BEAN TESTING

BECAUSE LIFE IS TOO SHORT FOR INTEGRATION TESTS

Carlos Barragan

IT Consultant

NOVATEC
make IT happen!

carlos.barragan@novatec-gmbh.de

@barraganc

Was

ist Bean-Testing?

kann ich alles mit Bean-Testing testen?

Warum

wird CDI verwendet?

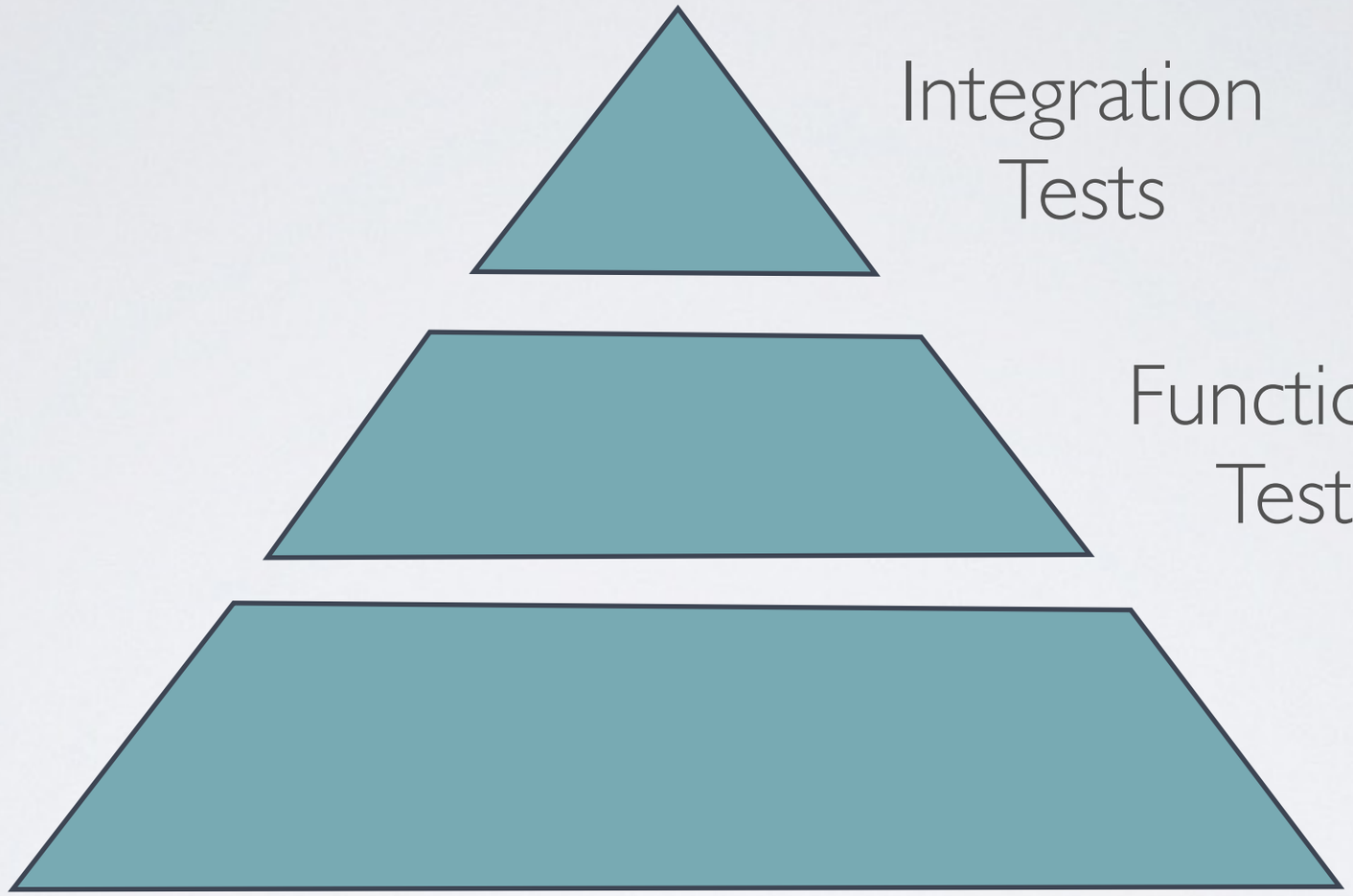
Wie

