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

Fall 2017

06 Program Families Continued

Dr. Spencer Smith

Faculty of Engineering, McMaster University

September 21, 2018

Program Families Continued

- Administrative details
- Questions?
- Proposed Family Methods
- Family of Mesh Generators
- Family of Linear Solvers
- Family of Material Behaviour Models

Administrative Details

- Marking scheme for SRS
  - You should be able to see it in Avenue
  - Do NOT submit on Avenue
- Grade columns in Avenue
  - Presentation questions worth 1%

Administrative Details: Deadlines

SRS or CA Present	Week 04	Week of Sept 24
SRS or CA	Week 05	Oct 4
Syst. VnV Present	Week 06	Week of Oct 15
System VnV Plan	Week 07	Oct 22
MG Present	Week 08	Week of Oct 29
MG	Week 09	Nov 5
MIS Present	Week 10	Week of Nov 12
MIS	Week 11	Nov 19
Unit VnV or Impl. Present	Week 12	Week of Nov 26
Unit VnV Plan	Week 13	Dec 3
Final Doc	Week 14	Dec 10

Administrative Details: Presentation Schedule

- SRS Present
  - Wednesday: Jennifer, Brooks
  - Friday: Vajiheh, Olu, Karol
- Syst V&V Plan Present
  - Wednesday: Malavika, Robert
  - Friday: Hanane
- MG Present
  - Wednesday: Karol, Malavika, Robert, Hanane
  - Friday: Brooks, Vajiheh, Olu, Jennifer
- MIS Present
  - Wednesday: Malavika, Robert
  - Friday: Hanane, Jennifer
- Unit VnV Plan or Impl. Present
  - Wednesday: Brooks, Vajiheh
  - Friday: Olu, Karol

Assigned “Questioners”

1. Jennifer: Brooks, Vajiheh, Olu
2. Brooks: Vajiheh, Olu, Karol
3. Vajiheh: Olu, Karol, Malavika
4. Olu: Karol, Malavika, Robert
5. Karol: Malavika, Robert, Hanane
6. Malavika: Robert, Hanane, Jennifer
7. Robert: Hanane, Jennifer, Brooks
8. Hanane: Jennifer, Brooks, Vajiheh

- Do not hesitate to ask anyone questions
- Consider setting up your own reviewing arrangements for your written documents

Questions?

- Questions about
  - SRS presentations?
  - SRS documents?
  - LaTeX?

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 [12, 18]

Background

- Program family idea since the 1970s (Dijkstra, Parnas, Weiss, Pohl, ...) - variabilities are often from a finite set of simple options [16, 17, 14]
- Families of algorithms and code generation in SC (Carette, ATLAS, Blitz++, ...) - not much emphasis on requirements [8, 33, 29, 6]
- Work on requirements for SC
  - Template for a single physical model [25, 24]
  - Template for a family of multi-purpose tool [21, 23, 22]
  - Template for a family of physical models [28, 27, 15]

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, 31, 32, 13, 30]

