hybris software
An SAP Company

Integration Testing from the Trenches

Joker<?>

Nicolas Fränkel

October 2014

Me, myself and I

Developer & Architect as consultant

Wide range of businesses & customers

Teacher & Trainer

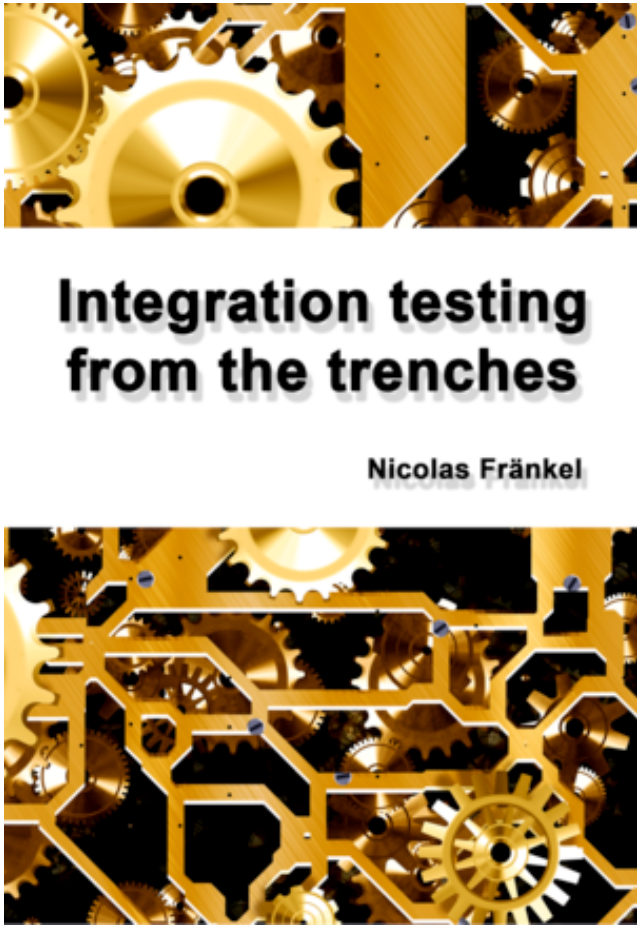
Speaker

Blogger

http://blog.frankel.ch/

(http://morevaadin.com/)

Also an author



https://leanpub.com/integrationtest

Plan

Integration Testing

What is that?

Challenges

Solution hints

Testing with resource dependencies

Database

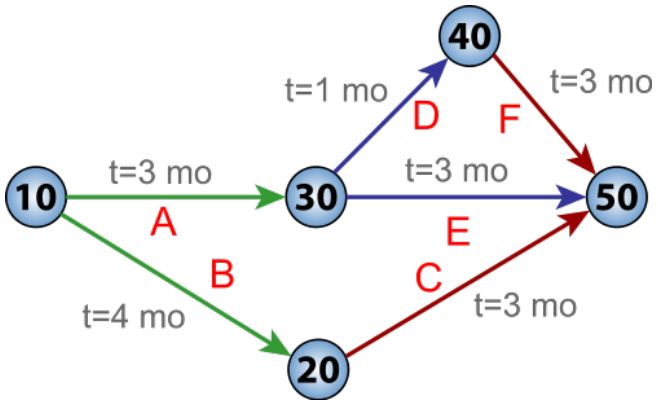
Web Services

Testing In-container

Spring & Spring MVC

JavaEE

hybris



hybris software
An SAP Company

Definitions

There are many different kinds of testing

Unit Testing

Mutation Testing

Integration Testing

GUI Testing

Performance Testing

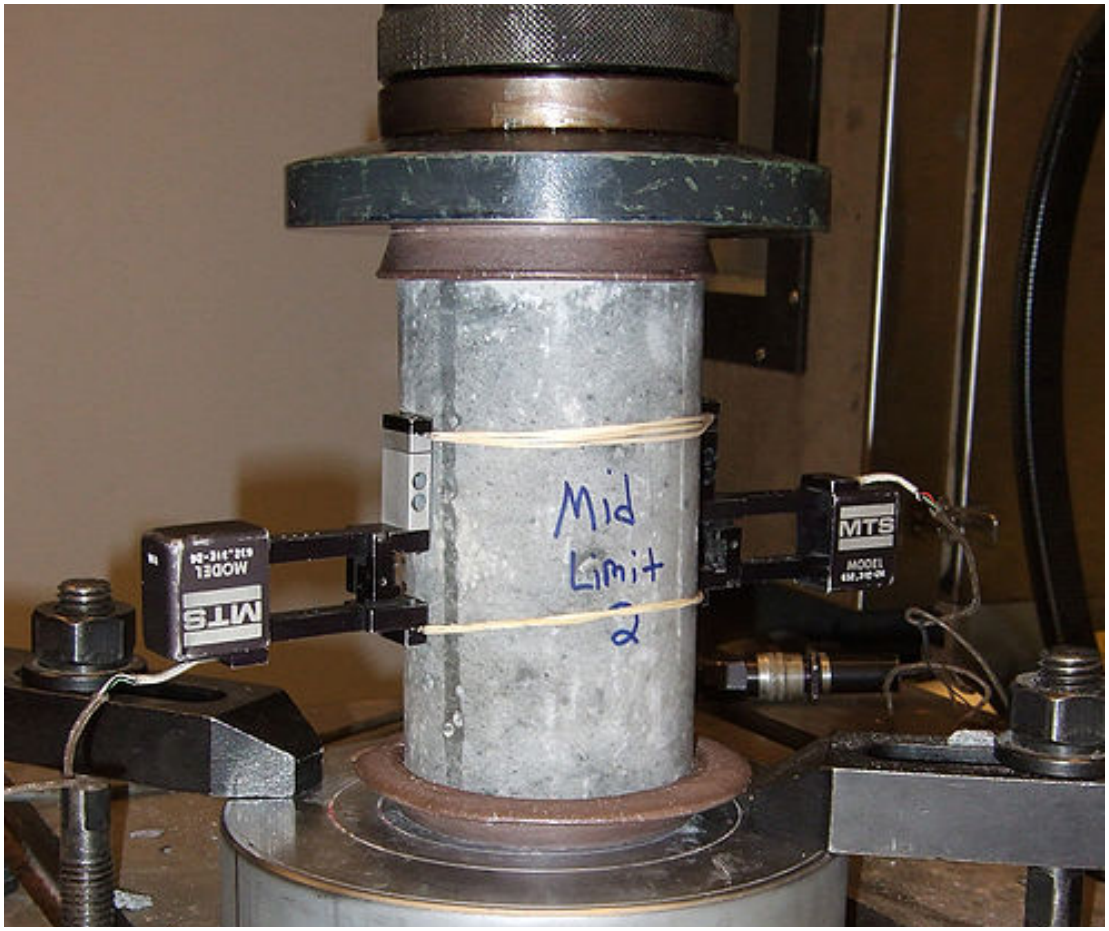
Load Testing

Stress Testing

Endurance Testing

Security Testing

etc.



Unit Testing vs. Integration Testing

Unit Testing

Testing a unit (i.e. a class) in isolation

Integration Testing

Testing the collaboration of multiple units

"Savate fouetté figure 1" by Daniel - Photo Daniel.

