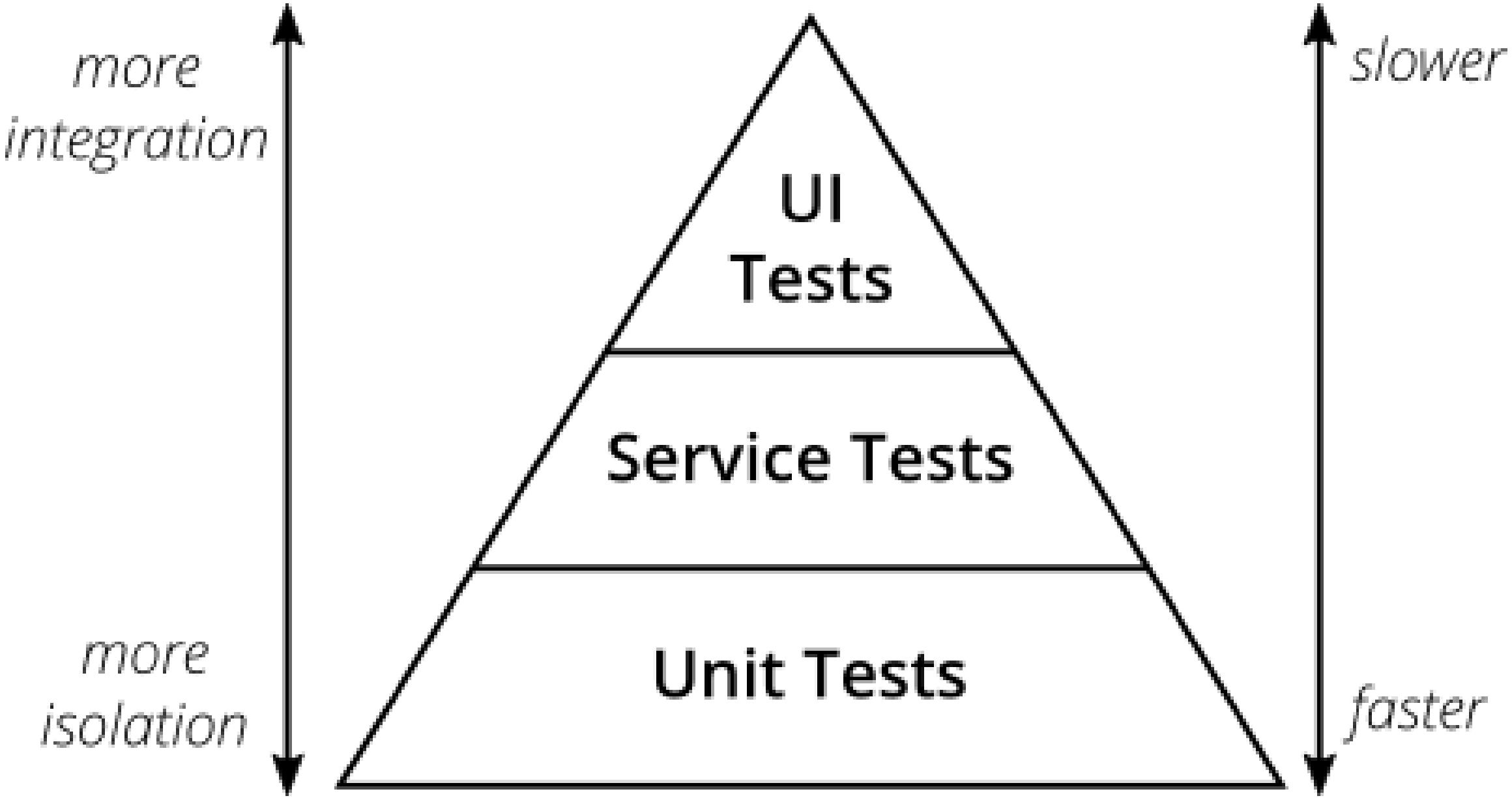
Contract Driven APIs

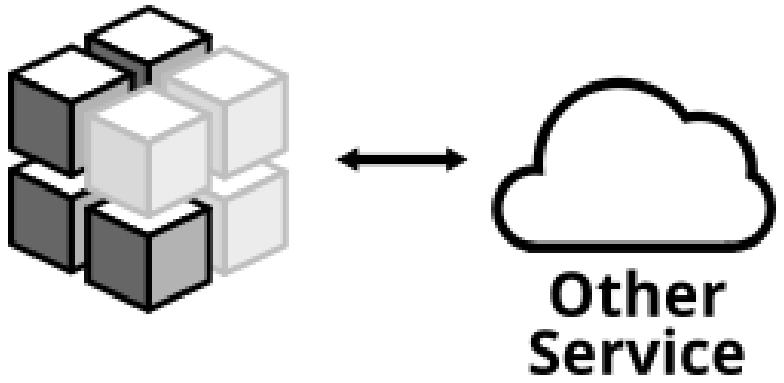
An Introduction by Kevin Denver

June 2022


more integration
more isolation
UI Tests
Service Tests
Unit Tests
slower
faster

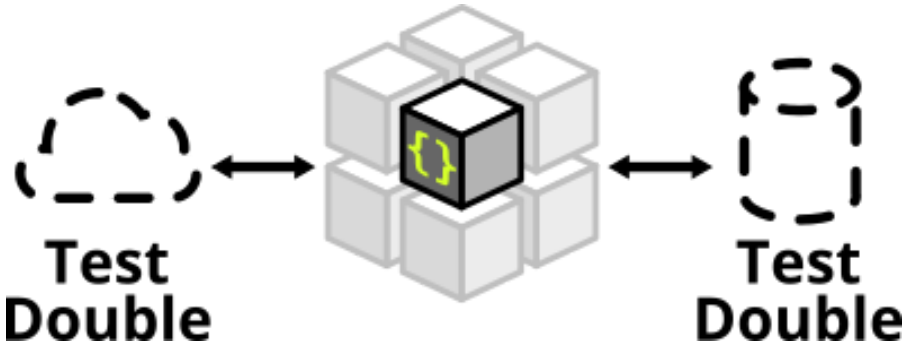
Disadvantages of End-To-End Test

- Slow
- Brittle / Flaky
- Introduces coupling between teams
- Increases blockers and dependencies
- You can't start testing until the dependency has been fulfilled
- Re-creating a test environment for a real-world scenario might not be possible



Test Doubles

