Test suites in scientific research

Pietro Berkes

Motivation

- Every scientific result (especially if important) should be independently reproduced at least internally before publication (DFG, 1999)
- Not very realistic in our context, but the problem does exist as simulation results are very sensitive to bugs
- Errors in source code can be largely avoided with appropriate programming practices

Unit testing

- One of the 12 XP practices (Kent Beck, 1999)
- Tests become part of programming cycle and are **automated**
- Write testing suite (collection of testing functions) in parallel with your code; external software runs the tests and provides reports
- Currently, software libraries for automating testing exist for almost every programming language

Benefits

- Encourages better code and optimization (code can change and consistency is assured by tests)
- Faster development; sounds like a paradox, but consider:
  - bugs are always pinpointed
  - tests are simple and don't need to look nice
  - avoids starting all over again with debugging when you modify code
- Installation check for users if you plan to distribute the code

5 min. guide to unit testing

1. Write code in small, testable units; write the code in the most straightforward way
2. Write simple tests to check your code (see next slides)
3. Run tests and debug until all tests pass
4. Optimize only at this point!
5. Go back to 3 until necessary

What to test, how to test it

- Test with simple (but general) cases using hard coded solutions
- Test special or boundary cases
- Test general routines with specific ones

Reacting to new bugs

1. Isolate the bug using previous tests
2. Write the simplest possible test that reproduces the bug
3. Solve the bug

Additional resources

- List of unit testing frameworks on wikipedia:
  http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks
- Other “programming for science” resources:
  http://www.gatsby.ucl.ac.uk/~berkes/oss/resources.html