2.– 5. September 2013
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Tut er's oder tut er's nicht

Wie ich Web-Apps richtig Testen kann

Stephan Müller

open knowledge GmbH

Who am I ...?

Enterprise Architect

- open knowledge GmbH
- Architekt / Projektleiter / Coach
- Autor & Konferenzspeaker

- Java EE (vor allem JSF)
- Clean Code & Testing
- Cloud Computing


WRONG
RIGHT
RIGHT

Warum testen?

„Wer testet, ist feige!“

„Ich weiß doch, dass mein Code funktioniert!“

„Wenn ich den Test selber schreibe, hat es eh keinen Sinn!“

Bekannte Testszenarien

eXtreme Clicking

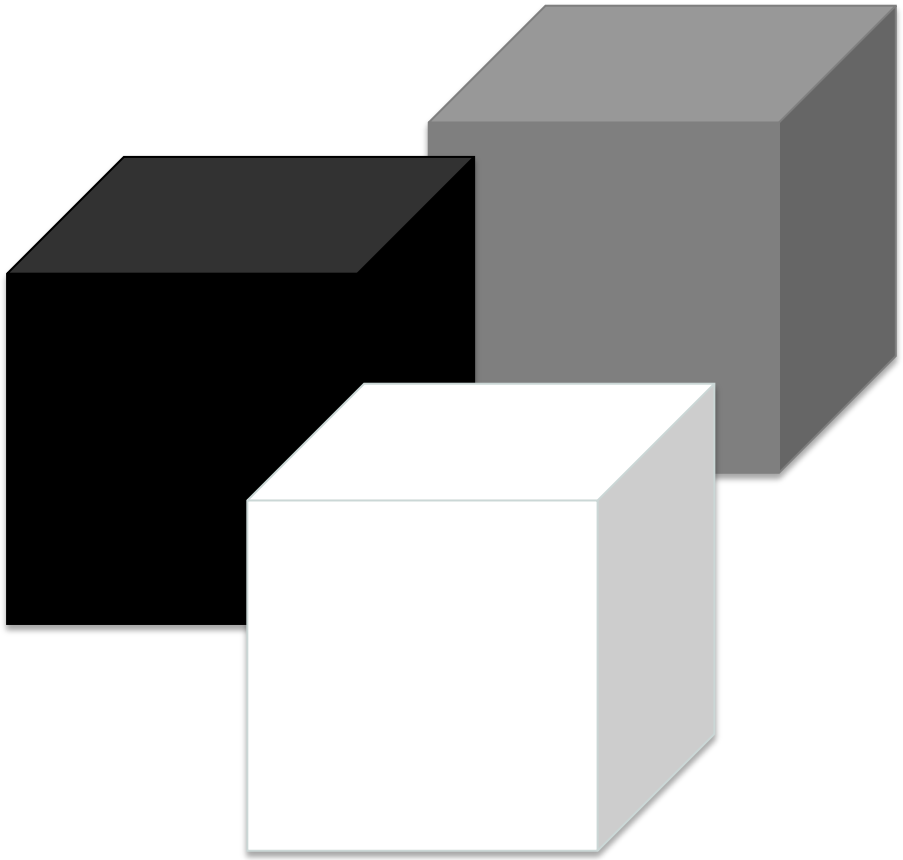
Bekannte Testszenarien

nur auf dem Entwicklerrechner

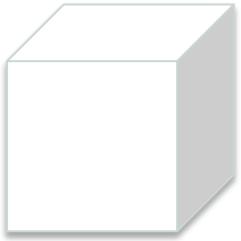
Bekannte Testszenarien

Kunden

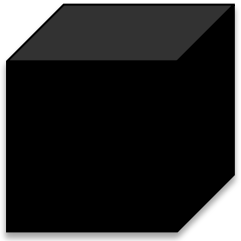
Ein paar Grundlagen ...



Ein paar Grundlagen ...

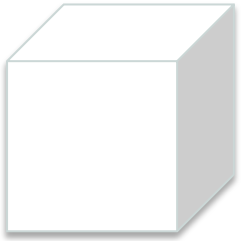


White-Box-Tests

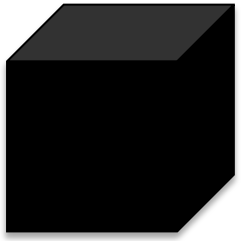


Black-Box-Tests

Ein paar Grundlagen ...

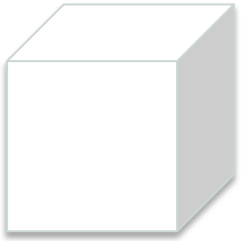


Tests mit Kenntnissen über die innere Funktionsweise

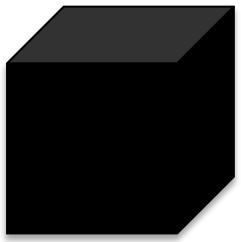


Tests ohne Kenntnisse über die innere Funktionsweise

Ein paar Grundlagen ...

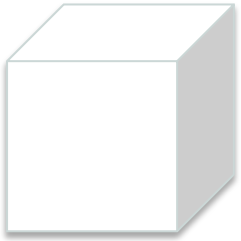


Sollen die korrekte Umsetzung der Implementierung sicherstellen

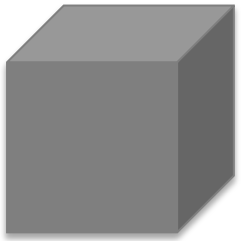


Sollen die korrekte Umsetzung der Anforderung sicherstellen, aber nicht die Implementierung

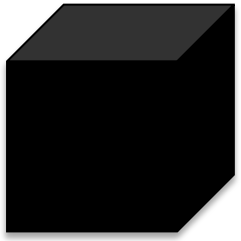
Ein paar Grundlagen ...



White-Box-Tests

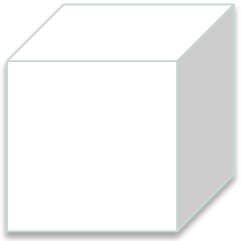


Grey-Box-Tests (Acryl-Box-Tests)

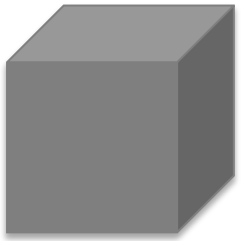


Black-Box-Tests

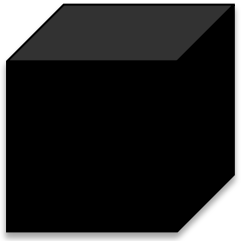
Ein paar Frameworks ...



