Real Java EE Testing
with Arquillian and ShrinkWrap

Don't mock me.

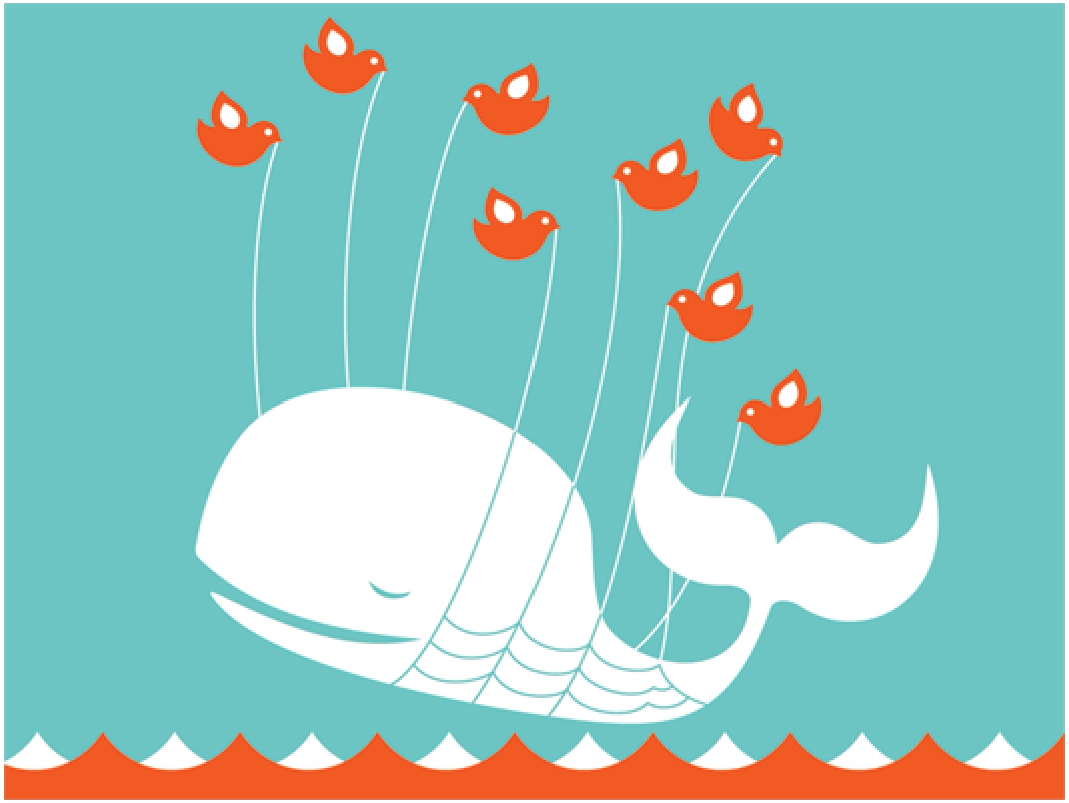
Dan Allen
Principal Software Engineer
JBoss, by Red Hat

Agenda

#arquillian

- Barriers to testing
- Rethinking integration testing
- ShrinkWrap
- Arquillian
- Demo, demo, demo
- Future
- Q & A

