Introduction to Testing scientific software

Stefano Cozzini

CNR/IOM Democritos and eXact lab srl

- Trieste

Outline

- Introducing the problem:
  - What does testing software mean ?
  - Why,where,when,who, what to test ?
- Scientific software testing..
  - What is scientific computing ?
  - Why is so difficult to test simulations ?
  - Reviewing verification/validation
  - Some idea/suggestions

Aim of this lecture

- Share some interesting ideas I read about and I tried to apply to my activity
- Tell about some current effort to apply some of such ideas to a scientific software development
- Convince you that testing scientific software is needed

- Disclaimer:
  - Content of these slides come from many sources all around the web; some slides are mine, some other are taken from other presentations without modification, some other are slightly modified

First definitions : errors and faults

Error

is a measure of the difference between a measured or calculated value of a quantity and what is considered to be its actual value.

Faults

Code faults are mistakes made when abstract algorithms are implemented in code.

Faults are not errors, but they frequently lead to errors.

Second definition: testing

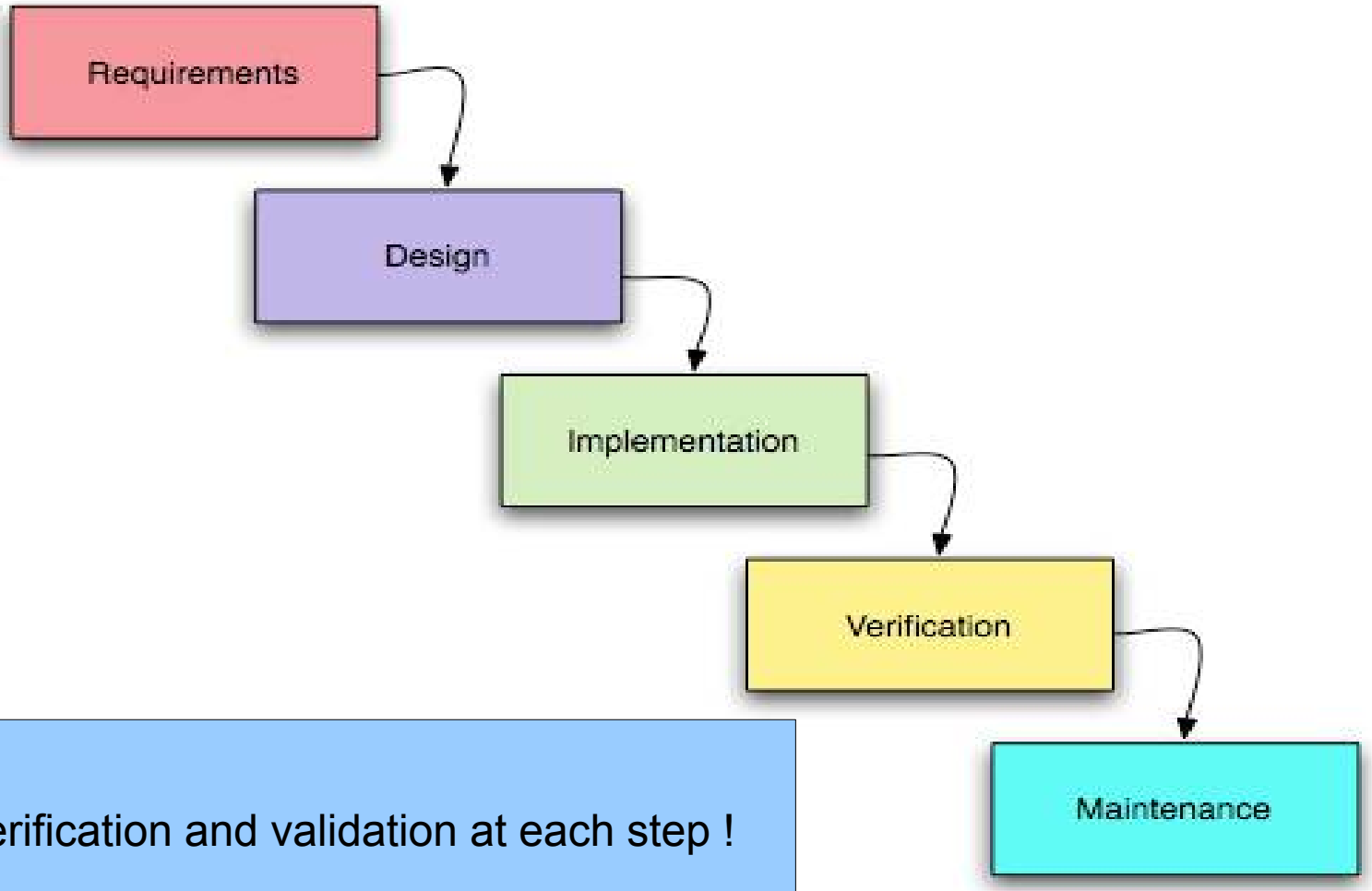
- Software testing is a process by which one or more expected behaviors and results from a piece of software are exercised and confirmed. Well chosen tests will confirm expected code behavior for the extreme boundaries of the input domains, output ranges, parametric combinations, and other behavioral edge cases.
- Software testing can be stated as the process of validating and verifying that a software program/application/product:
  - meets the requirements that guided its design and development;
  - works as expected; and
  - can be implemented with the same characteristics.

