

Software Testing, Quality Assurance & Maintenance—Lecture 26

Patrick Lam

March 10, 2017

Last Time

Practical techniques for writing tests.

Result verification:

- state verification;
- behaviour verification.

Also, techniques for improving your tests.

- reducing duplication
- simplifying tests

Today: More Test Design

- Mock objects;
- Flaky tests;
- Continuous integration;

Mock Objects



John Tenniel's original (1865) illustration for Lewis Carroll's "Alice in Wonderland". Alice sitting between Gryphon and Mock turtle.

Test Doubles

Meszaros proposes four kinds of test doubles:

- dummy objects (don't do anything);
- fake objects (do something, but no good in prod,
e.g. in-memory database);
- stubs (canned answers)
- spies (stubs/proxies that record interactions);
- mocks (objects with expectations)

Reference:

martinfowler.com/articles/mocksArentStubs.html

Mail Service Stub

```
public class MailServiceStub implements MailService {  
    private List<Message> messages =  
        new ArrayList<Message>();  
    public void send (Message msg) {  
        messages.add(msg);  
    }  
    public int numberSent() {  
        return messages.size();  
    }  
}
```

good for state verification:

```
assertEquals(1, mailer.numberSent());
```

Using Mocks

```
class OrderInteractionTester...

    public void testOrderSendsMailIfUnfilled() {
        Order order = new Order(TALISKER, 51);
        Mock warehouse = mock(Warehouse.class);
        Mock mailer = mock(MailService.class);
        order.setMailer((MailService) mailer.proxy());

        mailer.expects(once()).method("send");
        warehouse.expects(once()).method("hasInventory")
            .withAnyArguments()
            .will(returnValue(false));

        order.fill((Warehouse) warehouse.proxy());
    }
}
```

Creating Mock Objects with EasyMock¹

```
@RunWith(EasyMockRunner.class)
public class ExampleTest {

    @TestSubject
    private ClassUnderTest classUnderTest = new
        ClassUnderTest();

    @Mock // creates a mock object
    private Collaborator mock;

    @Test
    public void testRemoveNonExistingDocument() {
        replay(mock);
        classUnderTest.removeDocument("Does not exist");
    }
}
```

¹<http://easymock.org/user-guide.html>

Expecting behaviour: method calls

```
@Test
public void testAddDocument() {
    // recording phase:
    mock.documentAdded("New Document");
    replay(mock);
    // replaying phase; we expect the recorded actions
    to happen
    classUnderTest.addDocument("New Document",
                                new byte[0]);
    // check that the behaviour actually happened:
    verify(mock);
}
```

Expecting behaviour: return values

```
@Test
public void testVoteForRemoval() {
    // expect document addition
    mock.documentAdded("Document");
    // expect to be asked to vote for document removal,
    // 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);
}
```

Flakiness: Good for croissants², bad for tests



²thanks Pixabay

Reference

Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, Darko Marinov. “An Empirical Analysis of Flaky Tests”. In FSE '14.

What Are Flaky Tests?

Flaky test = sometimes fails (nondeterministically).

Dealing with Flaky Tests

- Label as flaky.
- Re-run tests, see if it ever passes.
- Ignore/remove flaky tests.

What causes flakiness?

Result of studying 201 fixes:

- improper wait for async responses;
- concurrency;
- test order dependency;
- etc.

Async Wait

Problem:

do something, then sleep for not-long enough.

Solution:

use a `wait` call to wait until the thing happens.

Concurrency

Usual concurrency problems:

- data races;
- atomicity violations;
- deadlocks.

May be in test or the system under test.

Test Order Dependency

Problem: test X expects test Y to have completed.
Solution: remove dependency.

Continuous Integration

Literally:

use a single shared master branch

Why This Is Continuous Integration

Integration = merge one's changes
back into master.

Continuous = do it all the time.

Why CI Is Awesome

Software stays in working state.

Developers don't integrate for months-on-end.

Things that go with CI

Continuous Builds

Test Automation

Continuous Deployment (optional)

What Happens In CI

- ① You clone the repo (which works).
- ② You make your changes.
- ③ You commit and push your changes (often!)
- ④ A machine pulls the changes, compiles them, and runs automated tests.
- ⑤ Everyone knows whether your changes passed tests or not.

Continuous Deployment

Minor variation to CI:

production machine also pulls changes as soon as tests pass.

Key Details

- Fix broken builds immediately!
- Keep the build fast (minutes):
parallelize it & tier your tests.
- Test in a prod-like environment.

Continuous Integration References

Bullet points from Gitlab:

about.gitlab.com/2015/02/03/

[7-reasons-why-you-should-be-using-ci/](#)

Mid-length article from Atlassian:

www.atlassian.com/agile/continuous-integration

Longer article by Martin Fowler:

martinfowler.com/articles/continuousIntegration.html

Serverless CI:

medium.com/@hichaelmart/lambci-4c3e29d6599b

Summary

More practical techniques for writing tests:

How to actually do behaviour verification (mock objects).

Pitfalls of bad test writing (flaky tests).

Making sure your code's always good (CI).