CAS 741 (Development of Scientific Computing Software)

Winter 2025

03 Requirements

Dr. Spencer Smith

Faculty of Engineering, McMaster University

January 14, 2025

Requirements

- Administrative details
- Questions: project choices?, software tools?
- Problem statement questions?
- 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

- Accounts requested for all non-CAS students to access the publications repo
- Use the GitHub template to create a new repo
  - Add `smiths` to your GitHub repo
  - Remove SRS-Meyer, SRS-Volere, Reflection, Hazard Analysis, Presentations
- Create a fork and a merge request to modify Repos.csv with your project details
- No projects in Repos.csv so far
- Is anyone still considering dropping the course?

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: , , , , ,
  - Jan 28: , , , , ,
  - Jan 31: , , , , ,
  - Feb 4: ,

Presentation Sched Cont'd

Presentation Schedule

- 3 or 4 presentations each
  - SRS everyone
  - VnV and POC subset of class
  - Design subset of class
  - Implementation everyone
- If you will miss a presentation, please trade with someone
- Implementation presentation could be used to run a code review, or code walkthrough

Questions?

- Questions about project choices?
- Questions about software tools?
  - git?
  - LaTex?
- Questions about Problem statement and goals? (also see prob state checklist and writing checklist)

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
- We will focus on qualities relevant for software, including qualities for the documentation, code and executable

Important Qualities for Scientific Computing Software

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

Definitions in [8].

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 practise 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 visit a hotel, especially in another country, do you expect some confusion with operating the shower? the microwave? the TV?

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
- **If you project is relatively simple, usability is a place where your project can "shine"**

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

Productivity

- The productivity of a software development process is the measure of how efficiently the process produces software
- Productivity highly depends on the skills and organization of the development team
- Productivity is very hard to measure
- The number of lines of code per unit time is a terrible metric for measuring software productivity
- Productivity can be greatly increased by the use of development tools, environments, and methods
- Software reuse decreases productivity in the short term, but increases productivity in the long term
- See SmithAndCarette2021 for some thoughts on productivity

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)

Reproducibility

- The cornerstone of the scientific method [5]
- QA has, "a bad name among creative scientists and engineers" [16, p. 352], but participating in QA also improves reproducibility
- Reproducibility benefits from a consistent and repeatable computing environment, version control and separating code from configuration/parameters [5]
- Historically not well done for SCS
- Need for action is highlighted by a study of 402 computer systems papers - only 48.3% of the code was both available and compilable [3].
- [4] point out potential roadblocks for reproducibility, including page length constraints and differing detail needs depending on the audience

Reproducibility

- Interest is growing [1, 2]
- Progress on re-running old code with docker, VMs
- Replicability is rarely achieved, as shown for microarray gene expression [10] and for economics modelling [11]
- Long way to go to replicability from original theory

Sustainability

- The latest "buzz word"
- Seems to mean maintainability + productivity

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?

