Giuseppe Piero Brandino

Introduction to software testing

Outline

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

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.[Wikipedia]
- Distinguishing features:
  - concerns with variables that are continuous rather than discrete
  - concerns with *approximations* and their effects
- Approximations are used not just by choice: they are inevitable in most problems

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 continuous entity*
  - Rounding: *computers offer only finite precision in representing real numbers*

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 ?

Errors and faults

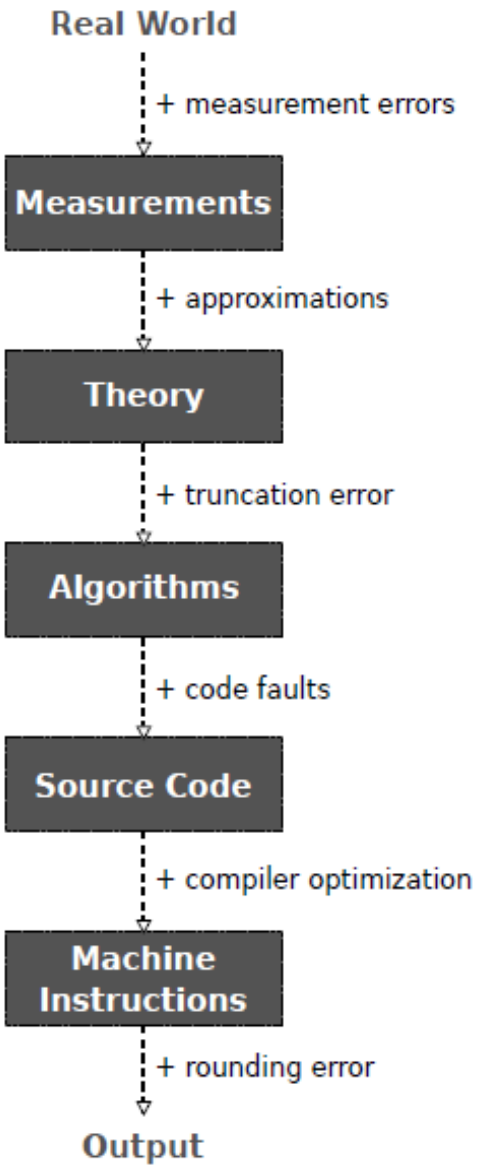
Error

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.



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.

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

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
  - can be implemented with the same characteristics.

(from wikipedia)

The 5 W of testing: Why testing ?

- To provide confidence
  - on reliability
  - on (probable) correctness
  - on 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, availability, etc..

The 5 W of testing: When testing ?

- When:
  - Always !
- Different granularity:
  - When I/they change of the code
  - Every night to check possible problem
  - 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 you 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 a portion of the code
    - Integration testing
  - On the code as a whole
    - Regression tests
    - Acceptance 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

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.

Scientific Computing

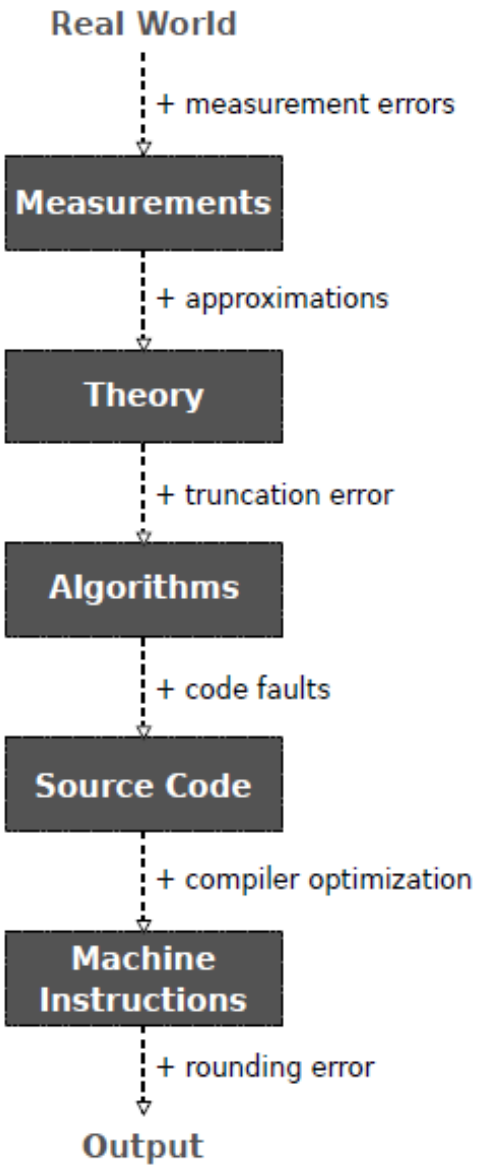
- Computational science (or scientific computing) is the field of study concerned with constructing mathematical models and quantitative analysis techniques and using computers to analyze and solve scientific problems.[Wikipedia]
- Distinguishing features:
  - concerns with variables that are continuous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

