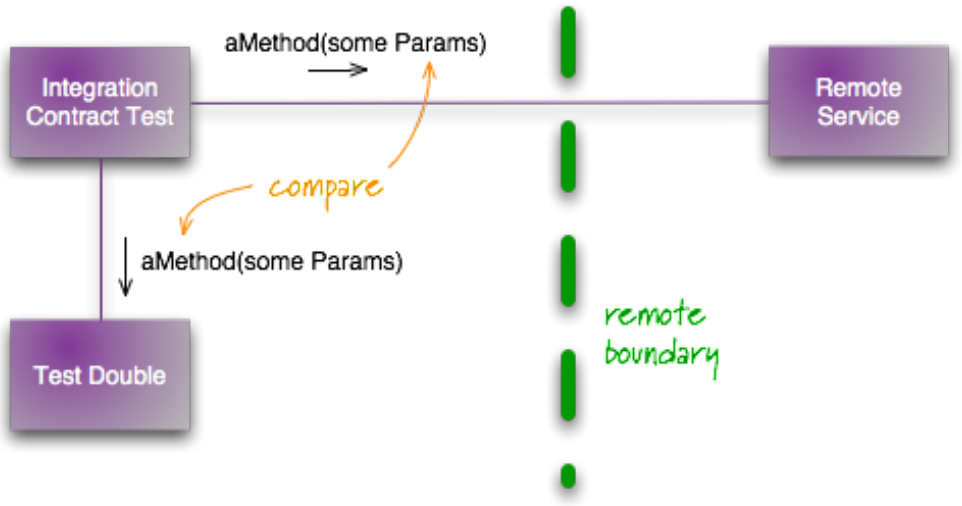
ContractTest

12 January 2011

Martin Fowler

TEST CATEGORIES

One of the most common cases of using a TestDouble is when you are communicating with an external service. Typically such services are being maintained by a different team, they may be subject to slow, and unreliable networks, and maybe unreliable themselves. That's why a test double is handy, it stops your own tests from being slow and unreliable. But testing against a double always raises the question of whether the double is indeed an accurate representation of the external service, and what happens if the external service changes its contract?



A good way to deal with this is to continue to run your own tests against the double, but in addition to periodically run a separate set of contract tests. These check that all the calls against your test doubles return the same results as a call to the external service would. A failure in any of these contract tests implies you need to update your test doubles, and probably your code to take into account the service contract change.

These tests need not be run as part of your regular deployment pipeline. Your regular pipeline is based on the rhythm of changes to your code, but these tests need to be based on the rhythm of changes to the external service. Often running just once a day is plenty.

A failure in a contract test shouldn't necessarily break the build in the same way that a normal test failure would. It should, however, trigger a task to get things consistent again. This may involve updating the tests and code to bring them back into consistency with the external service. Just as likely it will trigger a conversation with the keepers of the external service to talk about the change and alert them to how their changes are affecting other applications.

This communication with the external service supplier is even more important if this service is being used as part of a production application. In these cases a contract change may break a production application, triggering an emergency fix and an urgent conversation with the supplier team.

To reduce the chances of unexpected breaks in contracts, it's useful to move to a Consumer Driven Contracts approach. You can facilitate this by letting the supplier team have copies of your contract tests so they can run them as part of their build pipeline.

When testing an external service like this, it's usually best to do so against a test instance of the external service. Occasionally you'll have no choice but to hit the production instance, at that point you'll need to talk to the suppliers to let them know what's happening and be extra careful with what the tests do.

Contract tests check the contract of external service calls, but not necessarily the exact data. Often a stub will snapshot a response as at a particular date, since the format of the data matters rather than the actual data. In this case the contract test needs to check that the format is the same, even if the actual data has changed.

One of the best way to build these test doubles is to use a SelfInitializingFake.

Revisions

2018-01-01: Originally this bliki entry was entitled Integration Contract Test. Since it was written the term "contract test" has become widely used for these, so I changed the bliki entry.