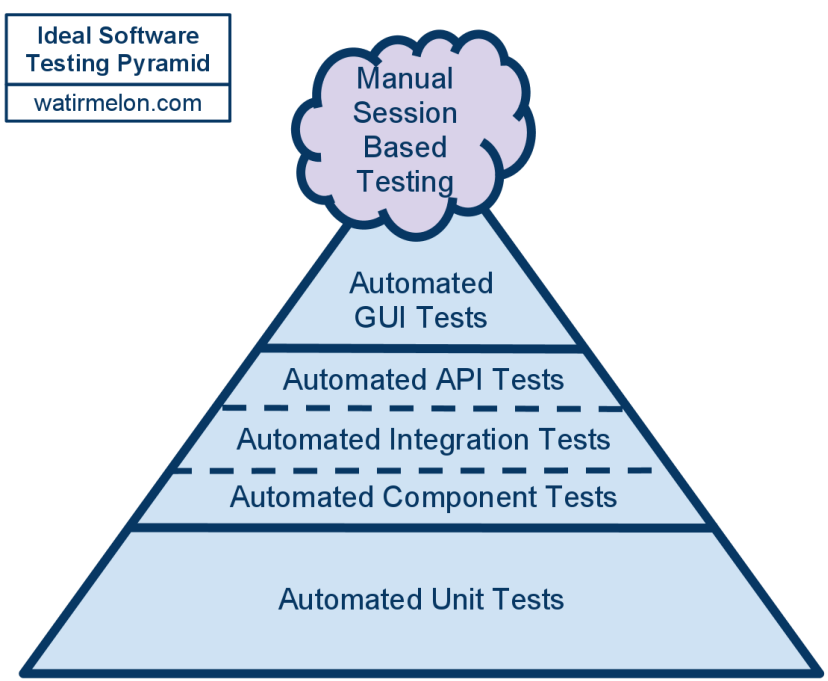
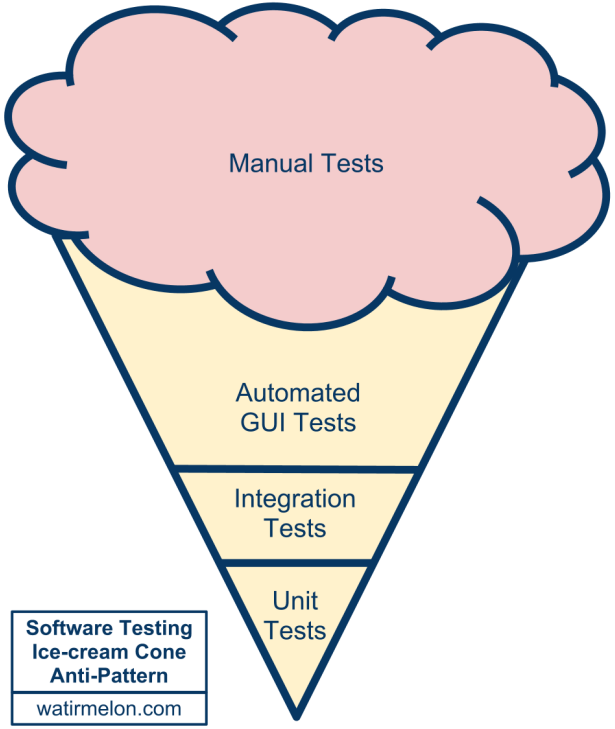
Referenten:
Joël Bader (bar), Thomas Kessler (tke)

Datum:
30.09.2015

Software Testing Ice Cream Cone

http://watirmelon.com/





Beispiel EJB

Was ist Arquillian

Testwerkzeug zum Entwickeln von Integrationstest für JavaEE-Anwendungen

Arquillian Features

- Kompatibel mit JUnit und TestNG
- Container Managed Resourcen in Test injecten
- “Skip the build”
- Micro Deployment
- Pass-by-reference zwischen Test und Container
- Test gegen alle Container ausführbar: z.B. JBoss, Glassfish, Jetty, Tomcat, WebSphere, Weld, OpenEJB, …

Demo

- Greeter

ShrinkWrap

- Erstellt dynamische Java Archive
  - JAR, WAR, EAR, ...
- Granularität
  - Einzelne Klassen
  - Packages
  - Dateien & Verzeichnisse
- Erstellt Deployment-Descriptors
- Demo

Testing Security

- Rollenbezogenes Testing ohne:
  - Echte security-domain
  - Login module
  - Konkrete Benutzer
- Delegating Bean mit @RunAs Annotation

