Lesson 9 & 10 Answers

Acceptance Tests vs Unit Tests

1 - Reflection questions

1.1 - Boy Scout Rule

In this lesson we applied the Boy Scout Rule to our code. What does this mean?

Boy scouts have a rule that says: Always leave the campground cleaner than you found it. If you find a mess on the ground, you clean it up regardless of who might have made the mess.

We tidied up the bug report scenario so that it became better documentation. This makes it easier for anyone to understand the current behaviour of our system when they come back to change it in the future.

1.2 - Who writes the tests?

We made a decision to write a unit test for the credit deduction. This led us to modify the design of the Network and Person classes.

If you are a tester, this would be hard to do without a developer to help you.

If you are a developer working on a team where most tests are UI tests written by testers, you might not be used to writing unit tests.

What changes would you need to make on your team to be able to work this way?

This depends on your context of course, but if you work in an environment where developers don't do much testing and testers don't do much coding, a good place to start is to work in pairs.

This way testers can transfer testing skills to developers, and developers can transfer programming skills to testers. Everyone wins.

1.3 - Purpose of acceptance tests

What is the purpose of acceptance tests? Who gets the most value from them?

Acceptance tests indicate what parts of a system are not working correctly, at a very high level. When they fail, they tell you what functionality a user will not be able to enjoy because of the bug. Acceptance tests can be useful for non-technical people such as product owners and business analysts to know how much is left to do.

1.4 - Purpose of unit tests

What is the purpose of unit tests? Who gets the most value from them?

Unit tests pinpoint precisely where and why something is not working. This is primarily useful to the people who will fix the problem: Developers.

1.5 - Mock Objects

We showed you that mock objects, when used in TDD, can give you design feedback about the communication protocols between objects. Why is it good design practice to keep these protocols simple?

When those protocols are simple, it means the smallest parts of your system are well decoupled and encapsulated. This has many benefits. Cleaning up your code (refactoring) becomes a lot easier, because a modification is less likely to affect other parts of the system.

Simple protocols also makes unit testing a lot easier, because it is easier to plug in a mock object without having to do a lot of configuration of that mock.

1.6 - Mock Objects - Abuse

How are mock objects abused? What steps can you take to avoid that?

Mock Objects is an extension to Test-Driven Development that supports good Object-Oriented design by guiding the discovery of a coherent system of types within a code base. It turns out to be less interesting as a technique for isolating tests from third-party libraries than is widely thought.[1]

When mock objects are used to retro-fit tests to an existing code base, you often end up with a very verbose and chatty interaction between the class being tested and the mock object(s).

You can avoid this by writing unit tests and mocks before you implement the code.

1.7 - Breaking tests on purpose

When we retrofitted the tests for Person's premium account responsibilities, they passed immediately. Why on earth did we then go and temporarily break them?

Never trust a test that you haven't seen fail! Whenever you add a new test or modify an existing one and it passes right away, get into the habit of breaking it temporarily. This gives you confidence that if the code breaks in the future, your test will give you a detailed error message telling you exactly what's wrong. Tests that never fail or give unhelpful error messages when they do fail aren't very useful!

[1] http://www.jmock.org/oopsla2004.pdf