

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

Fall 2017

03 Requirements

Dr. Spencer Smith

Faculty of Engineering, McMaster University

September 12, 2017



Requirements

- Administrative details
- Questions: project choices?, software tools?
- Problem statement and example
- Software Engineering for Scientific Computing literature
- Scientific Computing Software Qualities
- Motivation: Challenges to Developing Quality Scientific Software
- Requirements documentation for scientific computing
- A requirements template
- Advantages of new template and examples
- The template from a software engineering perspective
- Concluding remarks
- References

Administrative Details

- Add smiths to your GitHub repos
- Linked-In
- Assign the instructor an issue to review your problem statement

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

Questions?

- Questions about project choices?
- Questions about software tools?
 - ▶ git?
 - ▶ LaTeX?
- Partial tex files in the blank project template
- **Problem statement**

Problem Statement

- Written in LaTeX
- Due electronically (on GitLab) by deadline
- Comments might be typed directly into your source
- For later assignments with LaTeX source, include the LaTeX commands for comments
- **What** problem are you trying to solve?
- **Not how** you are going to solve the problem
- Why is this an important problem?
- What is the context of the problem you are solving?
 - ▶ Who are the stakeholders?
 - ▶ What is the environment for the software?
- A page description should be sufficient

Sample Project Statements

- CParser
- FloppyFish
- Screenholders
- Template in repo

SE For SC Literature

- CAS 741 process is document driven, adapted from the waterfall model [6, 26]
- Many say a document driven process is not used by, nor suitable for, scientific software.
 - ▶ Scientific developers naturally use an agile philosophy [1, 2, 5, 17],
 - ▶ or an amethodical process [9]
 - ▶ or a knowledge acquisition driven process [10].
- Scientists do not view rigid, process-heavy approaches, favorably [2]
- Reports for each stage of development are counterproductive [15, p. 373]
- Up-front requirements are impossible [2, 21]
- What are some arguments in favour of a rational document driven process?

Counter Arguments

- Just because document driven is not used, does not mean it will not work
- Documentation provides many benefits [14]:
 - ▶ easier reuse of old designs
 - ▶ better communication about requirements
 - ▶ more useful design reviews
 - ▶ easier integration of separately written modules
 - ▶ more effective code inspection
 - ▶ more effective testing
 - ▶ more efficient corrections and improvements.
- Actually faking a rational design process
- Too complex for up-front requirements sounds like an excuse
 - ▶ Laws of physics/science slow to change
 - ▶ Often simple design patterns
 - ▶ Think program family, not individual member

Definition of Software Qualities

- Measures of the excellence or worth of a software product (code or document) or process with respect to some aspect
- What are some important aspects (qualities) for scientific software?
- User Satisfaction = The Important Qualities are High + Within Budget

Important Qualities for Scientific Computing Software

- External qualities
 - ▶ Correctness (Thou shalt not lie)
 - ▶ Reliability
 - ▶ Robustness
 - ▶ Performance
 - ▶ Time efficiency
 - ▶ Space efficiency
- Internal qualities
 - ▶ Verifiability
 - ▶ Usability
 - ▶ Maintainability
 - ▶ Reusability
 - ▶ Portability

Correctness Versus Reliability Versus Robustness

What is the difference between these 3 qualities?

Can you assess correctness without a requirements specification?

Correctness

- A software product is correct if it satisfies its requirements specification
- Correctness is extremely difficult to achieve because
 - ▶ The requirements specification may be imprecise, ambiguous, inconsistent, based on incorrect knowledge, or nonexistent
 - ▶ Requirements often compete with each other
 - ▶ It is virtually impossible to produce “bug-free” software
 - ▶ It is very difficult to verify or measure correctness
- If the requirements specification is formal, correctness can in theory and possibly in practice be
 - ▶ Mathematically defined
 - ▶ Proven by mathematical proof
 - ▶ Disproven by counterexample

Reliability

- A software product is reliable if it usually does what is intended to do
- Correctness is an absolute quality, while reliability is a relative quality
- A software product can be both reliable and incorrect
- Reliability can be statistically measured
- Software products are usually much less reliable than other engineering products

Robustness

- A software product is robust if it behaves reasonably even in unanticipated or exceptional situations
- A correct software product need not be robust
 - ▶ Correctness is accomplished by satisfying requirements
 - ▶ Robustness is accomplished by satisfying unstated requirements

Question on Correctness. Reliability and Robustness

Reliable programs are a superset of correct programs AND robust programs are a superset of reliable programs. Is this statement True or False?