A concrete example

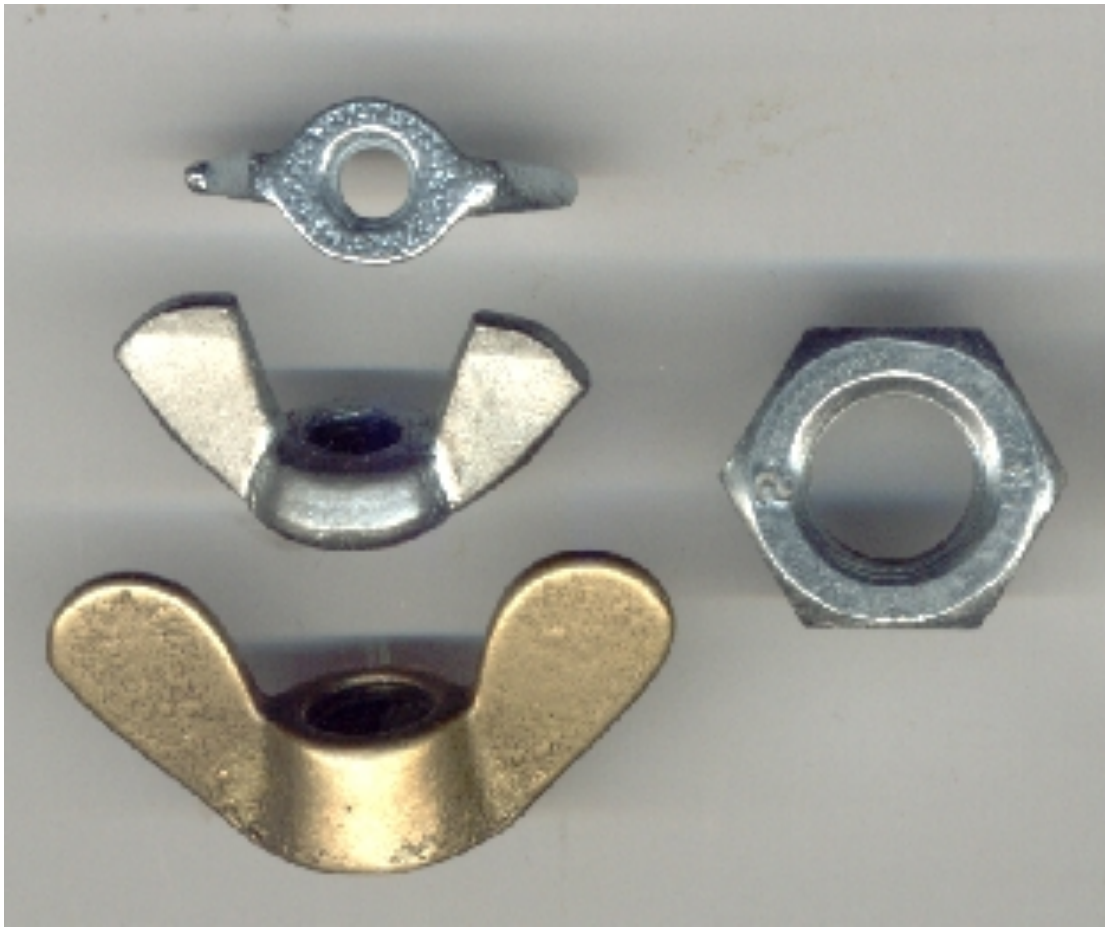
Let’s take an example

A prototype car

"2011 Nissan Leaf WAS 2011 1040" by Mariordo Mario Roberto Duran Ortiz - Own work

Unit Testing

Akin to testing each nut and bolt separately



Integration Testing

Akin to going on a test drive

"URE05e" by Marvin Raaijmakers - Own work.

Unit Testing + Integration Testing

Approaches are not exclusive but complementary

Would you take a prototype car on test drive without having tested only nuts and bolts?

Would you manufacture a car from a prototype having only tested nuts and bolts but without having tested it on numerous test drives?

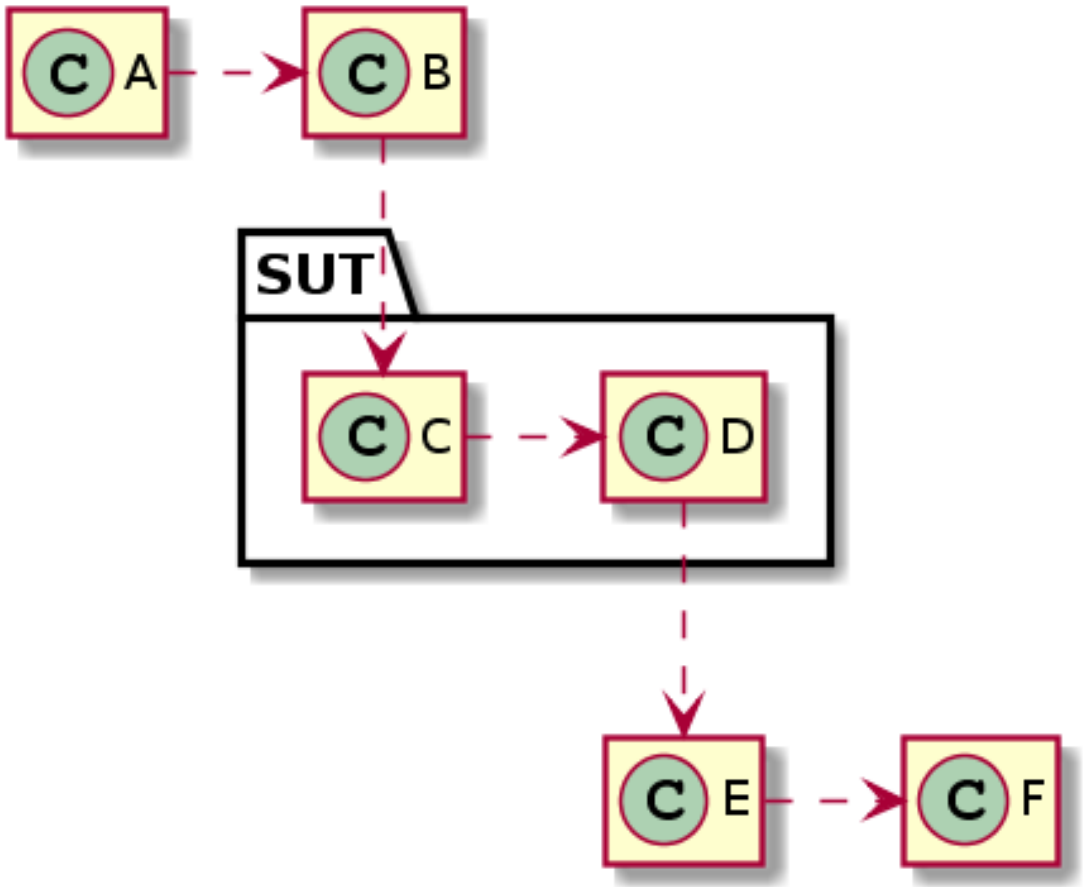
System Under Test

The SUT is what get tested

Techniques from Unit Testing can be re-used

Dependency Injection

Test doubles



Testing is about ROI

The larger the SUT

The more fragile the test

The less maintainable the test

The less the ROI

Thus, tests have to be organized in a pyramidal way

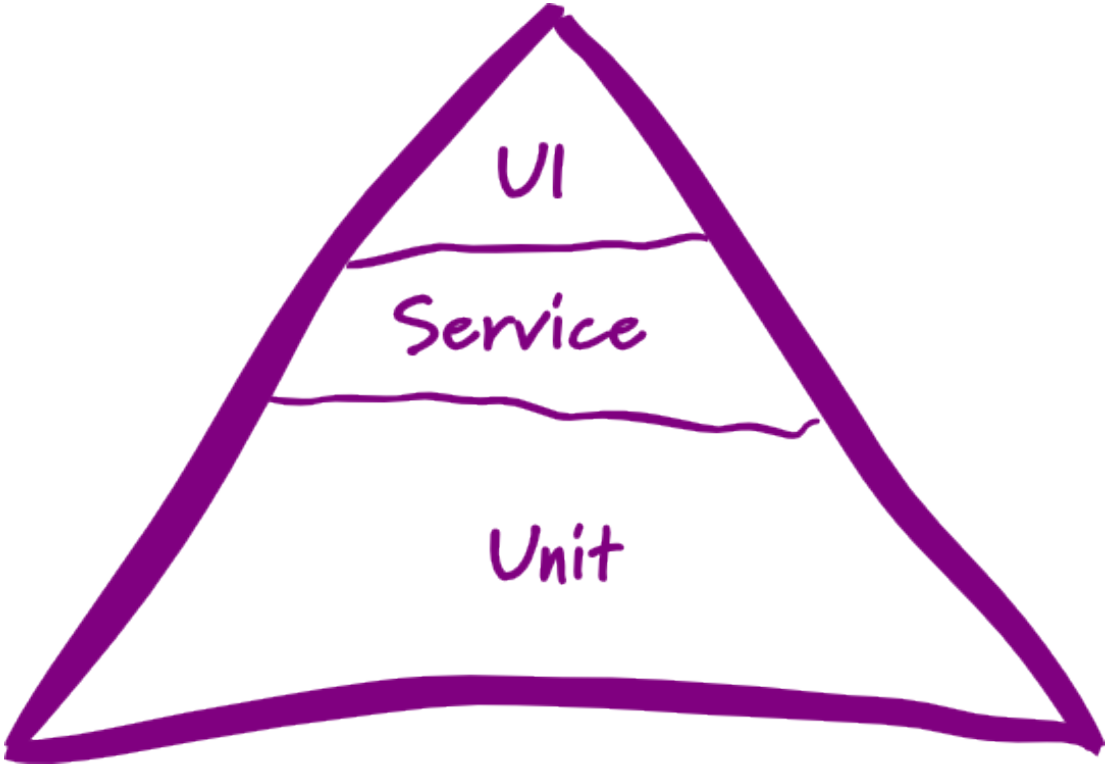
The bigger the SUT

The less the number of tests

Integration Testing

Test standard cases

Generally not error cases



http://martinfowler.com/bliki/TestPyramid.html

Integration Testing Challenges

Brittle

Dependent on external resources

- Database(s)
- etc.