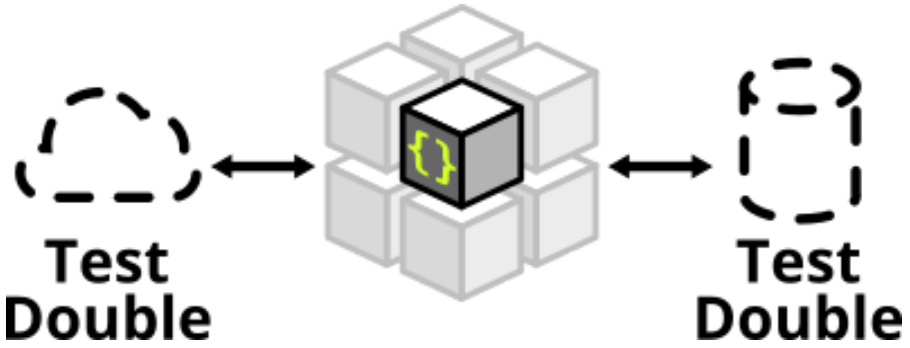
- **Mocks** and **Stubs** are two different kinds of Test Doubles.
- You can use test doubles to replace objects you'd use in production with an implementation that helps you with testing.
- **Stubs** provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test.
- **Mocks** are pre-programmed with expectations which form a specification of the calls they are expected to receive. They can throw an exception if they receive a call they don't expect and are checked during verification to ensure they got all the calls they were expecting.



