Contract testing
in theory and practice

Seb Rose
Some background
Test automation pyramid

How much of the application the test exercises

Number of tests

http://claysnow.co.uk/architectural-alignment-and-test-induced-design-damage-fallacy/

@sebrose

http://cucumber.io
Dependencies

@sebrose

http://cucumber.io
Dependencies replaced
What could possibly go wrong?
Contracts
Explicit contract
Implicit contract
interface ICalculate
{
    public int Add(
        int a,
        int b
    )
;
}

@sebros
http://cucumber.io
Design by contract

Contract
• an agreement between client and supplier

Characteristics
• expect some benefits
• incur some obligations
The need for contract testing
Collaboration tests

Test → A → Test double

test double
Collaboration tests - passes

Test → A → Test double

Test → A → Cloud
Collaboration tests - fails

@sebrose

http://cucumber.io
Collaboration tests - oops
Contract test

Test

Test double
Contract test through an interface
“Don’t mock what you don’t own”

Joe Walnes
Systematic contract testing

• Collaboration tests make assumptions about the contract

• Contract tests try to justify those assumptions

JB Rainsberger, via GOOS mailing list, “Unit-test mock/stub assumptions rots”
15 March 2012

@sebrosel http://cucumber.io
Example from first principles
Our system

A -> Interface

Test double

@sebros

http://cucumber.io
public interface IRevenueProvider {
    decimal GetRevenue(int customerId);
}
What’s our implicit contract?

• Pass in a valid customer ID, return a decimal value
  • Can we say anything about the magnitude of the returned value?

• Pass in an invalid customer ID…. then what?
Documenting our contract

[TestFixture]
public class RevenueProviderContract
{
    [Test]
    public revenue_provider_returns_revenue_for_valid_customer_id()
    {
    
    [Test]
    public revenue_provider_throws_exception_for_invalid_customer_id()
    {
    
    
}
Implementing the contract

[TestFixture]
public abstract class RevenueProviderContract
{
    private IRevenueProvider revenueProvider;

    protected abstract IRevenueProvider GetRevenueProvider();
}
protected abstract
    IRevenueProvider GetRevenueProvider();

[SetUp]
public void setup()
{
    revenueProvider = GetRevenueProvider();
}

[Test]
// Tests omitted for brevity
}
public class ProductionContract : RevenueProviderContract {

    public ProductionContract() {}

    protected override IRevenueProvider GetRevenueProvider() {
        return new ProductionRevenueProvider();
    }
}

@sebrose
public class TestDoubleContract : RevenueProviderContract
{
    public TestDoubleContract() {} 

    protected override IRevenueProvider GetRevenueProvider()
    {
        return new TestRevenueProvider();
    }
}
[Test]
public void valid_customer_id()
{
    revenueProvider.GetRevenue(VALID_ID);
}

[Test]
public void invalid_customer_id()
{
    Assert.Throws<CustomerIdException>((() =>
        revenueProvider.GetRevenue(INVALID_ID));
}
Communication, collaboration, and consumer driven contracts
Provider contract

- Closed and complete: Provider contracts express a service's business function capabilities in terms of the complete set of exportable elements available to consumers, and as such are closed and complete with respect to the functionality available to the system.

- Singular and authoritative: Provider contracts are singular and authoritative in their expression of the business functionality available to the system.

- Bounded stability and immutability: A provider contract is stable and immutable for a bounded period and/or locale. Provider contracts typically use some form of versioning to differentiate differently bounded instances of the contract.

https://martinfowler.com/articles/consumerDrivenContracts.html
Consumer driven contracts (CDC)

- **Open and incomplete** Consumer contracts are open and incomplete with respect to the business functionality available to the system. They express a subset of the system's business function capabilities in terms of the consumer's expectations of the provider contract.

- **Multiple and non-authoritative** Consumer contracts are multiple in proportion to the number of consumers of a service, and each is non-authoritative with regard to the total set of contractual obligations placed on the provider. Consumers may evolve at different rates.

- **Bounded stability and immutability** Like provider contracts, consumer contracts are valid for a particular period of time and/or location.

https://martinfowler.com/articles/consumerDrivenContracts.html
How do you agree a contract?
How do you agree a contract?
How do you agree a contract?

Provider team

Consumer team

@sebrose

http://cucumber.io
How do you agree a contract?

Provider team

Consumer team

@sebrose

http://cucumber.io
How do you agree a contract?

Provider team

Consumer teams

@sebrose

http://cucumber.io
CI/CD and microservices
Continuous integration

Dev → CI → EnvA → EnvB → EnvC → Prod
Continuous integration - single consumer

Dev  CI  Contract  EnvA  EnvB  Prod

P    P    P    P    P    P

C    C    C    C    C    C

http://cucumber.io
Simplifying micro-service testing

Pact provides a mechanism for creating a contract between a service consumer and a service provider, and then providing the tools to validate that the consumer and provider adhere to the contact independently of each other.

@sebrose

http://cucumber.io
DEMO

https://github.com/tdshipley/pact-workshop-dotnet-core-v1

@sebrose

http://cucumber.io
Key points

• Consumer creates contracts using Pact DSL
• Pact creates mock HTTP server
• Running the tests creates a Pact file
• Provider uses Pact file to verify compatibility
• Provider may offer “backdoor” interface
Pact broker
Continuous integration - micro services

Dev | CI | Contract | EnvA | EnvB | Prod

S → S → S → S → S

@sebrose

http://cucumber.io
Consumer CI → publish → Pact Broker ← fetch → Provider CI

@sebrose

https://www.youtube.com/watch?v=79GKBYSqMlo

http://cucumber.io
create
publish contract

webhook
task contract

webhook
fetch results

deploy

Provider CI

Consumer CI

webhook
publish results

verify

deploy

http://cucumber.io

https://www.youtube.com/watch?v=79GKBYSqMlo
If you can’t deploy services independently, you don’t have microservices.

You have a distributed monolith

Beth Skurrie
<table>
<thead>
<tr>
<th>Consumer version</th>
<th>Provider version</th>
<th>Verification result</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>31</td>
<td>success</td>
</tr>
<tr>
<td>12</td>
<td>31</td>
<td>failure</td>
</tr>
<tr>
<td>12</td>
<td>32</td>
<td>success</td>
</tr>
<tr>
<td>13</td>
<td>32</td>
<td>success</td>
</tr>
</tbody>
</table>

https://www.youtube.com/watch?v=79GKBYSqMIo  http://cucumber.io
can-i-deploy
Key points

- Pacts are published by Consumer
- Pacts are fetched by Provider
- Results are stored in the “Matrix”
- “Matrix” supports independent deployment
Contract testing

- Increases confidence
- Reduces need for integration tests
- Speeds up development

- No substitute for communication
- Does not replace all other testing
Seb Rose

Twitter:  @sebrose
Blog:  http://cucumber.io/blog
E-mail:  seb@cucumber.io