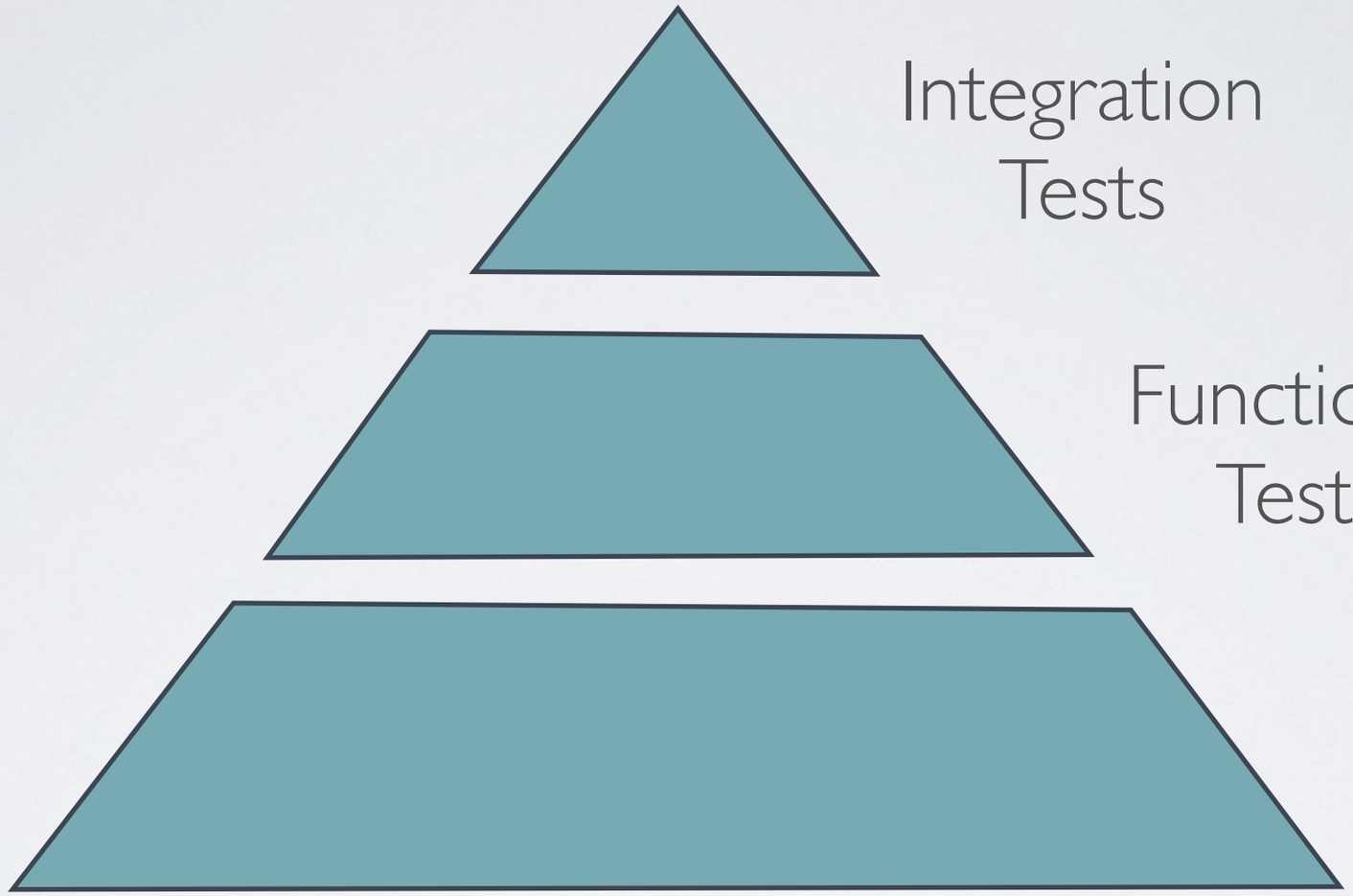
passt Bean-Testing zu Unit- und Integrationstesting?

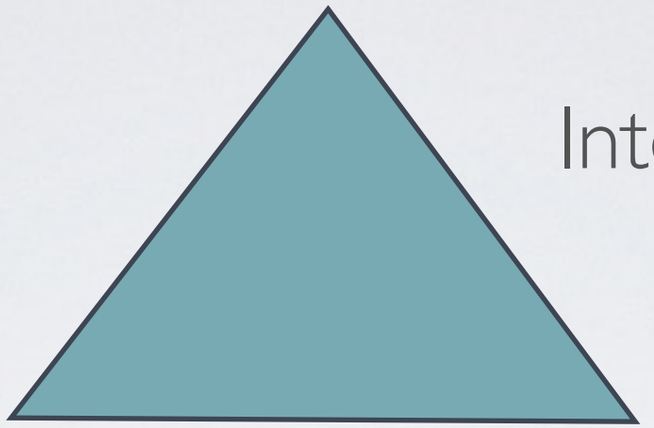
passt Bean-Testing zum Entwicklungsprozess?

kann man mit Bean-Testing Zeit und Geld sparen?

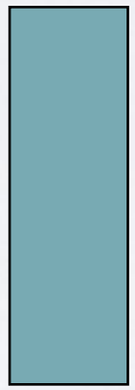
kann ich Bean-Testing in meinem aktuellen Projekt einsetzen?


