SIAM CSE 2019

ROLE OF REQUIREMENTS IN SCIENTIFIC SOFTWARE

SCIENTIFIC SOFTWARE:
PRACTICES, CONCERNS, AND SOLUTION STRATEGIES (PART II OF II)

JARED O'NEAL
Mathematics & Computer Science
Argonne National Laboratory

U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

25 February 2019
Spokane, Washington, USA

ACKNOWLEDGMENTS & FUNDING

- This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.
- This work was performed in part at the Argonne National Laboratory, which is managed by UChicago Argonne, LLC for the U.S. Department of Energy under Contract No. DE-AC02-06CH11357

IDEAS productivity

https://ideas-productivity.org

ECP EXASCALE COMPUTING PROJECT

https://www.exascaleproject.org

better scientific software

https://bssw.io

INFORMAL DEFINITION

A complete collection of well-defined, mutually-consistent statements that define what you want to build and why these statements are important.

- What qualifies as “complete” is up to team
- Well-defined & mutually-consistent should not be optional

Requirements

- help understand **what we want** before we address **how to build it**,
- should be **verifiable**, and
- should be documented.

FUNCTIONAL VS. NON-FUNCTIONAL

Functional Requirements communicate what services should or should not be provided. This can include how they react to

- inputs and
- to corner/edge cases.

Example: A new feature shall be added to the SW such that simulations Z can be configured at runtime to use a lower-order, but more performant solver.

Non-functional Requirements communicate constraints on the services and functionality. These could be related to performance, portability, process, *etc*.

Example: The SW shall be developed as an open source project that is hosted on a Git-based version control host and shall have automated testing integrated in the repository for use with Continuous Integration.

LOW-LEVEL REQUIREMENTS

- Technically-detailed or result of heavy constraints
- Possibly informed by implementation ideas & constraints
- Overly specific can hinder design, creativity, & freedom
- Functions, classes, and sub-systems can be developed through **design by contract** (interface specification)

Example: The SW architecture shall be upgraded such that a simulation can be run on nodes with Model X CPUs and Model Y GPUs. The use of GPUs shall be determined by the pre-processor.

HIGH-LEVEL REQUIREMENTS

- Broad ideas, concepts, constraints, and abstractions
- Little technical detail
- Can be understood by people from different disciplines
- Not affected as strongly by changes
- Can be difficult for non-experts to turn into implementations

Example: The SW architecture shall be upgraded such that a simulation can be built to run on a node with only CPUs or on a node with accelerators.

EXTERNALLY-IMPOSED

Functional or Non-functional requirements due to

- Use of third-party libraries
- Working as a team of teams, or
- Including standardization (e.g. xSDK Community Package Policies)

PARTICIPANTS

Requirements should capture viewpoints of different roles related to the development, maintenance, and use of the SW so that we discover more constraints & identify problems early

- Domain experts can define need, limits, & tolerances
- Developers & technical experts understand technical constraints
- Users define interfaces

EXAMPLE DESIGN WORKFLOW

- Science/Engineering Cases
- Derive Requirements from S/E Cases
  - Requirement elicitation, specification, & validation
  - Determine tests needed to confirm that requirements are satisfied
- Convert Requirements into Design
  - Generate low-level technical specifications
  - Create design that satisfies specifications
- Implement
- Verification – did we satisfy the requirements
- Validation – do the requirements result in SW that has correct/useful results

USER STORIES

A form of requirement elicitation

As a …, I would like … so that ….

These statements

- express what needs to be done or a constraint on what we can do and
- encapsulate the reasons why the need or constraint should be considered.

User stories should start a discussion that concludes with requirements and possibly tasks to start work.

ELICITATION & SPECIFICATION

As a user of the SW, I would like the storage of data to make good use of HPC resources and to leverage pre-existing libraries for reading data so that my simulations run in less time and time to results is reduced.