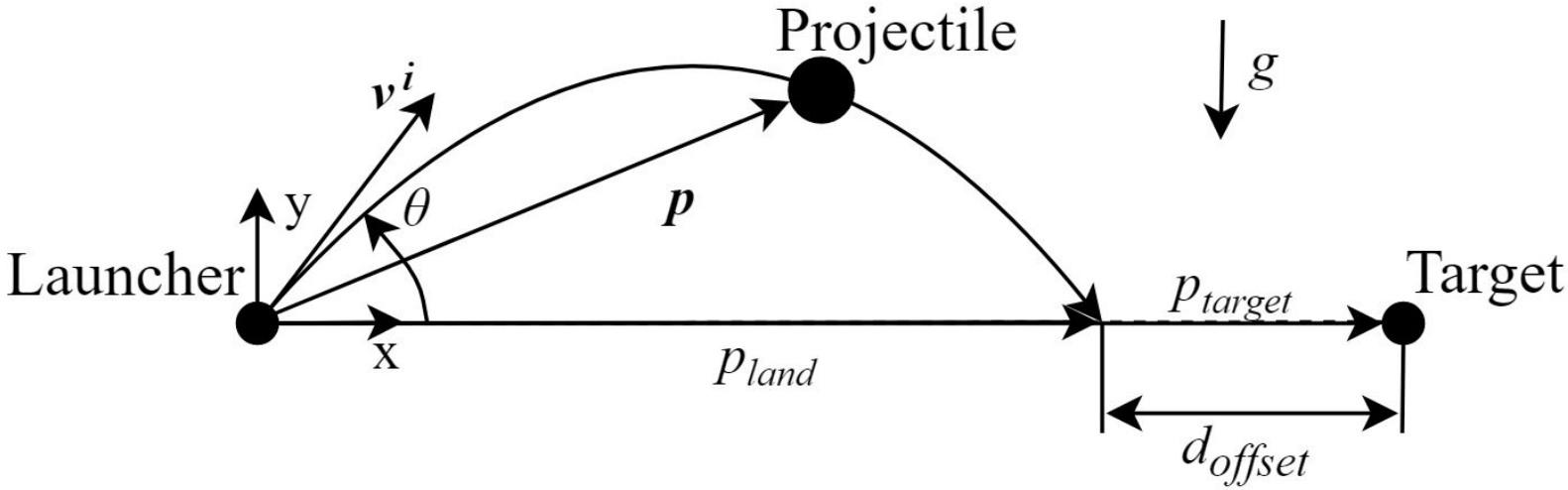
SRS versus CA

- SRS (Software Requirements Specification)
  - Requirements for a software product
  - Usually for specific physical problems
- CA (Commonality Analysis)
  - Requirements for a family of related software products
  - Sometime for specific physical problems
  - Commonly used for a library of general purpose tools
  - Distinguish commonalities, variabilities and parameters of variation

Big Picture View of SRS/CA

- Goal statement(s)
- Inputs and outputs

Requirements for Projectile



- Simplifying assumptions? (scope decisions, modelling decisions)
- Kinematic theories for translational motion?
- Data constraints on input? output?
- Types?
- Rationale?
- Refined Theories Projectile SRS

Goal Statements for SWHS

What are the goal statement for the Solar Water Heating System? (NoPCM SRS)

Think in terms of what are the inputs and outputs? For the goals, you might want to first think about the outputs.

Goal Statements for noPCM

Given the temperature of the heating coil, initial conditions for the temperature of the water, and material properties, the goal statements are:

GS1: Predict the water temperature over time.

GS2: Predict the change in the energy of the water over time.

- Consider using names instead of numbers for labels.
- For SWHS add goals related to the Phase Change Material

Goal Statements for GlassBR

For GlassBR:

Given the dimensions of the glass plane, glass type, the characteristics of the explosion, and the tolerable probability of breakage, the goal statements are:

GS1: Analyze and predict whether the glass slab under consideration will be able to withstand the explosion of a certain degree which is calculated based on user input.

Goal Statements for Game Physics

For Game Physics:

G_linear: Given the physical properties, initial positions and velocities, and forces applied on a set of rigid bodies, determine their new positions and velocities over a period of time (IM-IM_FT).

G_ang: Given the physical properties, initial orientations and angular velocities, and forces applied on a set of rigid bodies, determine their new orientations and angular velocities over a period of time. (IM-IM_FR).

G_dtcCol: Given the initial positions and velocities of a set of rigid bodies, determine if any of them will collide with one another over a period of time.

G_Col: Given the physical properties, initial linear and angular positions and velocities, determine the new positions and velocities over a period of time of rigid bodies that have undergone a collision (IM-IM_C).