Integration
Tests
Functional
Tests
Unit
Tests


Integration
Tests
Functional
Tests
Unit
Tests



Integration
Tests



Unit Tests

CDT

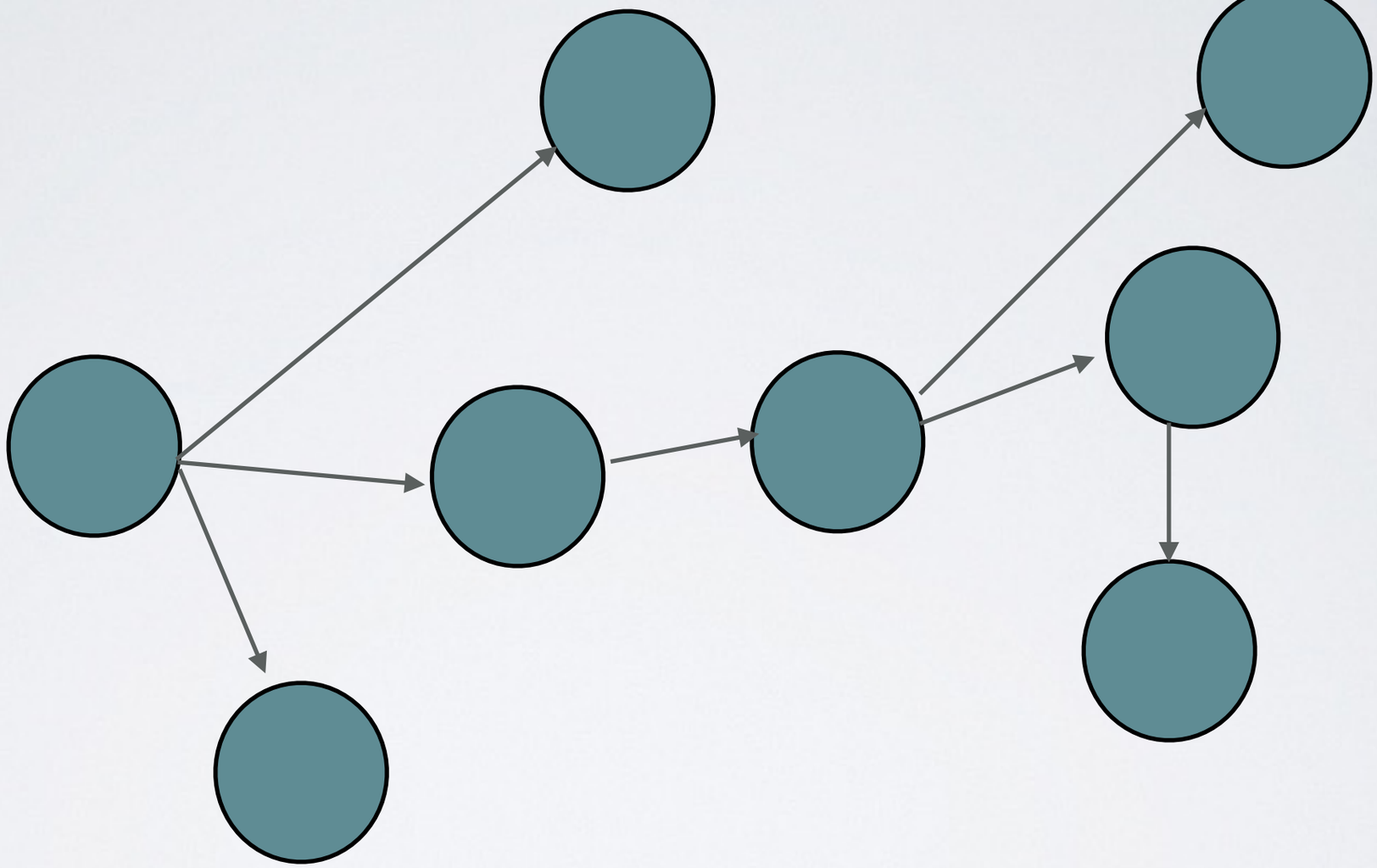
Customer Driven Tests

UNIT TESTS

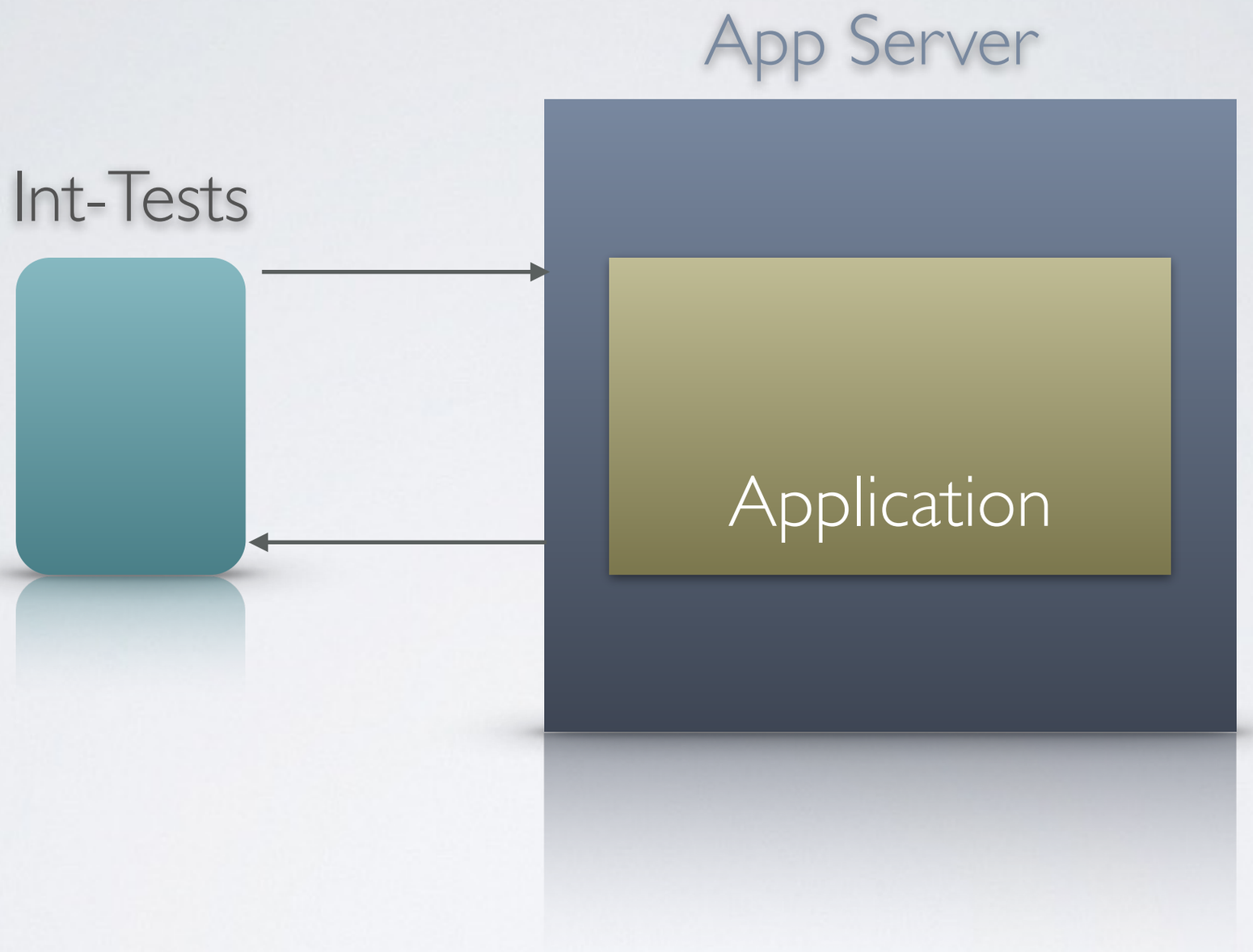
Schnell

Code Einheiten

Mock



INTEGRATION TESTS


App Server
Int-Tests
Application

- **Arquillian**
  - Deployment needed
- **Java EE embedded**
  - Configuration hell

There are only 10 types of Testing:

Unit Tests

Integration Tests

JPA Queries

Persistence Services

CDI Components

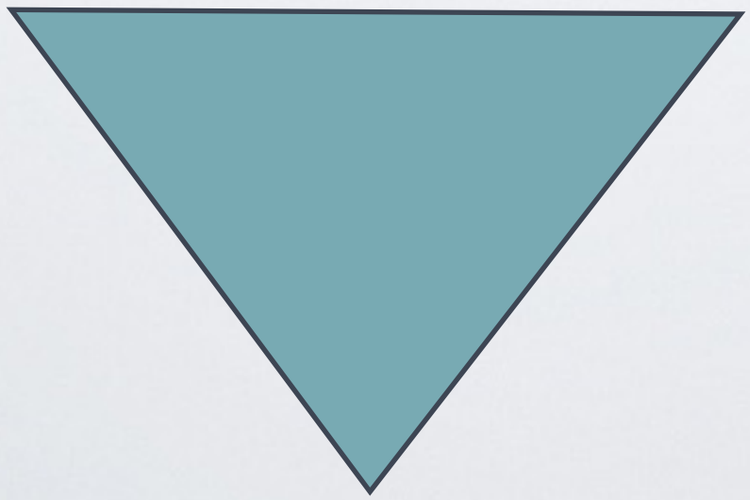
Bean Validation
Constraints

Distributed
Business Logic

Integration
Test ??

```java
@RunWith(Arquillian.class)
public class GamePersistenceTest {
    @Deployment
    public static Archive<?> createDeployment() {
        return ShrinkWrap.create(WebArchive.class, "test.war")
            .addPackage(Game.class.getPackage())
            .addAsResource("test-persistence.xml", "META-INF/persistence.xml")
            .addAsWebInfResource(EmptyAsset.INSTANCE, "beans.xml");
    }

    private static final String[] GAME_TITLES = {
        "Super Mario Brothers",
        "Mario Kart",
        "F-Zero"
    };

    @PersistenceContext
    EntityManager em;

    @Inject
    UserTransaction utx;

    // tests go here
}
```

BEAN TESTING



großer
Umfang

Integration
Tests

Bean
Tests

langsames
Feedback

schnelles
Feedback

Unit
Tests

geringer
Umfang

★ New in Java EE 6 (2009)

★ Origin JBoss Seam (Gavin King)

★ You can inject everything

EJB

Transactional	Interceptors
Security	Interceptors
Pooled	Scoped
Remote	?

