Introduction to Projects

Stefan Kluth
Max-Planck-Institut für Physik

Workshop on Advanced Programming Concepts
DESY, 8.10. - 13.10.12

Overview

- You all believe you can program (in C++)
  - Object-oriented: why and how
  - Lifetime of code and maintainance
  - Working in a team
  - “clean code”
- Majority of us are self-tought
  - Programming vs software engineering
  - This workshop tries to bridge the gap
  - Try out some of the ideas presented here

Structure

- Build teams of 6-8 people
  - In each team work in “promiscous” pairs
  - Teams sit together and collaborate
  - Discuss tasks, brake down work in small pieces, distribute work items in team
  - Here: repeat every hour
  - Teams guided by tutors
- Pair programming
  - “driver” and “navigator”
  - Change roles frequently

Structure

- Test-first programming
  - For a given “programming unit” write a unit test using a testing framework, test fails
  - Then write code to satisfy test
  - Repeat until done
- Unit test framework:
  - Organise and automate test running
  - We use boost_test for C++ (see project code)
- Input data
  - From other code
  - Or make it up to get going (“mock-up”)

Structure

- Code management with git (a la svn)
  - clone/pull https://github.com/skluth/<project.git>
  - Project version control in local repository
    - Git add, git commit in small increments
    - No interference with other developers
  - Project collaboration via central repository
    - Git pull, git push
    - Git push fails when local repo not in sync with central, need to pull and merge locally first
    - Git tag, git push --tags for tagging

.netrc

For “git push https:/github.com/... ” you need to authenticate

Create a $HOME/.netrc file (chmod 600):

```
machine github.com
login <your github user name>
password <your github password>
```

Not the most secure but works for us.
You didn't use your online banking pw for github

Structure

- “Continious integration”
  - Make target compiles and runs unit test executeables
  - Make fails when tests fail
  - Failing test only allowed in intermediate steps
- Only “push” working code
  - Tests could fail due to other package: pull changes, adapt your code (or fix other package)
  - Not formally enforced

Projects: INIParser

Read input data from simple config files (based on code “inih”)
Similar functionality to python ConfigParser

```
# comment
[section1]
Item1 = value
Name = Tom

[section2]
…
```

Code is complete and working with tests. Needed by other package RooAverageTools to parse inputs

github.com/skluth/INIParser.git

Projects: RooAverageTools

Calculation of error weighted average from several measurements. Consider statistical and systematic errors with correlations

See material on averaging

AverageDataParser.py: read input data, prepare for averaging
blue.py: matrix inversion solution
clsqAverage.py: solution using constrained fit
minuitAverage.py: solution using minuit fit

Provide C++/ROOT version of existing python code, use TVectorD and TMatrixD(Sym) for linear algebra

C++: github.com/skluth/RooAverageTools.git
Python: github.com/skluth/AverageTools.git

Projects: RooConstrainedFit

Implement linear least squares fit with constraints. Needed by RooAverageTools contrained fit averaging method

See material on averaging for details

Clsq.py: classes for handling of constraints and of fit

Provide C++/ROOT version of existing python code, use TVectorD and TMatrixD(Sym) for linear algebra

C++: github.com/skluth/RooConstrainedFit.git
Python: github.com/skluth/ConstrainedFit.git

Projects: RooUnfold

ROOT based implementation of several unfolding methods with a common interface.
Contains high level “acceptance tests” (a.k.a examples) but no unit tests

See previous Terascale workshops on statistics for unfolding

Implement unit tests using boost_test for class methods.
For code under test try some refactoring

github.com/skluth/RooUnfold.git

Pair programming

- Well established industry practice
- Advantages
  - Avoids obvious (”stupid”) mistakes
  - Cleaner more structured code with fewer errors
  - Pass knowledge between team members
  - Fewer distractions, higher attention level
- Disadvantages
  - 2 FTEs instead of one for the same task?
  - Both partners at similar level
  - Distributed teams?