1 Reading 6: HFSD Ch 7. Testing, pp. 235-251

There are three ways to look at your system

- Your users see the system from the outside: your system is a black box to them, it either does what they asked it to do, or it doesn't. Functionality!
- Testers peek under the covers a little: they're poking underneath to make sure things are really happening the way you said they would. Your system is more of a grey box to them. Testers might look at data in your database to make sure things are being cleaned up correctly. They might be checking that ports are closed, network connections are dropped, etc.
- Developers let it all hang out: they see good and bad class design, patterns, duplicated code, inconsistencies in how everything is represented.

1.1 Black Box Testing Focusing on Input and Output

- Functionality: does the system do what the user story says it's supposed to do.
- User input validation: Feed system values that don't make sense.
- Output results: Hand-check numerical values that your system returns. Make sure all functional paths have been tested.
- State transitions: some systems need to move from one state to another according to very specific rules.
- Boundary cases, off-by-one errors: test your system with value that's a little too small or just outside the maximum allowable value.

1.2 Grey Box Testing Gets You Closer to Code

- Verifying auditing and logging: log viewing tool or auditing report, or query some datbase tables directly.
- Data destined for other systems: check output format and data you're sending.
- System-added information: Make sure system-generated information (e.g., timestamps) are created correctly.
- Scraps left laying around: make sure data is being deleted if it's supposed to be. No memory leaks either.

1.3 White-Box Testing Uses Inside Knowledge

- Testing all different banches of code: you should look at all code.
- Proper error handling: if you do feed invalid data into a method, are you getting the right error back?
- Working as documented
- Proper handling of resource constraints: what happens if your app needs something it can't get?

1.4 Coming Up With Tests is Your Job

- Frameworks run your test, not write them.
- You still need to know what you want to test.
- Then, your testing framework choice is going to impact how you test.
- Hanging your tests on a framework
  - Automate tests so we can be more effective

1.5 Testing Everything with One Step

1. Build a suite of tests.
2. Run all your tests with one command: the easier it is to run your tests, the better.
3. Get regression tests for free: as you add more functionality, your existing tests should still work. If they don't, your software has regressed!

Tailor your test suites to suit the occasion. So, break out fast and slow tests so developers can run all the fast tests often while they are chaning and adding tests, but only run the full suite when they think they need to.

1.6 Automate Tests w/ Testing Framework

- JUnit can be used to test your code.
- Make sure to import it.
- `@Before`: called before each test.
- `@After`: called after each test.