Application Server

Proxies

AOP
(Interceptors)


Interceptor
Inject
Java
Dependency
Type-safe
CDI
standard
context
Produce
Qualifier
extensible
Enterprise
Decorator
portable
Alternative


Weld


OpenWebBeans


resin®

http://cdi-spec.org/

Interceptors

Decorators

CDI Extensions

Dependency Injection

CDI Events

Producer Methods

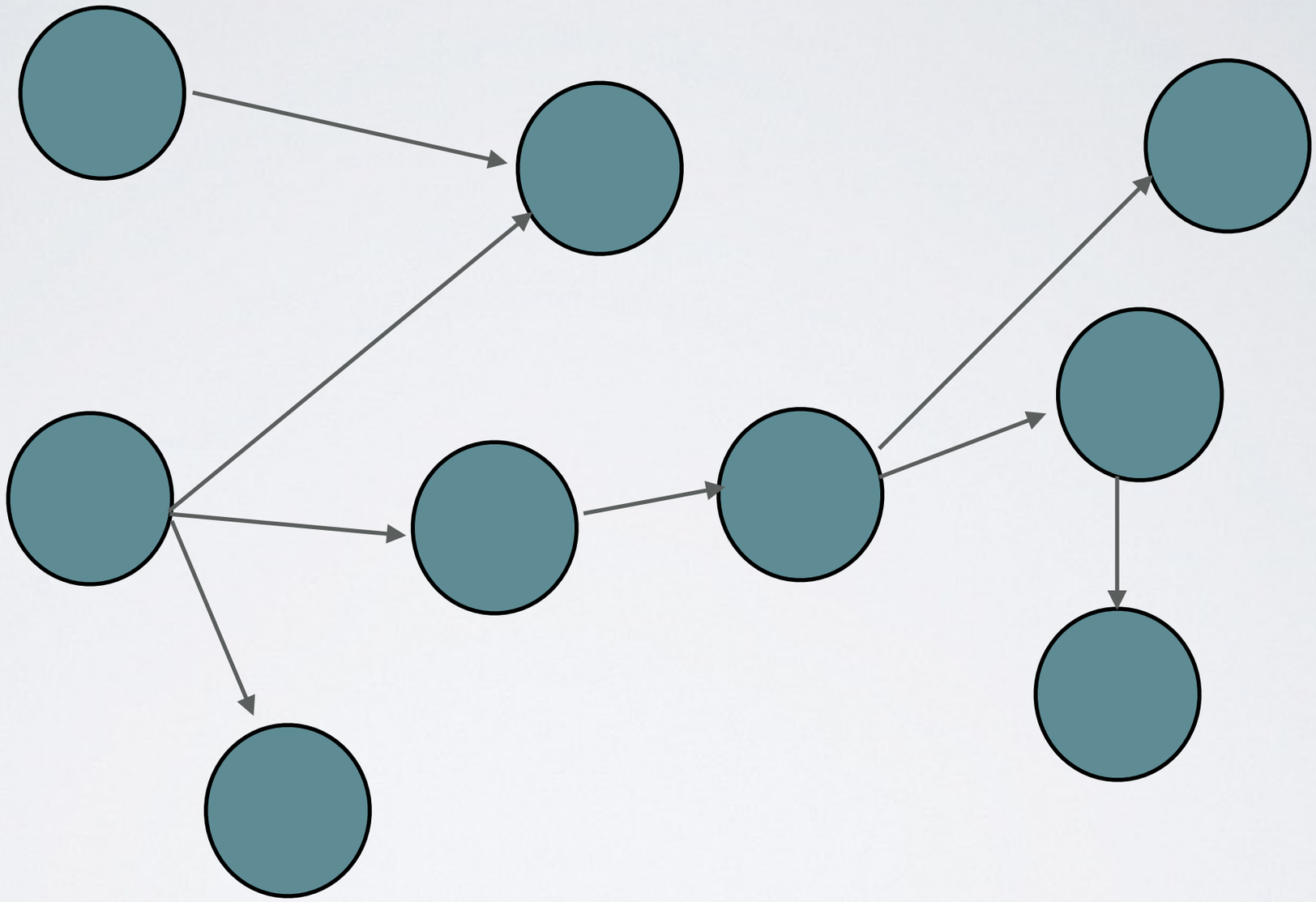
CDI Extensions

```
@Stateless
public class MyService{}
```

```
@RequestScoped
@Transactional
public class MyService{}
```

```
@EJB
private MyService myService;
```