- 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

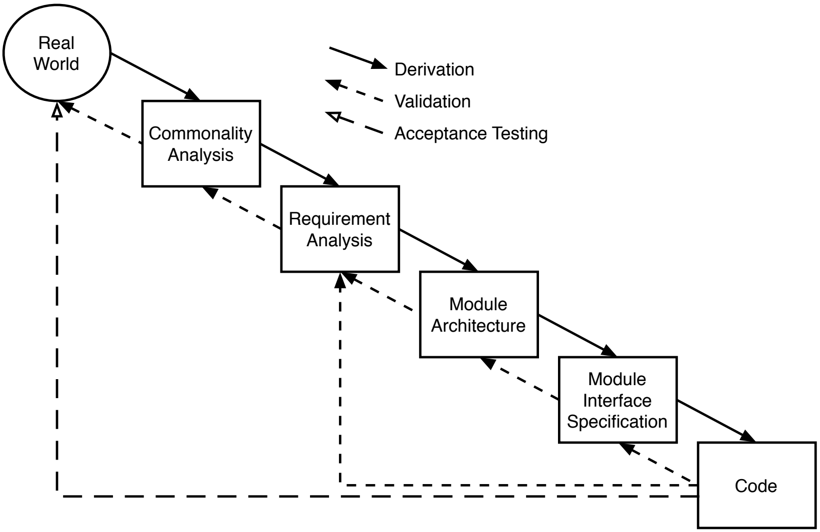
Challenges

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

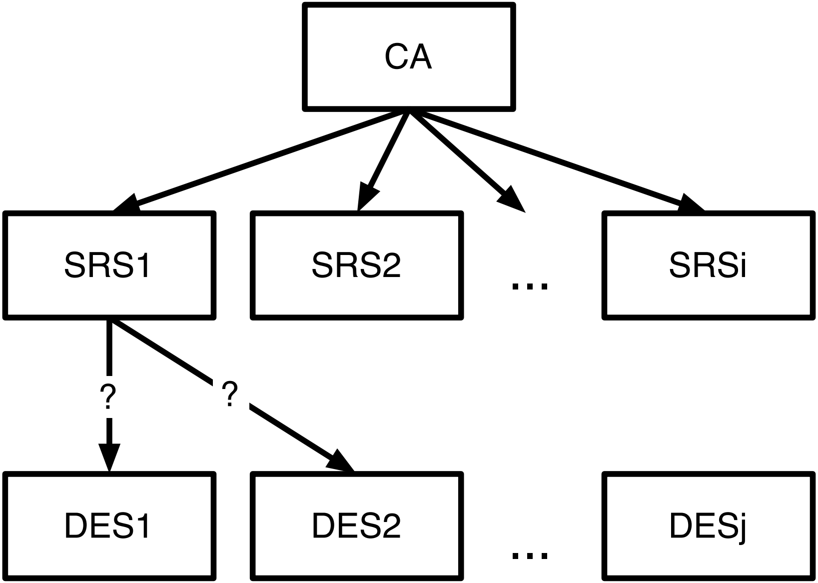
Challenges (Continued)

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

Overview of Process



CA to SRS to Design



Proposed Methodology

1. Identify family of interest
   - Specific physical model?
   - Multipurpose tool?
2. Commonality Analysis (CA)
   - Terminology
   - Commonalities
   - Variabilities
   - Parameters of variation
   - Binding time
3. Domain Specific Language (DSL)
4. Generation of family members

CA Template From [21]

1. Reference Material: a) Table of Contents b) Table of Symbols c) Abbreviations and Acronyms
2. Introduction: a) Purpose of the Document b) Organization of the Document
3. General System Description: a) Potential System Contexts b) Potential User Characteristics c) Potential System Constraints
4. Commonalities: a) Background Overview b) Terminology Definition c) Goal Statements d) Theoretical Models
5. Variabilities: a) Input Assumptions b) Calculation c) Output
6. Requirements (added to template)
7. Traceability Matrix

Abstract Requirements

- Appropriate level of abstraction by refining from goal to theory to input assumptions
- A goal is a functional objective the software should achieve:
  G1: Find the roots of an equation
- Goals are refined into theoretical models:
  T1: Given a function $f(x)$ and an interval $\{x | x_{lower} \leq x_{upper}\}$, return the points where $f(x) = 0$
- Introduce simplifying assumptions to allow theoretical model to be solved:
  VA1,2: $f(x)$ is continuous on the interval and/or $f(x)$ has at least one sign change on the interval

Abstract Requirements (Continued)

- Each variability has an associated parameter of variation and a binding time
  - Specification time
  - Compile time
  - Generation time
  - Run time

Capture Existing Knowledge

- Systematic consideration from general to specific
- Communication between experts
- Standard template allows comparison
- Convenient framework for summarizing existing literature
- Eventually a library of requirements documentation
- CA refined by a family of SRSs

System Requirements Specification (SRS)

- Based on IEEE Standard 830 and Volere requirements specification template
- Sections from CA are refined in SRS
- "Potential" descriptions are made specific
- Variabilities are set
- Binding times are set

SRS Template

1. Reference Material
2. Introduction
3. General System Description
4. Specific System Description: a) Background Overview, b) Terminology Definition, c) Goal Statements d) Theoretical Models, e) Assumptions, f) Data Constraints, g) System Behaviour
5. Non-functional Requirements: a) Accuracy of Input Data, b) Sensitivity of the Model, c) Tolerance of Solution, d) Performance, ... i) Portability,
6. Solution Validation Strategies, (moved to separate document)
7. Other System Issues:
8. Traceability Matrix