- A. True
- B. False

Performance

What are some ways you could measure software performance?

What are some ways you could specify performance requirements to make them unambiguous and verifiable?

Performance

- The performance of a computer product is the efficiency with which the product uses its resources (memory, time, communication)
- Performance can be evaluated in three ways
 - ▶ Empirical measurement
 - ▶ Analysis of an analytic model
 - ▶ Analysis of a simulation model
- Poor performance often adversely affects the usability and scalability of the product

Usability

What are some examples of excellent usability?

When you go to a friend's house, you can likely operate their microwave without reading the manual. What did human factors engineers do to make this possible?

Usability

- The usability of a software product is the ease with which a typical human user can use the product
- Usability depends strongly on the capabilities and preferences of the user
- The user interface of a software product is usually the principle factor affecting the product's usability
- Human computer interaction (HCI) is a major interdisciplinary subject concerned with understanding and improving interaction between humans and computers

Verifiability

- The verifiability of a software product is the ease with which the product's properties (such as correctness and performance) can be verified
- Verifiability can be both an internal and an external quality

Maintainability

- The maintainability of a software product is the ease with which the product can be modified after its initial release
- Maintenance costs can exceed 60% of the total cost of the software product
- There are three main categories of software maintenance
 1. Corrective: Modifications to fix residual and introduced errors
 2. Adaptive: Modifications to handle changes in the environment in which the product is used
 3. Perfective: Modifications to improve the qualities of the software
- Software maintenance can be divided into two separate qualities
 1. Repairability: The ability to correct defects
 2. Evolvability: The ability to improve the software and to keep it current

Maintainability

What do software developers do to promote maintainability?

Reusability

What are the advantages of reusing code?

Why doesn't it happen more often?

Reusability

- A software product or component is reusable if it can be used to create a new product
- Reuse comes in two forms
 1. Standardized, interchangeable parts
 2. Generic, instantiable components
- Reusability is a bigger challenge in software engineering than in other areas of engineering

Portability

- A software product is portable if it can run in different environments
- The environment for a software product includes the hardware platform, the operating system, the supporting software and the user base
- Since environments are constantly changing, portability is often crucial to the success of a software product
- Some software such as operating systems and compilers, is inherently machine specific

Understandability

- The understandability of a software product is the ease with which the requirements, design, implementation, documentation, etc. can be understood
- Understandability is an internal quality that has an impact on other qualities such as verifiability, maintainability, and reusability
- There is often a tension between understandability and the performance of a software product
- Some useful software products completely lack understandability (e.g. those for which the source code is lost)

Relationship between Qualities

Draw a diagram showing the relationships between the various software qualities

Measurement of Quality

- A software quality is only important if it can be measured
 - without measurement there is no basis for claiming improvement
- A software quality must be precisely defined before it can be measured
- Most software qualities do not have universally accepted
- Can you directly measure maintainability?
- How might you measure maintainability?

Problems with Developing Quality Scientific Computing Software

- Need to know requirements to judge reliability
- In many cases the only documentation is the code
- Reuse is not as common as it could be
 - ▶ [Meshing software survey](#)
 - ▶ [Public domain finite element programs](#)
 - ▶ etc.
- Many people develop “from scratch”
- Cannot easily reproduce the work of others
- Neglect of simple software development technology [27]

Adapt Software Engineering Methods

- Software engineering improves and quantifies quality
- Successfully applied in other domains
 - ▶ Business and information systems
 - ▶ Embedded real time systems
- Systematic engineering process
- Design through documentation
- Use of mathematics
- Reuse of components
- Warranty rather than a disclaimer

