A Brief Introduction to Unit Testing

Marc Paterno
Scientific Computing Division/Fermilab
Revision 1

Contents

Please feel free to interrupt with questions or comments of interest to you.

1 Purpose of testing

Why do we test? That is, what are the *end goals* of testing? There are many:

- Helping assure our scientific results are correct.
- Helping assure our papers are correct.
- Justification of physics conclusions.
- Verification of physics algorithms.
- Helping to improve measurements.
- Selection of optimal algorithms: most precise measurements, highest efficiencies, fastest calculations.

All these are related to *doing physics*. We can group them all into one category: *physics validation*. Nobody seriously considers neglecting physics validation.

We do testing to help get the best physics results, and so that we can believe our physics results.

2 Types of testing

There are many ways to categorize tests. I want to distinguish two different types of testing:

1. *Physics validation*. This type of testing answers the question: “Is my algorithm the right one for the physics task at hand?”
2. *Coding verification*, in the form of *unit testing*. This type of testing answers the question: “Does my software encode the algorithm I intended for it to encode?”

The value of unit testing is to make sure we don’t have logic errors in the code; it frees us to spend our intellectual effort in making sure our algorithms are the right algorithms (physics validation).

Unit testing can help find (and so fix) logic errors more quickly than looking at histograms or summary statistics.

3 Constructing code for (unit) testability

Unit testing should be taken into account when you *start* coding, not when you are "done"[1] with coding. Adding testing as an afterthought is much more difficult than starting with testing in mind. So, what needs to be tested?

- Some code is so trivial that it can be seen to be correct "by inspection". There is rarely a need to test an accessor for some member datum.
- Some code does not need to be unit tested because it has already been tested by others. You don't need to test algorithms of the C++ Standard Library, nor do you need to test functions of the **art** framework.[2]
- Non-trivial code needs to be tested. Deciding what is trivial is a matter of taste, to be guided by experience.

Small, focused functions help to reduce the amount of unit testing needed, because they increase the amount of code that can be verified by inspection.

[1]You are not really done until you've verified your code is what you need. So you're not done coding until you've done validation.

[2]As of version 1.10.00b, **art** contains 240 test programs. We test it so you don't have to.

4 An example

The example code we will inspect is from the 2014 C++ course offered recently at Fermilab, taught by Prof. Micheal Herndon. The code is available at GitHub, at `https://github.com/herndon/FNALComp`, on the branch called `day1a`; this code was not directly part of the course. You can get the code using `git`:

```
git clone https://github.com/herndon/FNALComp
cd FNALComp
git checkout day1a
```

The handout contains the code from:

- `Modules/include/HitRecoModule.hh`
- `Modules/src/HitRecoModule.cc`
- `Modules/include/HitAccum.hh`
- `Modules/src/HitAccum.hh`
- `Modules/include/Strip.hh`
- `Modules/include/Layer.hh`