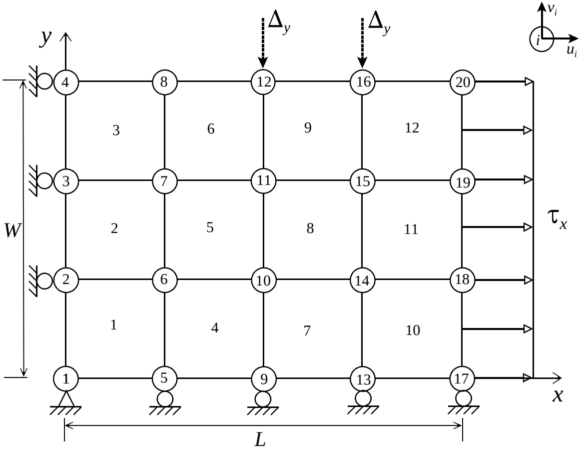
NFRs

- Rather than absolute quantification of NFRs, use relative comparison between other program family members
- Specify requirements in big O notation
- Relative importance between NFRs using Analytic Hierarchy Process (AHP) [20]
  - Addresses challenge of comparing attributes that are measured in different (or hard to quantify) units
  - Series of pair-wise comparisons between attributes
  - 1 for equal importance, 3 for moderately strong importance, ..., 9 for extreme importance

Validatable Requirements

- Relative comparison between programs is a validatable requirement
- Focus on a posteriori description, rather than a priori specification
- Solution validation strategies
  - Solve using different techniques
  - Identify benchmark test problems
  - Test cases built starting from assumed solutions (Method of Manufactured Solutions)
  - Partially validate for a simpler subset where the solution is known

Mesh Generating Software



Commonality Analysis for a Mesh Generator

From Chen's work [11, 23, 22]. Alternate approach in [5, 19, 2, 3, 4]

- Terminology
  - requirement
  - structured mesh, ...
- Commonalities
  - discretization
  - input from user is required, ...
- Variabilities
  - shape of elements
  - coordinate system used, ...
- Parameters of variation
  - line, triangle, quadrilateral, tetrahedral, hexahedral
  - Cartesian, polar, spherical, ...

Definition of a Mesh

Let Ω be a closed bounded domain in $\mathbb{R}$ or $\mathbb{R}^2$ or $\mathbb{R}^3$ and let K be a simple shape, such as a line segment in 1D, a triangle or a quadrilateral in 2D, or a tetrahedron or hexahedron in 3D. A mesh of Ω, denoted by τ, has the following properties:

1. $\Omega \approx \cup(K | K \epsilon \tau : K)$, where $\cup$ is first closed and then opened
2. the length of every element K, of dimension 1, in τ is greater than zero
3. the interior of every element K, of dimension 2 or greater, in τ is nonempty
4. the intersection of the interior of two elements is empty

Example Commonality

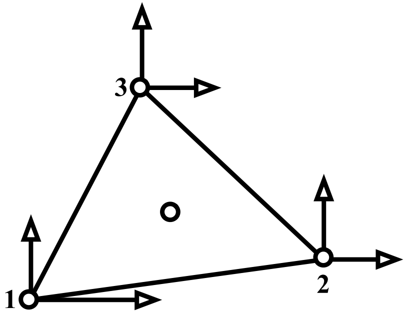
Item Number	C1
Description	A mesh generator discretizes a given computational domain (closed boundary Ω) into a covering up of a finite number of simpler shapes.
Related Variability	V6, V8, V12, V14, V15, V16, V17, V18
History	Created - May 7, 2004