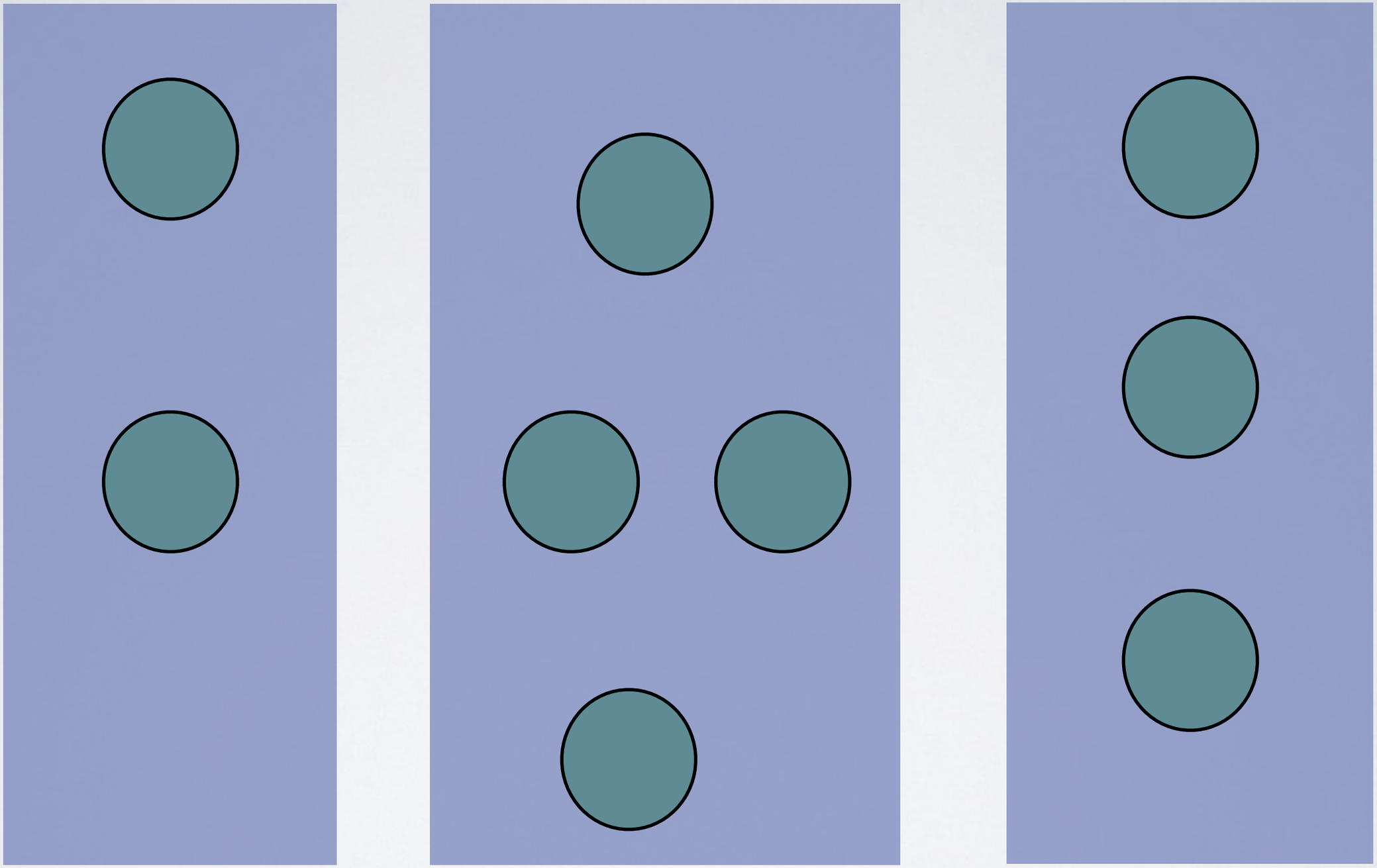
```
@Inject
private MyService myService;
```



Services (Façade)

Business Functions

Persistence Services



Backing Beans (JSF)

Services (Façade)

Business Functions

Persistence Services

SubcutaneousTest

Martin Fowler
14 February 2011

I use *subcutaneous test* to mean a test that operates just under the UI of an application. This is particulary valuable when doing functional testing of an application: when you want to test end-to-end behavior, but it's difficult to test through the UI itself.

Simulierter EJB Container in Java SE

Manipulation von Bean Metadata

Zukunftssicher (Java EE 8 and beyond)

Wie sieht ein Bean-Test aus?

```
@Test
public void shouldFindCustomer() {

    CustomerPersistenceService customerPs = getBean(CustomerPersistenceService.class);

    assertThat(customerPs.findAll(), is(empty()));

    Customer customer = new Customer("John", "123456", CustomerStatus.NORMAL);

    customer = customerPs.save(customer);
    assertThat(customer.getId(), not(0L));

    Customer customerFromDb = customerPs.find(customer.getId());

    assertThat(customerFromDb, is(notNullValue()));
    assertThat(customer.getId(), is(customerFromDb.getId()));
    assertThat(customer.getName(), is(customerFromDb.getName()));
    assertThat(customer.getCustomerId(), is(customerFromDb.getCustomerId()));

}
```

Was kann ich mit Bean-Test testen?

Was ist verfügbar?	Was kann man testen?
Dependency Injection	Dependencies (@EJB)
JPA Runtime	Queries, Beziehungen, Constraints
Interceptors & Decorators	Security
CDI Events, CDI Producers	CDI Events, CDI Producers (@Resource, @PersistenceContext)

Was kann ich mit Bean-Test testen?

