Integration Test

16 January 2018

Martin Fowler

TEST CATEGORIES

Integration tests determine if independently developed units of software work correctly when they are connected to each other. The term has become blurred even by the diffuse standards of the software industry, so I've been wary of using it in my writing. In particular, many people assume integration tests are necessarily broad in scope, while they can be more effectively done with a narrower scope.

As often with these things, it's best to start with a bit of history. When I first learned about integration testing, it was in the 1980's and the waterfall was the dominant influence of software development thinking. In a larger project, we would have a design phase that would specify the interface and behavior of the various modules in the system. Modules would then be assigned to developers to program. It was not unusual for one programmer to be responsible for a single module, but this would be big enough that it could take months to build it. All this work was done in isolation, and when the programmer believed it was finished they would hand it over to QA for testing.

The first part of testing would be unit testing, which would test that module on its own, against the specification that had been done in the design phase. Once that was complete, we then move to integration testing, where the various modules are combined together, either into the entire system, or into significant sub-systems.

The point of integration testing, as the name suggests, is to test **whether many separately developed modules work together as expected**. It was performed by activating many modules and running higher level tests against all of them to ensure they operated together. These modules could parts of a single executable, or separate.

Looking at it from a more 2010s perspective, these conflated two different things:

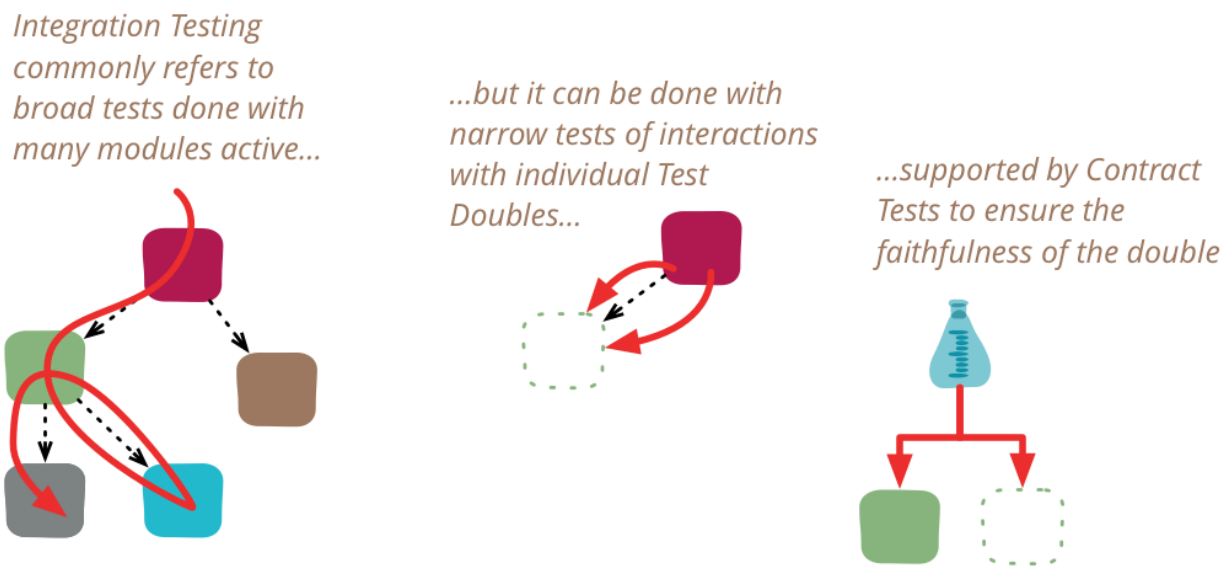
- testing that separately developed modules worked together properly
- test that a system of multiple modules worked as expected.

These two things were easy to conflate, after all how else would you test the shopping cart and catalog modules without activating them both into a single environment and running tests that exercised both modules?

The 2010s perspective offers another alternative, one that was rarely considered in the 1980s. In the alternative, we test the integration of the shopping cart and catalog modules by exercising the portion of the code in shopping cart that interacts with catalog, executing it against a TestDouble of catalog. Providing the test double is a faithful double of catalog, we can then test all the interaction behavior of catalog without activating a full catalog instance. This may not be a big deal if they are separate modules of a monolithic application, but is a big deal if catalog is a separate service, which requires its own build tools, environments, and network connections. For services, such tests may run against an in-process test double, or against an over-the-wire double, using something like mountebank

An obvious catch with integration testing against a double is whether that double is truly faithful. But we can test that separately using ContractTests.

Using this combination of using narrow integration tests and contract tests, I can be confident of integrating against an external service without ever running tests against a real instance of that service, which greatly eases my build process. Teams that do this, may still do some form of end-to-end system test with all real services, but if so it's only a final smoke test with a very limited range of paths tested. It also helps to have a mature QA in Production capability, and if that is mature enough, there may be no end-to-end system testing done at all.



The problem is that we have (at least) two different notions of what constitutes an integration test.

narrow integration tests

- exercise only that portion of the code in my service that talks to a separate service
- uses test doubles of those services, either in process or remote
- thus consist of many narrowly scoped tests, often no larger in scope than a unit test (and usually run with the same test framework that's used for unit tests)

broad integration tests

- require live versions of all services, requiring substantial test environment and network access
- exercise code paths through all services, not just code responsible for interactions

And there is a large population of software developers for whom “integration test” only means “broad integration tests”, leading to plenty of confusion when they run into people who use the narrow approach.

If your only integration tests are broad ones, you should consider exploring the narrow style, as it's likely to significantly improve your testing speed, ease of use, and resiliency. Since narrow integration tests are limited in scope, they often run very fast, so can run in early stages of a DeploymentPipeline, providing faster feedback should they go red.

As if this terminological confusion isn't enough, things have worsened in the late 2010s with yet another usage of “integration test”. This comes from a divergence in the

meaning of Unit Test. Some people define unit test to be what I refer to as a solitary unit test, one where all program elements other than the one under test are replaced with test doubles. Given this narrow definition some writers defining “integration test” to mean sociable unit tests.

All this is why I’m wary with “integration test”. When I read it, I look for more context so I know which kind the author really means. If I talk about broad integration tests, I prefer to use “system test” or “end-to-end test”. I don’t have any better name for narrow integration tests, so I do use that (but with “narrow” to help signal to the reader the nature of these tests). I continue to use “unit test” for both kinds, using solitary/sociable when I need to make a distinction.

Acknowledgements

Birgitta Böckeler, Brian Oxley, Dave Rice, Deepti Mittal, Jonny Leroy, Kief Morris, Raimund Klein, Rogerio Chaves, and Tiago Griffo discussed drafts of this post on our internal mailing list.

Revisions

Updated on June 3 2021 to move use of “integration test” to mean sociable unit tests from footnote into main text