JUnit, Mockito, DBUnit



Arquillian, Drone, Warp, Selenium



Selenium, JMeter

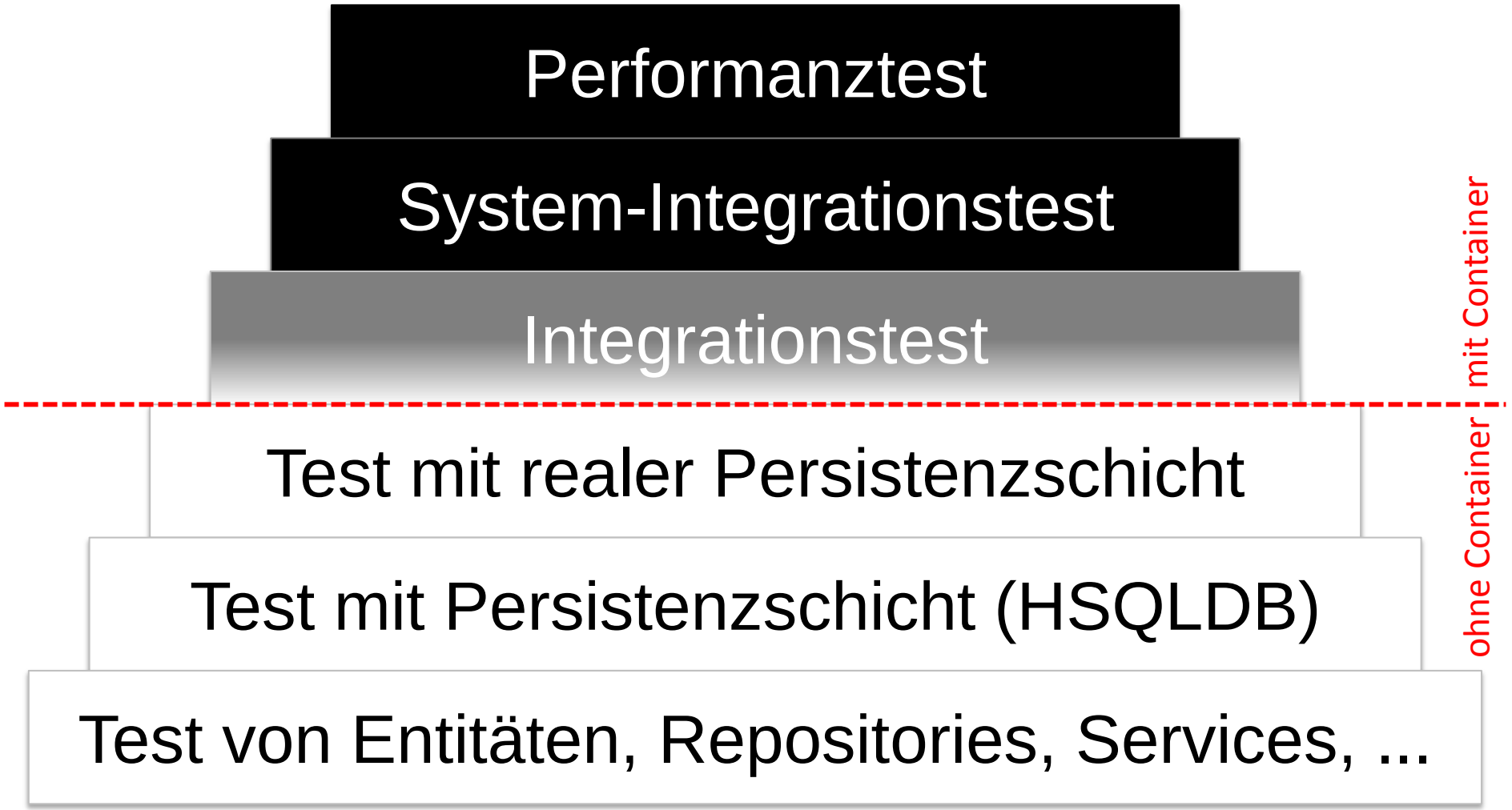
Ein paar Anforderungen ...

Was man beachten sollte ...

- Auswahl angemessenen Testwerkzeuge
- Festlegung was wie getestet werden soll
- Ausführbarkeit aus der IDE heraus
- Integration der Tests in den Buildprozess
- Regressionsfähigkeit der Tests
- realitätsnahe Testdaten


WRONG
RIGHT
RIGHT

Sechs Kategorien ...



Sechs Kategorien ...

Nicht alle Tests sind gleich

- Anzahl
- Ausführungszeit
- Laufzeitumgebung (IDE / Testserver)
- ohne / mit Container


Kann ich
die Tests nicht
unterscheiden?

JUnit Categories

```
public interface DevelopmentTest { /* category marker */ }
public interface IntegrationTest { /* category marker */ }

@Category( DevelopmentTest.class )
public class ExampleTest {

  @Test
  public void function() { ... }

  @Test(expected = NullPointerException.class )
  public void functionWithException() { ... }
}
```

https://github.com/junit-team/junit/wiki/Categories

JUnit Categories

```
public interface DevelopmentTest { /* category marker */ }
public interface IntegrationTest { /* category marker */ }

@RunWith( Categroies.class )
@IncludeCategory( DevelopmentTest.class )
@SuiteClasses( { AllCoreTests.class } )
public class AllDevelopmentTests {}

@RunWith( Categroies.class )
@IncludeCategory( IntegrationTest.class )
@SuiteClasses( { AllCoreTests.class } )
public class AllIntegrationTests {}
```

https://github.com/junit-team/junit/wiki/Categories

JUnit Categories

Maven Konfiguration

```
<build>
 <plugins>
  <plugin>
   <artifactId>maven-surefire-plugin</artifactId>
   <configuration>
    <groups>de.openknowledge.archetype.core.test.DevelopmentTests</groups>
   </configuration>
  </plugin>
 </plugins>
</build>
```