Slow

Dependent on external resources

Hard to diagnose

How to cope

Separate Integration Tests from Unit Tests

Fake required infrastructure resources

Test in-container

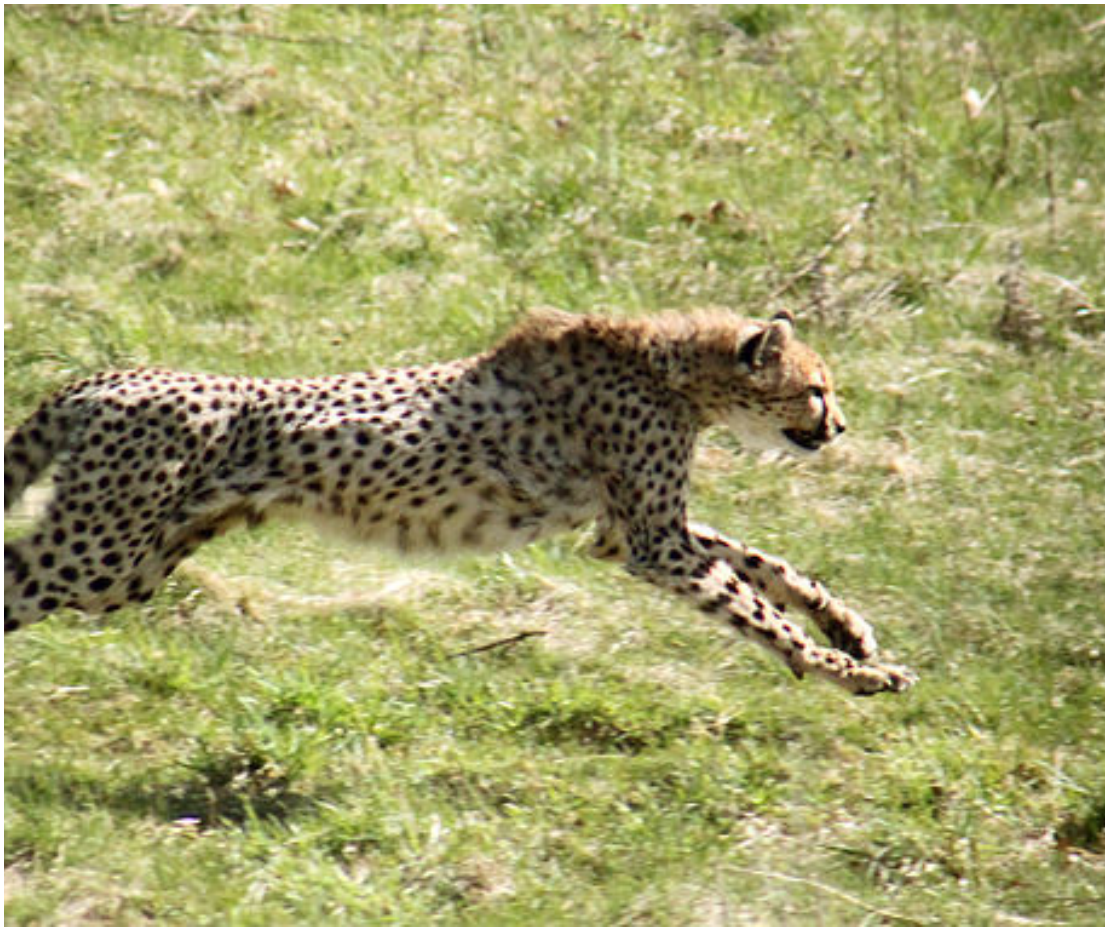
But IT are still slow?!

Separating UT & IT doesn’t make IT run faster

But you can uncover errors from UT faster

Fail Fast

It will speed testing



"Gepardjagt2 (Acinonyx jubatus)" by Malene Thyssen - Own work.

Integration Testing and build

Available tools

Ant

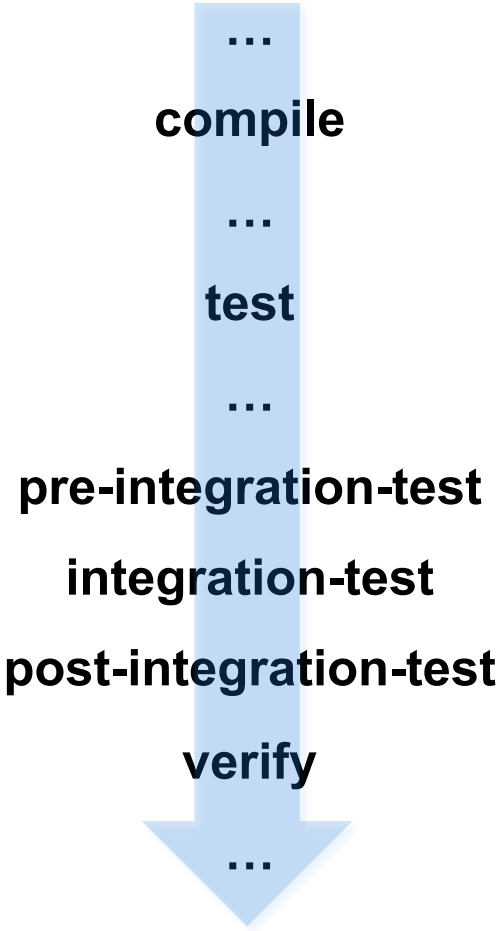
Maven

Gradle

etc.

New Development Recently Finished on Bristol's City Centre by Brizzleboy

Maven lifecycle



Reminder on Surefire

Bound to the `test` phase

Runs by default

*Test

Test*

*TestCase

Failsafe

“Copy” of Surefire

Different defaults

*IT

IT*

*ITCase

One goal per lifecycle phase

pre-integration-test

integration-test

post-integration-test

verify

Must be bound explicitly

Binding Failsafe - sample

```
<plugin>
  <artifactId>maven-failsafe-plugin</artifactId>
  <version>2.17</version>
  <executions>
    <execution>
      <id>integration-test</id>
      <goals>
        <goal>integration-test</goal>
      </goals>
      <phase>integration-test</phase>
    </execution>
    <execution>
      <id>verify</id>
      <goals>
        <goal>verify</goal>
      </goals>
      <phase>verify</phase>
    </execution>
  </executions>
</plugin>
```

Continuous Integration

Needs a build configured

Suggestions

Unit Tests run at each commit

Integration Tests run “regularly”

- Daily
- Hourly
- Depending on the context

hybris software
An SAP Company

Infrastructure resources

Infrastructure dependencies

Database

Filesystem

Time

Message Oriented Middleware

Mail server

FTP server

etc.

Mocks and infrastructure dependencies

To test your Service

Mock your DAO/repository

- Mockito

To test your DAO/repository

Mock your database???