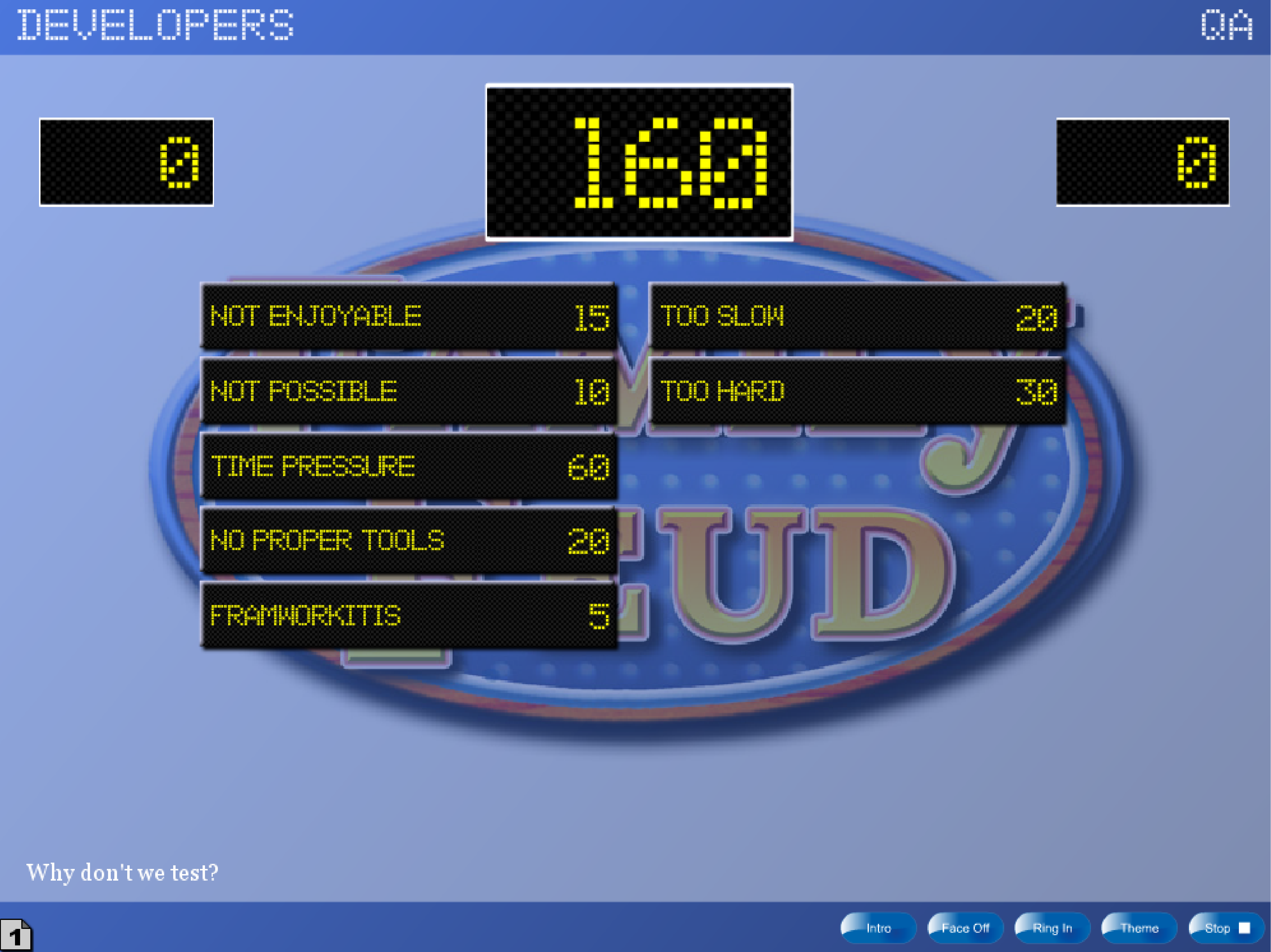
Why don't we test?



DEVELOPERS QA

Why don't we test?

Intro Face Off Ring In Theme Stop ■



DEVELOPERS

QA

0

160

0

NOT ENJOYABLE	15
NOT POSSIBLE	10
TIME PRESSURE	60
NO PROPER TOOLS	20
FRAMWORKITIS	5
TOO SLOW	20
TOO HARD	30

Why don't we test?