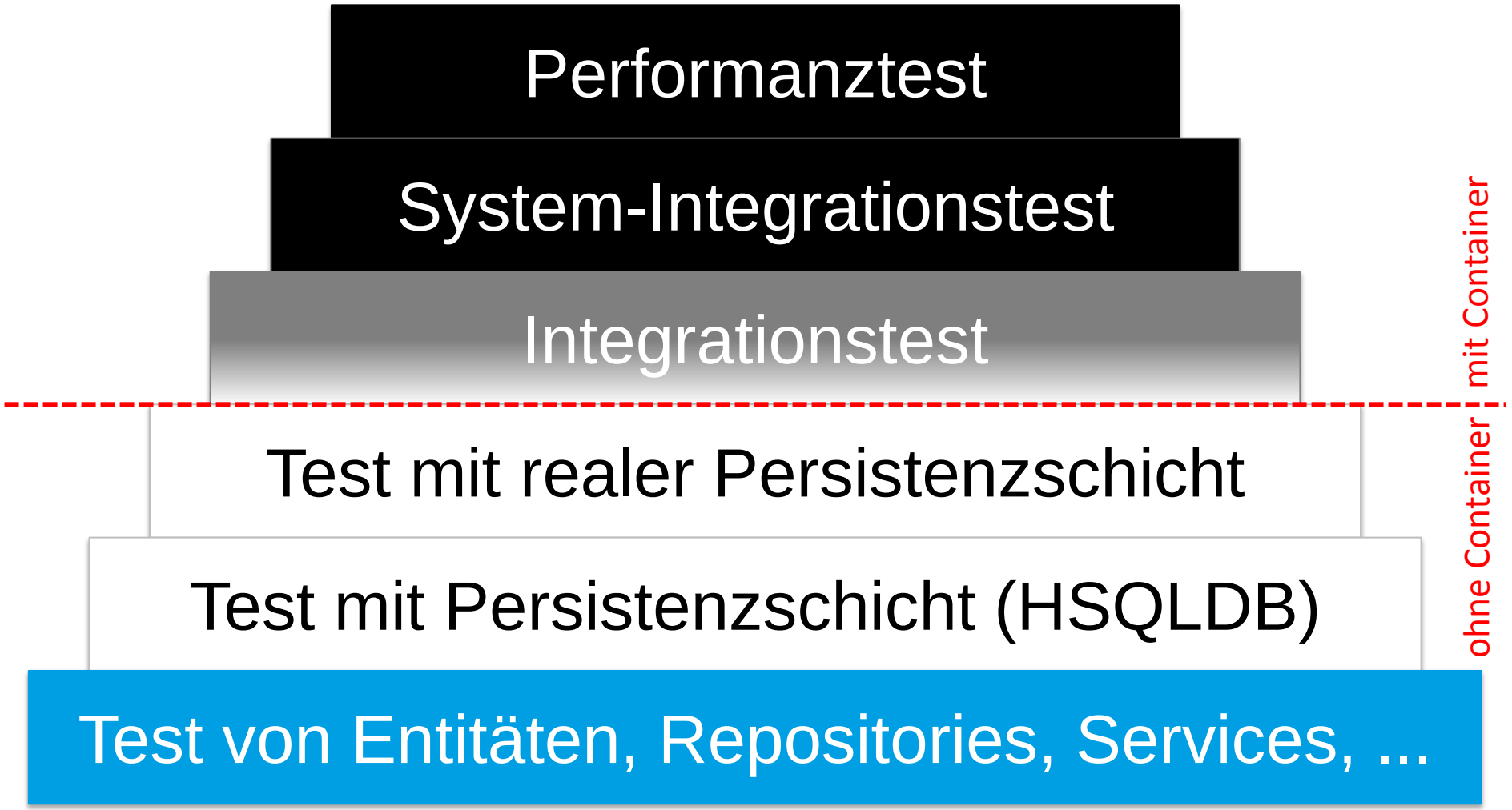
JUnit Rules

Name der Testklasse & Methode

```
public class ExampleTest {

  private static Logger LOGGER = ...;

  @Rule
  public TestName name = new TestName();

  @Test
  public void function() {
    LOGGER.debug( name.getMethodName() );
  }
}
```

https://github.com/junit-team/junit/wiki/Rules

Kategorie 1: @DevelopmentTest



Kategorie 1: @DevelopmentTest

Entwicklertest

- Klassischer Unit-Test
- Assertions
  - assertEquals, assertThat, ...
  - Hamcrest API (is, not, nullValue, startsWith, ...)
- Mock-Objekte via Mockito
  - mock, when, verify, spy
  - @Mock (benötigt MockitoJUnitRunner)

Kategorie 1: @DevelopmentTest

Mocks

```
@Category( DevelopmentTests.class )
public class RoleTest {

  private User user;

  @Before
  public void setUp()
      throws Exception {

    user = Mockito.mock( user );
    Mockito.when( user.getName ).thenReturn( ... );
  }
```

Kategorie 1: @DevelopmentTest

Mocks

```
@RunWith( MockitoJUnitRunner.class )
@Category( DevelopmentTests.class )
public class RoleTest {

  @Mock
  private User user;

  @Test
  public void function() {

    Mockito.when( user.getName ).thenReturn( ... );
  }
```

Kategorie 1: @DevelopmentTest

Problem: Dependency Injection (Field Injection)

```
@Model
public class UserBean {

   @Inject
   private UserService service;

   public User create(User user) {

      ...
   }
}
```

Kategorie 1: @DevelopmentTest

Lösung: Dependency Injection (Konstruktor Injection)

```
public class UserBean {

   private UserService service;

   protected UserBean() {}

   @Inject
   public UserBean(UserService service) {

      ...
   }
}
```

CDI Proxy Constructor

CDI Test Constructor

Kategorie 1: @DevelopmentTest

Lösung: Dependency Injection (Setter Injection)

```
public class UserBean {

   private UserService service;

   protected UserBean() {}

   @Inject
   void setUserService(UserService service) {

      ...
   }
}
```

