System Constraints
- Specific System Description: a) Problem Description b) Solution Characteristics Specification c) Non-functional Requirements
- Other System Issues
- Traceability Matrix
- List of Possible Changes in the Requirements
- Values of Auxiliary Constants
- References

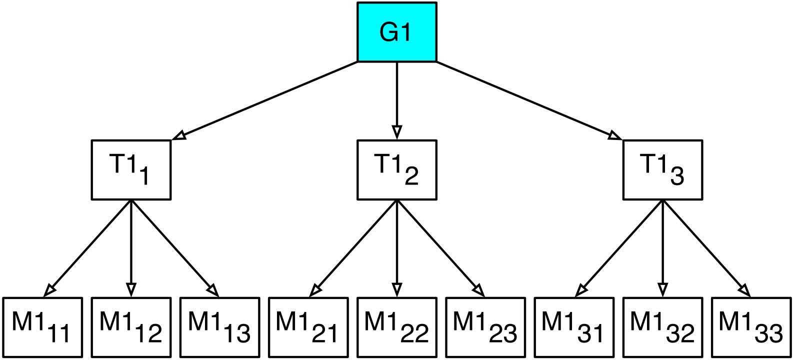
Excerpts from Specific System Description

- Problem Description
  - Physical system description (**PS**)
  - Goals (**G**)
- Solution Characteristics Specification
  - Assumptions (**A**)
  - Theoretical models (**T**)
  - Data definitions
  - Instanced models (**M**)
  - Data constraints
  - System behaviour
- Non-functional Requirements
  - Accuracy of input data
  - Sensitivity of the model
  - Tolerance of the solution
  - Solution validation strategies (now moved to a separate document)

Refinement from Abstract to Concrete

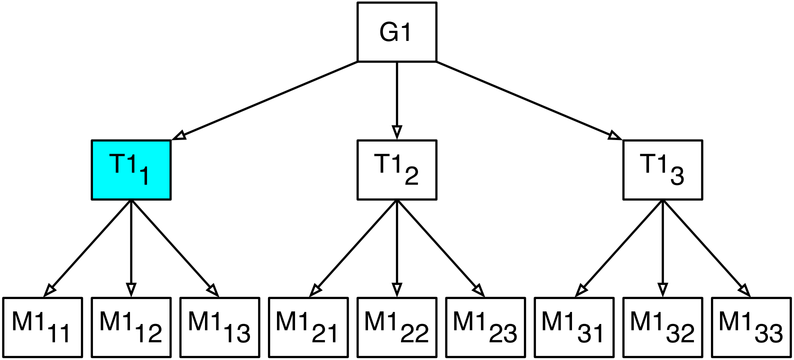


Refinement from Abstract to Concrete



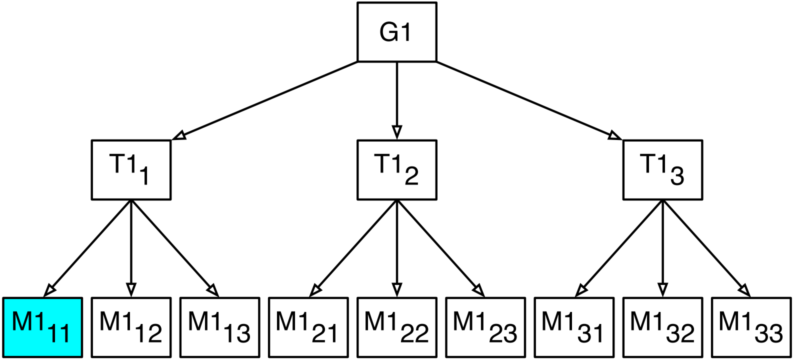
G1: Solve for unknown forces

Refinement from Abstract to Concrete



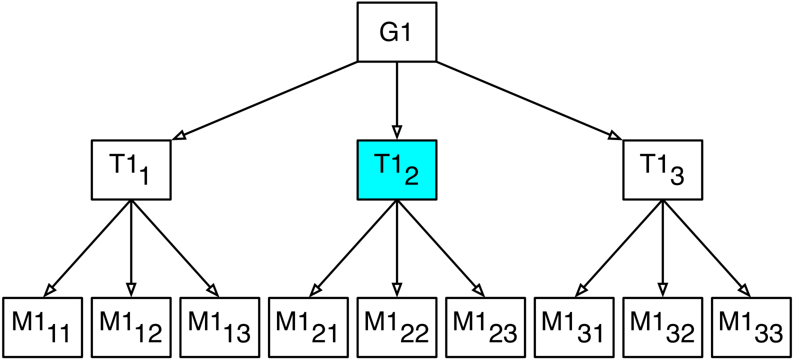
$$(\mathbf{T1_1}) \quad \begin{cases} \sum F_{xi} = 0 \\ \sum F_{yi} = 0 \\ \sum M_i = 0 \end{cases}$$

