Assessing the Quality of Scientific Software

1st Int'l Workshop SECSE08
May 2008 Leipzig

Royal Military College of Canada

Diane Kelly
Kelly-d@rmc.ca

Queen's University Kingston, Canada

Rebecca Sanders
Sanders@cs.queensu.ca

Statement of the Problem

- What are reasonable quality assessment techniques/methods/practices for scientific software?
  - "Reasonable"= seen as useful and useable by scientists

What qualities are important?

Correctness

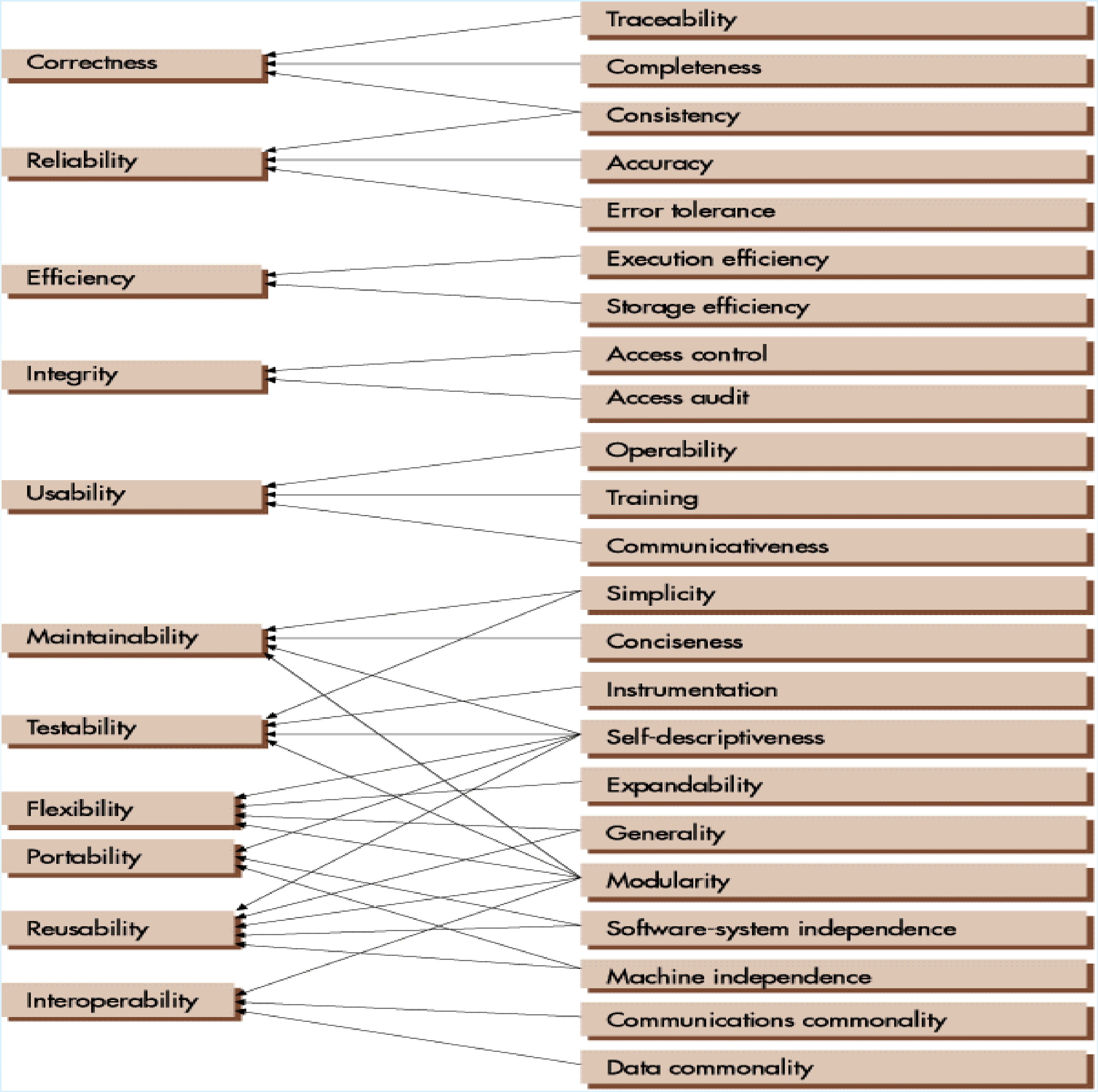
- If this is not acceptable, entire system is useless
- Qualities such as on-time, under-budget, maintainability, usability, portability, efficiency, reusability take a back seat

However ….