Kategorie 1: @DevelopmentTest

Problem: Statics

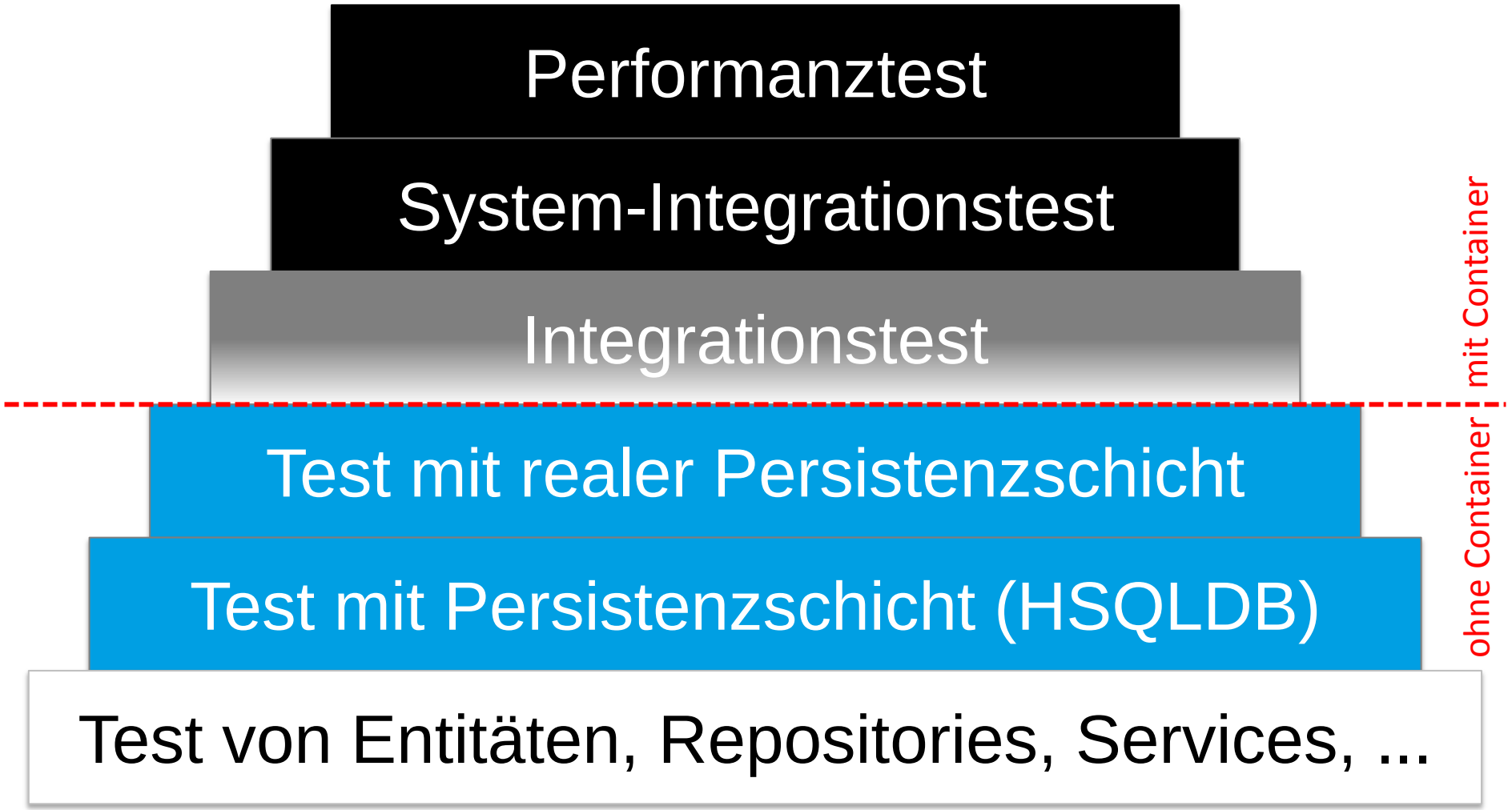
```
@Model
public class UserBean {

   private UserService service;

   ...

   public User create(User user) {

      FacesContext ctx = FacesContext.getCurrentInstance();
      ...
   }
}
```

Kategorie 1: @DevelopmentTest

Lösung: Statics vermeiden oder abstrahieren

```
@Model
public class UserBean {

  private UserService service;

  private FacesContextProducer producer;

  public User create(User user) {

    FacesContext ctx = producer.getContext();
    ...
  }
```

Kategorie 2 & 3: @PersistenceTest



Kategorie 2 & 3: @PersistenceTest

Persistenztest

- Klassischer Unit-Test mit EntityManager
- In-Memory-DB & RDBMS
- DBUnit

- JPA-Mapping
- Repositories
- Criteria API & JPQL Queries

Kategorie 2 & 3: @PersistenceTest

In-Memory DB & RDBMS

- Persistence Unit dynamisch auswählen
  - Mehrere Persistence Units in persistence.xml
  - Name wird bei der Testausführung übergeben
- Vorteile
  - Performanz auf Entwicklerrechner
  - Tests auf unterschiedlichen Datenbanken möglich
  - DB-spezifische Probleme können erkannt werden

Kategorie 2 & 3: @PersistenceTest

DBUnit

- stellt Testzustand her
- prüft Endzustand mit Erwartungswert
- CSV, XML, YAML

Kategorie 2 & 3: @PersistenceTest

Problem: EntityManager aus PersistenceContext

```
@Repository
public class UserRepository {

   @PersistenceContext
   private EntityManager entityManager;

   public User create(User user) {

      ...
   }
}
```

Kategorie 2 & 3: @PersistenceTest

Lösung: Konstruktor Injection

```
@Repository
public class UserRepository {

   private EntityManager entityManager;

   protected UserRespository() {}

   @Inject
   public UserRespository(EntityManager entityManager) {
      ...
   }
}
```

CDI Proxy Constructor

CDI Test Constructor

Kategorie 2 & 3: @PersistenceTest

```
public class AbstractJpaTest {

  private static EntityManagerFactory emf;

  protected EntityManager entityManager;

  @BeforeClass
  public static void createEntityManagerFactory() {
    emf = Persistence.createEntityManagerFactory( ... );
  }

  @Before
  public void createEntityManager() {
    entityManager = emf.createEntityManager();
  }
```

Kategorie 2 & 3: @PersistenceTest

```
@Category( PersistenceTest.class )
public class UserRepositoryTest
    extends AbstractJpaTest {

  private UserRepository repository;

  @Before
  public void setUp()
    throws Exception {

    repository = new UserRepository( entityManager );
  }

  ...
}
```

