

Agenda

- Iteration #1/#2
- Iterative Automated Testing
- FIT Revisited
- JUnit
- GERT
- Tutorial time

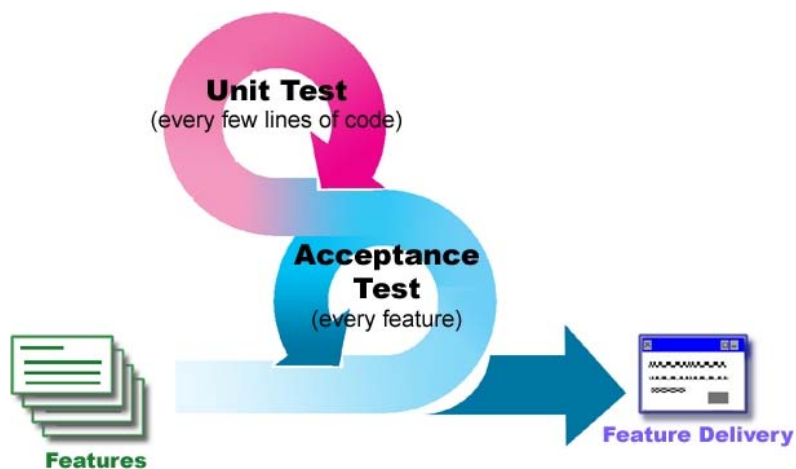
Iteration #1/#2

- **Iteration #1 user stories**
 - Add/edit requirements, defects, test cases
- **Iteration #1 deadline: Monday, 9/13**
 - Turn in everything you have for feedback, no grade
- **Iteration #2 deadline: Monday, 9/27**
 - All iteration #1 user stories
 - FIT Table for all “non-display” requirements
 - JUnit for 80% statement of non-GUI code
 - Black box test plan
 - 20% of grade (Iteration #1 and Iteration #2 grade)

Skills vs. tools

- **Skills:**
 - **Black box testing**
 - Equivalence class partitioning, boundary value analysis, diabolical, acceptance tests
 - Brought to life: FIT Tool → Automation
 - **White box testing**
 - Equivalence class partitioning, boundary value analysis, diabolical, coverage, basis set, cyclomatic complexity
 - Brought to life: JUnit Tool → Automation

Test Cases



FIT

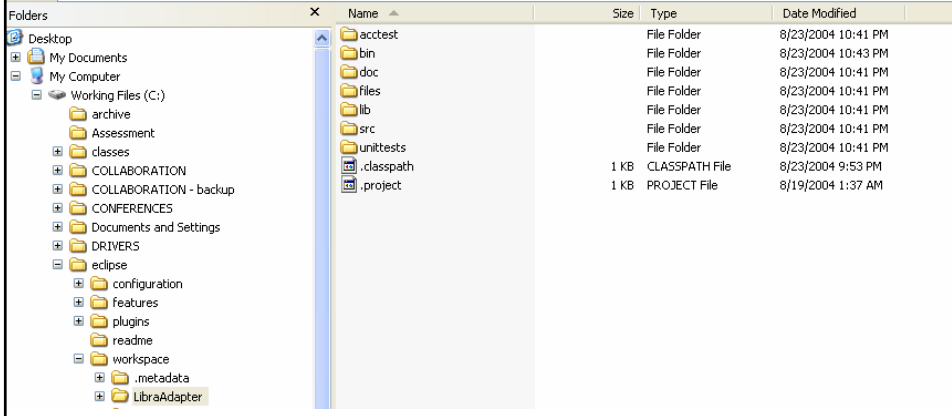
- **Action Fixture**
 - Transaction oriented
 - Through a series of events, something happens, want to make sure whatever happens is correct
- **Column Fixture**
 - Computation oriented
 - A computation is performed and want to make sure the outcome is correct given as many different inputs as possible