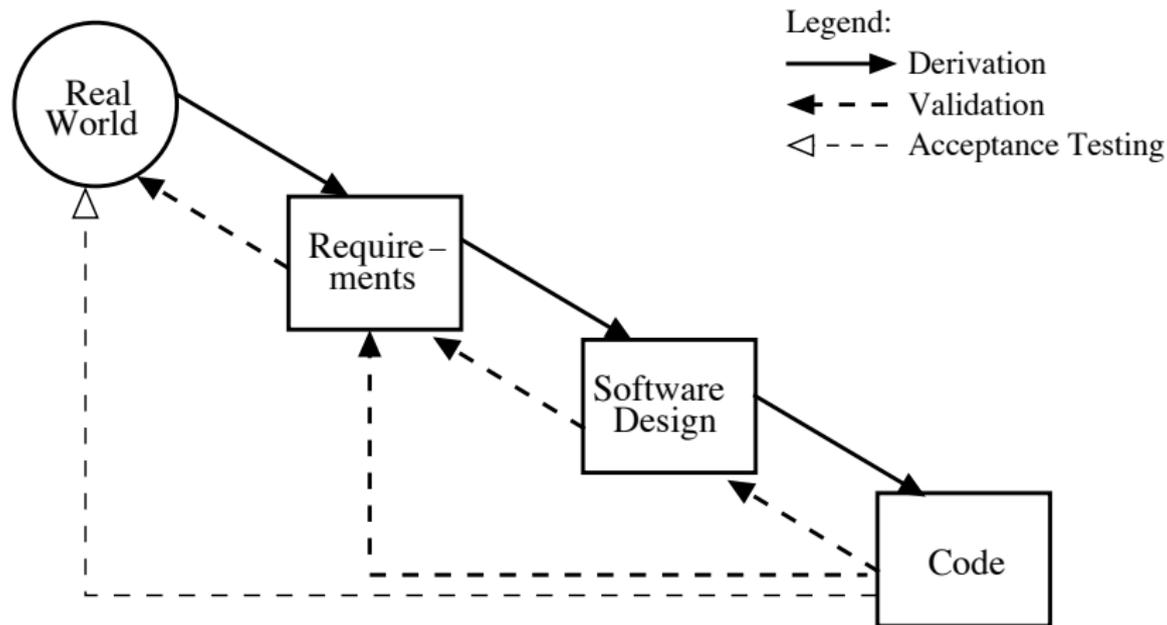
Developing Scientific Computing Software

- Facilitators
 - ▶ One user viewpoint for specifying a physical model
 - ▶ Assumptions can be used to distinguish models
 - ▶ High potential for reuse
 - ▶ Libraries
 - ▶ Already mathematical
- Challenges
 - ▶ Verification and Validation
 - ▶ Acceptance of software engineering methodologies
 - ▶ No existing templates or examples