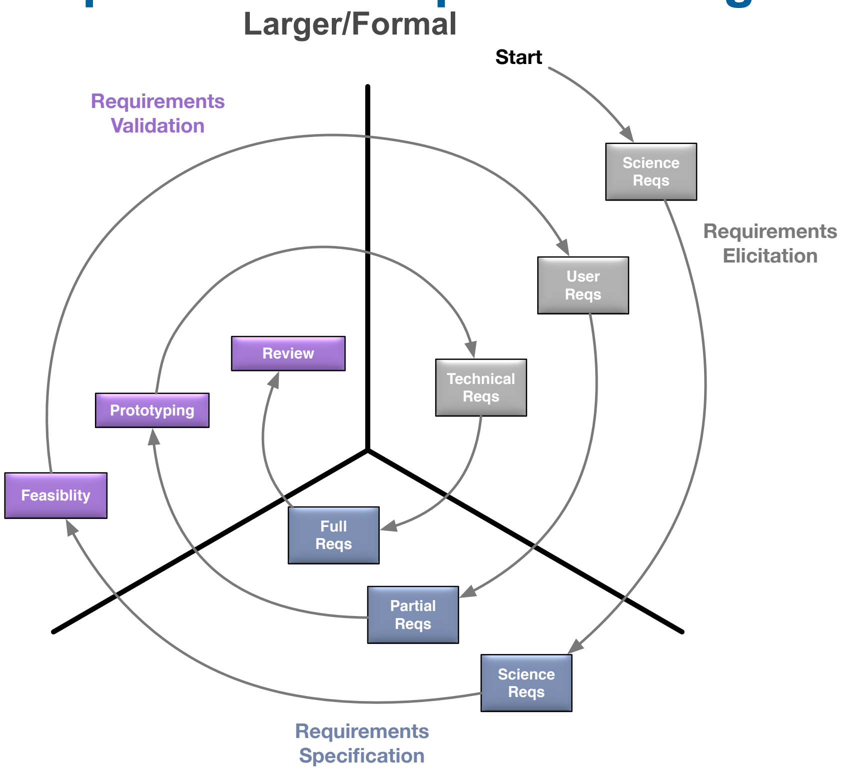
V1: The SW shall record simulation results, configuration values, hardware information, and telemetry via a parallel IO library and using a standard file format.

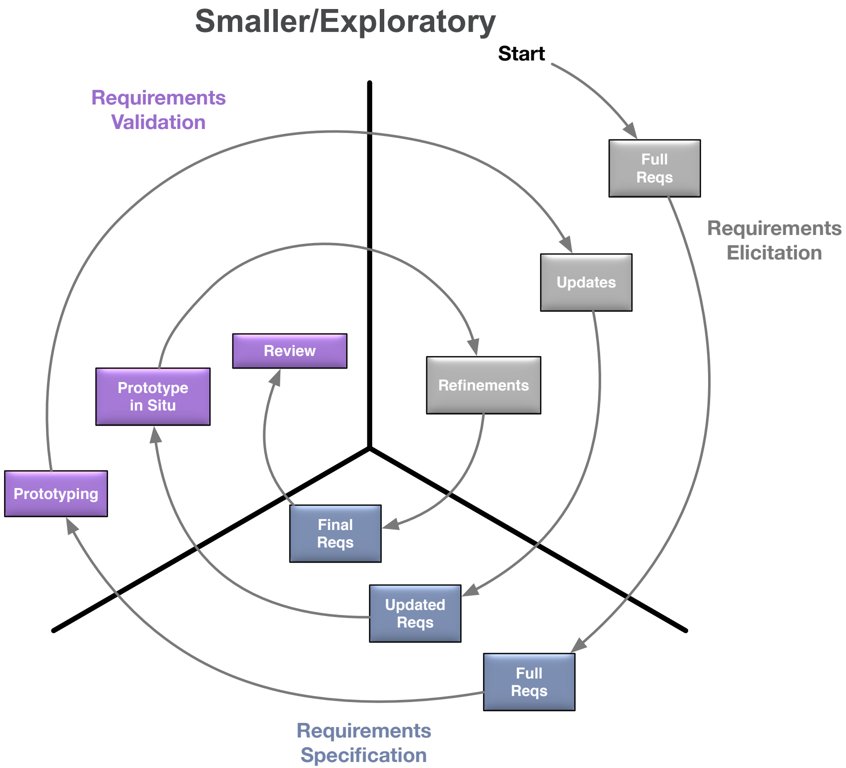
V2: The SW shall record simulation results, configuration values, hardware information, and telemetry via a parallel IO library and using a file format that is included in python, R, MATLAB, and C/C++.

V3: The SW shall record simulation results, configuration values, hardware information, and telemetry via parallel IO library XYZ v1.2.3 or greater.

