Testing with Mock Objects

CMSC 433
Mock Objects

- Easy to test a method that doesn’t call any other methods, just returns a value
  - easy to test binary search, add, etc.
- Can be harder to test code that interacts with a substantial chunk of code and devices/services
  - code may not be written yet
  - devices/services may be slow, unavailable or unreliable
  - hard to test device/service failures
Test JUnit creation

• In VisualAge (a Java IDE)
• Want to check that if you attempt to create a JUnit test case with the classname that is already in use
• a ClassExistsException is thrown
public void testCreationWithExistingClass() {
    myMockPackage.addContainedType(
            new MockType(EXISTING_CLASS_NAME));
    myMockWorkspace.addPackage(mockPackage);
    JUnitCreatorModel creatorModel =
            new JunitCreatorModel(myMockWorkspace, PACKAGE_NAME);
    try {
        creatorModel.createTestCase(EXISTING_CLASS_NAME);
        fail("Should generate an exception for existing type");
    } catch (ClassExistsException ex) {
        assertEquals(EXISTING_CLASS_NAME, ex.getClassName());
    }
    myMockWorkspace.verify();
}
Advantages of Mock testing

- Deferring infrastructure choices
- Coping with scale
- No stone unturned
Better tests

• Failures fail fast

• You can run the test against the real database, and then after the test, check that the database contains the correct values

• But that doesn’t tell you which call to do database corrupted the result

• Dangers here too...
Better tests...

- Refactored assertions

- You can put a lot of consistency checking logic into the mock, and it will get automatically reused whenever
mock = createMock(Collaborator.class);
classUnderTest = new ClassUnderTest();
classUnderTest.addListener(mock);// expect document addition
mock.documentAdded("Document");
// expect to be asked to vote, and vote for it
expect(mock.voteForRemoval("Document")).andReturn((byte) 42);
// expect document removal
mock.documentRemoved("Document");

replay(mock);
classUnderTest.addDocument("Document", new byte[0]);
assertTrue(classUnderTest.removeDocument("Document"));
verify(mock);
Dangers of Mocking

• Don’t Mock Me: Design Considerations for Mock Objects, Jeff Langr, Langr Software Solutions

• The result of this rampant mocking was an extremely inflexible system.

• The mock-based tests depended upon the specifics of the implementation.

• Even small changes to the implementation adversely impacted dozens of tests.

• A simple refactoring took hours instead of minutes.