Simple database use-case

Oracle database

Use an in-memory datasource and hope for the best

Use Oracle Express and hope for the best

Use a dedicated remote schema for each developer

- And your DBAs will hate you

Reducing database gap risk

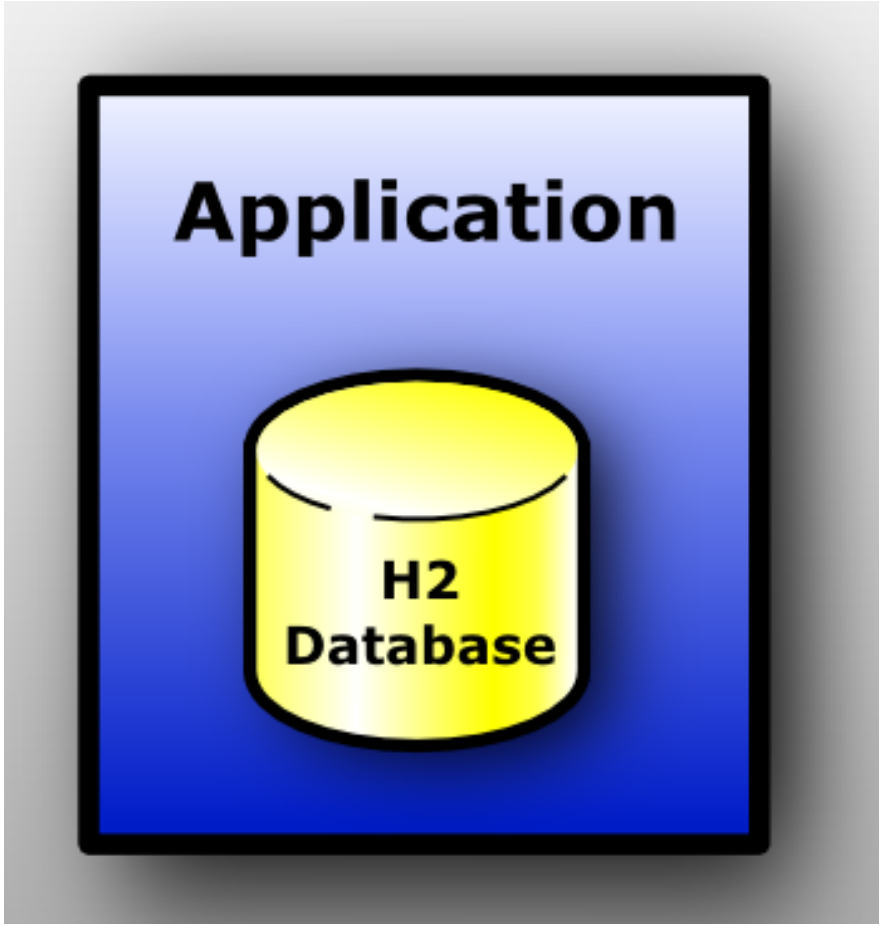
In-memory databases are easy to setup

h2 is such a database

(successor of HSQL)

Compatibility modes for most widespread DB

- `jdbc:h2:mem:test;MODE=Oracle`



Parameterizing properties

Update
`local.properties`

```
db.url=
db.driver=
db.username=
db.password=
```

And use your favorite build tool

Maven

- Resource filtering

Ant

Gradle

Web Services also are an infrastructure resource

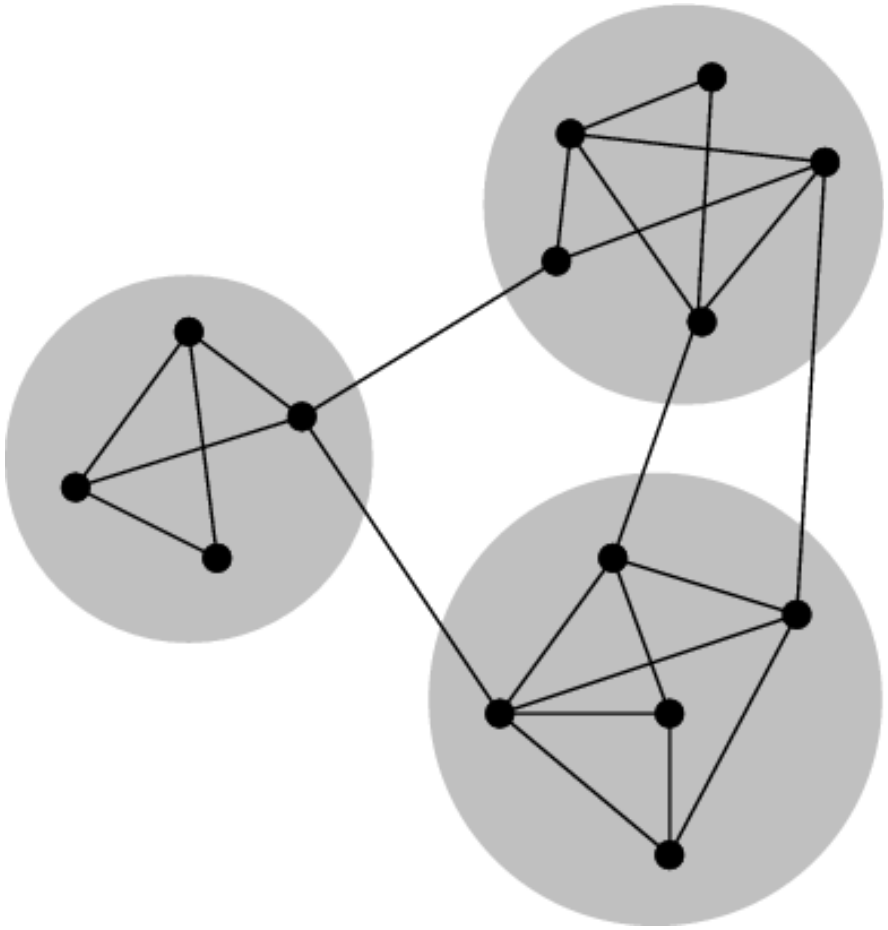
Hosted on-site

Or outside

Different Web Services types have different solutions

RESTful

SOAP



Faking RESTful WS

Require an HTTP server

Requirements

Easy setup

Standalone

Embeddable in tests

Spring MVC?

Requires a servlet container

- (Not with Spring Boot)

Some code to write

Author: Dwight Sipler from Stow, MA, USA

Spark to the rescue

Micro web framework

A la *Sinatra*

http://www.sparkjava.com/

Very few lines of code

Just wire to serve JSON files

Spark sample

```
import static spark.Spark.*;
import spark.*;

public class SparkSample{
  public static void main(String[] args) {
    setPort(5678);
    get("/hello", (request, response) -> {
      return "Hello World!";
    });
    get("/users/:name", (request, response) -> {
      return "User: " + request.params(":name");
    });
    get("/private", (request, response) -> {
      response.status(401);
      return "Go Away!!!";
    });
  }
}
```

Faking SOAP web service

Possible to use Spark for SOAP

But unwieldy

SOAPUI

SOAPUI is the framework to test SOAP WS

Has a GUI

Good documentation

Understands

- Authentication
- Headers
- Etc.

Can be used to Fake SOAP WS

SOAPUI usage

Get WSDL

Either online

Or from a file

Create MockService

Craft the adequate response

Run the service

Point the dependency to localhost

MockResponse can be (very) dynamic

Craft multiple response, serve one depending on request

In a sequence

Randomly

From XPath

- Matching the SOAPUI name for the response

From Query

- Same as above with a level of indirection

Script (yes, we can)

Craft a single response, with dynamic placeholder(s)

Script the placeholder value

Whireling Dervishes of the Mevlevi order, danse 1 by Claude Valette

Challenges to the previous scenario

Craft the adequate response?

More likely get one from the real WS

And tweak it

Running in an automated way

Save the project

Get the SOAPUI jar

Read the project and launch

SOAPUI automation

```
WsdlProject project = new WsdlProject();
  String wsdlFile = "file:src/test/resources/chapter7/
ip2geo.wsdl”;
  WsdlInterface wsdlInterface = importWsdl(project,
wsdlFile, true)[0];
  WsdlMockService fakeService =
project.addNewMockService("fakeService");
  WsdlOperation wsdlOp =
wsdlInterface.getOperationByName("ResolveIP");
  MockOperation fakeOp =
fakeService.addNewMockOperation(wsdlOp);
  MockResponse fakeResponse =
fakeOp.addNewMockResponse("fakeResponse");
 fakeResponse.setResponseContent("<soapenv:Envelope ...</
soapenv:Envelope>");
  runner = fakeService.start();
```

Faking Web Service in real-life

Use the same rules as for UT

Keep validation simple

Test one thing

- One Assert
- Or a set of related ones

Keep setup simple

Don’t put complex logic

- Don’t put too much logic
- Don’t put logic at all

Duplicate setup in each test

- Up to a point

Author: I, rolf B

hybris software
An SAP Company

In-container Testing

Upping the ante

Testing collaboration is nice

Faking infrastructure dependencies is nice

But didn’t we forget the most important dependency?

The container!

“Proprietary” container

Spring

Application Server

Tomcat

JBoss

<Place your favorite one here>

So far, we can use:

Real beans

- Service
- Controller

Test beans on fake resources

- Datasource

What about the configuration?

In Unit Tests, we set dependencies

- The real configuration is not used
- *Ergo*, not tested!