ITERATION & PROTOTYPING

Requirements require refining





DOCUMENTATION

Requirements Management

- Documents should be clear, readable by many, & living
- Documentation maintenance should be easy & simple
- Design-by-contract requirements & motivation can be comments and inline documentation
- Should high-level or system-level requirements
  - Go into dedicated document?
  - Be included in the developer's guide or adapted for user guide?
  - Be a history of static requirements documents?
  - Be encoded in system-level test cases?

ARE REQUIREMENTS FOR CSE?

The Bad & Ugly

- Can be challenging and frustrating
- Can be seen as impediment to immediate progress
- Requirements change
  – Due to changing environment
  – Due to improved understanding
- Hard to know when enough is enough

ARE REQUIREMENTS FOR CSE?

The Good

- Achieve a clear & shared understanding of what needs to be done,
- Arrive at definitions & concepts that are understood by all,
- Bring out in the open ideas that seem obvious to some and usually go unstated,
- Bridge differences between disciplines & levels of expertise,
- Discover constraints/problems early,
- Link requirements with verification,
- Build a team where members feel like an important part of the process, and
- Arrive at idea of SW architecture through structuring/grouping requirements.

SOURCES

Selected Books

Textbooks

1. Ian Sommerville, **Software Engineering.**
2. Benjamin S. Blanchard and Wolter J. Fabrycky, **Systems Engineering and Analysis.**

Popular books

1. Andrew Hunt and David Thomas, **The Pragmatic Programmer.**
2. Steve McConnell, **Code Complete 2.**

Chapters

1. Alberto Sillitti and Giancarlo Succi, ”Requirements Engineering for Agile Methods” in **Engineering and Managing Software Requirements**.

SOURCES

Selected Articles

1. Yang Li, Emitza Guzman & Bernd Brügge, **Effective Requirements Engineering for CSE Projects: A Lightweight Tool**, 2015.
2. Dustin Heaton & Jeffrey C. Carver, **Claims about the use of software engineering practices in science: A systematic literature review**, 2015.
3. Yang Li, Matteo Harutunian, Nitesh Narayan, Bernd Brügge and Gerrit Buse, **Requirements Engineering for Scientific Computing: A Model-Based Approach,** 2011.
4. Sarah Thew, Alistair Sutcliffe, Rob Procter, Oscar de Bruijn, John McNaught, Colin C. Venters, & Iain Buchan, **Requirements Engineering for E-science: Experiences in Epidemiology**, 2009.

DESIGN BY CONTRACT

Example of designing smart iterator interface

Specify the interface of a function, class, or sub-system – fix a contract between the code and the user.

- Iterators shall be initialized for use by the `Grid_getBlkIterator` public routine and in accord with the values provided for the dummy input parameters `nodetype`, `level`, and `tiling`. All code that initializes an iterator with `Grid_getBlkIterator` shall release the iterator with the `Grid_releaseBlkIterator` public routine once the iterator is no longer needed.
- The iterator shall abort program operation if it is used before being initialized with the `Grid_getBlkIterator` public routine.
- The iterator may provide access to blocks without ensuring any particular ordering.
- The iterator shall implement a method called `isDone` that returns TRUE if the iterator has already walked all its associated boxes and shall return FALSE otherwise.

COST OF SKIPPING REQUIREMENTS

From Code Complete 2 by Steve McConnell

A (possibly outdated) summary of cost to solve problem for large, SW projects from many organizations

Problem Discovered During

Problem Introduced	Requirements	Architecture	Implementation	System Test	Post-Release
Requirements	1	3	5-10	10	10-100*
Architecture		1	10	15	25-100
Implementation			1	10	10-25

* For smaller projects this is closer to 5-10 times the cost.

REQUIREMENTS IN SW

Results of study by Standish Group

Problem	Main Cause of Project Failure
Incomplete requirements	13.1%
Low customer involvement	12.4%
Lack of resources	10.6%
Unrealistic expectations	9.9%
Lack of management support	9.3%
Changes in the requirements	8.7%
Lack of planning	8.1%
Useless requirements	7.5%

Alberto Sillitti and Giancarlo Succi, "Requirements Engineering for Agile Methods" chapter in **Engineering and Managing Software Requirements,** Springer, 2005.