Unit tests vs integration tests

Unit

- Fine-grained
- Simple
- Single API call
- Fast, fast, fast

- Easily run in an IDE

Integration

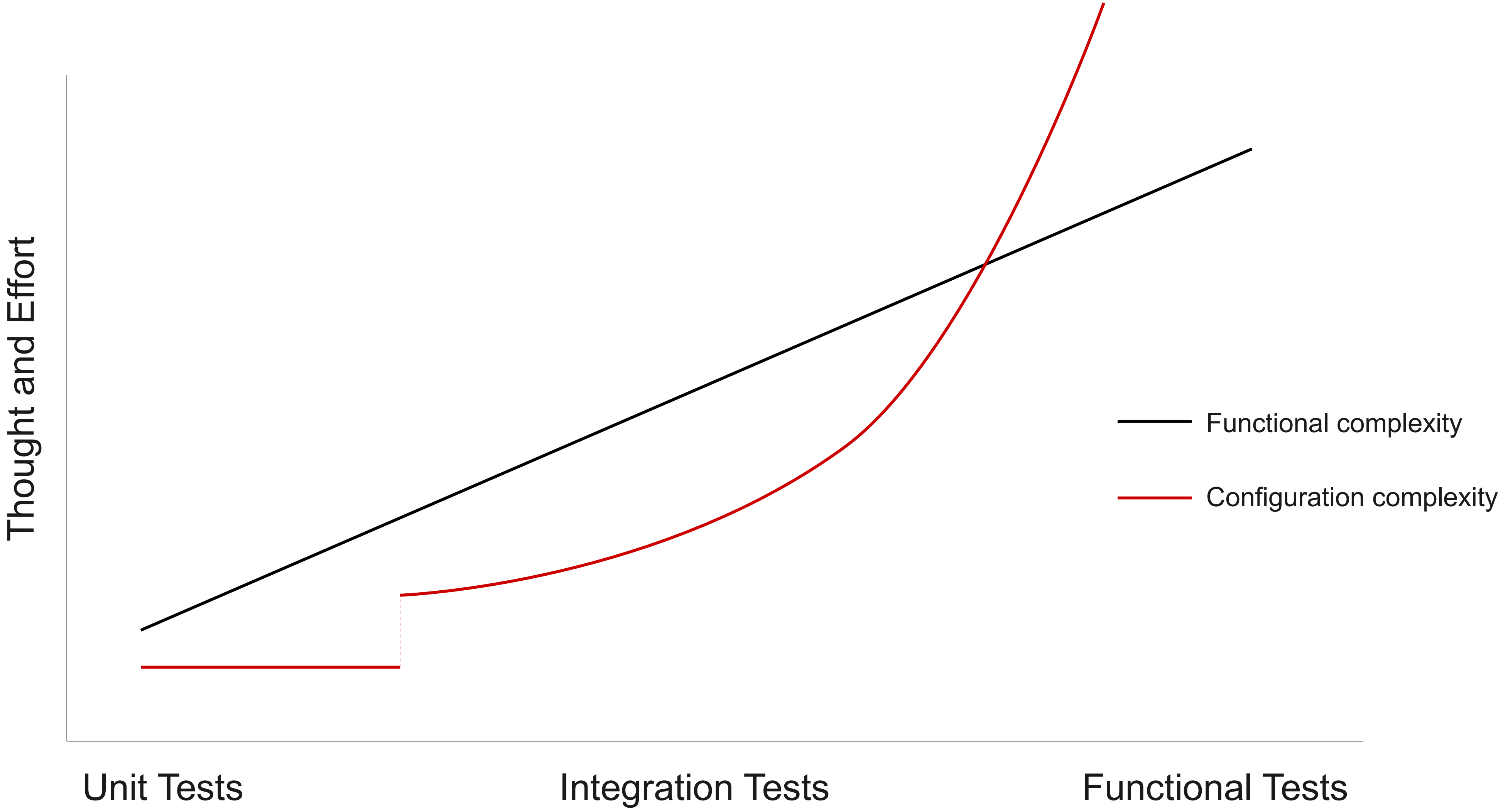
- Coarse-grained
- Complex
- Component interactions
- Sloooooooow

- Run in an IDE? How?

Common integration testing challenges

- Start a container environment
- Run a build to create/deploy application archive
- Mock dependent components
- Configure application to use test data source(s)
- Deal with (lack of) classpath isolation

The “testing bandgap” and compounding effort



What if integration testing could be...

- as easy as writing a unit test
- run in the IDE, leveraging incremental builds
- limited to select components running in isolation
    - ...avoiding the “big bang” integration roadblock

Skip the Build!

Test in-container!

arquillian

An in-container approach to integration testing

1. Start or connect to a container
2. Package and deploy test case* as archive
3. Run test in-container
4. Collect results
5. Undeploy test archive

container

n. *a process that manages a runtime environment and provides resources, a component model and a set of services*

Containers and components

- Component
  - Follows standard programming model
  - Encapsulates business logic
  - Packaged in deployable archive
- Container
  - Loads and executes components
  - Provides services and resources

What's been ^missing from Java EE?