Kategorie 2 & 3: @PersistenceTest

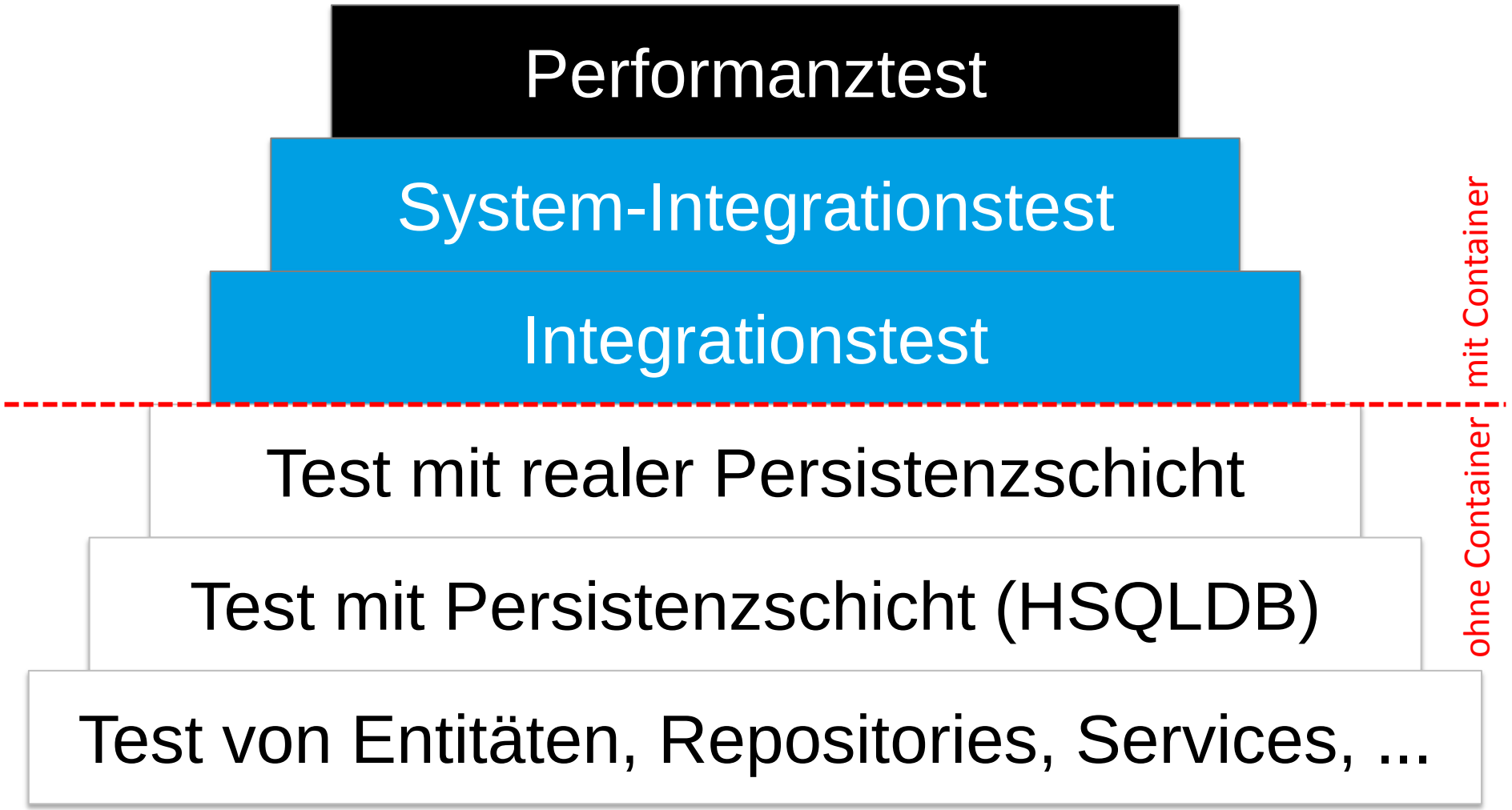
```
@Category( PersistenceTest.class )
public class UserRepositoryTest
    extends AbstractJpaTest {

  @Test
  public void create() {

    User user = ...
    ...
    entityManager.getTransaction().begin();
    user = repository.create(user);
    entityManager.getTransaction().commit();
    User foundUser = repository.find(user.getId());
    assertNotNull( foundUser );
}
```

Kategorie 3: @PersistenceTest

Was man beachten sollte ...

- Auswahl angemessenen Testwerkzeuge
- Festlegung was wie getestet werden soll
- Ausführbarkeit aus der IDE heraus
- Integration der Tests in den Buildprozess
- Regressionsfähigkeit der Tests
- realitätsnahe Testdaten

Kategorie 4 & 5: @IntegrationTest



Kategorie 4: @IntegrationTest

Integrationstest

- Test in Laufzeitumgebung
- Testausführung in einem (Embedded) Container
  - Glassfish, JBoss, Tomcat, WebSphere, ...
  - Apache DeltaSpike
- Arquillian, Drone & Warp
- Selenium

Kategorie 4: @IntegrationTest

Integrationstest (White-Box-Test)

- Test von CDI Beans & Interceptoren
- Testausführung in einem (Embedded) Container

- Arquillian

Kategorie 4: @IntegrationTest

Integrationstest (White-Box-Test)

```
@RunWith( Arquillian.class )
@Category( IntegrationTests.class )
public class LoggerProducerTest {

  @Deployment
  public static Archive< ? > createDeployment() {
    return ShrinkWrap.create( JavaArchive.class )
      .addClass( LoggerProducer.class )
      .addAsManifestResource( INSTANCE, "beans.xml" );
  }
  ...
}
```

Kategorie 4: @IntegrationTest

Integrationstest (White-Box-Test)

```
@RunWith( Arquillian.class )
@Category( IntegrationTests.class )
public class LoggerProducerTest {

  @Inject
  private Logger logger;

  @Test
  public void injectedLogger() {

    assertNotNull( logger );
  }
}
```

Kategorie 4: @IntegrationTest

Integrationstest (White-Box-Test)

- Test von CDI Beans & Interceptoren
- Testausführung in einem (Embedded) Container