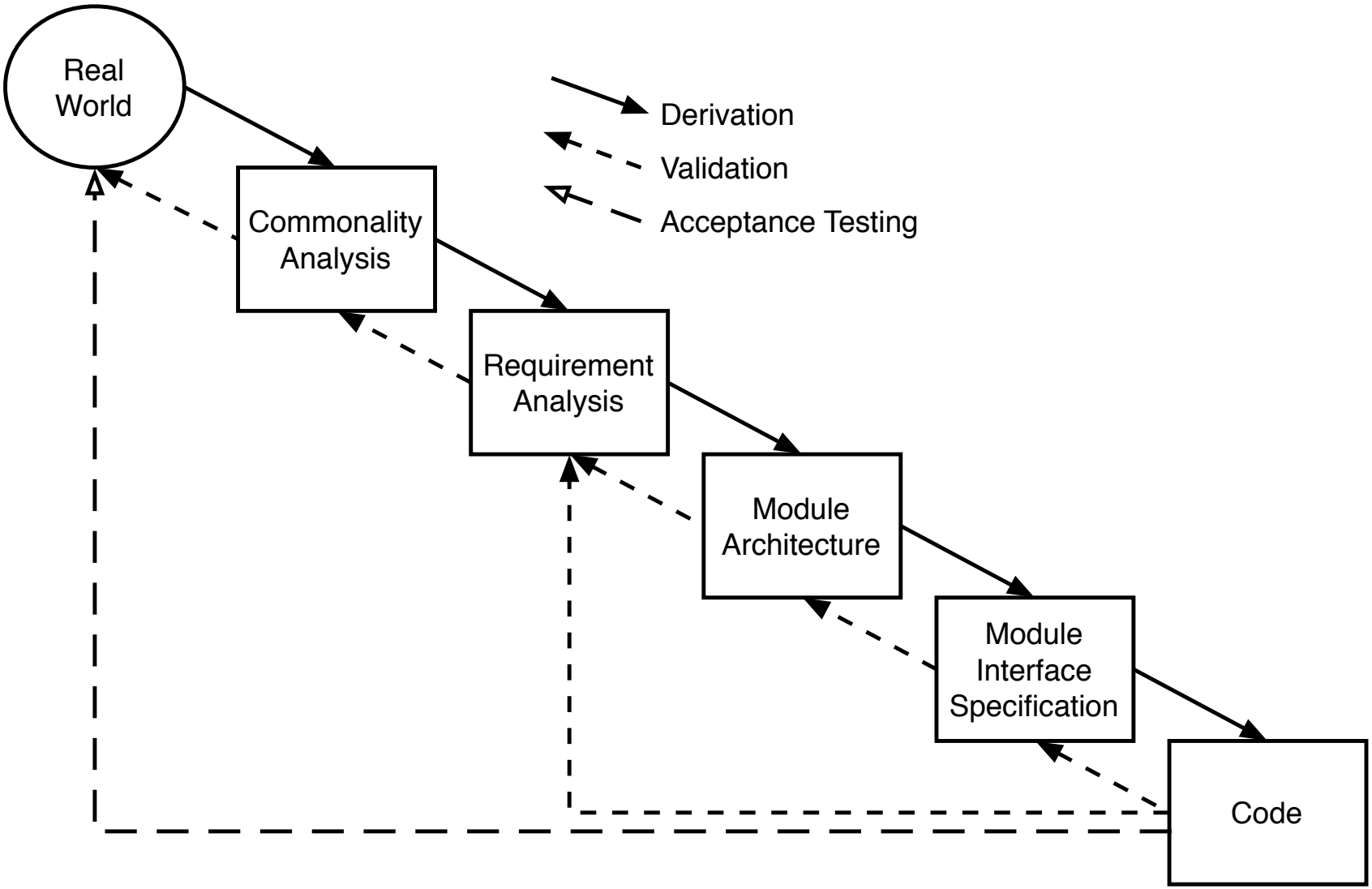
Goal Statements for Linear Solver

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

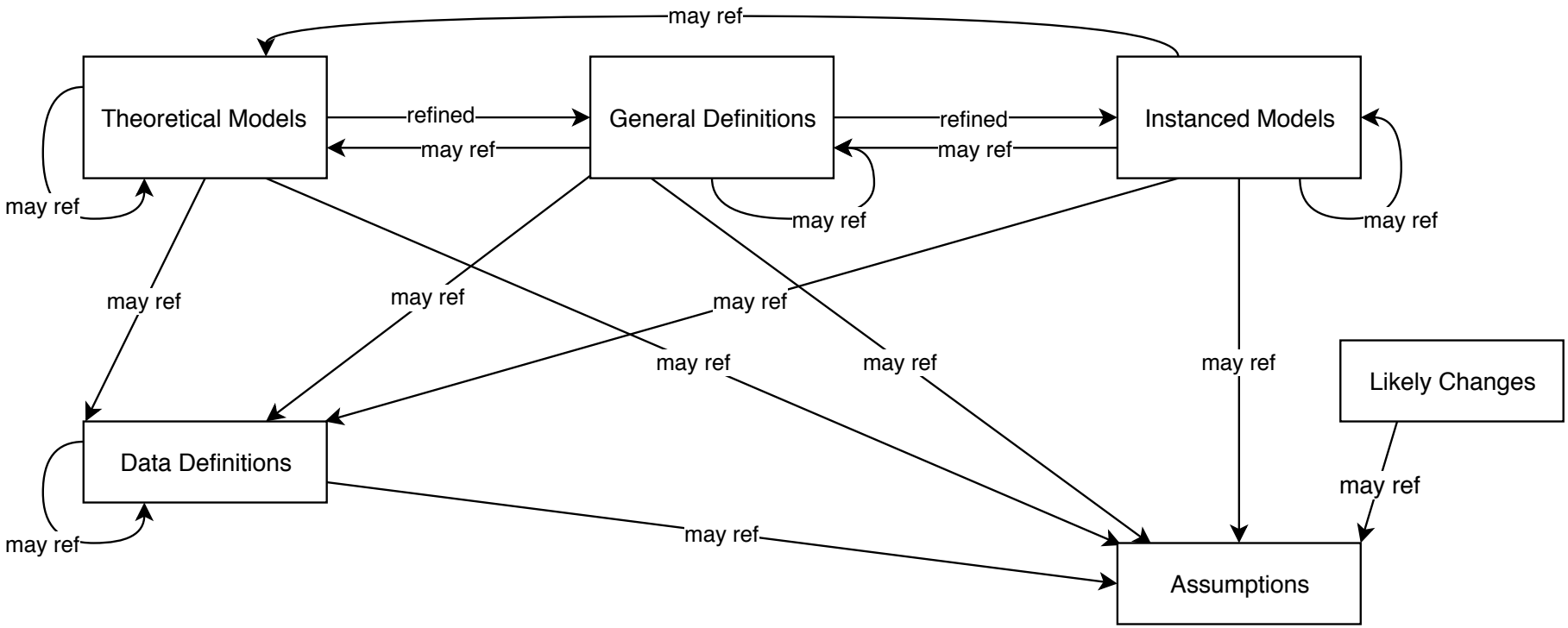
Goal Statements for Linear Solver

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

Relationship Between SRS and CA



Major Conceptual Parts of SRS/CA



Also Goal Statements and Requirements

Examples, Checklist and Template

- Projectile Example
- Refined Theories Projectile SRS
- GlassBR Example
- SWHS Example
- Aorta Example
- 3dfim+
- IMU-based Attitude Estimation
- Blank SRS from Template
- Checklist

References I

David H. Bailey, Jonathan M. Borwein, and Victoria Stodden.
Reproducibility: Principles, Problems, Practices, chapter Facilitating reproducibility in scientific computing: principles and practice, pages 205–232.
John Wiley and Sons, New York, 2016.

F. Benureau and N. Rougier.
Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions.
ArXiv e-prints, August 2017.

References II

Christian Collberg, Todd Proebsting, and Alex M Warren. Repeatability and benefaction in computer systems research.
Technical Report TR 14-04, Department of Computer Science, University of Arizona, Tucson, AZ, 2015.

Tom Crick, Benjamin A. Hall, and Samin Ishtiaq. "Can I implement your algorithm?": A model for reproducible research software.
CoRR, abs/1407.5981, 2014.

References III

A. P. Davison.
Automated capture of experiment context for easier reproducibility in computational research.
Computing in Science & Engineering, 14(4):48–56, July-Aug 2012.

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.

References IV


Paul F. Dubois.
Designing scientific components.
Computing in Science and Engineering, 4(5):84–90, September 2002.


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, October 1998.

References V