Mesh Generator (MG) Goals

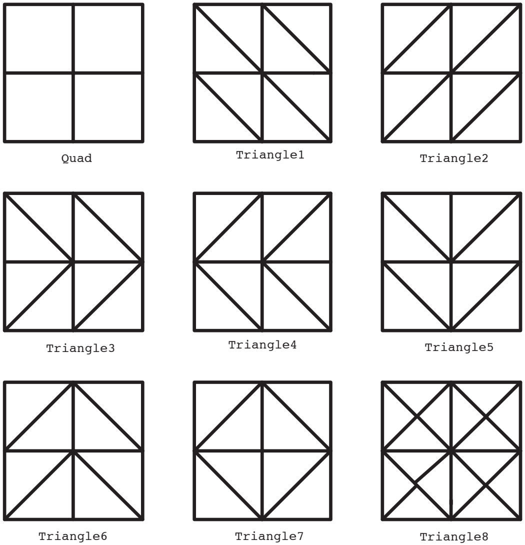
- G1 Input spatial domain Ω output a mesh M that covers this domain.
- G2 Transform information on the materials, material properties and the locations of the different materials
- G3 Transform information on the boundary condition types, values and locations
- G4 Transform system information, such as numerical algorithm parameters

Element Variability

Location of nodes: sequence of LocationT
Number of dof at nodes: sequence of $\mathbb{N}$
LocationT = tuple of $(L_1 : \text{natT}, L_2 : \text{natT}, L_3 : \text{natT})$
natT = $\{ s : \mathbb{R} | 0 \leq s \leq 1 : s \}$



Local Topology Variability



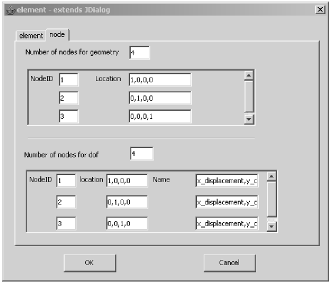
DSL Using XML

```
<elementSet>
    <geometrySpec>
        <shape>triangle1</shape>
        <nodeGeo count="3">
            <node id="1">
                <location>1,0,0</location>
            </node>
            <node id="2">
                <location>0,1,0</location>
            </node>
            ...
        </nodeGeo>
    </geometrySpec>
</elementSet>
```

Proof of Concept Implementation

From Cao's work [7, 26]

- XML document that customizes a Java object
- The Java object customizes the general purpose MG as it is loaded
- General purpose MG
  - All variabilities bound at run-time
  - Corresponds to an empty XML specification



Linear Systems of Equations

$$Ax = b$$

Commonality analysis presented in [21]

Goal and Theoretical Model

G1: Given a system of n linear equations represented by matrix A and column vector b, return x such that $Ax = b$, if possible
T1: Given 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 $b \notin span(A)$

Variabilities for Input Assumptions

Variability	**Parameter of Variation**
Allowed structure A	Set of $\{$ full, sparse, banded, tridiagonal, block triangular, ..., Hessenberg $\}$
Allowed definiteness A	Set of $\{$ not definite, positive definite, ..., negative semi-definite $\}$
Allowed class of A	Set of $\{$ diagonally dominant, Toeplitz, Vandermonde $\}$
Symmetry assumed?	boolean
Possible values for n	set of $\mathbb{N}$
Possible entries in A	set of $\mathbb{R}$
...	...

Variabilities for Calculation

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 $\}$

Variabilities for Output

Variability	**Parameter of Variation**
Destination for output x	Set of { to file, to 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 $\mathbb{R} \cup \{-\infty, \infty, \mathit{undef}\}$

Analytic Hierarchy Process

- Example 1
  - Embedded real-time system for digital signal processing
  - $n = 10$
  - A is assumed to be Toeplitz

	Speed	**Accuracy**	**Portability**	**Priority**
Speed	1	3	5	0.64
Accuracy	1/3	1	3	0.26
Portability	1/5	1/3	1	0.11

Solution Validation Strategies (for VnV Plan)

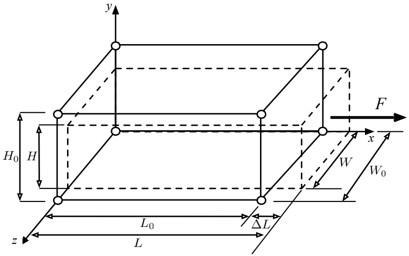
- Create test cases with known solutions
  - Assume A and x, calculate b
  - Given A and b calculate x^* and compare to the assumed x
- Comparison with Matlab
- Comparison with NAG library
- Where possible compare solution to interval arithmetic solution
- Experiments to describe how accuracy changes with increasing condition number

