Software Testing in Science

Dr. Axel Kohlmeyer

Senior Scientific Computing Expert

Information and Telecommunication Section
The Abdus Salam International Centre
for Theoretical Physics

http://sites.google.com/site/akohlmey/

akohlmey@ictp.it

Scientific Software Development Idiosyncrasies

- Scientific software is developed to solve specific problems, not to generate revenue
  => less pressure to prove a feature is working
- The developer is often also the customer
  => superficial testing is considered adequate, since you have confidence in your capabilities
  => How can the software be wrong if it gives the right answer, anyway?
- Many developers have no formal training in software engineering, so they don't even know

More Scientific Software Development Idiosyncrasies

- There is little credit to be had for software development compared to using the software => any additional effort invested besides the minimum is reducing the competitiveness
- For the same reason, it is difficult to obtain funding for scientific software development
- The bulk of the software development work is done by inexperienced people (students, post docs); advisers are not trained in management (of software development projects)

Even More Scientific Software Development Idiosyncrasies

- The correctness of the result is not affected by the quality or efficiency of the implementation
- A specific application may only be needed once
- Goals and tasks of scientific software for research are not always well defined
- Applications are often a complex composite of many units and it is difficult to test this anything but the complete application
- Applications use “inexact” floating-point math

Why Worry About This Now?

- Computers become more powerful all the time and more complex problems can be addressed
- Use of computational tools becomes common among non-developers and non-theorists
  -> many users could not implement the whole applications that they are using by themselves
- Current hardware trends (SIMD, NUMA, GPU) make writing efficient software complicated
- Solving complex problems requires combining expertise from multiple domains or disciplines

Ways to Move Forward

- Write more modular, more reusable software
  => build frameworks and libraries
- Write software that can be modified on an abstract level or where components can be combined without having to recompile
  => combine scripting with compiled code
- Write software where all components are continuously (re-)tested and (re-)validated
- Write software where consistent documentation is integral part of the development process

Unit- and Regression Testing are Opportunities

- Gain more confidence in your application
- Writing unit tests helps to understand problems => conceptual problems surface before use
- Writing unit tests encourages modular code => more code reuse, better maintainability
- Automated regression testing helps discovering inconsistencies before a software is released
- Adding tests is easy while implementing a feature; writing a test library after is daunting

What to Test

- Many fast tests for single feature/unit
  - Output does not have to be meaningful science; it is sufficient to get the correct “wrong” result
  - Tests should be as independent as possible
  - Tests need to cover all code paths
  - Also need to test for all cases that should raise warnings or error conditions
- Few specific reference runs of the whole code
- Test parallel applications in serial and parallel