John PA Ioannidis, David B Allison, Catherine A Ball, Issa Coulibaly, Xiangqin Cui, Aedín C Culhane, Mario Falchi, Cesare Furlanello, Laurence Game, Giuseppe Jurman, et al.

Repeatability of published microarray gene expression analyses.

Nature genetics, 41(2):149–155, 2009.

Cezar Ionescu and Patrik Jansson.

Dependently-Typed Programming in Scientific Computing — Examples from Economic Modelling.

In *Revised Selected Papers of the 24th International Symposium on Implementation and Application of Functional Languages*, volume 8241 of *Lecture Notes in*

References VI

Computer Science, pages 140–156. Springer International Publishing, 2012.


R. Janicki and R. Khedri.
On a formal semantics of tabular expression.
Science of Computer Programming, 39(2-3):189–213, 2001.


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 VII

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.

Patrick J. Roache.
Verification and Validation in Computational Science and Engineering.
Hermosa Publishers, Albuquerque, New Mexico, 1998.

References VIII

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.

Judith Segal.
When software engineers met research scientists: A case study.
Empirical Software Engineering, 10(4):517–536, October 2005.

References IX

Judith Segal.
End-user software engineering and professional end-user developers.
In *Dagstuhl Seminar Proceedings 07081, End-User Software Engineering*, 2007.

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.

References X

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).

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,*

References XI

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.
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.

References XII


The Institute of Electrical and Electronics Engineers, Inc.
Software Requirements Engineering.
IEEE Computer Society Press, 2nd edition, 2000.