Testing Your Code

22 Feb 2019
15-441 Recitation
Ray

You cannot do well on the projects in this class if you do not **test your code well**.

For P2, your implementation should be able to correctly handle a large file transfer between a sender and receiver.

For P2, your implementation should be able to correctly handle a large file transfer between a sender and receiver. **What is correct?**

We learned in class how TCP does connection setup and teardown, flow control, and congestion control.

Tests should define what is the correct behavior of your code/implementation.

The projects in this class are hard but testing your code does not have to be!

Tip #1 - Write tests that define expected behavior and to help you find bugs.

Tip #2 - Automate your tests whenever possible to make them easy to run and re-run as you change your code.

Tip #3 - Tests should test one thing and one thing well.

Tip #4 - Parametrize your tests to reuse test code for different inputs, and expected outputs. Example: test_GET(get_example1, get_expected1), test_GET(get_example2, get_expected2)

Tip #5 - Document what your tests are suppose to do. Verify.

In groups of 2 or 3 discuss (10min):
Brainstorm tests you should write to test your P2 implementation.

Example functionality you should test: connection setup, connection tear down, flow control, congestion control.

Now let's discuss as a group.
Brainstorm tests you should write to test your P2 implementation.

Example functionality you should test: connection setup, connection tear down, flow control, congestion control.

Let’s look at the starter code together:

P2 starter code had a few simple starter tests.
What does each test do?

For P2, `make test` must run your tests.

You need more than just the starter tests.

You can continue to use pytest or whatever your favorite testing tool is.

You should also make use of standard C debugging tools gdb and Valgrind.

Ask the course staff for help
if you're having trouble testing your code.