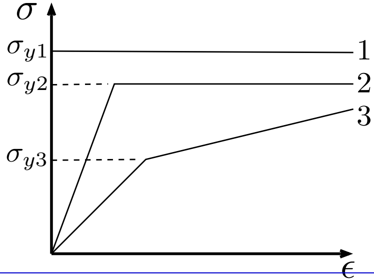
Connection to Design

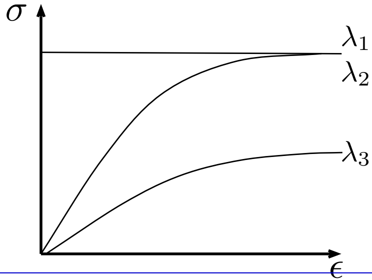
- Abstract requirements to concrete design decisions
- Reuse existing packages within the program family
- Summarize existing software by the parameters of variation and binding time
- If functional requirements match, then use NFRs
  - AHP to compare each design against each of the NFRs
  - Contribution of each NFR for each design alternative is found by multiplying the contribution of each alternative to the given NFR with the corresponding priority of that NFR
  - Sum the contributions
  - The highest overall score is the “winning” alternative

A Family of Material Models

From McCutchan's work [10, 26, 27, 9, 28, 15]

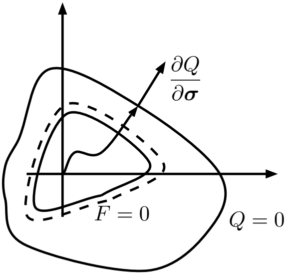






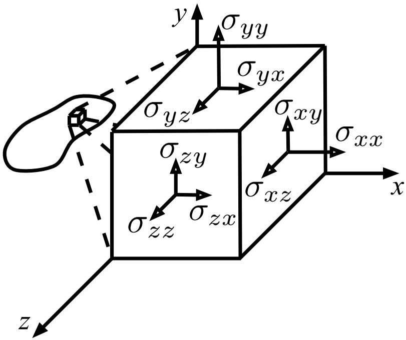
Terminology Definitions

Label:	D_YieldFunction
Symbol:	$F = F(\boldsymbol{\sigma}, \kappa)$
Type:	(tensor2DT $\times$ $\mathbb{R}$) $\rightarrow$ $\mathbb{R}$
Related:	D_Stress, D_HardeningParameter
Sources:	...
Descrip:	The yield function defines a surface $F = 0$ in the six dimensional stress space ...



Goal Statement

Label:	G_StressDetermination
Descrip:	Given the initial stress and the deformation history of a material particle, determine the stress within the material particle.
Refine:	T_ConstitEquation



Assumptions

Label:	A_AdditivityPostulate
Related:	D_StrainRate
Equation:	$\dot{\boldsymbol{\epsilon}} = \dot{\boldsymbol{\epsilon}}^{e} + \dot{\boldsymbol{\epsilon}}^{vp}$ with the following types and units $\dot{\boldsymbol{\epsilon}}$: tensor2DT (1/t) (1/s) $\dot{\boldsymbol{\epsilon}}^{e}$: tensor2DT (1/t) (1/s) $\dot{\boldsymbol{\epsilon}}^{vp}$: tensor2DT (1/t) (1/s)
Descrip:	The total strain rate ($\dot{\boldsymbol{\epsilon}}$) is assumed to decompose into elastic ($\dot{\boldsymbol{\epsilon}}^{e}$) and viscoplastic ($\dot{\boldsymbol{\epsilon}}^{vp}$) strain rates.
Rationale	This is a standard assumption for elastoplastic and elastoviscoplastic materials. The appropriateness of this assumption is born out by the success of theories built upon it.
Source:	[6, page 339]; [7, page 181]