Need an operational definition of Correctness.

McCall's Quality Model

Quality Factors



Quality Criteria

Operational Definition of Correctness

- Disconnect between idealized concept of quality and associated assessable quantity(s)

- McCall's Quality Model suggests:
  - Traceability, Completeness, Consistency

- Every software system will have a different definition even for correctness
  - Eg. No known 'severe' errors where

    'severe' = error that affects conclusions in safety reports to regulators

  How do you operationalize this?

What do scientists currently do?

- Mostly “big bang” testing
  - Check results from entire software against
    - Algebraic calculations
    - Data from physical experiments
    - Measurements or observations from real world events
    - Stored benchmark sets
    - Other software results

- Obvious problems with this
    - Oracle may be wrong
    - Testing is seriously incomplete
    - There are errors masking other errors
    - …

Assessment confounded by (1)

- Scientific mindset
  - They test to show the theoretical models are correct
    - They never test to show the software is WRONG
  - The software is invisible
    - They only see their models
  - They want to do science
    - They don’t want to be spending huge amounts of time doing software without obvious progress towards their science
    - Recognition is for their science, not the software

Assessment confounded by (2)

- Problems with test oracles
  - Data is difficult or impossible to gather
  - Scenarios are limited
  - Use test data to do model fitting or fine tuning
    - No data left over for independent tests
  - Errors in oracles due to faulty measurements and misinterpretations
  - Answering the question of what is close enough
  - Misjudgment
    - TLAR

Assessment confounded by (3)

- Errors hiding in each version and translation of the model
  - Procedural part of model
    - Idealized real, continuous scientific model
    - Discretized/computationalized approximations
    - Code on a machine with limited precision and arithmetic
  - Data part of the model
    - Measurements or observations from physical world
    - Discretized/computationalized approximations
    - Data identifiers and data structures in the code

Assessment confounded by (4)

- Confusion about V&V
  - Definitions for V&V based on process (eg. ISO standards)
    - Ignore the computational problems
  - Model centric definitions from the computational community
    - Ignore software problems
  - Confused combinations of the two
    - Eg. CSA standards for computational code
      - Composed by non-software people reading out-of-date software engineering textbooks

Assessment confounded by (5)

- Existing assessment methods not examined/refined specifically for computational software
  - Inspecting/code reading
    - Some work done here but not wide spread in industry
  - Formal methods
    - How can we address correctness?
  - Testing
    - Need for effectiveness
      - Addressing correctness
    - Need for efficiency
      - Time- and effort- efficient
    - Need for acceptance by the scientists
      - "Why didn't I think of this?"

Assessment confounded by (6)

- Lack of communication between the software engineering and computational science communities
  - Computer science curricula does not include sciences or even calculus
  - Deep domain knowledge embedded in software
    - Content adds to complexity of software
      - Addressed by keeping the code as simple as possible
    - Software is not readily accessible to software engineers
  - Software engineers lack of interest in computational software
  - Computational scientists lack of trust of software engineering

Assessment confounded by (7)

- Different understandings of priorities
  - Computational science communities
    - Correctness
      - Over a very long lifetime of the software
  - Software engineering community
    - Computational speed
      - Eg. high performance computing
    - Usability
      - User interface design and GUI languages
    - Rapid development, first delivery, time, budget
      - Agile methods, OOD and OOP
    - Reliability
      - Process standards, formal methods, probabilistic testing

Where do we go from here?

a) What software engineering topics are suitable to be included in courses for scientists?
    - Now?
    - Later?

b) How can we leverage ideas from the scientific method?

c) How do we create operational definitions for correctness?

d) How do we fit assessment activities into the established practices of scientists?

e) What good things do scientists do now?

f) What inspection/reading techniques are amenable to scientists?

g) How do we address correctness with formal methods?

h) How do we address correctness with testing?

Thank-you
from
Diane and Rebecca

Questions?