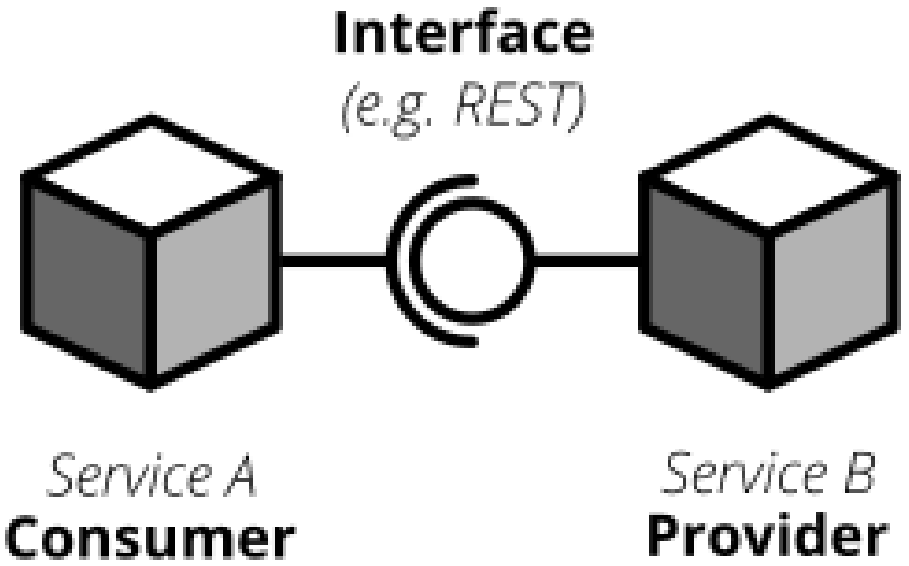
https://martinfowler.com/bliki/TestDouble.html

Disadvantages of Test Doubles

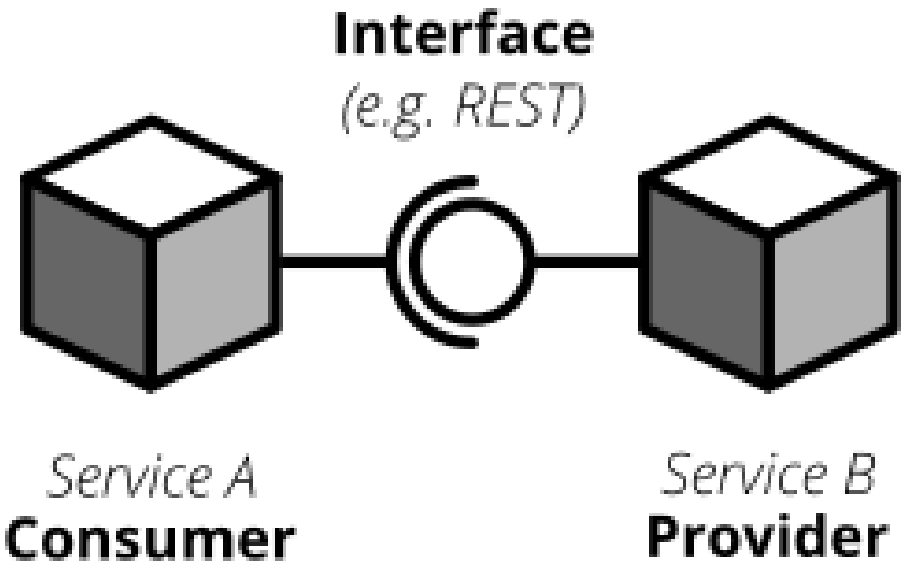
- No way of knowing whether it'll work in production
- No way of knowing when the dependency changes



As you often spread the consuming and providing services across different teams you find yourself in the situation where you have to clearly specify the interface between these services (the so called **contract**).

