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

Fall 2017

03 Requirements

Dr. Spencer Smith

Faculty of Engineering, McMaster University

September 12, 2018

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

- Can everyone access GitLab?
  https://gitlab.cas.mcmaster.ca/smiths/cas741
- Create a GitHub account if you don't already have one
- 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 14
SRS Present	Week 04	Week of Sept 24
SRS	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

Questions?

- Questions about project choices?
- Questions about software tools?
  - git?
  - LaTex?
- Partial tex files in the blank project template
- Problem statement
- Copy the folder structure and README files from the blank project, but wait for tex files for deliverables

Problem Statement

- Written in LaTeX
- Due electronically (on GitHub) by deadline
- Generated files should NOT be under source control (except pdf)
- Comments might be typed directly into your source
- For all 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

- SpectrumImageAnalysisPy
- Aqueous Speciation Diagram Generator
- System of ODE solver library
- CParser
- FloppyFish
- Screenholders
- Template in repo

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

Definitions in [6].

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 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?

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

Goal Statements for SWHS

What are the goal statement for the Solar Water Heating System?

Goal Statements for SWHS

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

GS1: Predict the water temperature over time.

GS2: Predict the PCM temperature over time.

GS3: Predict the change in the energy of the water over time.

GS4: Predict the change in the energy of the PCM over time.

(Consider using names instead of numbers for labels.)

Goal Statements 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

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

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. 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 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.