Testing configuration

Configuration cannot be monolithic

Break down into fragments

Each fragment contains a set of either

- Real beans
- Fake beans

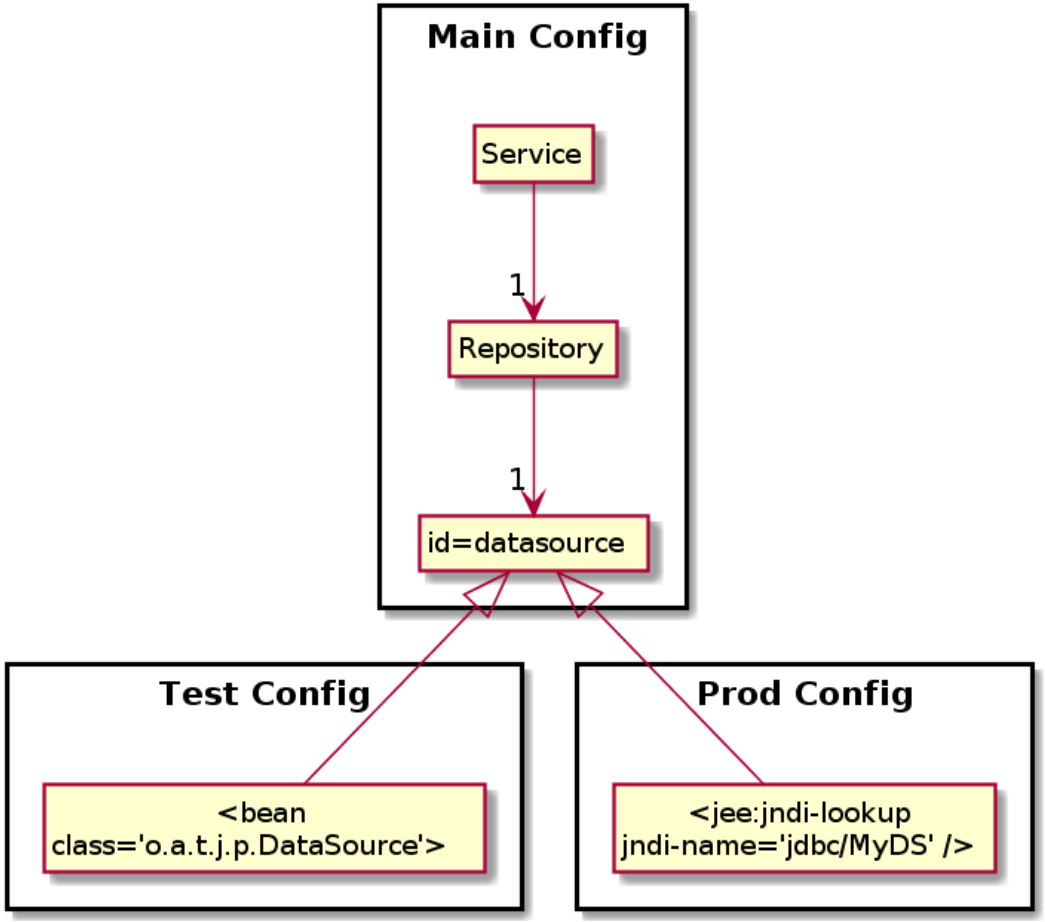
Rudston Monolith May 2013 by Angela Findlay

Data source configuration fragment management example

Different configuration fragments

Production JNDI fragment

Test in-memory fragment



Data source configuration sample

```
<beans ...>
  <jee:jndi-lookup id="ds" jndi-name="jdbc/MyDS" />
</beans>
```

```
<beans ...>
  <bean id="ds" class="o.a.t.jdbc.pool.DataSource">
    <property name="driverClassName”
              value="org.h2.Driver" />
    <property name="url" value="jdbc:h2:~/test" />
    <property name="username" value="sa" />
    <property name="maxActive" value="1" />
  </bean>
</beans>
```

Fragment structure

1. **Main fragment**

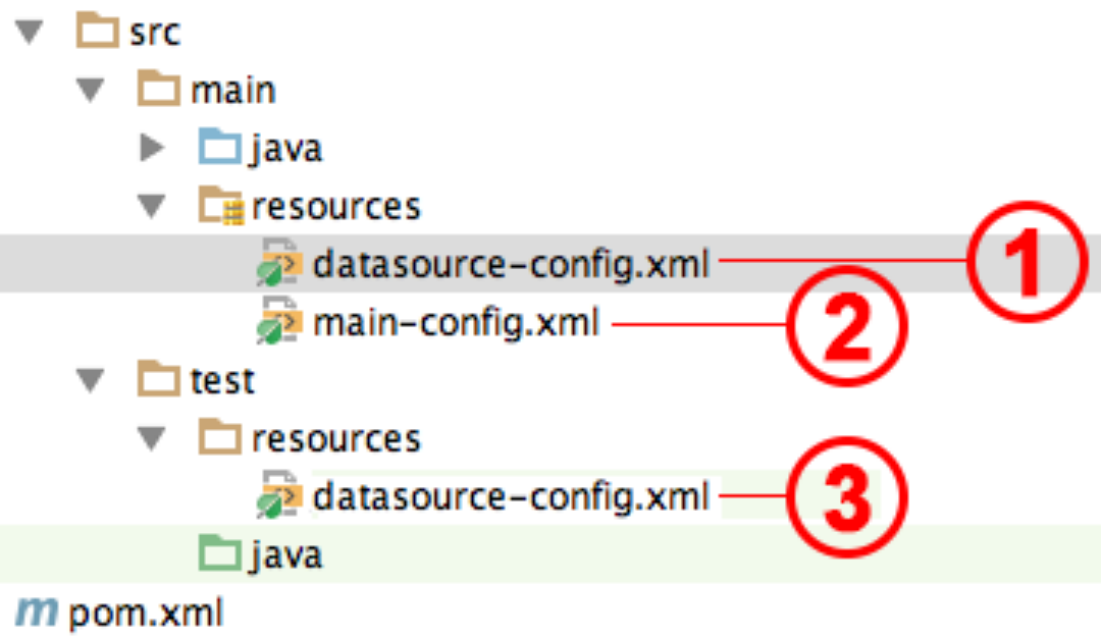
Repository

Service

etc.

2. **Prod DB fragment**

3. **Test DB fragment**



Tips

Prevent coupling

No fragments reference in fragments

Use top-level assembly instead

- Tests
- Application Context
- Webapps

Pool exhaustion check

Set the maximum number of connections in the pool to 1

Compile-time safety

Use JavaConfig

Not related to testing ☺

And now, how to test?

Get access to both

The entry point

And the “end” point

Spring Test to the rescue

Integration with common Testing frameworks

- ~~JUnit~~
- TestNG

St Louis Gateway Arch 1916" by Dirk Beyer - Own work.