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
- $\mathbf{T3_1}$: $\frac{d^2y}{dx^2} = \frac{M}{EI}$, $y(0) = y(L) = 0$
- $\mathbf{T3_2}$: y determined by moment area method
- $\mathbf{T3_3}$: y determined using Castigliano's theorem
- $\mathbf{M3_{11}}$: $y = \frac{12 \int_0^L (\int_0^L M dx) dx}{Eeh^3}$, $y(0) = y(L) = 0$

Kreyman and Parnas Five Variable Model

- See [7]
- An alternative approach
- Unfortunately the numerical algorithm is not hidden in the requirements specification
- The analogy with real-time systems leads to some confusion

Examples

- Solar Water Heating System
- GlassBR

Summary of Template

- Quality is a concern for scientific computing software
- Software engineering methodologies can help
- Motivated, justified and illustrated a method of writing requirements specification for engineering computation to improve reliability
- Also improve quality with respect to usability, verifiability, maintainability, reusability and portability
- Tabular expressions to reduce ambiguity, encourage systematic approach
- Conclusions can be generalized because other computation problems follow the same pattern of *Input* then *Calculate* then *Output*
- Benefits of approach should increase as the number of details and the number of people involved increase

Summary of Template (Continued)

- A new template for scientific computing has been developed
- Characteristics of scientific software guided the design
- Designed for reuse
- Functional requirements split into “Problem Description” and “Solution Characteristics Specification”
- Traceability matrix
- Addresses nonfunctional requirements (but room for improvement)

Specification Qualities

- What are the important qualities for a specification?

Specification Qualities

- The qualities we previously discussed (usability, maintainability, reusability, verifiability etc.)
- Clear, unambiguous, understandable
- Consistent
- Complete
  - Internal completeness
  - External completeness
- Incremental
- Validatable
- Abstract
- Traceable

Summarized in [16, 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?

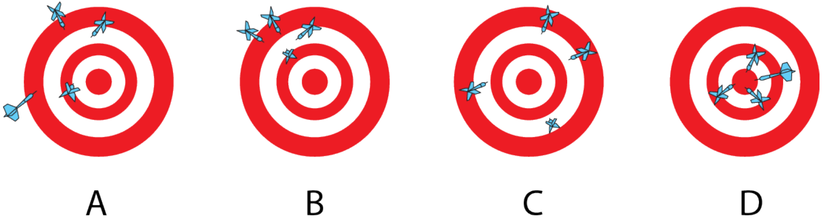
Incremental

- 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 matricies
- Common sense suggests traceability improves maintainability
- Shows consequence of change
- Minimizes cost of recertification
- Additional advantages
  - Program comprehension
  - Impact analysis
  - Reuse

Accuracy Versus Precision



What is the distinction between accuracy and precision?

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 [4, 12]

Background

- Program family idea since the 1970s (Dijkstra, Parnas, Weiss, Pohl, ...) - variabilities are often from a finite set of simple options [10, 11, 6]
- Families of algorithms and code generation in SC (Carette, ATLAS, Blitz++, ...) - not much emphasis on requirements [3, 25, 21, 2]
- Work on requirements for SC
  - Template for a single physical model [18, 17]
  - Template for a family of multi-purpose tool [13, 15, 14]
  - Template for a family of physical models [20, 19, 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 [1, 23, 24, 5, 22]

- 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

References I

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.

Jacques Carette.
Gaussian elimination: A case study in efficient genericity with MetaOCaml.
Science of Computer Programming, 62(1):3–24, 2006.

References II

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

Lei Lai.
Requirements documentation for engineering mechanics software: Guidelines, template and a case study.
Master's thesis, McMaster University, Hamilton, Ontario, Canada, 2004.

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

References V

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 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 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 VII

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 VIII

W. Spencer Smith and Lei Lai.
A new requirements template for scientific computing.
In J. Ralyté, P. Ȧgerfalk, 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 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.

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 X

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 XI

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 XII

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.