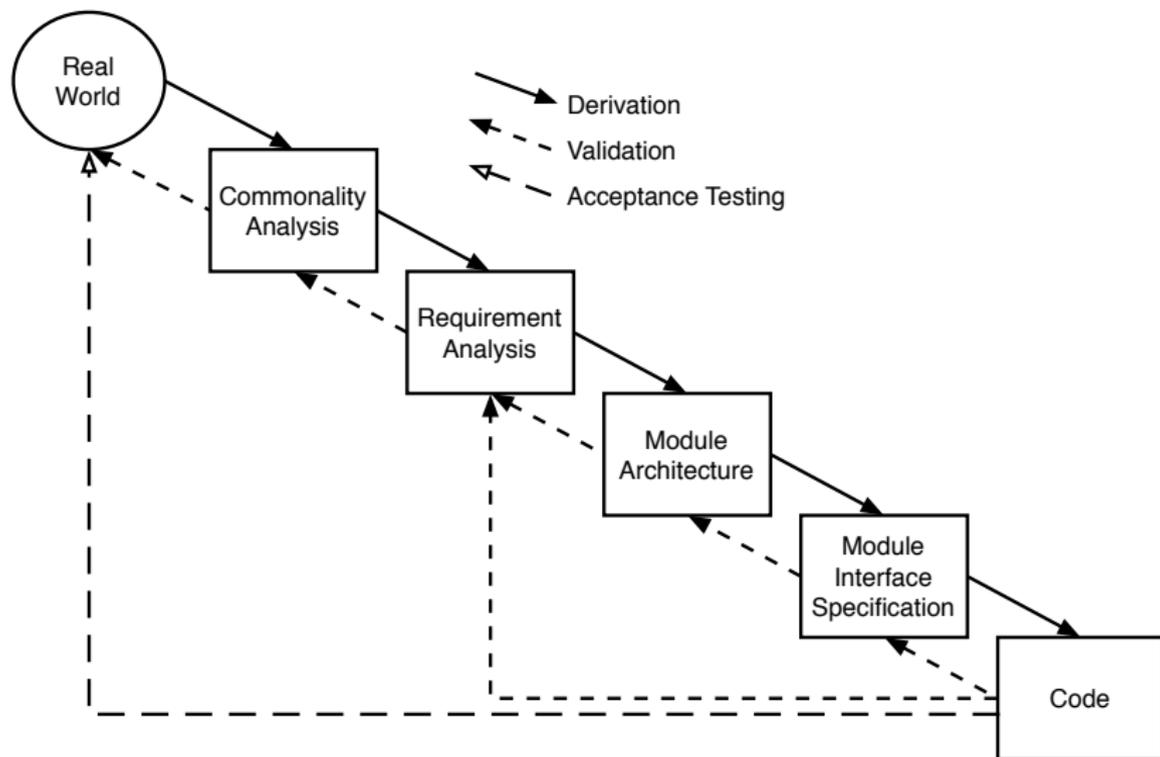
Outline of Discussion of Requirements

- Background on requirements elicitation, analysis and documentation
- Why requirements analysis for engineering computation?
- 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



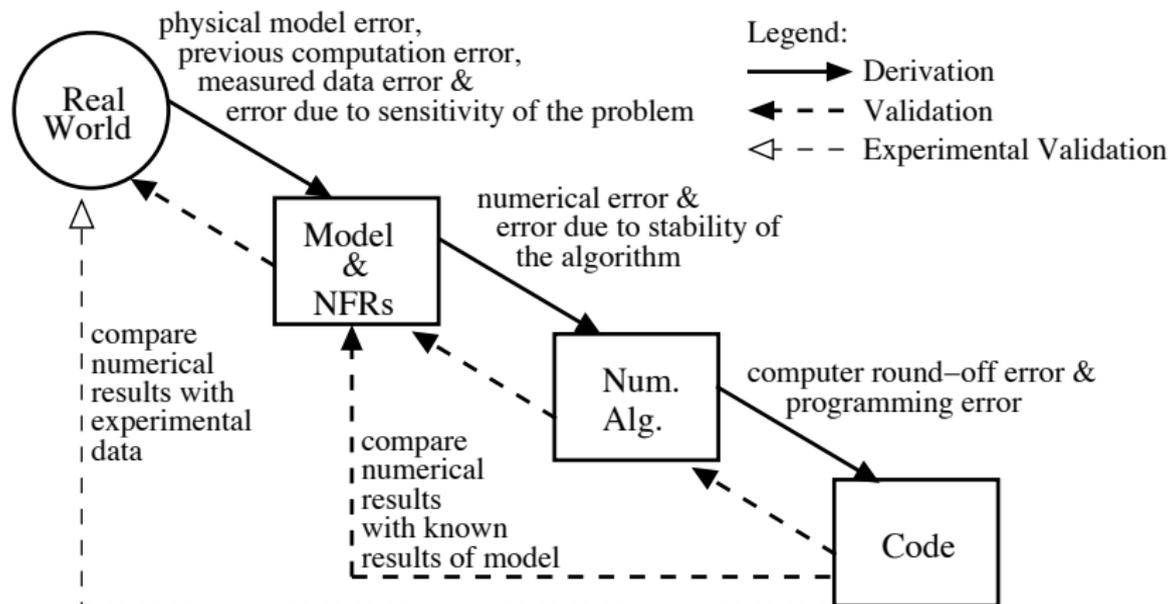
Sometimes Include Commonality Analysis



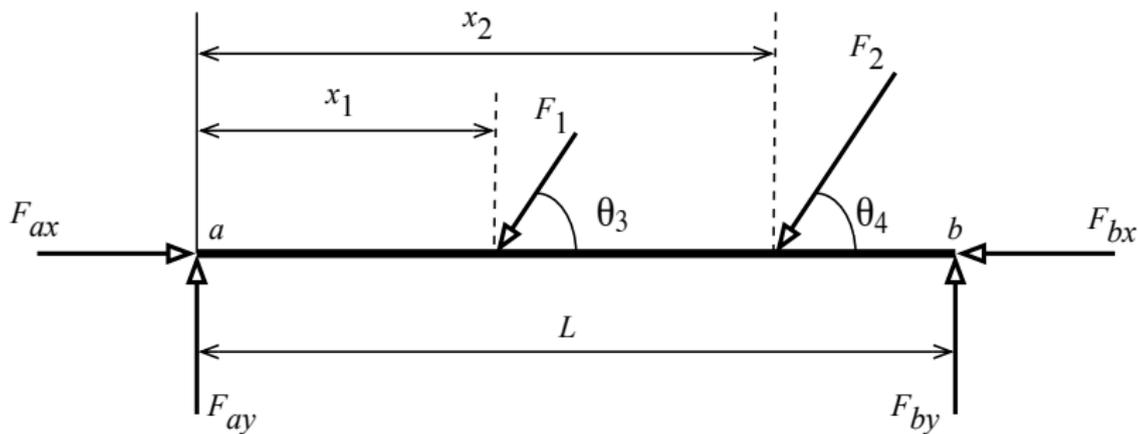
Software Requirements Activities

- A software requirement is a description of how the system should behave, or of a system property or attribute
- Requirements should be 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?