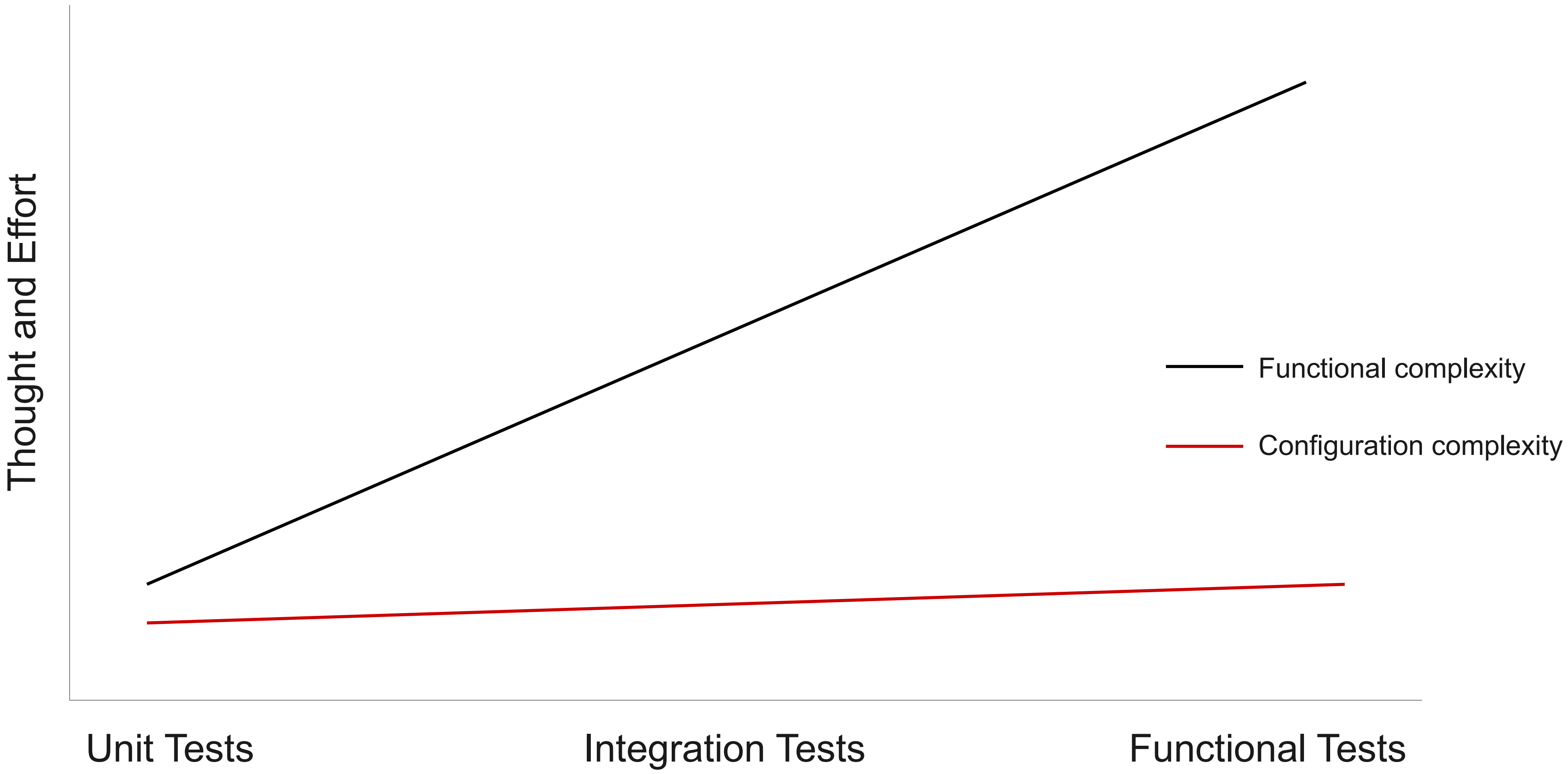
arquillian

A component model

for your tests

Arquillian's testing continuum

Reducing enterprise testing to child's play



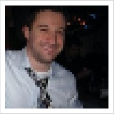
How do we get there?

Step 1: Liberate tests from the build!

- Adds overhead to development cycle
- Slows down test execution
- Uses coarse-grained classpath/package

- Keep it manageable!
    - Well-defined “unit”
    - Classpath control

Project lead: ***A****ndrew* ***L****ee* ***R****ubinger*



ShrinkWrap

n. *a fluent Java API for programmatic creation of archives like JARs, WARs and EARs; developed by the JBoss Community*

Benefits of ShrinkWrap

- Incremental IDE compilation
  - Save and re-run
  - Skip the build!
- Simple, fluent API
- Tooling views
- Export and debugging
- Micro deployments

Fluent archive creation

```
final JavaArchive archive = ShrinkWrap.create(JavaArchive.class, "slsb.jar")
     .addClasses(Greeter.class, GreeterBean.class);
System.out.println(archive.toString(true));
```