Theoretical Model

Label:	T_ConstitEquation
Related:	A_CauchyStress, A_DeformationHistory, A_PerzynaConstit, A_AdditivityPostulate, A_ElasticConstit, A_DescriptionOfMotion, V_MaterialProperties
Input:	$\boldsymbol{\sigma}_0$: tensor2DT (StressU) (Pa) t_{begin} : $\mathbb{R}$ (t) (s) t_{end} : $\mathbb{R}$ (t) (s) $\dot{\epsilon}(t)$: $\{t : \mathbb{R} \| t_{begin} \leq t \leq t_{end} : t\} \rightarrow$ tensor2DT (1/t) (1/s) mat_prop_val : string $\rightarrow \mathbb{R}$ E : $\mathbb{R}^+$ (StressU) (Pa) ν : poissonT (dimensionless)

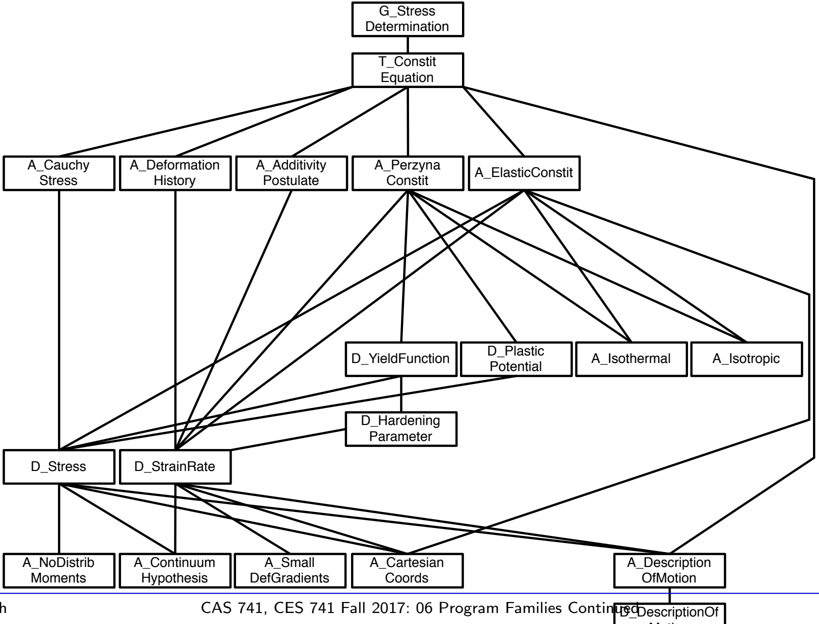
Theoretical Model Continued

Label:	T_ConstitEquation
Output:	$\boldsymbol{\sigma}(t) : \{t : \mathbb{R} \| t_{begin} \leq t \leq t_{end} : t\} \rightarrow$ tensor2DT such that $\dot{\boldsymbol{\sigma}} = \mathbf{D}\left(\dot{\boldsymbol{\epsilon}} - \gamma < \varphi(F(\boldsymbol{\sigma}, \kappa)) > \frac{\partial Q(\boldsymbol{\sigma})}{\partial \boldsymbol{\sigma}}\right)$ and $\boldsymbol{\sigma}(t_{begin}) = \boldsymbol{\sigma}_0$, the components of $\boldsymbol{\sigma}$ have the units of StressU (Pa)
Derive:	The governing differential equation is found by first solving for $\dot{\boldsymbol{\epsilon}}^e$ in A_AdditivityPostulate and then ...
Descrip:	The theoretical model is only completely defined once the associated variabilities (V_MaterialProperties) that define the material have been set. ...
History:	Created – June 14, 2007