Beam Analysis Software



Proposed Template

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

$S_{unkF} \notin \mathbb{P}_3$	-
$S_{unkF} = \{\textcircled{F}_{ax}, \textcircled{F}_{bx}, \textcircled{F}_{ay}\}$	-
$S_{unkF} = \{\textcircled{F}_{ax}, \textcircled{F}_{ay}, \textcircled{F}_1\}$	$x_1 \neq 0$ $\wedge \theta_3 \neq 0$ $\wedge \theta_3 \neq 180$
	otherwise

H_2

H_1	
$S_{GET} = S_{sym} - S_{unkF}$	$S_{GET} \neq (S_{sym} - S_{unkF})$
$(ErrorMsg' = InvalidUnknown) \wedge ChangeOnly(ErrorMsg)$	FALSE
$ErrorMsg' = NoSolution \wedge ChangeOnly(ErrorMsg)$	
$F'_{ax} = \frac{-\cos \theta_3 F_2 x_2 \sin \theta_4 + \cos \theta_3 F_{by} L + F_2 \cos \theta_4 x_1 \sin \theta_3 + F_{bx} x_1 \sin \theta_3}{x_1 \sin \theta_3}$ \wedge $F'_{ay} = -\frac{F_2 x_2 \sin \theta_4 - F_{by} L - F_2 \sin \theta_4 x_1 + F_{by} x_1}{x_1 \sin \theta_3}$ $\wedge F'_1 = \frac{-F_2 x_2 \sin \theta_4 + F_{by} L}{x_1 \sin \theta_3} \wedge ChangeOnly(S_{unkF})$	
$(ErrorMsg' = Indeterminant) \wedge ChangeOnly(ErrorMsg)$	

G

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

A10. The deflection of the beam is caused by bending moment only, the shear does not contribute.

More on the Template

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

Why a New Template?

