cucumber school

Lesson 11 & 12

Web Automation

1 - Adding more adapters

Suppose there is a new requirement to develop a native mobile application that communicates with the shouty server over a RESTFUL (HTTP/JSON) interface. What modifications would you make to the Shouty codebase and tests?

We wouldn't have to change a thing about our core domain. All we would have to do is to modify our existing HTTP adapter (`ShoutyServlet.java` or `web_app.rb`) to support JSON. You could write unit tests for that, and you could complement those by reusing the existing Cucumber Scenarios by adding a `RESTShoutSupport` or `RestWorld` adapter.

2 - What's the difference between impact and risk?

When we evaluated what scenarios to include in the top part of the test pyramid, we evaluated impact and risk. What is the difference between those terms? Why did we decide to only include high risk and high impact scenarios?

Business Impact is the likely cost (in money or reputation) to the business if a bug leaked out. For example, if you run an online store, then a test that ensures someone can check out and pay has high impact. By comparison, a test for the validation on a preferences form is probably not so high impact.

Technical Risk on the other hand assesses the likelihood that something might go wrong. Behaviour that's implemented in a library that hasn't been touched in years is very low risk, whereas behaviour that sits in the gnarly 1500-line method that everyone touches every day is at a very high risk of breaking.

When you can't test everything, focus your testing effort where technical risk *and* business impact are high.

© Cucumber Limited

3 - Test Depth

We talked about test depth in this episode. How would you define test depth?

Test depth refers to the number of layers of code being exercised by the test. A shallow test would only exercise a single method. A deeper test might test one method, but when it exercises that method, it causes the method to create other objects, and invoke methods on those objects too. A really deep test would exercise every layer of your application, from the user interface right through to the database.

4 - Cost of tests

The benefits of tests come at a cost. What are typical costs of a test?

Tests carry a few costs.

First you have the cost of writing the test. If you're not very practiced at this there may be some cost of learning how to write the test, then the cost of the implementation itself.

Once the test is implemented, it may need maintenance in the future. It may start to fail for unwanted reasons, and will need to be changed. This is a cost.

You'll also have the cost of waiting for the test. If you have thousands of tests, each individual one needs to be quick, otherwise you're waiting a long time for the feedback you care about.

Fixing a failing test also has a cost. If the test tells you exactly where the bug is, the cost is lower. If you have to spend a lot of time hunting for the source of the bug, that carries an extra cost.

5 - How can you speed up your UI tests?

When we ran our scenarios through the UI using Selenium it made our test suite slow. What could you do to speed it up?

Often, the best way to speed up a UI test is to remove it, or rather, refactor it to a domain layer test that doesn't go through the UI, but talks directly to the domain layer.

If you're unable to do that, there are still things you can do to speed up UI tests. Start by identifying what steps and hooks are slow.

© Cucumber Limited

Are you starting your web app in a Before hook for every scenario? Try starting your app only once, for all scenarios.

Are you launching a new browser for every scenario? Try reusing the browser instance.

Do your scenarios need to go through a lot of pages before you get to the UI you really want to test? Try to go directly to the page you need to test, and set up the data you need “underneath” the UI.

Many of these optimisations introduce a new problem - you might have state leaking between scenarios. When state leaks between scenarios they might pass when run in a certain order, but they might fail when run in a different order. This can make it very difficult and time consuming to fix failing scenarios. To avoid this we recommend you use Before hooks to explicitly clear state that might have been left over from the previous scenario.

© Cucumber Limited