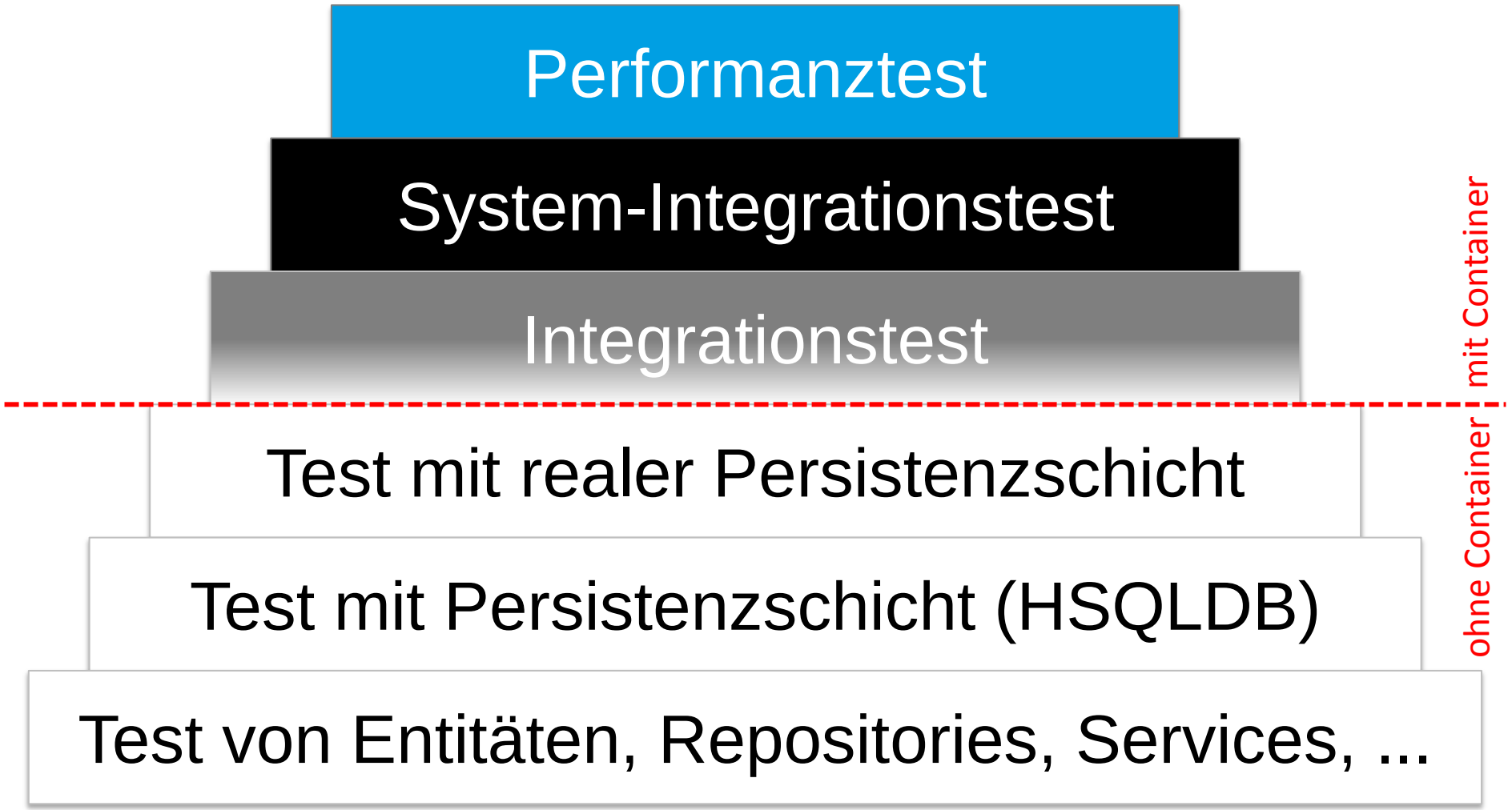
- Arquillian

Kategorie 5: @IntegrationTest

System-Integrationstest

- Testet mehrere Seiten im Verbund
  - „Schön-Wetter-Flug“
  - Fokus liegt auf vollständiger Pfadabdeckung
- Selenium
  - Kombination mit DBUnit für Setup & Clean-Up

Kategorie 6: @PerformanceTest



Kategorie 6: @PerformanceTest

Performancetest

- Simulation von Lastspitzen
- Nachweis nicht-funktionaler Anforderungen
  - Antwortzeiten
  - parallele Benutzer
- JMeter

Kategorie 6: @PerformanceTest

JMeter

- unterstützt verschieden Protokolle
  - HTTP/HTTPS, FTP, JDBC, LDAP, SOAP/XML-RCP
- ermöglicht Erstellung von Testplänen
- Maven-Integration über 3rd Party Plugin

Kategorie 6: @PerformanceTest

Problem: Infrastruktur

- erfordert produktionsnahe Testumgebung
  - Server, Storage, Host, ...
- häufig nur Kompromisslösungen