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

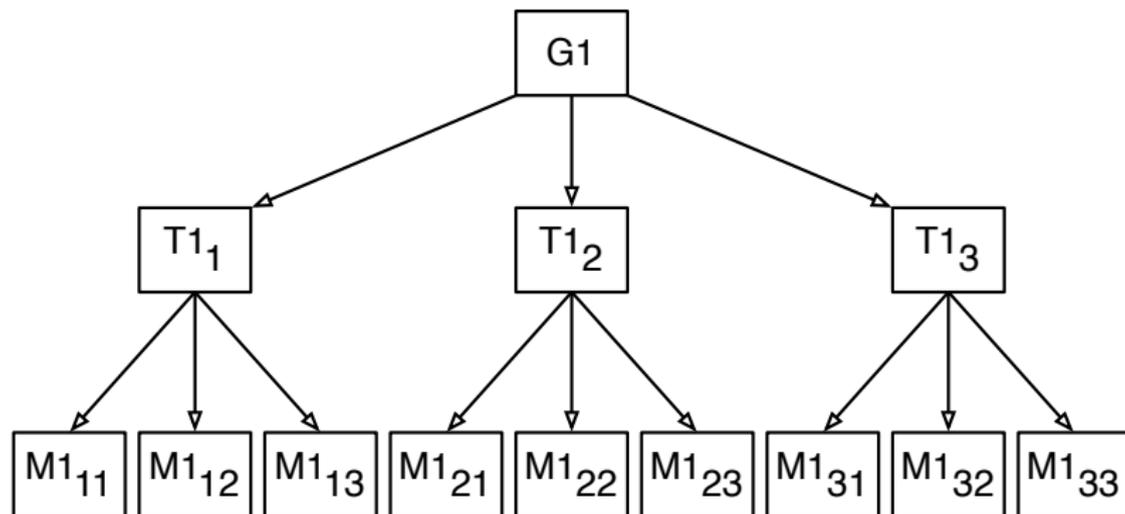
Overview of the New Template

- 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

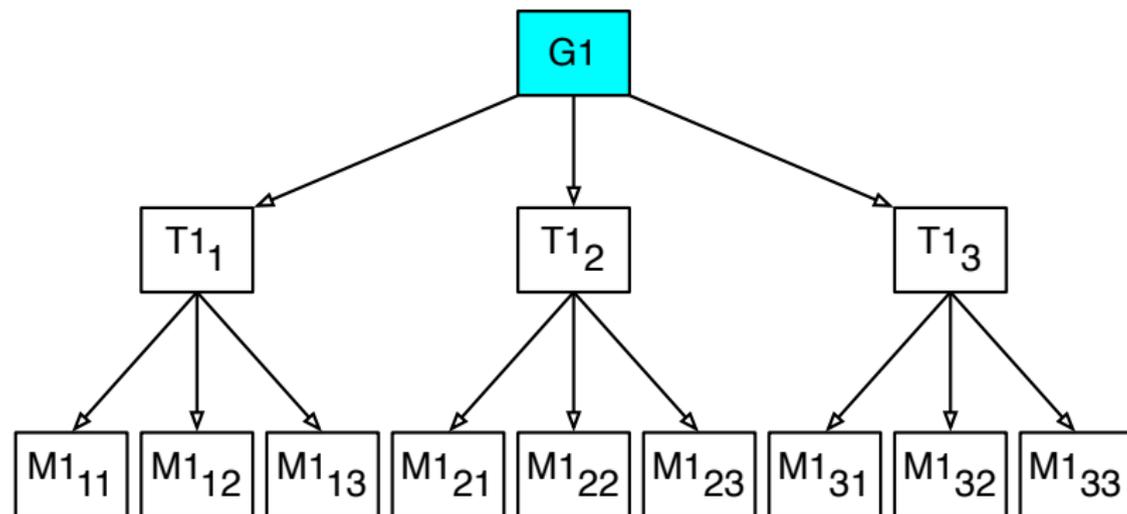
Excerpts from Specific System Description