(from wikipedia)

Third definition: Verification validation

- Verification: "Are we building the product right ?"
  - The software should conform to its specification
- Validation: "Are we building the right product ? "
  - The software should do what the user really requires
- V & V must be applied at each stage in the software process
- Two main objectives:
  - Discovery of defects in a system
  - Assessment of whether the system is usable in an operational situation

Software stages: waterfall model



Verification and validation at each step !

Is all the above true for scientific codes/packages/ simulation method ???

- Sure !
- But something more is required...

The 5 W of testing: Why testing ?

- To find faults
- To provide confidence
  - of reliability
  - of (probable) correctness
  - of detection (therefore absence) of particular faults
- Other issues include:
  - Performance of systems (i.e. use of resources like time, space, Bandwidth,...).
  - "...ilities" can be the subject of test e.g. usability, learnability, reliability,availability, etc..

The 5 W of testing: When testing ?

- When:
  - Always !
- Different granularity:
  - When I/they change of the code
  - at every night to check possible problems
  - When a new feature/release is available
  - When we start using a new package for scientific research

The 5 W of testing: Who should test ?

- Who:
  - You as developer
  - You as part of a developing team:
    - Try to test things you did not write
    - Find some other to test your own software
  - You as user:
    - Is this software/package/routines/code what I really need ?
  - You as scientific user (never use scientific code without your own test !!!)

The 5 W of testing: Where to test ?

- Software point of view
  - On any single function of your code:
    - Unit testing
  - On the code as a whole
    - Regression tests
- Hardware point of view
  - On all the possible platforms you have at disposal
    - Ensure portability of software and of scientific output !

The 5 W of testing: What should I test ?

- Software characteristics:
  - Usability
  - Portability
  - Performance
  - Reliability
  - Scalability
- Scientific Software correctness

Yet another definition: Scientific Computing

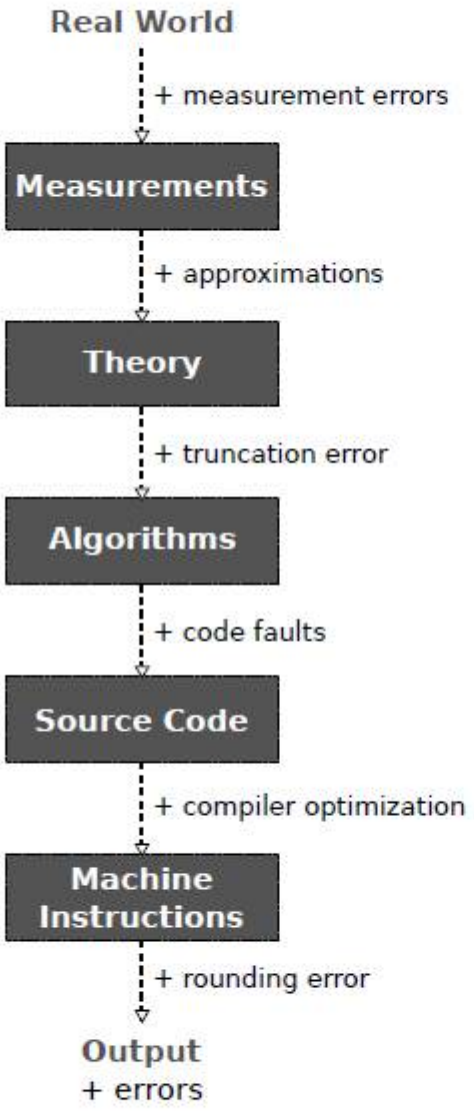
- Computational science (or scientific computing) is the field of study concerned with constructing mathematical models and quantitative analysis techniques and using computers to analyze and solve scientific problems.[Wikipidia]
- Distinguish features:
  - concerned with variables that are continuous rather than discrete
  - concerned with *approximations* and their effects
- Approximations are used not just by choice: they are inevitable in most problems

What do you need to do scientific computing ?