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 continuous entity*
  - Rounding: *computers offer only finite precision in representing real numbers*

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.

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.


exact
Giuseppe Piero Brandino
Testing in details

Type of testing

- Unit testing
- Integration testing
- Regression testing
- Acceptance testing

Unit testing

- is a level of the software testing process where individual units/components of a software/system are tested
- The purpose is to validate that each unit of the software performs as designed

Unit testing

- concerned with functional correctness and completeness of individual program units
- typically written and run by software developers to ensure that code meets its design and behaves as intended.
- Its goal is to isolate each part of the program and show that the individual parts are correct.

Unit testing

Concerned with

- Functional correctness and completeness
- Error handling
- Checking input values (parameter)
- Correctness of output data (return values)
- Optimizing algorithm and performance

Unit testing shoul be

- Isolatable
- Repeatable
- Automatable
- Easy to Write

Unit testing

- Unit testing allows the programmer to refactor code at a later date, and make sure the module still works correctly.
- By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier.
- Unit testing provides a sort of living documentation of the system.

Unit test guidelines

- Keep unit tests small and fast
  - Ideally the entire test suite should be executed before every code push. Keeping the tests fast reduce the development turnaround time.

- Unit tests should be fully automated and non-interactive
  - The test suite is normally executed on a regular basis and must be fully automated to be useful. If the results require manual inspection the tests are not proper unit tests.

Unit test guidelines

- Make unit tests simple to run
  - Configure the development environment so that single tests and test suites can be run by a single command or a one button click.
- Keep tests independent
  - To ensure testing robustness and simplify maintenance, tests should never rely on other tests nor should they depend on the ordering in which tests are executed.

Integration testing

- Testing in which software components, hardware components, or both together are combined and tested to evaluate interactions between them
  - Big-bang
  - Incremental
    - Bottom-up
    - Top-down
    - Sandwich

Integration testing - Bing bang

Here all component are integrated together at once and then tested.

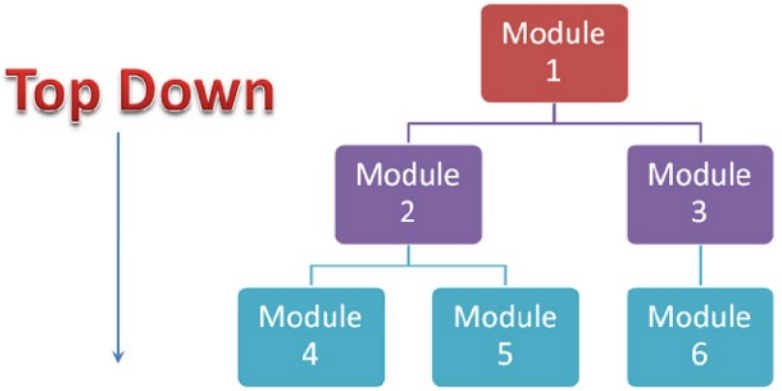
- Advantages:
  - Convenient for small systems.
- Disadvantages:
  - Fault Localization is difficult.
  - Given the sheer number of interfaces that need to be tested in this approach, some interfaces link to be tested could be missed easily.