Yields output:

```
slsb.jar:
/com/
/com/acme/
/com/acme/app/
/com/acme/app/ejb3/
/com/acme/app/ejb3/Greeter.class
/com/acme/app/ejb3/GreeterBean.class
```

Build, what build?

```
@Deployment
public static Archive<?> createDeployment() {
  return ShrinkWrap.create(JavaArchive.class)
     .addClasses(Greeter.class, GreeterBean.class);
}
```

Skip the build!

```
@Deployment
public static Archive<?> createDeployment() {
  return ShrinkWrap.create(JavaArchive.class)
    .addPackage(TemperatureConverter.class.getPackage())
    .addManifestResource(EmptyAsset.INSTANCE, "beans.xml");
}
```

Step 2: Gut the plumbing!

- Manage lifecycle of container
- Enrich test class
- Package test archive
- Deploy test archive
- Invoke tests
- Capture test results

- What's left?
  - Pure test logic

Project lead: ***Aslak*** *Knutsen*

arquillian

n. *a container-oriented testing framework that abstracts away container lifecycle and deployment from test logic so developers can easily develop a broad range of integration tests for their enterprise Java applications; developed by the JBoss Community*

Arquillian project mission

Make integration testing a breeze!

Prove it.

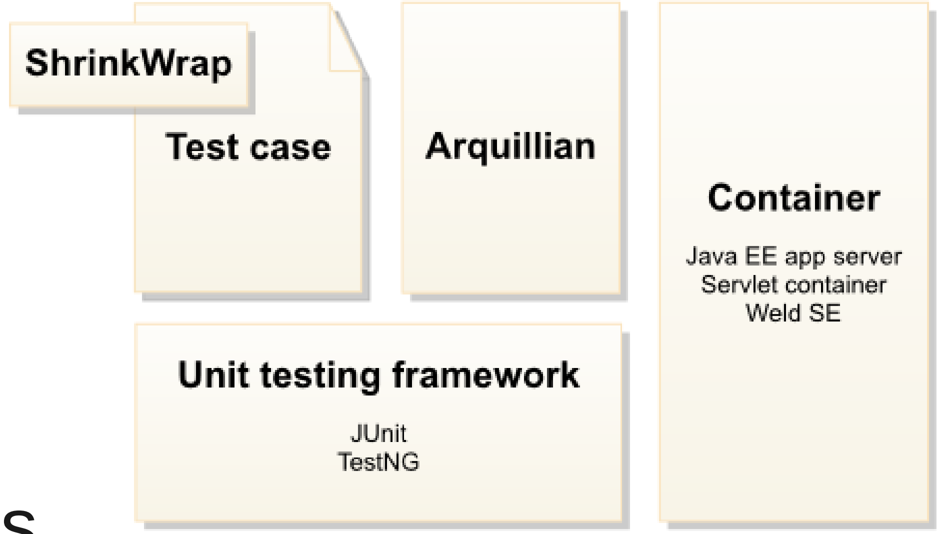
```java
@RunWith(Arquillian.class)
public class GreeterTestCase {

   @Deployment
   public static Archive<?> createDeployment() {
      return ShrinkWrap.create(JavaArchive.class)
         .addClasses(Greeter.class, GreeterBean.class);
   }

   @EJB private Greeter greeter;

   @Test
   public void shouldBeAbleToInvokeEJB() throws Exception {
      assertEquals("Hello, Earthlings", greeter.greet("Earthlings"));
   }
}
```

Benefits of Arquillian

- Write less (test) code
- As much or as little “integration” as you need
- Looks like a unit test, but you're in a real environment!
  - Easily lookup component to test
  - You no longer hesitate when you need a resource
- Run same test in multiple containers

Arquillian tactical unit

- Unit testing framework runner
- Deployable test archive
- Test enrichers
- Test run mode
- Target container
- Test framework integrations



Supported unit testing frameworks

JUnit

>= 4.6

TestNG

>= 5.10

Test archive

- Assembled using ShrinkWrap API
- Declared with static @Deployment method
- Bundles:
  - Class/component to test
  - Supporting classes
  - Configuration and resource files
  - Dependent libraries

Micro deployments

- Deploy components in isolation
- Test one functional unit at a time
- Don't need to wait for full application build/startup
- Incremental integration
  - Hand pick components & resources
  - No “big bang” integration