BPM Engine
Integration
(Camunda, Activiti)

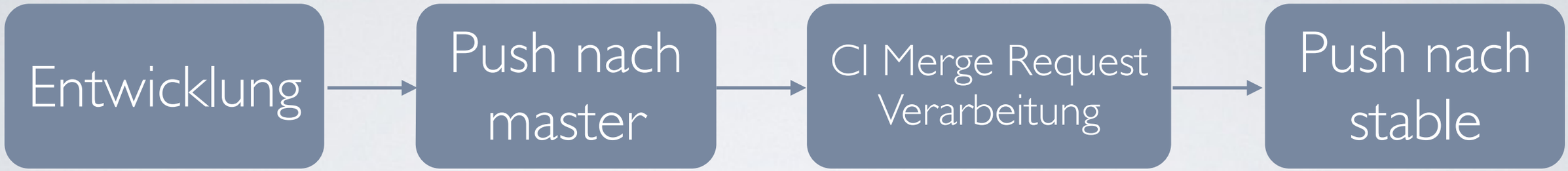
3rd Party Libraries
Integration

JSF Backing Beans

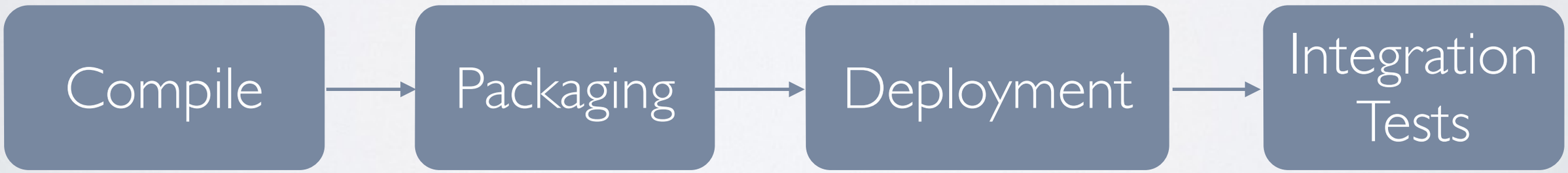
Wo kann ich Bean-Test einsetzen?