Favor TestNG

Extra grouping

Per layer

Per use-case

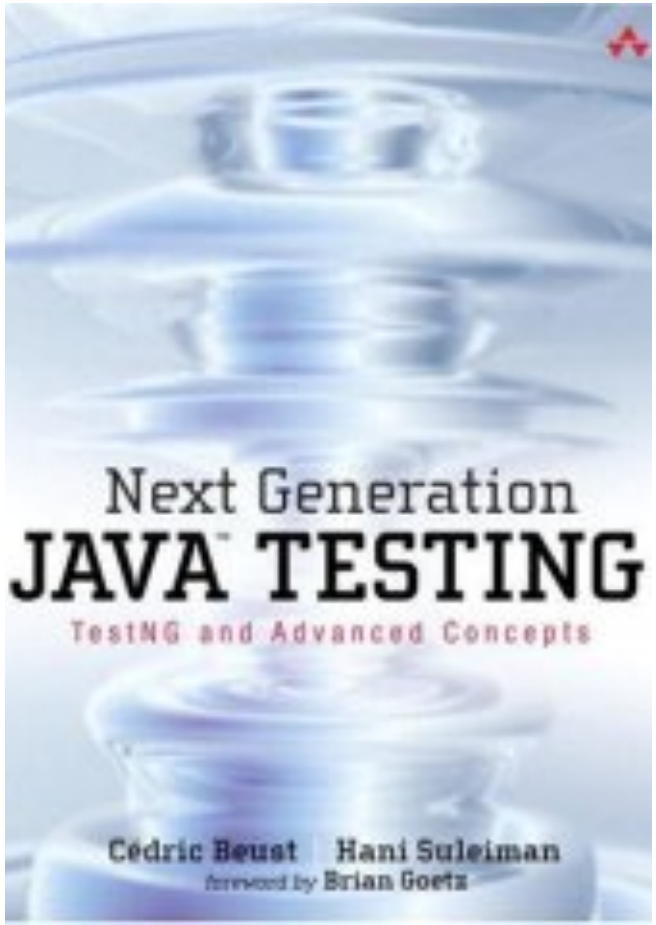
Name your own

Extra lifecycle hooks

Better parameterization

Data Provider

Ordering of test methods



Spring TestNG integration

AbstractTestNGSpringContextTests

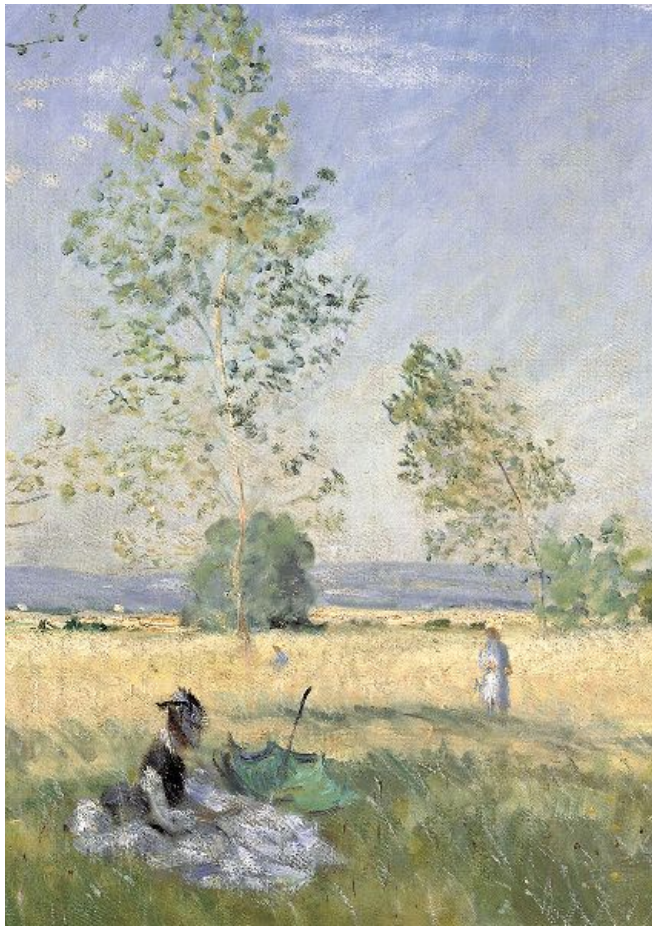
AbstractTransactionalTestNGSpringContextTests

Configurable context fragments

@ContextConfiguration

Inject any bean in the test class

If necessary, `applicatonContext` member from superclass



Sample TestNG test with Spring

```
@ContextConfiguration(
  classes = { MainConfig.class, AnotherConfig.class
})
public class OrderIT extends
AbstractTestNGSpringContextTests {

  @Autowired
  private OrderService orderService;

  @Test
  public void should_do_this_and_that() {
    orderService.order();
    ...
  }
}
```

Profiles

Profiles are an alternative to fragments

Instead of putting beans in different files, tag them

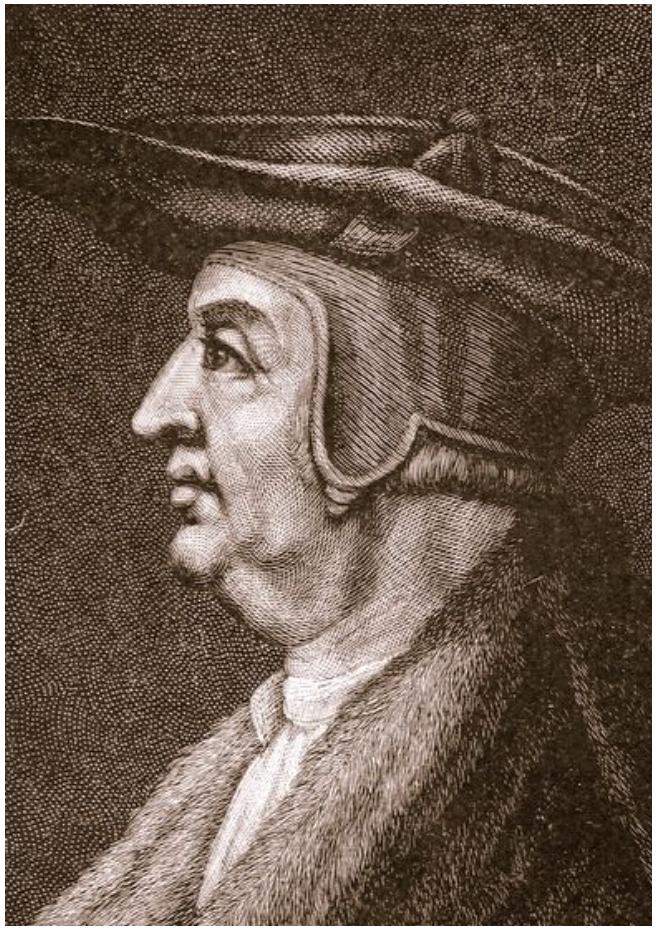
A profile is just a label

Each bean (or config class) can be tagged with a profile

Activating said profile at context creation will make Spring create the bean and put it in the context

Bean tagged with inactivated profiles won't be created

Beware, you're shipping test config into production!!!



Managing profiles

```
@Bean
@Profile("aProfile")
public Object aBean() {
   ...
}
```

```
@ActiveProfiles
public class MyTest
extends... {
   ...
}
```

Testing with the DB (or other transactional resources)

Transactions

Bound to business functionality

Implemented on Service layer

With DAO

Use explicit transaction management

```
@Transactional
```

Transaction management tip

Tests fail… sometimes

How to audit state?

By default, Spring rollbacks transactions

General configuration

```
@TransactionConfiguration(
  defaultRollback = false
)
```

Can be overridden on a per-method basis

- @Rollback(true)

Sample Transaction management

```
@ContextConfiguration
@TransactionConfiguration(defaultRollback = false)
public class OverrideDefaultRollbackSpringTest
  extends AbstractTransactionalTestNGSpringContextTests {

  @Test
  @Rollback(true)
  public void transaction_will_be_rollbacked() { ... }

  @Test
  public void transaction_wont_be_rollbacked() { ... }
}
```

Spring MVC webapps Testing

Require a context hierachy

Parent as main context

Child as webapp context

@ContextHierarchy

Require a webapp configuration

@WebAppConfiguration

Spring MVC test sample

```
@WebAppConfiguration
@ContextHierarchy({
  @ContextConfiguration(classes = MainConfig.class),
  @ContextConfiguration(classes = WebConfig.class)
})
public class SpringWebApplicationTest
    extends AbstractTestNGSpringContextTests {

  ...
}
```

Entry points for testing Spring webapps

At the HTML level

At the HTTP level

At the Controller level

Like standard Java testing

"Lahntunnel Weilburg" by rupp.de - Own work.

Tools for testing webapps

HTML testing tools

Interact with HTML/CSS

- Fill this field
- Click on that button

HTTP testing tools

- Send HTTP requests
- Get HTTP responses

Drawback of previous approaches

Very low-level

Fragile!

Remember that testing is about ROI

- Breaking tests with every HTML/CSS change is the worst way to have positive ROI
- (There are mitigation techniques → out of scope)

Attribution: © Milan Nykodym, Czech Republic

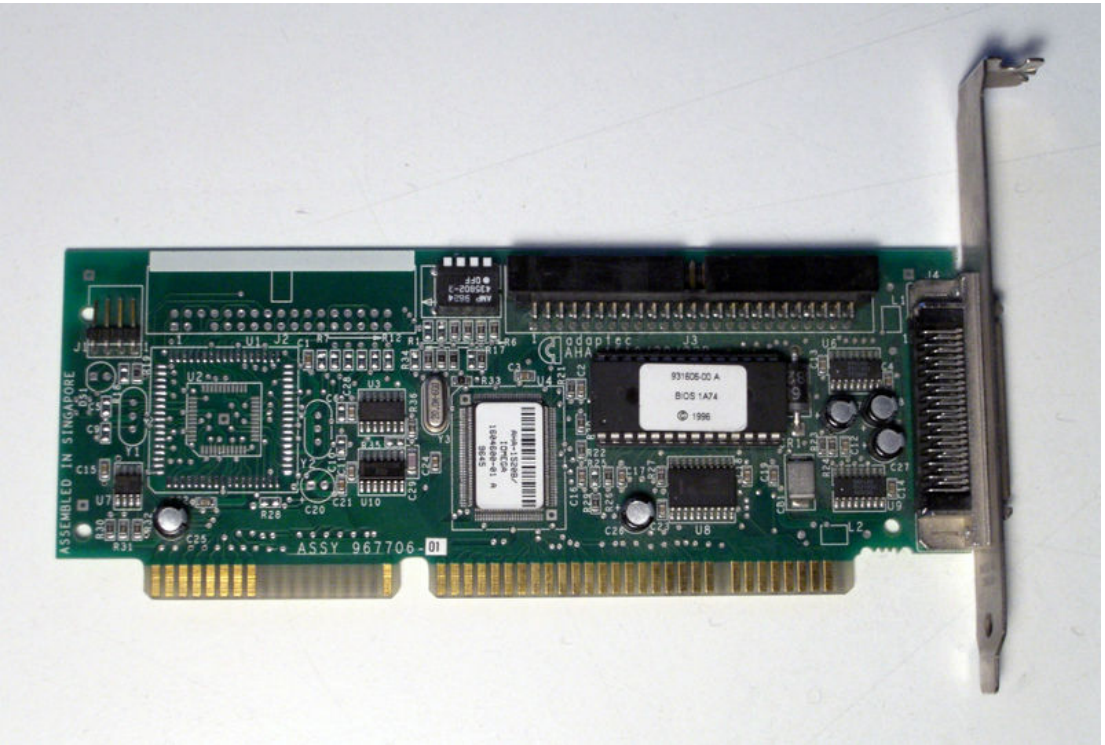
Drawback of Testing with controllers as entry point

Bypass many URL-related features

Interceptors

Spring Security

etc.