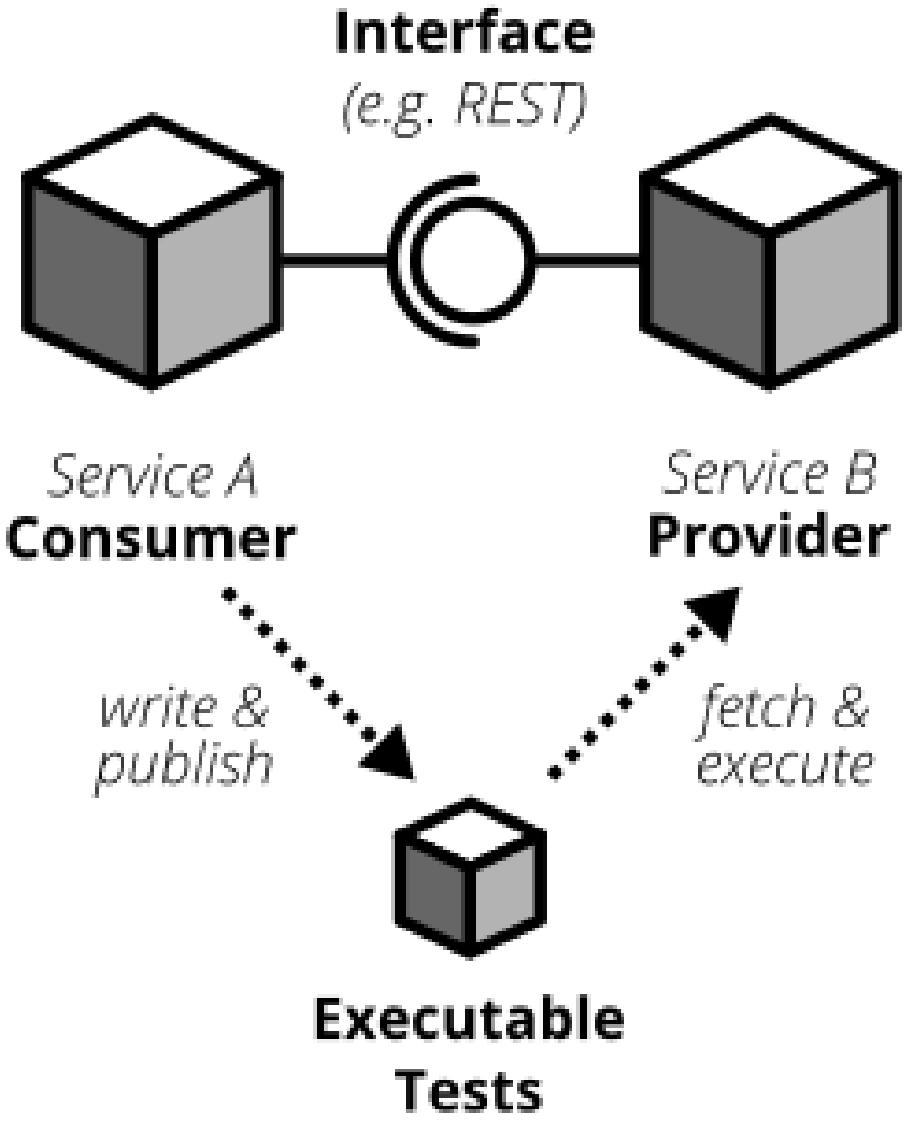


- Write a long and detailed interface specification (**the contract**)
- Implement the providing service according to the defined contract
- Throw the interface specification over the fence to the consuming team
- Wait until they implement their part of consuming the interface
- Run some large-scale manual system test to see if everything works
- Hope that both teams stick to the interface definition forever and don't screw up
- OpenAPI or API Blueprint