- Computers
  - (generally very huge/large and/or many of them)
- Software:
  - Something that allow you to use the computer (operating system/ middleware/compilers/libraries)
  - Something that allow you to simulate/compute what you needed (scientific programs / codes)
- Goodwill:
  - Learn how to use the right tools among a huge number of them
  - Learn how to write/use scientific program

Source of approximations

- Before computation begins:
  - Modeling: *neglecting certain physical features*
  - empirical measurements: *can't always measure input data to the desired precision*
  - previous computations: *input data may be produced from error-prone numerical methods*
- During computation:
  - truncation: *numerical method approximate a continuos entity*
  - rounding: *computers offer only finite precision in representing real numbers*



Scientific software development involves a number of model refinements in which errors may be introduced.

Program outputs include the accumulation of all these errors.

Stupid example

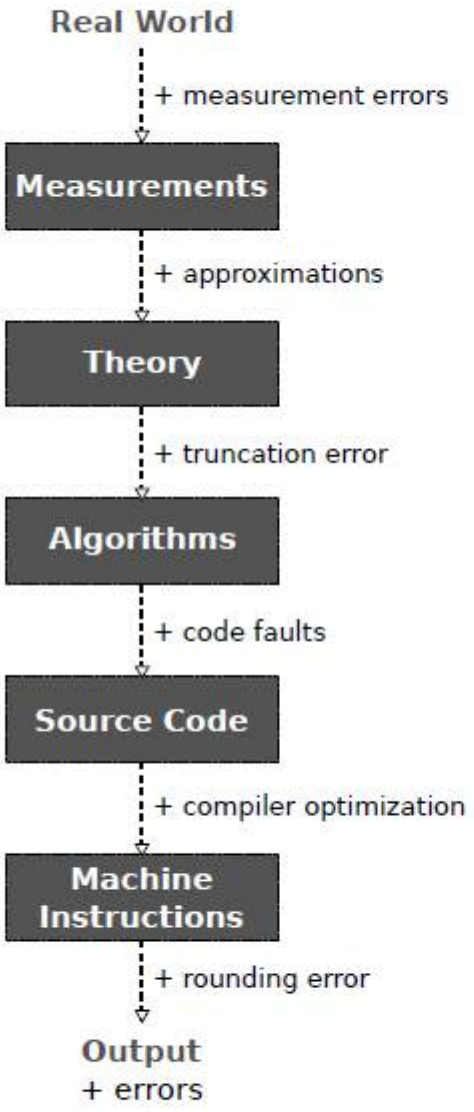
- Computing surface area of Earth using formula

$$A=4\pi r^2$$

- This involves several approximations:
  - Modeling: Earth is considered as a sphere…
  - Measurements: value of radius is based on empirical methods
  - Truncation: value for π is truncated at a finite number..
  - Rounding: values for input data and results of arithmetic operations are rounded in computer.