Controller SCSI.JPG by Rosco

Spring Test to the rescue

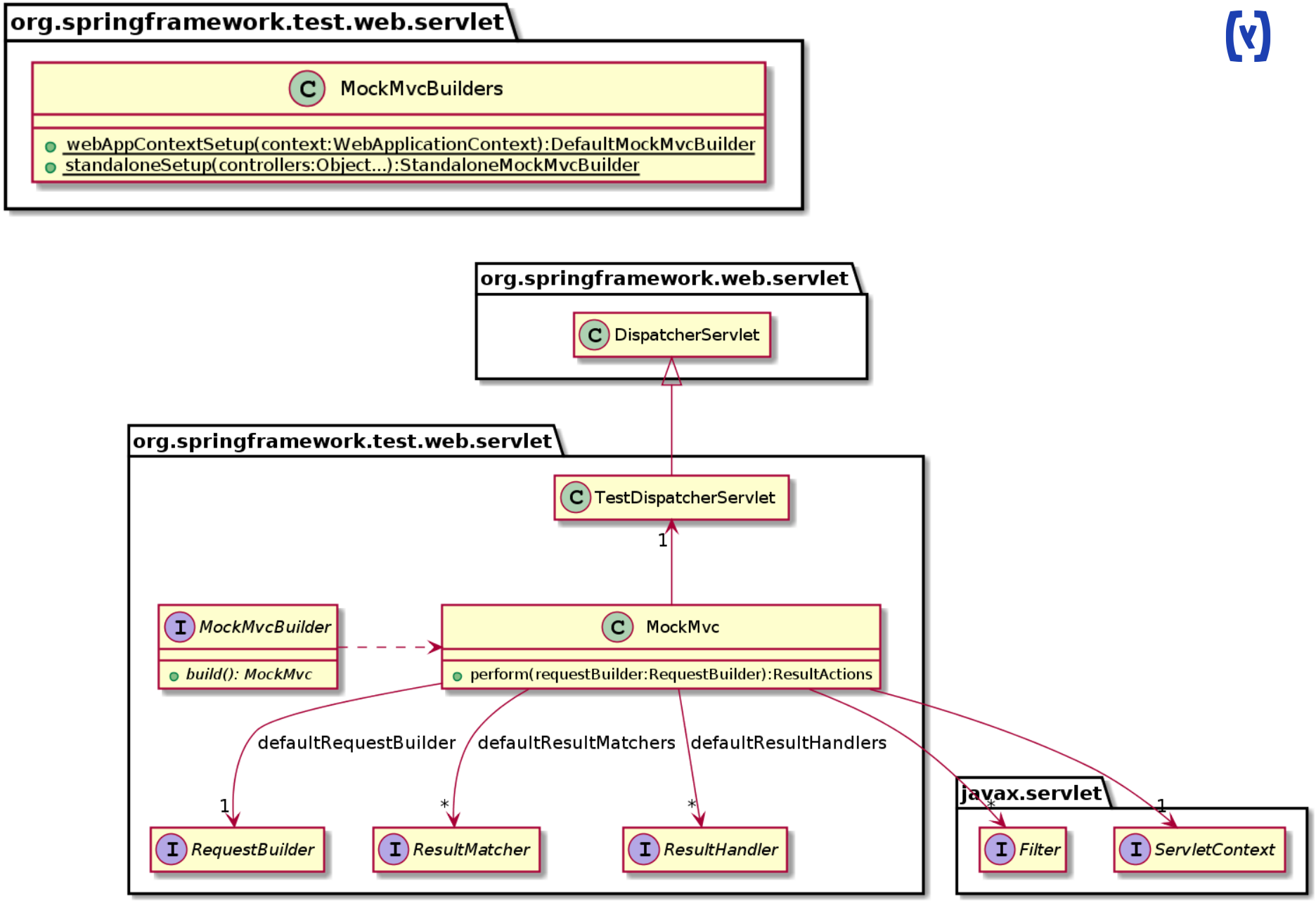
Spring Test has a large chunk dedicated to MVC

Since 3.2

Can test with URL as entry-points

Fluent API with static imports

Coastguard Helicopter (8016050677)" by Paul Lucas from Leicestershire, UK - Coastguard Helicopter


org.springframework.test.web.servlet
MockMvcBuilders
webAppContextSetup(context:WebApplicationContext):DefaultMockMvcBuilder
standaloneSetup(controllers:Object...):StandaloneMockMvcBuilder
org.springframework.web.servlet
DispatcherServlet
org.springframework.test.web.servlet
TestDispatcherServlet
1
MockMvcBuilder
build(): MockMvc
MockMvc
perform(requestBuilder:RequestBuilder):ResultActions
defaultRequestBuilder
defaultResultMatchers
defaultResultHandlers
1
*
*
RequestBuilder
ResultMatcher
ResultHandler
javax.servlet
*
1
Filter
ServletContext

MockMvc class responsibilities

Request builder

Configures the Fake request

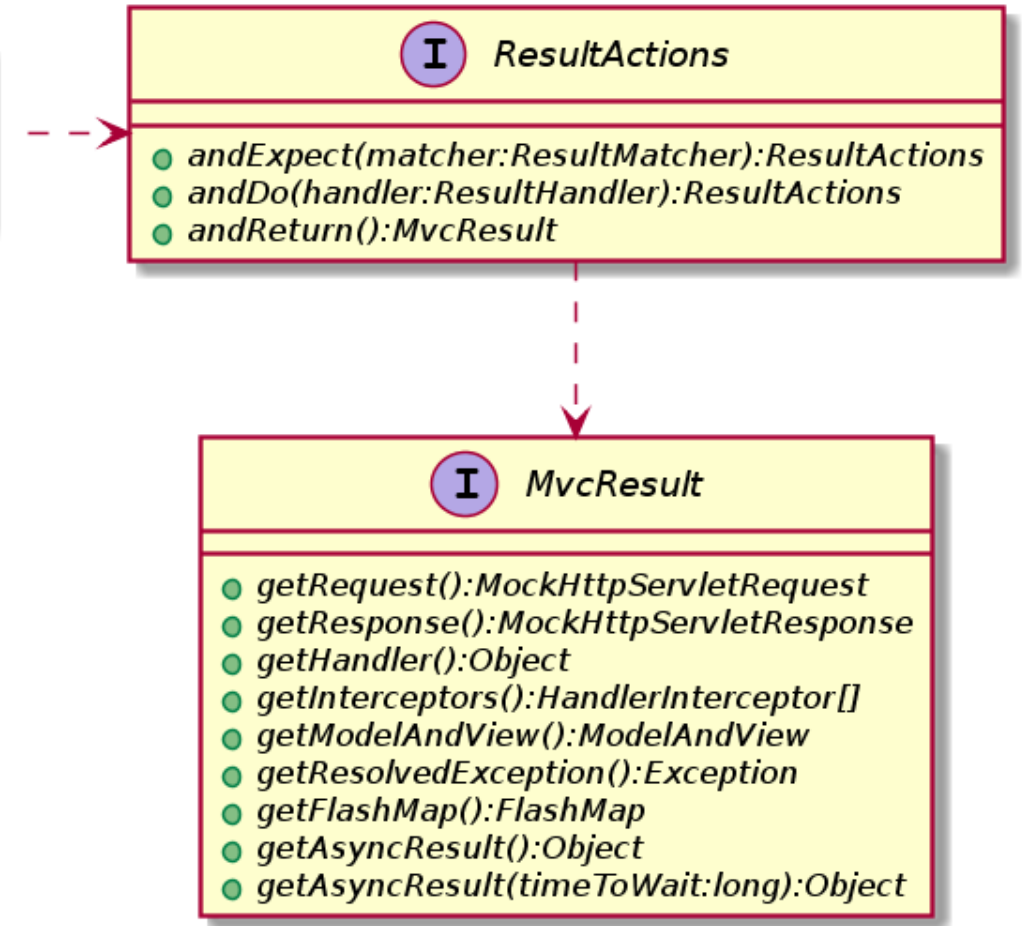
Request matcher

Misc. assertions

Request handler

Do something

- OOB logger



Available configuration on Request Builder

HTTP method

GET

POST

etc.

