Testing

UCB-THW
Rachel Slaybaugh
March 11, 2015

How do you (or anyone else for that matter)

know

that your code **works**?

Motivation

- Verification: Have we built the software correctly (i.e. does it match the specification)?

- Validation: Have we built the right software (i.e. is this what the customer wants)?

- Two different but important goals
- Use different kinds of tests to answer

Testing Levels

- Unit: verify the functionality of a specific section of code
- Integration: test interfaces between units
- System: verify completely integrated system
- System integration (if applicable): works properly with 3rd party systems
- Regression: ensure that new code changes don’t break anything

Test Driven Development

Write all the tests

before you write the code.

Why?

Error Checking

- One way to force the code to “test as you go” is to write checks into the code itself
- E.g. make sure the right sizes and data types are being used everywhere
- Example: https://github.com/rachelslaybaugh/JellyBeanCode

Exceptions

- Separate the “normal” flow and the “exceptional” cases
- TRY to execute a block of code
- CATCH any errors that are THROWn
- Handle different kinds of errors
- Limit error checking redundancy
- Add meaning and readability to errors

Unit Tests

- Nosetests: http://nose.readthedocs.org/en/latest/testing.html
- Example: https://github.com/pyne/pyne/blob/develop/tests/test_data.py
  is testing
  https://github.com/pyne/pyne/blob/develop/pyne/data.pyx

Regression Tests

- An extra awesome way to do this is with Continuous Integration
- Variation of running all unit tests before code is integrated and possibly every night
- Example: BatLab and PyNE, https://github.com/pyne/pyne/pulls

Summary

- Testing can help you catch errors right away
- Tests give you a way to track all the parts of your code
- You can add checking and testing as you go
- Tests help with debugging and maintaining code
- Tests save you time in the long run