Test enrichment

- Injection
  - Fields & method arguments
  - @Inject, @Resource, @EJB, @PersistenceContext, ...
  - CDI producer methods in test class
- Context management
  - Request & Conversation ➔ Test method
  - Session ➔ Test class
  - Application ➔ Test class
- *More to come...*

Test run modes

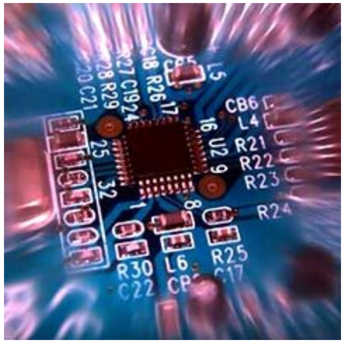
- *In-container*
  - Test bundled with @Deployment archive
  - Archive deployed to container
  - Test runs inside container alongside application code
  - Test invokes application code directly (same JVM)
- *As client*
  - @Deployment archive is test archive (unmodified*)*
  - Archive deployed to the container
  - Test runs in original test runner
  - Test interacts as a remote client (e.g., HTTP client)

Containers

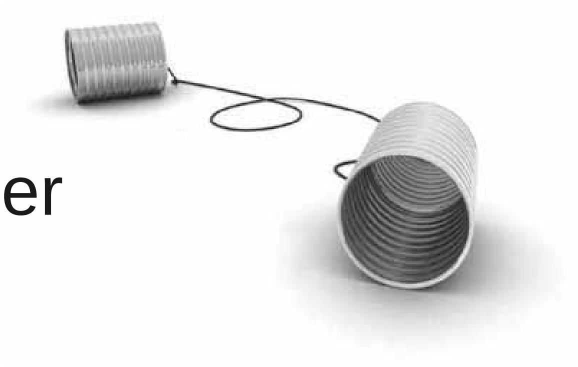
- By mode
- By compliance

Container modes

- *Embedded*
  - Same JVM as test runner
  - Test protocol either local or remote
  - Lifecycle controlled by Arquillian



- *Remote*
  - Separate JVM from test runner
  - Arquillian connects to running container
  - Tests executed over remote protocol



- *Managed*
  - Remote with lifecycle management

Container compliance

- Java EE application server (JBoss AS, GlassFish, etc)
- Servlet container (Tomcat, Jetty)
- Managed bean container (Weld SE, Spring)
- OSGi
- *Whatever else comes along...*

Supported containers

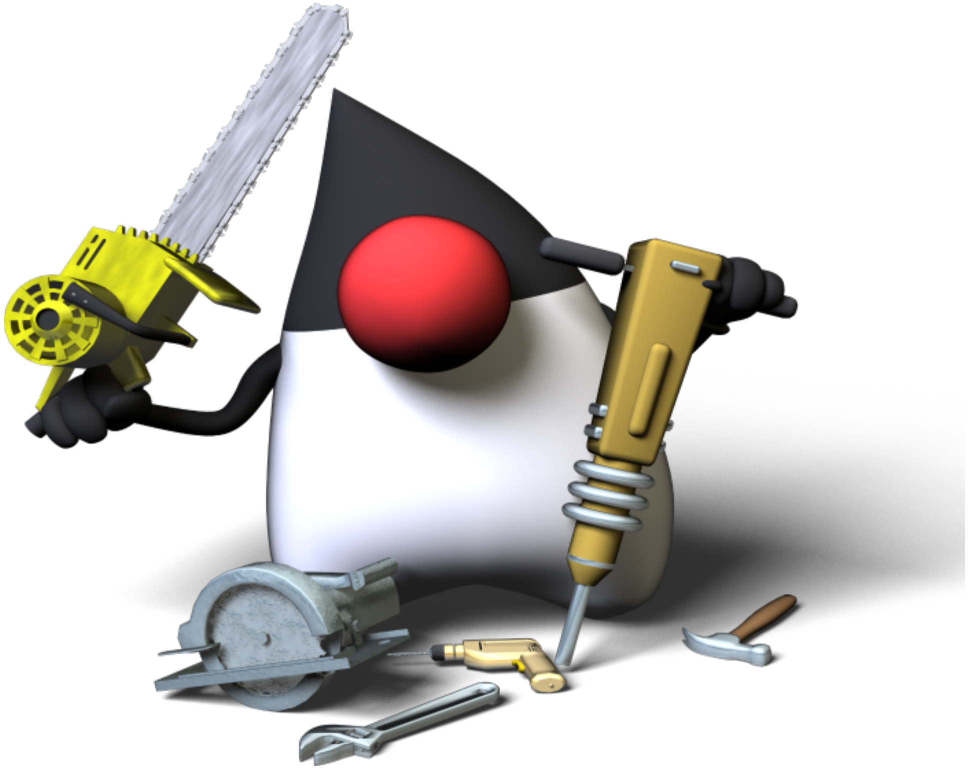
- JBoss AS 5.0 & 5.1 – Managed and remote
- JBoss AS 6 – Managed, remote and embedded
- JBoss JCA – Embedded
- GlassFish 3 – Remote and embedded
- Weld – SE embedded and EE mock embedded
- OpenWebBeans – embedded
- OpenEJB 3.1 – embedded
- OSGi – embedded
- Tomcat 6, Jetty 6.1 and Jetty 7 – embedded
- *More on the way...*