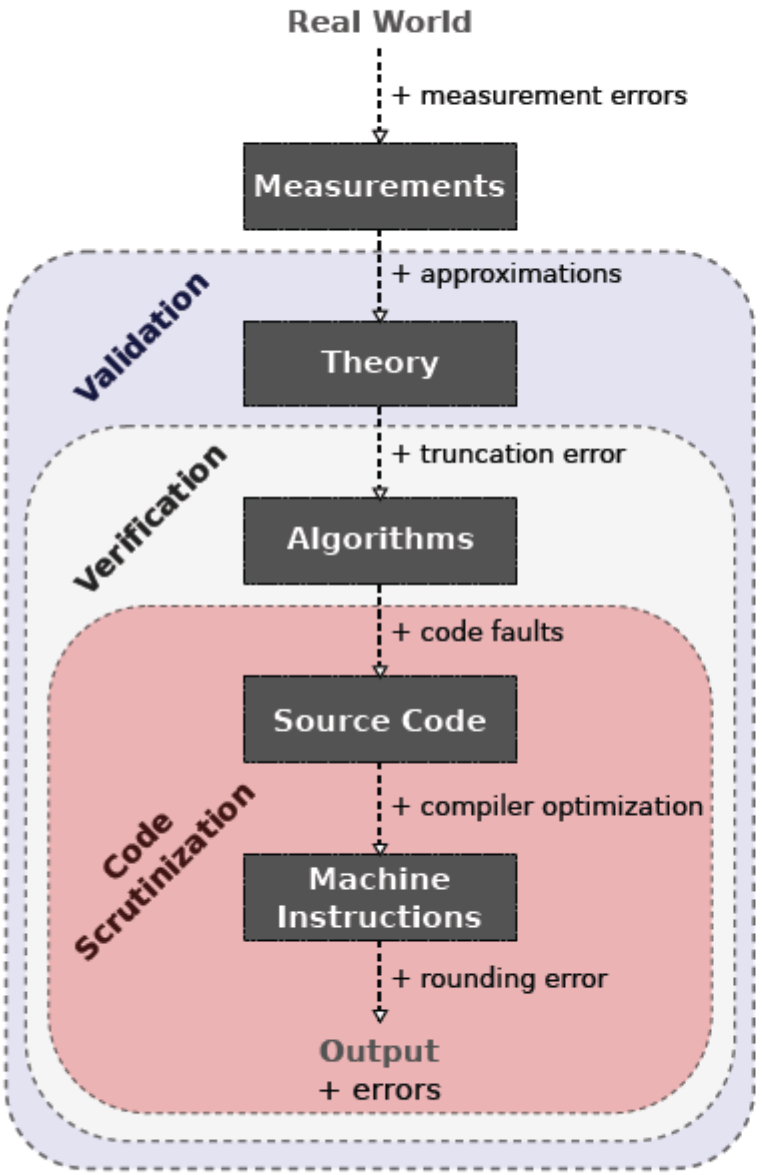
Rounding error

- Difference between result produced by a given algorithm using exact arithmetic and result produced by the same algorithm using rounded arithmetic
- Due to the inexact representation of real numbers and arithmetic operations upon them
- To understand where and how they turn out we need to know how computers deals with numbers..
- Error analysis techniques: how are your equations sensitive to roundoff errors ?
  - Forward error analysis: what errors did you make ?
  - Backward error analysis: which problem did you solve exactly ?



Scientific software development involves a number of model refinements in which errors may be introduced.

Program outputs include the accumulation of all these errors.



Validation and verification activities are applied in an attempt to identify or bound these errors.

In addition to validations and verifications, we also suggest that computational scientists should conduct *code scrutinizations*.

Scientific simulation Context

Our simulations provide *approximate* solutions to problems for which we do not know the exact solution.

This leads to two more questions:

- How good are the approximations?
- How do you test the software?

Verification and Validation (V&V) Definitions

Verification: Are the equations solved correctly?
(Math)

Validation: Are the equations correct?
(Physics)

- Verification: The process of determining that a model implementation accurately represents the developer's conceptual description of the model and the solution to the model.
- Validation: The process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model.

V&V targets applications of codes, not codes.

a few more informal Definitions

Software Quality Engineering (SQE)	Manage software complexity
Code Verification	Assess algorithms and their implementation vs. exact solutions
Solution Verification	Estimate discretization error
Validation	Assess physics models vs. experimental data
SA / UQ	Assess sensitivity of answer to input parameters

An ingredients list for predictive simulation, *not a menu*.

Code Verification As A Continuous Process

- To set up a verification problem once takes significant effort – steep learning curve, infrastructure is not in place
- Running a verification analysis you have maintained takes minimal work
- Without regular, automated verification testing, verification results go stale quickly - they do not reflect the current state of the code

```
while 1:
  …
  run verification_suite
```

Code Verification Is Not Free

Principal Costs:

- Infrastructure development
- Test development

Recurring Costs – A tax on development:

- Maintenance of existing tests
- Code development becomes a very deliberate process

Sustainable verification: Benefits outweigh costs

The Most Efficient Code Verification is Done by Code Developers

- Code developers best understand the numerical methods they are using

- Code developers are best able to use the results of code verification (and other forms of assessment) to improve the algorithms they use

Verification Testing Must Be a Team Ethic

- Discipline is required to keep a "clean" test suite – to keep all tests passing; "stop-the-line" mentality
- If only part of the team values verification, that part is always carrying the other developers
- Maintaining an automated verification test suite is probably necessary but *definitely not sufficient*
- Developers should be using verification tests interactively

Verification is orthogonal to UQ and SA

- Uncertanty Quantification/Sensitivity Analysis :
  - Given code ("model") compute outputs to examine uncertainty, sensitivity.
- Code Verification: Given inputs compute exact error to examine code.
-
  -

Alternative statement: UQ/SA and Code Verification are complementary.

Final ideas: what should we provide....

A software infrastructure to support testing...

- Ability to run a large number of simulations repeatedly
- Ability to compare results with a baseline
- Ability to report pass/diff/fail
- Run on platforms of interest at regular intervals