ENTWICKLUNGSPROZESS

CI Merge Request

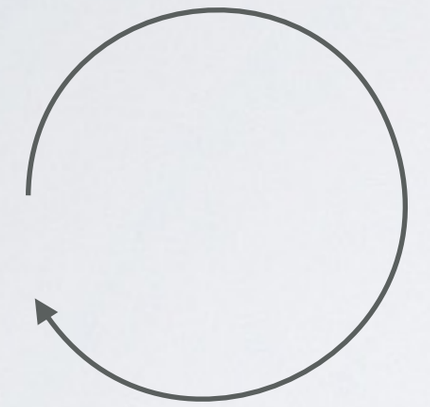


CI Integrationstest



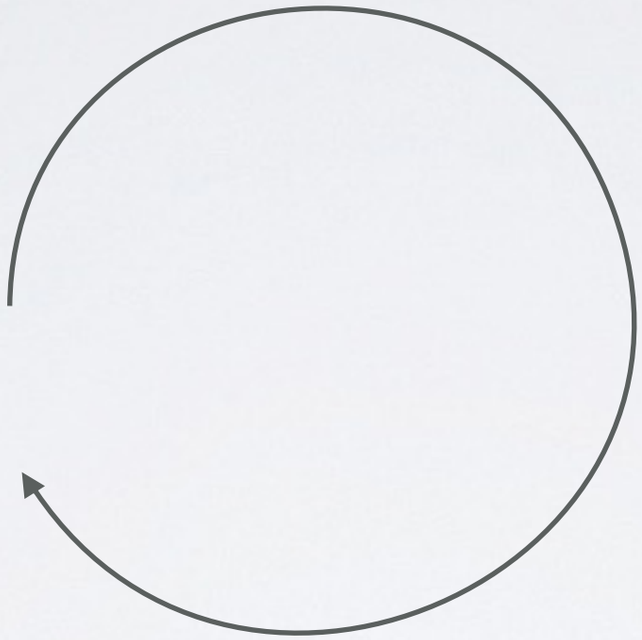
FEEDBACK

Unit Tests



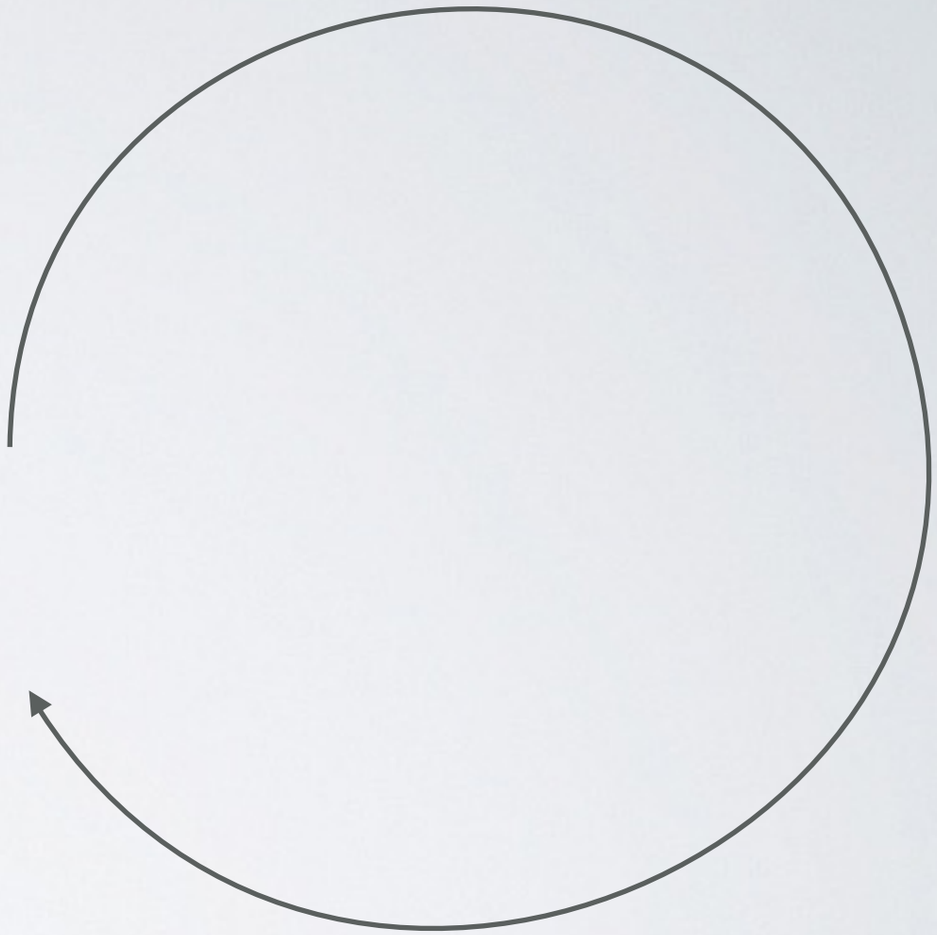
milliseconds

Bean Tests

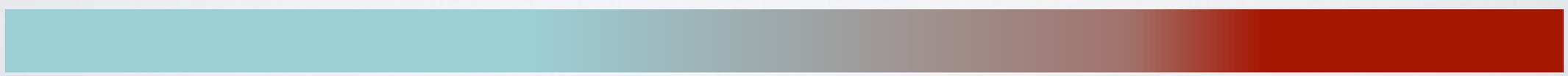


seconds

Integration Tests



(several)
minutes



BEAN TESTING

Hohe Geschwindigkeit (Feedback) mit großem Umfang

DRY – Alles ist schon da (JPA, EJB, CDI, Interceptors, TestNG)

nichtinvasive

TDD ist möglich

Unabhängig von Application Server

Verbesserte Qualität

Zeit und Geld (und Nerven) sparen.

Wie kann ich Bean-Test in meinem Projekt einsetzen ?

1

```
<dependency>
	<groupId>info.novatec</groupId>
	<artifactId>bean-test</artifactId>
	<version>0.1</version>
	<scope>test</scope>
</dependency>

<repository>
	<id>Novatec</id>
	<name>Novatec Repository</name>
	<url>http://repository.novatec-gmbh.de/content/repositories/novatec</url>
</repository>
```

2 Persistence-Unit “beanTestPU” in `src/test/resources/META-INF`

3 Empty beans.xml in `src/test/resources/META-INF`

GitHub

https://github.com/NovaTecConsulting/BeanTest

README.md

Bean Testing for Java EE Applications using CDI

This project attempts to show an interesting approach on testing Java EE Applications. It uses a CDI Container to resolve dependencies like EJBs or Resources when running unit test in a standard environment.

The name "Bean Testing" is used, since it isn't about proper unit tests. However, the feedback speed is very close to unit test and the tests look undistinguishable too.

Main advantages:

- Very fast test feedback (very close to unit test feedback speed).
- Dependencies are solved automatically without the need of a JEE Application Server (or Embedded Server).
- Everything is CDI so you can easliy extend the functionality.
- You get basic transaction propagation support.
- You can provide your own mocks to test external dependencies.
- You use the usual stuff for configuration: persistence.xml, beans.xml, Junit, etc.

JavaEE 8 ?

https://java.net/jira/browse/JAVAEE_SPEC-35

javaee-spec

Provide a CDI-Embedded Container to faciliate testing

Created: 30/Jan/14 04:13 PM Updated: 31/Jan/14 12:08 PM

Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Time Tracking:	Not Specified

Tags:	testing cdi
Participants:	Bill Shannon, cbarragan, ldemichiel and reza_rahman

Description « Hide

It would be quite easy to test JEE Applications using a CDI container that runs outside the Application Server. Weld already provides a CDI Container for Java SE (Weld-SE). I have been using this approach to test JEE Applications in several customer projects and the feedback has been good.

Please see this post for detailed information about this approach:
http://blog.novatec-gmbh.de/unit-testing-jee-applications-cdi/

The source code can be found on:
https://github.com/NovaTecConsulting/BeanTest

The example described in the post uses standard CDI features to make it possible to test JEE applications with (almost) all its dependencies resolved by the CDI Container. It runs like a normal Unit test, which means that the tests need milliseconds to run. It uses local transactions which it is ok for testing purposes. It also uses CDI extensions to "convert" EJBs into normal CDI Beans, so it aligns very well in the direction where JEE8 is going, namely, CDI instead of EJBs.

I think this feature will help a lot with testing. Besides, the approach uses other well established specs like JPA. I think that the technical aspects for this feature are already solved. The specs and infrastructure needed are already out there.

I know there is an Embedded Container already but it is very hard (almost impossible) to properly configure it in large projects.

Arquillian is a very good framework but I think it is used more for integration tests (deployment is still required in most cases).

Danke!