- 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

Refinement from Abstract to Concrete

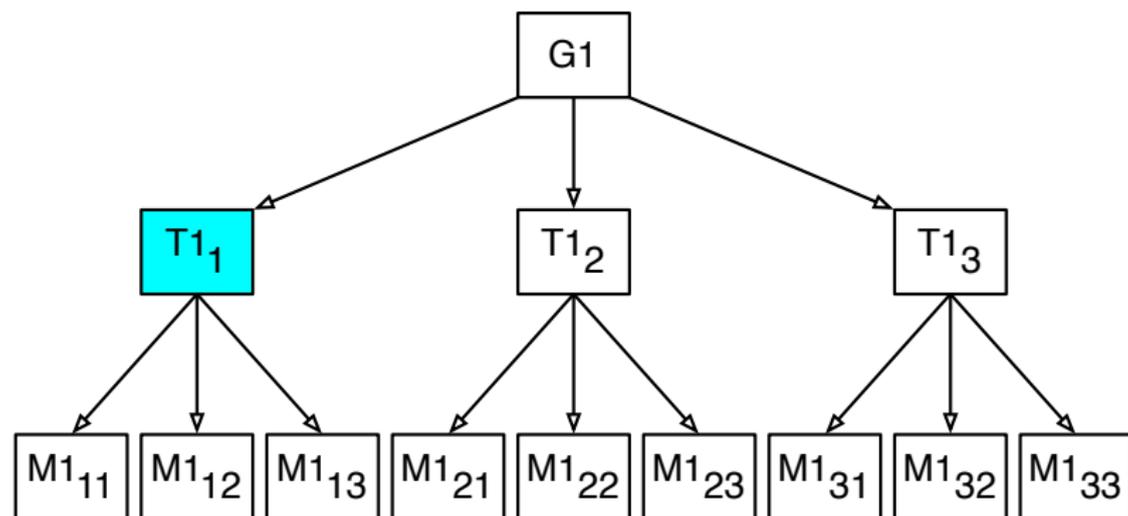


Refinement from Abstract to Concrete



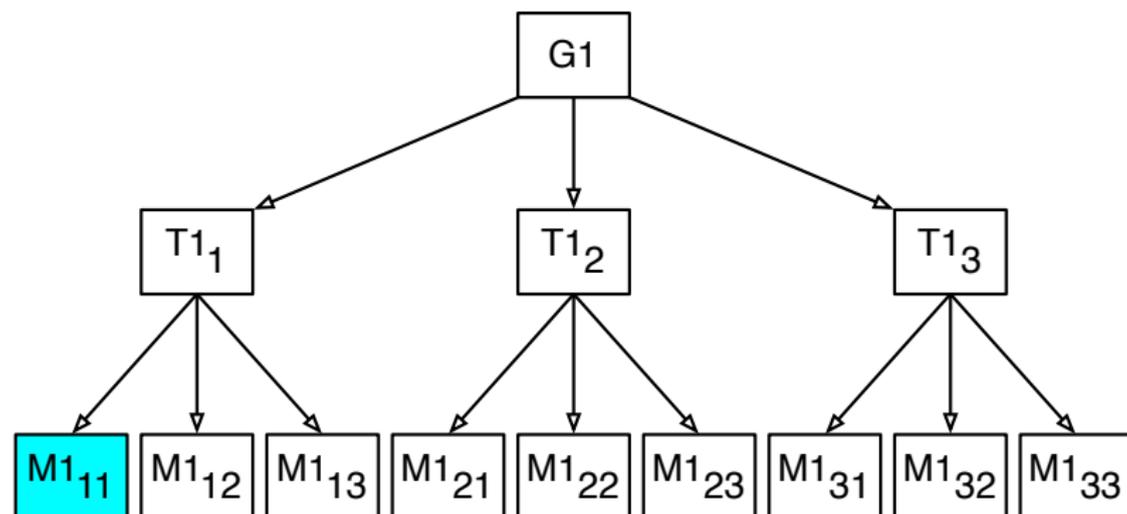
G1: Solve for unknown forces

Refinement from Abstract to Concrete



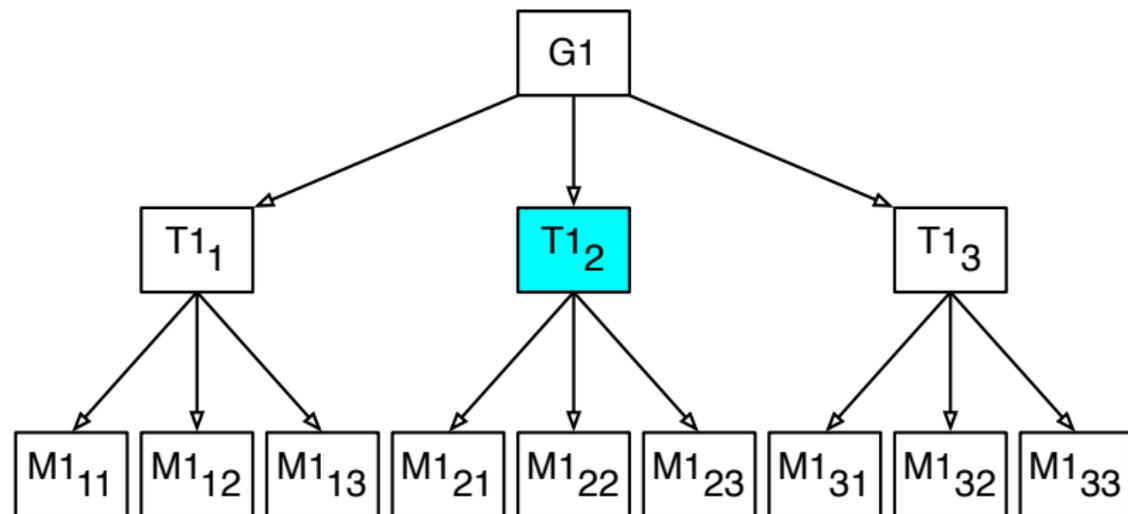
$$(\mathbf{T1}_1) \begin{cases} \sum F_{xi} = 0 \\ \sum F_{yi} = 0 \\ \sum M_i = 0 \end{cases}$$

Refinement from Abstract to Concrete



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

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
- **T3₁**: $\frac{d^2y}{dx^2} = \frac{M}{EI}$, $y(0) = y(L) = 0$
- **T3₂**: y determined by moment area method
- **T3₃**: y determined using Castigliano's theorem
- **M3₁₁**: $y = \frac{12 \int_0^L (\int_0^L M dx) dx}{Eeh^3}$, $y(0) = y(L) = 0$

Kreyman and Parnas Five Variable Model

- 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

Concluding Remarks

- 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

Concluding Remarks (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)

References I



Karen S. Ackroyd, Steve H. Kinder, Geoff R. Mant, Mike C. Miller, Christine A. Ramsdale, and Paul C. Stephenson.