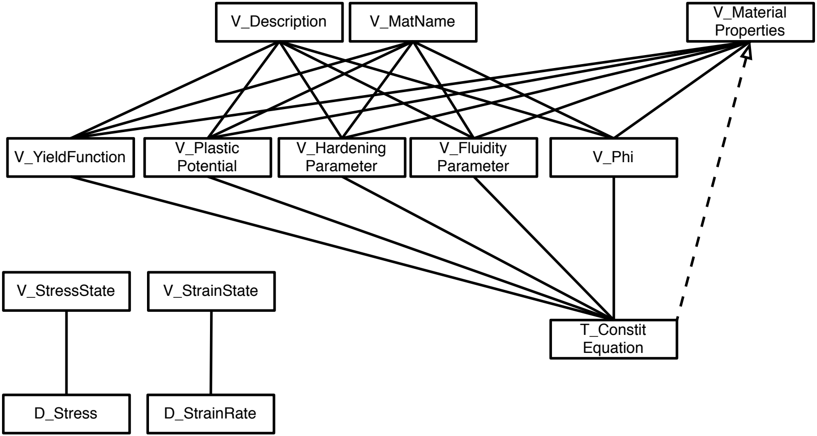
Variabilities

- $F = F(\boldsymbol{\sigma}, \kappa) : \mathbb{R}^6 \times \mathbb{R} \rightarrow \mathbb{R}$
- $Q = Q(\boldsymbol{\sigma}) : \mathbb{R}^6 \rightarrow \mathbb{R}$
- $\kappa = \kappa(\boldsymbol{\epsilon}^{vp}) : \mathbb{R}^6 \rightarrow \mathbb{R}$
- $\varphi = \varphi(F) : \mathbb{R} \rightarrow \mathbb{R}$
- $\gamma : \mathbb{R}$
- *mat_prop_names* : set of string

Dependency Graph



Dependency Graph Between Commonalities and Variabilities



Example

Label:	E_StrainHardening
V_MatName	*name* = "Strain-Hardening Viscoelastic"
V_YieldFunct	$F = q\kappa^{\frac{n-1}{m}}$ (StressU) (Pa)
V_PlasticPot	$Q = q$ (StressU) (Pa)
V_HardParam	$\kappa = \epsilon_q^{vp}$ (L/L) (m/m)
V_Phi	$\varphi = F^{\frac{m}{n}}$ (StressU$^{\frac{m}{n}}$) (Pa$^{\frac{m}{n}}$)
V_FluParam	$\gamma = nA^{\frac{1}{n}}$ (StressU^{-m}t^{-1}) (Pa^{-m}s^{-1})
V_MatProps	*mat_prop_names* = { "A", "m", "n" }, where the type of the material properties are ...
V_Description	*descript* = "This constitutive equation combines a power-law viscoelastic material with a strain hardening (softening) material. ..."

Code Generation

- Specify variabilities
- Symbolically calculate terms needed by numerical algorithm, including $\frac{\partial Q}{\partial \boldsymbol{\sigma}}$, $\frac{\partial F}{\partial \boldsymbol{\sigma}}$, etc.
- Symbolic processing avoids tedious and error-prone hand calculations
  - Reduces workload
  - Allows non-experts to deal with new problems
  - Increases reliability
- Use Maple Computer Algebra System for model manipulation
- Convert math expressions into C expressions using "CodeGeneration"
- Inline into a C++ class defining the material model
- A finite element program can this interface to realize the numerical algorithm

BNF of DSL for F

$\langle expression \rangle \rightarrow \langle number \rangle |$
$(\langle expression \rangle)|$
$\langle expression \rangle \,\hat{}\, \langle expression \rangle |$
$\langle expression \rangle * \langle expression \rangle |$

...

$\langle simulation\text{-}variable\text{-}F \rangle | \langle user\text{-}defined\text{-}constants \rangle$
$\langle simulation\text{-}variable\text{-}F \rangle \rightarrow$ **Kappa** $| \langle simulation\text{-}variable\text{-}stress \rangle | \langle simulation\text{-}variable\text{-}stress\text{-}macros \rangle$
$\langle simulation\text{-}variable\text{-}stress \rangle \rightarrow$ **SigmaXX**|**SigmaYY**|**SigmaZZ**|**SigmaXY**|**SigmaYZ**|**SigmaXZ**
$\langle simulation\text{-}variable\text{-}stress\text{-}macros \rangle \rightarrow$ **Sxx**|**Syy**|**Szz**|**Sxy**|**Syz**|**Sxz**|**Sm**|**J2**|**J3**|**q**
$\langle user\text{-}defined\text{-}constants \rangle \rightarrow \langle string \rangle$

Concluding Remarks

- Case studies of applying software engineering methodologies to mesh generating systems and linear solvers
- Appropriate and advantageous to apply program family strategy
- Challenges for software engineers
- General purpose scientific software is best studied as a program family
  - Variabilities are assumptions about problems that can be handled
  - Derive requirements from commonality analysis
- Eventually hope for automatic code generation