HTTP related stuff

Headers

Content

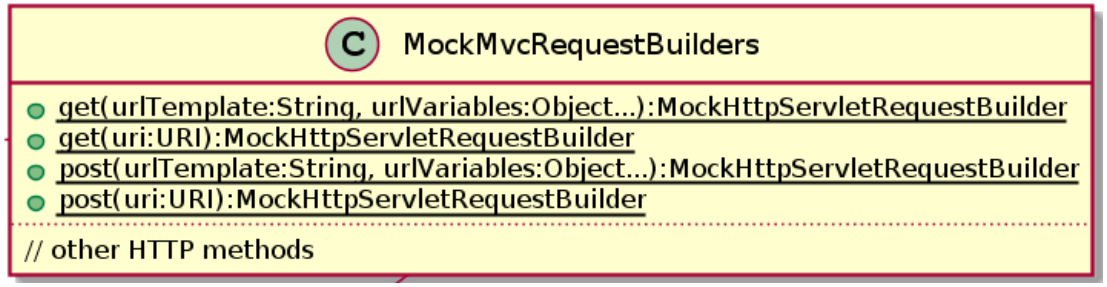
JavaEE related stuff

Parameters

Request attributes

Session

etc.



Request Builder sample

```
MockHttpServletRequestBuilder builder =
  get("/customer/{id}", 1234L)
    .accept("text/html")
    .param("lang", "en")
    .secure(true);
```

```
GET /customer/1234?lang=en HTTP/1.1
Accept: text/html
```

You can use constants in your @RequestMapping

```
@Controller
public class MyController {

  public static final PATH = "/customer/${id}";

  @RequestMapping(PATH)
  public String showCustomer() {...}
}
```

```
MockHttpServletRequestBuilder builder = get(PATH, 1L);
```

Available Request Matcher

Entry point is `MockMvcResultMatchers`

Provides static methods returning

`RequestMatcher` implementations

“Grouping” classes that return them

Methods returning matchers

Checks result is a

Forward

- Either exact
- Or regexp

Redirect

- Either exact
- Or regexp

JSON payload

a safety wax match box and matches by Aathavan jaffna

Methods returning grouping classes

Request class

Handler class
Controller

Content class

Cookie class

Status class
HTTP code

Flash class
(Attributes, not the techno)

View class

Model class

"Ovejas en Patagonia - Argentina" by writtecarlosantonio

Spring Pet Clinic

Integration Testing on Spring Pet Clinic

```
@WebAppConfiguration
@ContextHierarchy({
  @ContextConfiguration("classpath:spring/business-config.xml"),
  @ContextConfiguration("classpath:spring/mvc-core-config.xml")
})
@ActiveProfiles("jdbc")
public class PetControlIT extends
AbstractTestNGSpringContextTests {

  @Test
  public void should_display_create_form() throws Exception {
    WebApplicationContext wac = (WebApplicationContext)
applicationContext;
    MockMvc mvc =
      MockMvcBuilders.webAppContextSetup(wac).build();
    MockHttpServletRequestBuilder newPet =
      get("/owners/{ownerId}/pets/new", 1);
    mvc.perform(newPet)
      .andExpect(view().name("pets/createOrUpdatePetForm"))
      .andExpect(model().attributeExists("pet"));
    }
}
```

JavaEE has unique challenges

CDI has no explicit wiring

- You can `@Veto` you own classes
- But no compiled ones

Different application servers

- Same specifications
- Different implementations

Deploy only what you want

Standalone API to deploy only resources relevant to the test

Just pick and choose

ShrinkWrap

Maven Integration

Gradle too…

Shrinkwrap sample

```
String srcMainWebapp = "src/main/webapp/";
ShrinkWrap.create(WebArchive.class, "myWar.war")
    .addClass(MyService.class)
    .addPackage(MyModel.class.getPackage())
    .addAsWebInfResource("persistence.xml",
        "classes/META-INF/persistence.xml")
    .addAsWebInfResource(
        new File(srcMainWebapp, "WEB-INF/page/my.jsp"),
       "page/my.jsp")
    .addAsWebResource(
        new File(srcMainWebapp, "script/my.js"),
       "script/my.js")
    .setWebXML("web.xml");
```

Maven integration sample

```
File[] libs = Maven.resolver()
    .loadPomFromFile("pom.xml")
    .importDependencies(COMPILE, RUNTIME).resolve()
    .withTransitivity().asFile();
ShrinkWrap.create(WebArchive.class, "myWar.war")
    .addAsLibraries(libs);
```

Different application servers

Abstraction layer to

Download

Deploy applications

Test

Container adapters

TomEE

JBoss

Weld

etc.

Full Maven integration

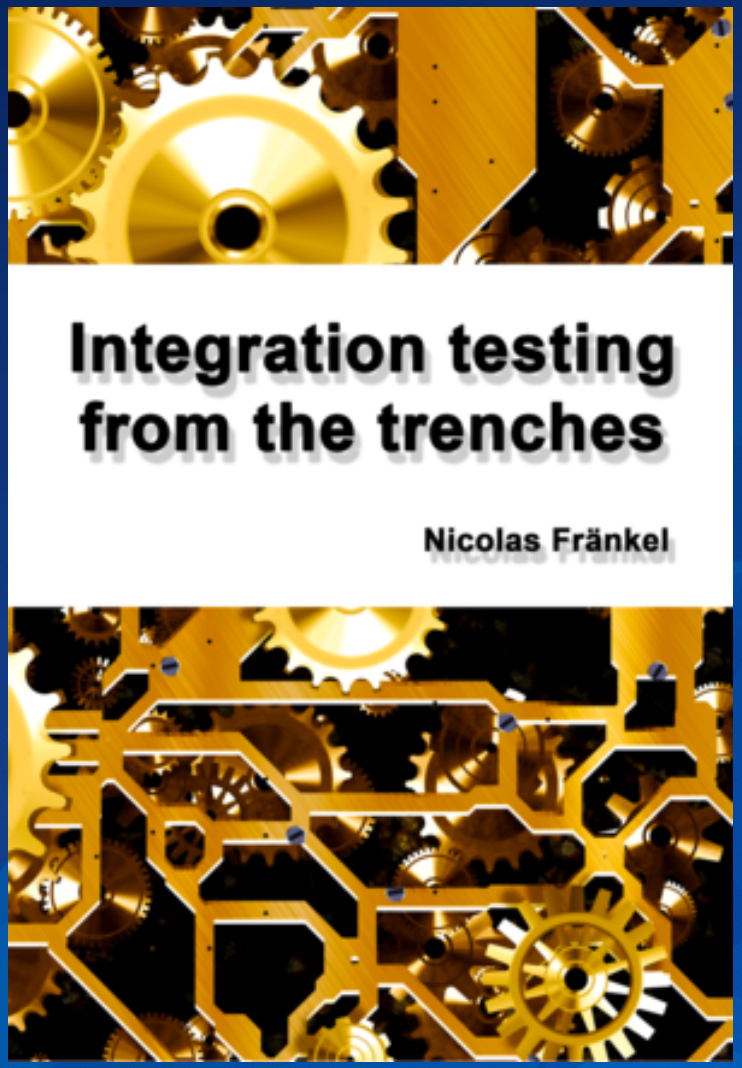
Arquillian Test sample

```
public class ArquillianSampleIT extends Arquillian {

  @Inject
  private MyService myService;

  @Deployment
  public static JavaArchive createDeployment() {
      return ...;
   }

  @Test
  public void should_handle_service() {
    Object value = myService.handle();
    Assert.assertThat(...);
  }
}
```

Arquillian configuration sample

```
<arquillian xmlns="http://jboss.org/schema/arquillian"
  xmlns:xsi="..."
  xsi:schemaLocation="
    http://jboss.org/schema/arquillian
http://jboss.org/schema/arquillian/arquillian_1_0.xsd">
  <container qualifier="tomee" default="true">
    <configuration>
      <property name="httpPort">-1</property>
      <property name="stopPort">-1</property>
  </configuration>
</arquillian>
```

https://leanpub.com/integrationtest



Twitter: @itfromtrenches