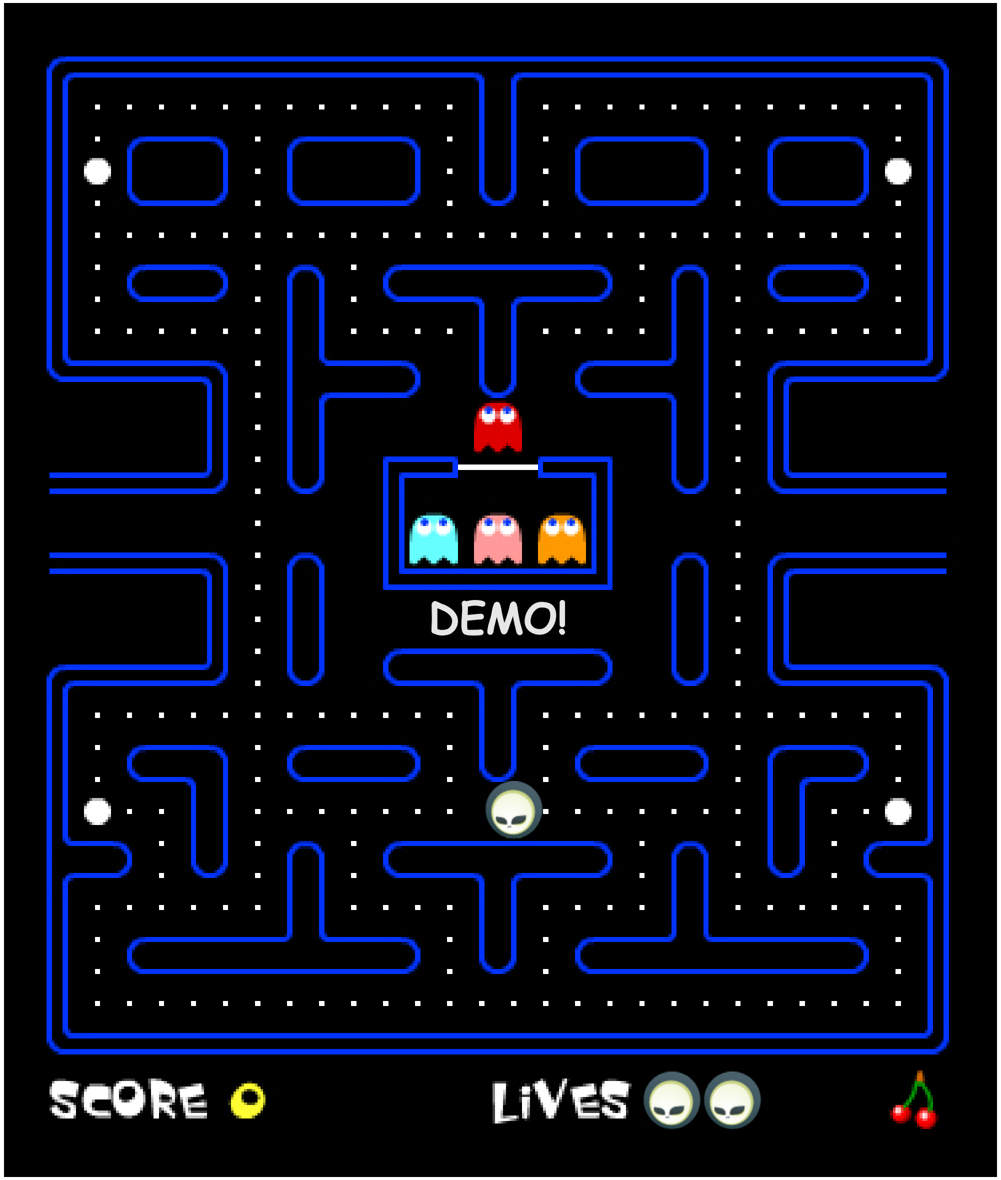
Container SPI, not just for Java EE

```
public interface DeployableContainer {

   void setup(Context context, Configuration configuration);

   void start(Context context) throws LifecycleException;

   ContainerMethodExecutor deploy(Context context, Archive<?> archive)
         throws DeploymentException;

   void undeploy(Context context, Archive<?> archive)
         throws DeploymentException;

   void stop(Context context) throws LifecycleException;

}
```