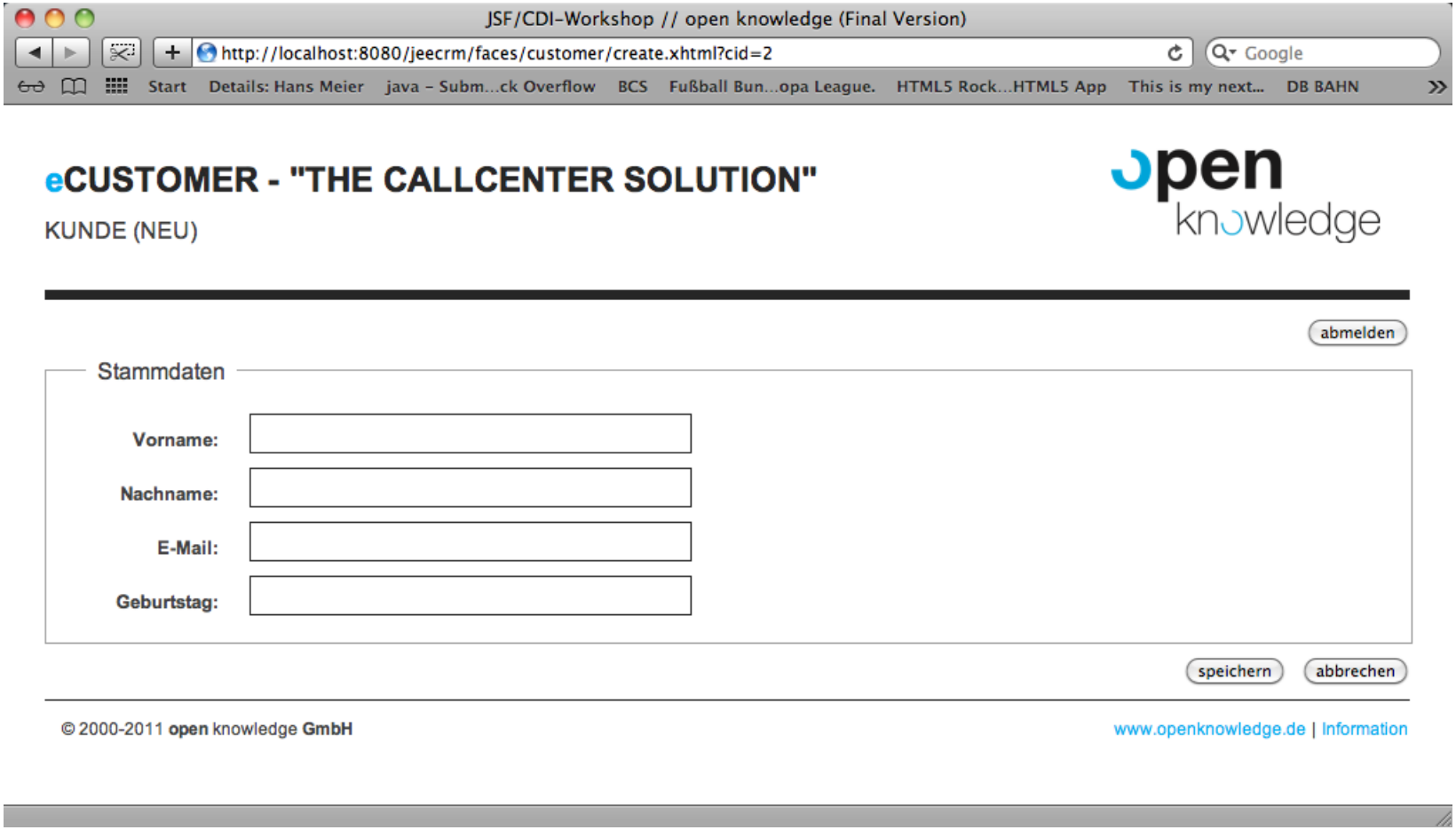
Kategorie 6: @PerformanceTest

Was man beachten sollte ...

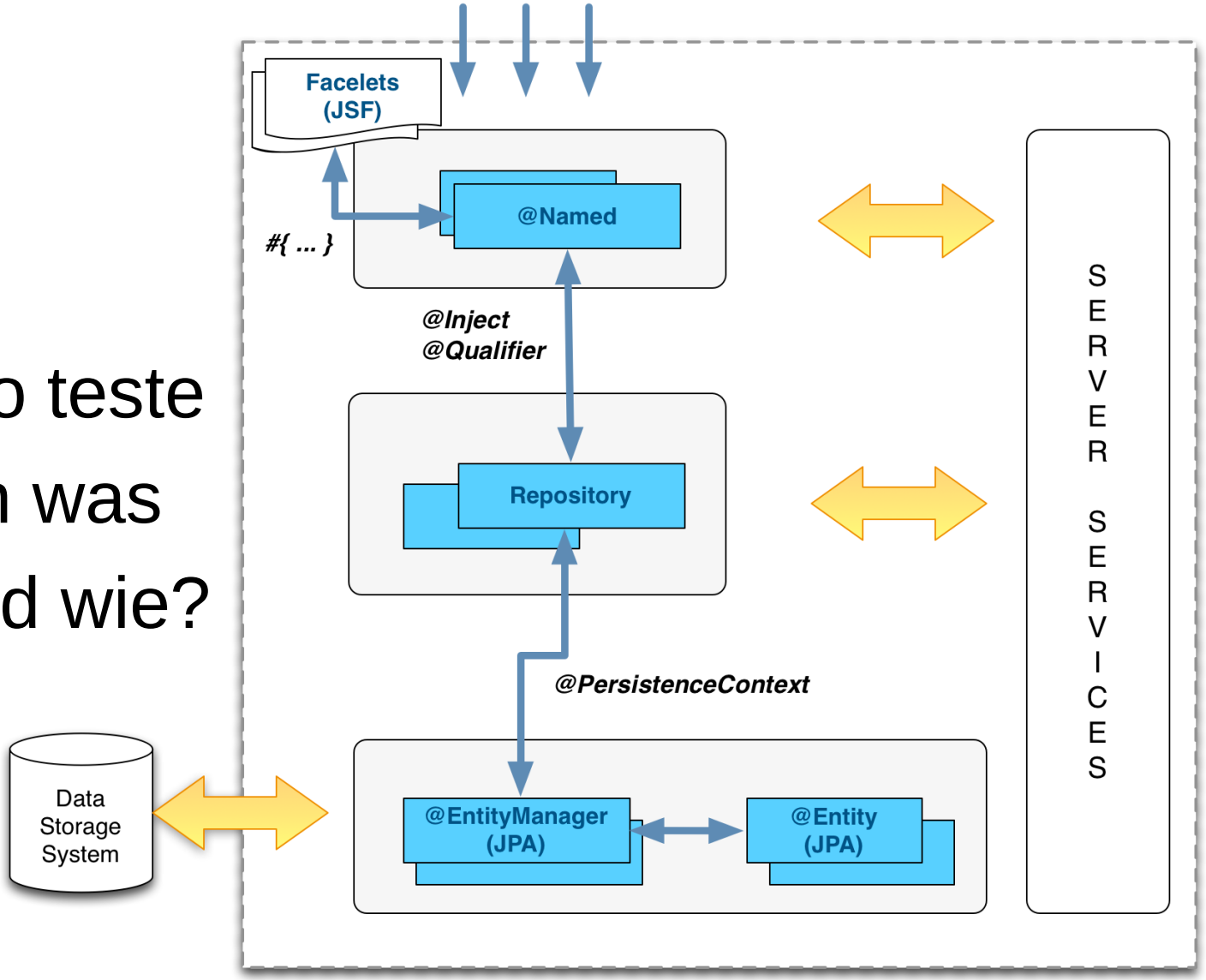
- Auswahl angemessenen Testwerkzeuge
- Festlegung was wie getestet werden soll
- Ausführbarkeit aus der IDE heraus
- Integration der Tests in den Buildprozess
- Regressionsfähigkeit der Tests
- realitätsnahe Testdaten


WRONG
RIGHT
RIGHT

Beispielanwendung



Wo teste ich was und wie?