You do the FIT Tables

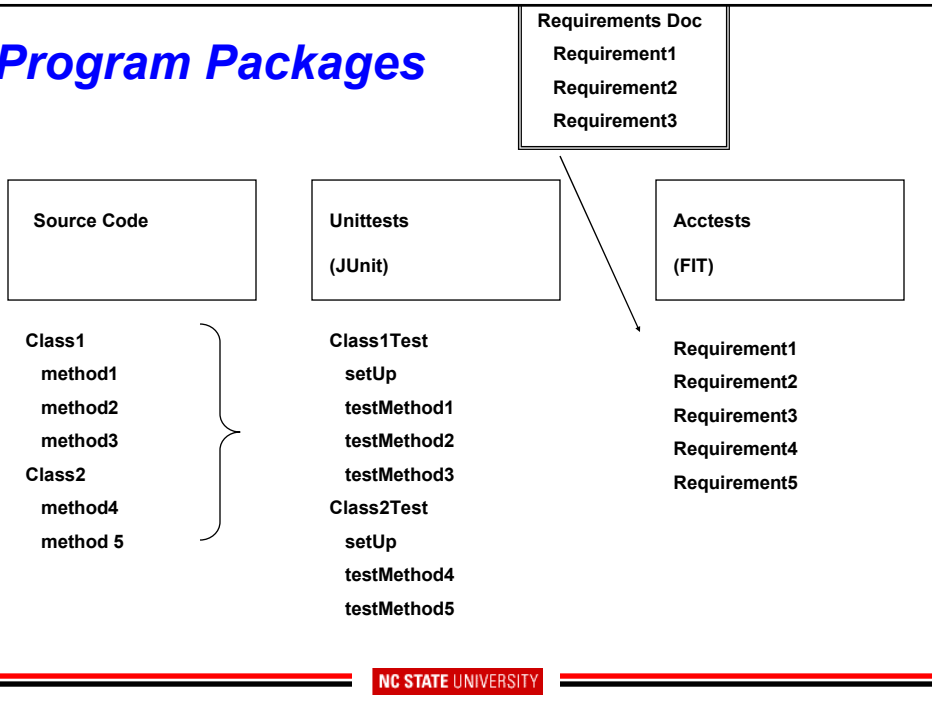
- **Userid must be of the format ID@someplace.something**
- **Passwords must be of 6-12 characters long**
- **New user obtains a password by supplying a good userid and a good password. If successful, the new user is added to the database.**

JUnit

- Similar to FIT → you add JUnit package to your class path and extend JUnit class (TestCase)



Program Packages



TestCase

- Any class that contains test methods should subclass the TestCase class.
- testXxx() methods are defined in this class.
- When you want to check the expected and actual test results, you invoke assert() and pass a boolean expression which will indicate if the test case is passed or not.
- When a test class contains multiple testXxx() methods, you can use the setup() and tearDown() methods to initialize and release any common objects under test.

Writing a Test Case

1. Define a subclass of TestCase
2. Override the setUp() method to initialize object(s) under test (optional)
3. Override the tearDown() method to release any permanent resources you allocated in setUp() (optional)
4. Define one or more testXxx() methods that exercise the objects under test

setUp and tearDown

- Run before (setUp) and after (tearDown) every test.
- Tests do not run in any order
- Asserts
 - assertEquals(a, b, delta)
 - assertTrue(a)
 - assertFalse(a)
- fail() test whether the program throws an exception to verify that the tests execution path ends up in the exception handler.

```
public void testIndexOutOfBoundsException() {  
    Vector v= new Vector(10)  
    try {  
        Object o= v.elementAt(v.size());  
        fail("Should raise an ArrayIndexOutOfBoundsException");  
    } catch (ArrayIndexOutOfBoundsException e) {  
    }  
}
```

GERT

- **Good Enough Reliability Tool**
 - We'll understand why it is "good enough" later in the course
 - For now, we'll use it for computing coverage
- Don't forget Project→Clean

To Do

- **Tutorials:**
 - FIT
 - JUnit
 - GERT
- **Do the tutorials on your choice**
 - Files suggested in tutorial
 - Monopoly
 - Your own program
- **Ask for help when you need it**