https://martinfowler.com/articles/practical-test-pyramid.html

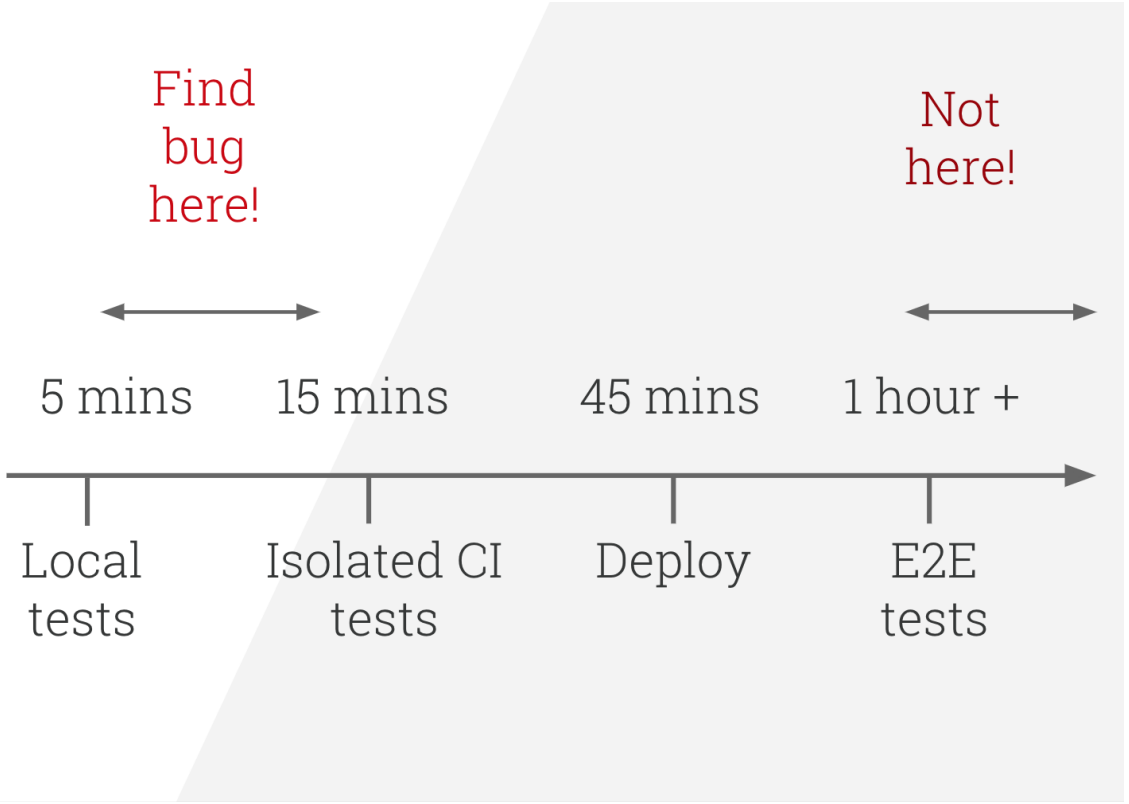
Consumer-Driven Contract tests (CDC tests) let the **consumers drive the implementation of a contract**. Using CDC, consumers of an interface write tests that check the interface for all data they need from that interface. The consuming team then **publishes** these tests so that the publishing team can fetch and execute these tests easily. The providing team can now develop their API by running the CDC tests. Once all tests pass they know they have implemented everything the consuming team needs.



https://martinfowler.com/articles/practical-test-pyramid.html

Benefits

- The ability to develop the consumer before the API
- The ability to drive out the requirements for your provider first, meaning you implement exactly and only what you need in the provider.
- The ability to immediately see which consumers will be broken if a change is made to the provider API.



https://pactflow.io/what-are-the-benefits-of-contract-testing/

Tooling

- https://pact.io/
- https://github.com/stoplightio/prism
- https://github.com/apiaryio/dredd/

References

- Consumer-Driven Contracts: A Service Evolution Pattern
- The Practical Test Pyramid
- OpenAPI Specification
- API Blueprint
- A Comprehensive Guide to Contract Testing APIs in a Service Oriented Architecture