Integration testing – Stubs and Drivers

- Incremental Approach is carried out by using dummy programs called Stubs and Drivers.
- Stubs and Drivers do not implement the entire programming logic of the software module but just simulate data communication with the calling module.
  - Stub: Is called by the module under test.
  - Driver: Calls the module to be tested.

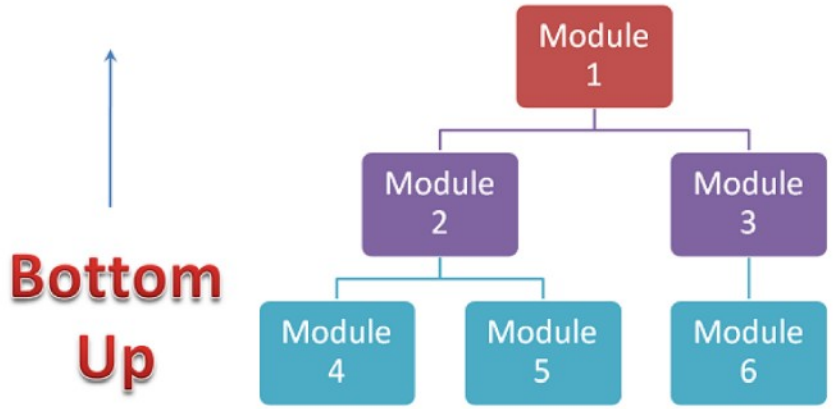
Integration testing – Top-down

- Advantages:
  - Fault Localization is easier.
  - Possibility to obtain an early prototype.
  - Critical Modules are tested on priority; major design flaws coul be found and fixed first.
- Disadvantages:
  - Needs many Stubs
  - Modules at a lower level are tested inadequately.



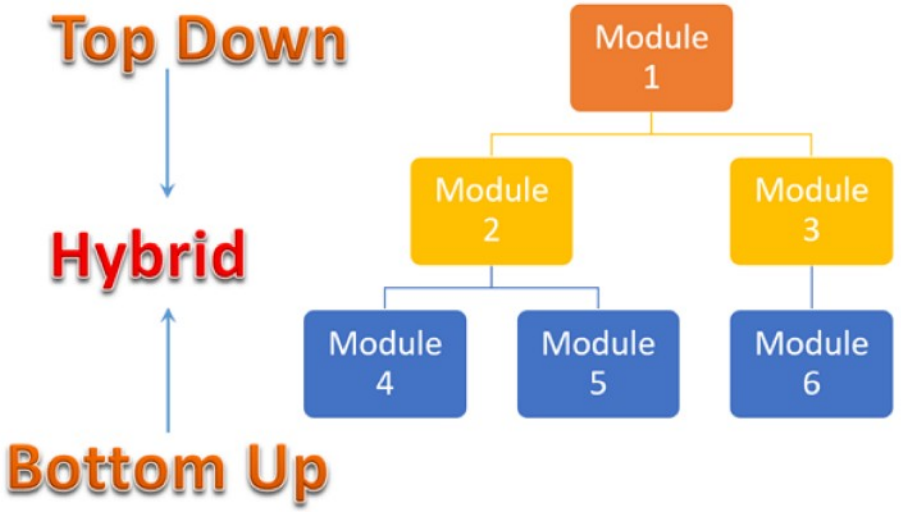
Integration testing – Bottom-up

- Advantages:
  - Fault localization is easier.
  - No time is wasted waiting for all modules to be developed unlike Big-bang approach
- Disadvantages:
  - Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
  - An early prototype is not possible



Integration testing – Sandwich

- In the sandwich/hybrid strategy is a combination of Top Down and Bottom up approaches.
- Here, top modules are tested with lower modules at the same time lower modules are integrated with top modules and tested.
- This strategy makes use of stubs as well as drivers.



Regression testing

- is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features.
- is nothing but a full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.
- This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that the old code still works once the new code changes are done.

Acceptance test

- Formal testing with respect to user needs, requirements, and business/scientific processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.

Testing framework

- C
  - CTest
  - Cunit
- Python
  - Unittest
  - Nose
  - pytest