

The Zen of Unit Testing

L02, Aug 28 2008

Wisdom o' the day

Program testing can be used to show
the presence of bugs, but never to
show their absence!

-- E.W. Dijkstra

The zen of unit testing

- Old world view:
 - Build full program
 - Test it

The zen of unit testing

- New world view
 - Build single “unit” (e.g., class)
 - Test that in isolation

Being agile

- Even newer
 - Figure out what unit you're going to build
 - Build test cases for that
 - Then build the unit itself
 - Use tests to measure progress on building the unit

The test-first procedure

```
procedure testFirstImplement(Interface I) {  
    // Stage 1  
    understand specification of I  
    // Stage 2  
    for each (method m in I) {  
        write test cases, t(m), for m {  
            check common use cases for m  
            check edge cases for m  
            check corner cases for m  
            check exception cases for m  
        }  
    }  
    // Stage 3  
    make skeleton class C implements I  
    while (t(m) fails for any m in I) {  
        add code for methods to C  
        fix bugs in C  
    }  
}
```

Unit Testing FAQ

- Q: What methods should be tested?
- A: Any “non-trivial” methods
 - Maybe even test the trivial ones
- Test to make sure that object *obeys the specification*
 - => You have to *understand* the spec...

Unit Testing FAQ

- Q: What makes a good test case?
- A: Your goal is to show that method works under as many conditions as you possibly can
 - Common cases -- ways you expect it to be used
 - Uncommon cases -- things that you don't expect, but are legal
 - Edge cases -- edge of what's legal
 - Corner cases -- multiple edges
 - Illegal cases! Make sure it fails the right way.
 - Bugs

Unit Testing Case

- Q: What do you mean test bugs?
- A: Every time you find bug in running code:
 - Replicate it -- figure out how to make it happen consistently
 - Turn it into a test case -- repeatably test to demonstrate its presence/absence
 - Localize it -- figure out where it is in your code
 - Fix it
 - Run tests to demonstrate that it's gone
 - Tests *also* show that you haven't broken other stuff...

JUnit 3

- A java library for automating tests
- Simply makes testing more convenient
- Interesting design... we will get to that later
- Create test using class: `TestCase`
 - `setUp()` : init any commonly used variables
 - `runTest()` : run “the” test
 - `tearDown()` : cleanup any member variables

JUnit

- You have a class: `MyClass` to test...
- Subclass from `TestCase`: `MyClassTest` extends `TestCase`
- Setup any member variables, override: `void setUp()`
- Handle removal of variables, override: `void tearDown()`
- Write test functions: `boolean testMyFunction()`
- Invoke a test: `new MyClassTest("testMyFunction")`

More JUnit

- You will probably have lots of tests in MyClassTest
- You can invoke multiple tests using a Test Suite
- Example:
 - `TestSuite suite = new TestSuite(MyClassTest.class);`
 - `TestResult result = suite.run();`
- This automatically finds all the test[funcname] functions.
- You can add multiple [classname]Test classes to a Suite

Yet More JUnit

- JUnit provides a user interface for tests: `TestRunner`
- `java junit.swingui.TestRunner MyClassTest`
- If you organize Suites into a class, you can run them
- `java junit.swingui.TestRunner MyTestSuite`
- ... see documentation for suite class definition

Unit Testing FAQ

- Q: When should I run my tests?
- A: As often as possible
 - Every time you make (even a minor) change to your class
 - Between every change (test one change at a time)
 - Every time you add a new test case
 - Every time you discover/fix a bug

Testing is Good!

- Marks your progress as a developer
- Greatly improves stability of software
- Often provides important examples of usage!
- Improves coding efficiency