Scientific software development at a research facility.
IEEE Software, 25(4):44–51, July/August 2008.



Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post.

Software development environments for scientific and engineering software: A series of case studies.

In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 550–559, Washington, DC, USA, 2007. IEEE Computer Society.

References II



Jules Desharnais, Ridha Khedri, and Ali Mili.
Representation, validation and integration of scenarios
using tabular expressions.

Formal Methods in System Design, page 40, 2004.

To appear.



Paul F. Dubois.
Designing scientific components.

Computing in Science and Engineering, 4(5):84–90,
September 2002.

References III



Steve M. Easterbrook and Timothy C. Johns.
Engineering the software for understanding climate
change.

Comuting in Science & Engineering, 11(6):65–74,
November/December 2009.



Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli.
Fundamentals of Software Engineering.

Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition,
2003.



IEEE.

Recommended practice for software requirements
specifications.

IEEE Std 830-1998, pages 1–40, Oct 1998.

References IV



R. Janicki and R. Khedri.

On a formal semantics of tabular expression.

Science of Computer Programming, 39(2-3):189–213, 2001.



Diane Kelly.

Industrial scientific software: A set of interviews on software development.

In *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research, CASCON '13*, pages 299–310, Riverton, NJ, USA, 2013. IBM Corp.

References V



Diane Kelly.

Scientific software development viewed as knowledge acquisition: Towards understanding the development of risk-averse scientific software.

Journal of Systems and Software, 109:50–61, 2015.



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 VI



Lei Lai.

Requirements documentation for engineering mechanics software: Guidelines, template and a case study.

Master's thesis, McMaster University, Hamilton, Ontario, Canada, 2004.



David L. Parnas and P.C. Clements.

A rational design process: How and why to fake it.

IEEE Transactions on Software Engineering,
12(2):251–257, February 1986.



David Lorge Parnas.

Precise documentation: The key to better software.

In *The Future of Software Engineering*, pages 125–148,
2010.

References VII



Patrick J. Roache.

Verification and Validation in Computational Science and Engineering.

Hermosa Publishers, Albuquerque, New Mexico, 1998.



Suzanne Robertson and James Robertson.

Mastering the Requirements Process, chapter Volere
Requirements Specification Template, pages 353–391.

ACM Press/Addison-Wesley Publishing Co, New York,
NY, USA, 1999.

References VIII



Judith Segal.

When software engineers met research scientists: A case study.

Empirical Software Engineering, 10(4):517–536, October 2005.



Judith Segal.

End-user software engineering and professional end-user developers.

In *Dagstuhl Seminar Proceedings 07081, End-User Software Engineering*, 2007.

References IX



Judith Segal.

Some problems of professional end user developers.

In *VLHCC '07: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 111–118, Washington, DC, USA, 2007. IEEE Computer Society.



Judith Segal.

Models of scientific software development.

In *Proceedings of the First International Workshop on Software Engineering for Computational Science and Engineering (SECSE 2008)*, pages 1–6, Leipzig, Germany, 2008. In conjunction with the 30th International Conference on Software Engineering (ICSE).

References X



Judith Segal and Chris Morris.

Developing scientific software.

IEEE Software, 25(4):18–20, July/August 2008.



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 XI

-  W. Spencer Smith, Lei Lai, and Ridha Khedri.
Requirements analysis for engineering computation.
In R. Muhanna and R. Mullen, editors, *Proceedings of the NSF Workshop on Reliable Engineering Computing*, pages 29–51, Savannah, Georgia, 2004.
-  R. H. Thayer and M. Dorfman, editors.
IEEE Recommended Practice for Software Requirements Specifications.
IEEE Computer Society, Washington, DC, USA, 2nd edition, 2000.
-  The Institute of Electrical and Electronics Engineers, Inc.
Software Requirements Engineering.
IEEE Computer Society Press, 2nd edition, 2000.

References XII



Hans van Vliet.

Software Engineering (2nd ed.): Principles and Practice.
John Wiley & Sons, Inc., New York, NY, USA, 2000.



Gregory V. Wilson.

Where's the real bottleneck in scientific computing?
Scientists would do well to pick some tools widely used in
the software industry.
American Scientist, 94(1), 2006.