Concluding Remarks (Continued)

A new methodology for documenting requirements for general purpose scientific computing software

1. Validatable requirements
   - Relative comparison between program family members
   - Focus on description rather than specification
   - Solution validation strategy
2. Abstract
   - Refine goal statement to theoretical model to input assumptions
   - In some cases one may want to turn off input checking
   - Connection to design

Concluding Remarks (Continued)

3. NFRs
   - Relative comparison
   - AHP
4. Capture and reuse
   - Systematic consideration from general to specific
   - CA refined by a family of SRSs
   - CA and SRS summarize existing knowledge and currently available software
   - Standard template allows comparison
   - Convenient framework for summarizing existing literature

Concluding Remarks

- A new template for a family of models of physical phenomena
- Refinement of Goals to Theoretical Models using Data Definitions and Assumptions
- Variabilities are identified in the Theoretical Model
- A constitutive equation can be written using a (declarative) DSL and the code can be generated
- A DSL has been built, using Maple, for a virtual material testing laboratory

Concluding Remarks

- SC software is a great candidate for development as a program family
- Produce programs that are as special or general purpose as needed
- Improve reusability, usability and reliability
- Potential to improve performance
- A commonality analysis facilitates the design of a DSL
- Symbolic processing and code generation are very useful techniques
- We will return to code generation later

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.

M Cecilia Bastarrica and Nancy Hitschfeld-Kahler.
An evolvable meshing tool through a flexible object-oriented design.
In *International Meshing Roundtable*, pages 203–212. Citeseer, 2004.

References II

- María Cecilia Bastarrica.
  Base architecture in a software product line.
  In *Proceedings of the XXVIII Latin American Conference of Informatics, CLEI'2002, Montevideo, Uruguay*, page 119, 2002.

- María Cecilia Bastarrica and Nancy Hischfeld-Kahler.
  An evolvable meshing tool through a flexible object-oriented design, september 2004.
  In *Proceedings of the 13th International Meshing Roundtable*, pages 203–212, Williamsburg, Virginia, 2004. Sandia National Laboratories.

References III

María Cecilia Bastarrica and Nancy Hischfeld-Kahler.
Designing a product family of meshing tools.
Advances in Engineering Software, 37(1):1–10, 2006.

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

Fang Cao.
A program family approach to developing mesh generators.

Master's thesis, McMaster University, April 2006.

References IV

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

Jacques Carette, W. Spencer Smith, John McCutchan, Christopher Anand, and Alexandre Korobkine.
Model manipulation as part of a better development process for scientific computing code.
Technical Report 48, Software Quality Research Laboratory, McMaster University, December 2007.
41 pp.

References V

Jacques Carette, W. Spencer Smith, John McCutchan, Christopher Anand, and Alexandre Korobkine.
Intelligent Computer Mathematics, 9th International Conference, AISC 2008, chapter Case Studies in Model Manipulation for Scientific Computing, pages 24–37.
Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Birmingham, UK, 2008.

Chien-Hsien Chen.
A software engineering approach to developing mesh generators.
Master's thesis, McMaster University, Hamilton, Ontario, Canada, 2003.

References VI

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 VII

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 VIII

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 IX

Pedro O. Rossel, María Cecilia Bastarrica, Nancy Hitschfeld-Kahler, Violeta Díaz, and Mario Medina.
Domain modeling as a basis for building a meshing tool software product line.
Advances in Engineering Software, 70:77–89, 2014.

T. L. Saaty.
The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation.
McGraw-Hill Publishing Company, New York, New York, 1980.

References X

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 XI

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 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,*

References XII

SREP'05, pages 107–121, Paris, France, 2005. In conjunction with 13th IEEE International Requirements Engineering Conference.

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 Fang Cao.
Program families in scientific computing.
In Jonathan Sprinkle, Jeff Gray, Matti Rossi, and Juha-Pekka Tolvanen, editors, *7^{th} OOPSLA Workshop on Domain Specific Modelling (DSM'07)*, pages 39–47, Montréal, Québec, October 2007.

References XIII

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).
8 pp.

References XIV

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 XV

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 XVI

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.