Power tools: test framework integration

- Test frameworks are services too
- Extend component model for tests
- Examples:
  - HTTPUnit
  - JSFUnit
  - DBUnit
  - Cobertura




DEMO!
SCORE 0
LIVES

Arquillian...

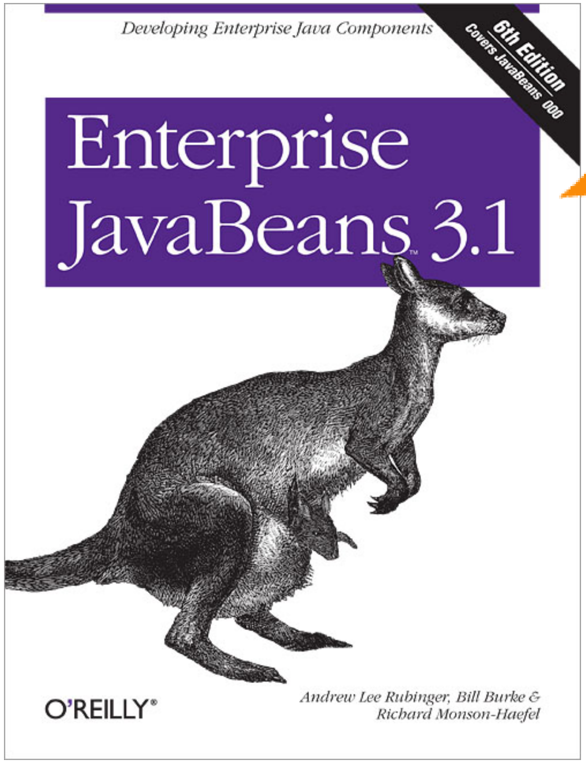
- is a container-oriented testing framework
- provides a component model for tests
- handles test infrastructure & plumbing
- ships with a set of container implementations
- provides a little bit of magic ;)

Arquillian invasion strategy

- **1.0.0.Alpha3 out!**
- More containers
- More framework integrations
  - DBUnit, HTTPUnit, Selenium
- Cloud
- Performance extension
- Component coverage
- Package-time bytecode manipulation
  - Exception, callback, assertion injection
- Tooling

We're in print!

LOOK INSIDE!



Developing Enterprise Java Components

6th Edition
Covers JavaBeans 000

Enterprise JavaBeans 3.1

O'REILLY®

Andrew Lee Rubinger, Bill Burke & Richard Monson-Haefel

Enterprise JavaBeans 3.1, Sixth Edition
O'Reilly - *Andrew Lee Rubinger, et al*

http://community.jboss.org/groups/oreillyejb6th

Get involved!

- Participate with us!
  - Share ideas on the forums (or IRC)
  - Give us feedback on releases – still in alpha!
- Fork us! github SOCIAL CODING
  - http://github.com/arquillian
- Meet us!
  - #jbosstesting channel on irc.freenode.net
- Write for us!
  - Share your stories – Blog! Tweet!
  - Document how it works

The JBoss Testing Initiative

- Comprehensive testing tool “stack”
- Establish a testing culture in Java EE
  - #jbosstesting channel on irc.freenode.net
- Filling voids
  - ShrinkWrap – Programmatic archive creation
  - Arquillian – Managed integration testing
  - JSFUnit – Gray-box JSF testing
  - Placeebo – Mock Java EE API implementations
  - Unit testing framework enhancements?

Q & A

http://jboss.org/arquillian
http://jboss.org/shrinkwrap