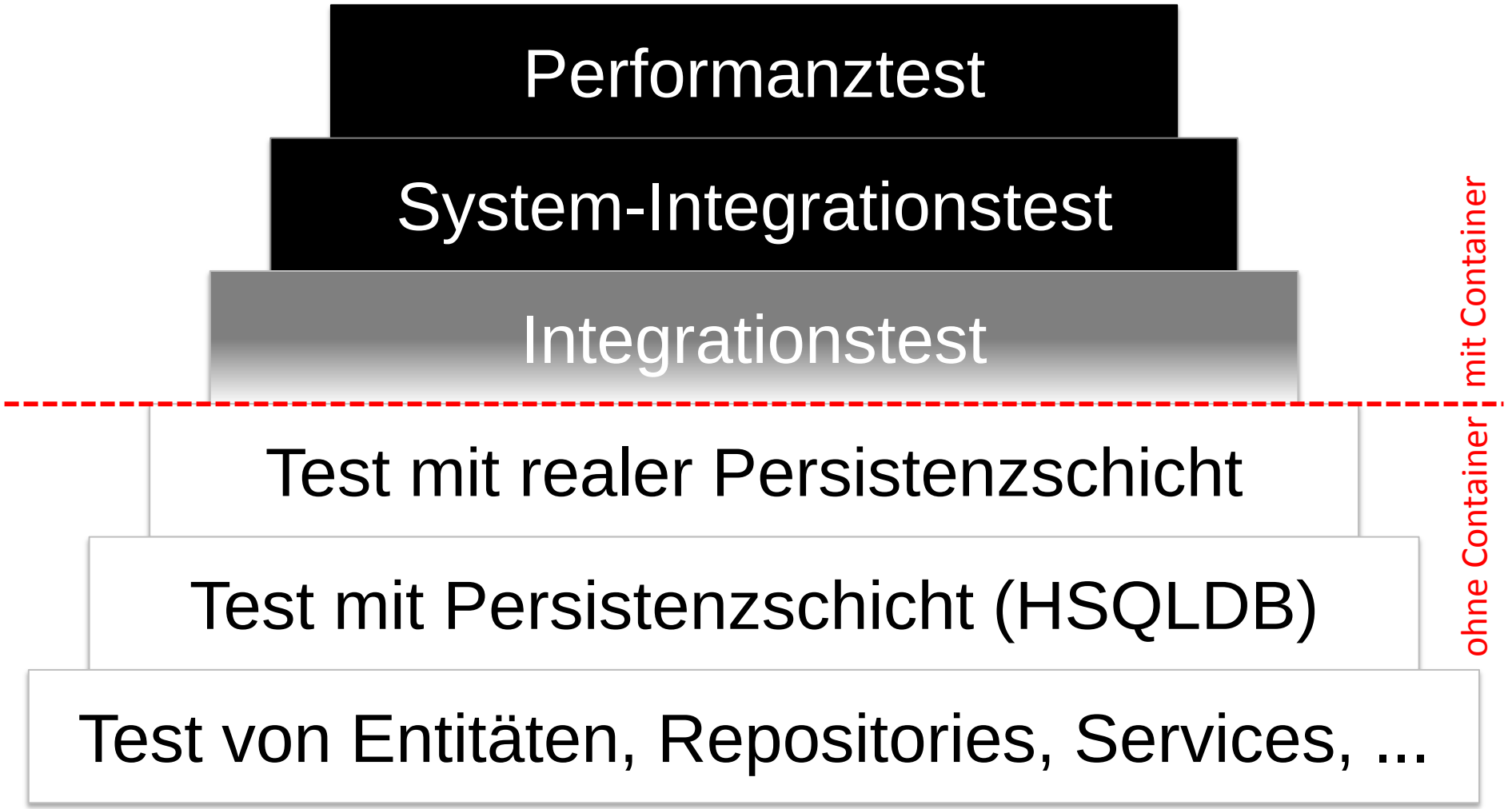
Projektstruktur

Application	Web-Schicht
Domain	Persistenz- & Businessschicht
Integration	Funktionalität ohne Domain Bezug

Wo teste ich was und wie?

Application	CDI Beans, JSF Components, Converter & Validator, ...
Domain	Entities, Value Objects, Repositories, JPA Mapping, ...
Integration	Services, Interceptors, Util, ...

Wo teste ich was und wie?



Wo teste ich was und wie?

Application	Kategorie 1, 4, 5 & 6
Domain	Kategorie 1, 2, 3 & 4
Integration	Kategorie 1 & 4

Ein paar Regeln zum Schluss ...

Clean-Code gilt auch für Tests

- für jede Klasse eine **eigene** Testklasse
- jede Methode testet nur eine Funktion
- verständliche Methodennamen
- getter & setter nur wenn **notwendig** testen

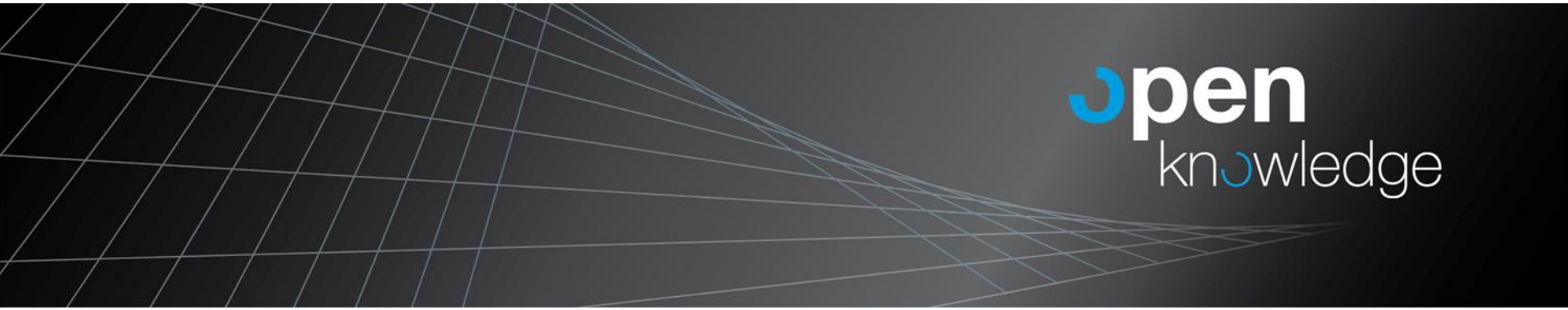
Gibt es noch Fragen?

Dann los ...

Vielen Dank!

Stephan Müller
open knowledge GmbH

KONTAKT



Bismarckstraße 13
26122 Oldenburg
Tel. +49 (0) 441 4082-0
Fax +49 (0) 441 4082-111
kontakt@openknowledge.de

www.openknowledge.de

@_openknowledge
@muellerst

facebook.de/openknowledge