Testing Persistence

- Testing von JPA
- Constraints
- Entity Validation
- Cascading
- Transactional Behaviour (commit, rollback)

-> Demo

Arquillian & Mocks

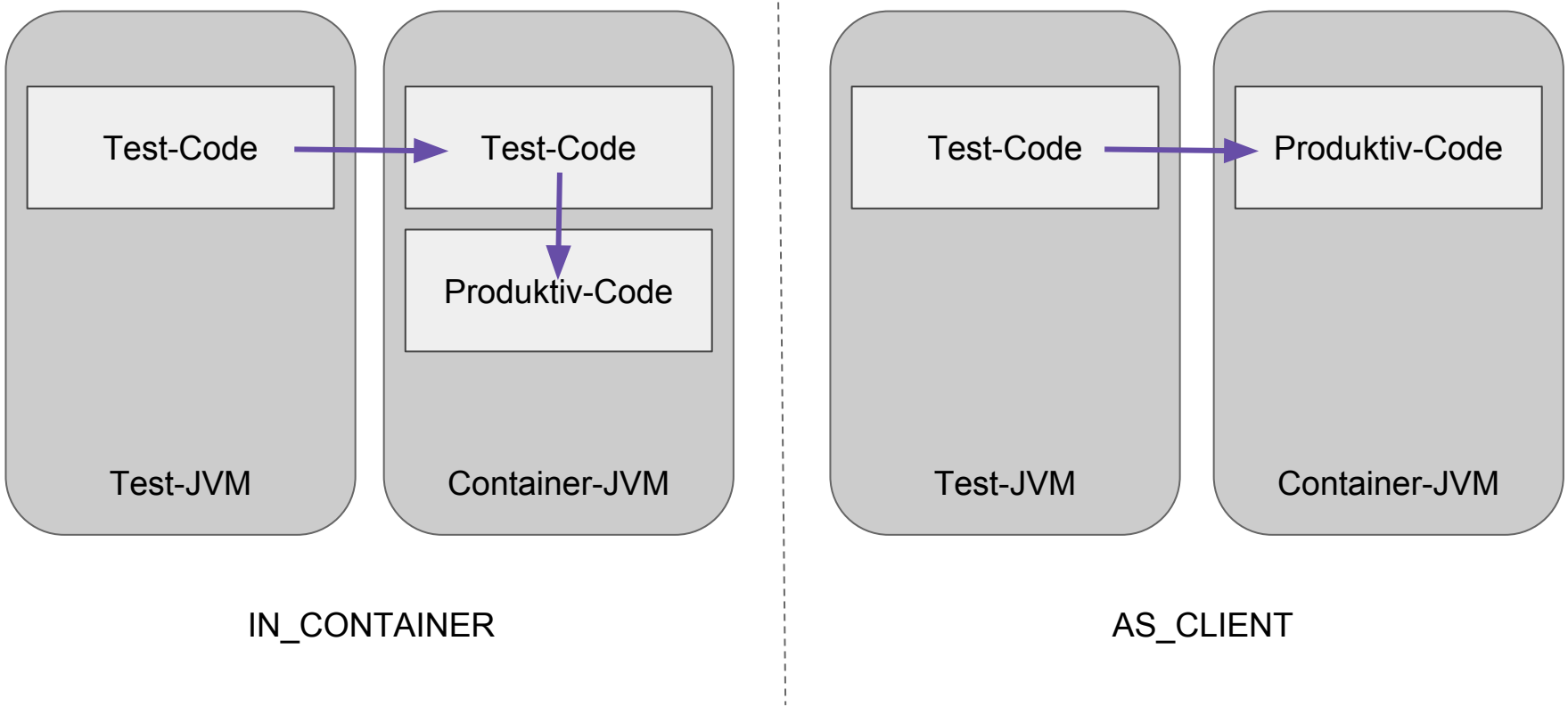
- Eigene Mocks/Stubs deployen
- Mocking Framework (z.B. Mockito)
- CDI Alternatives
- CDI Bean Specialization

-> Demo

Container

- Verschiedene Container-Adapter
  - Konfiguration über Maven Profile

Run Modes


Test-Code
Test-Code
Produktiv-Code
Test-JVM
Container-JVM
IN_CONTAINER
Test-Code
Produktiv-Code
Test-JVM
Container-JVM
AS_CLIENT

Debugging Arquillian Tests

- Embedded Container
  - Container und IDE laufen in derselben VM
  - Debugging funktioniert out-of-the-box
- Remote Container
  - Container und IDE in unterschiedlichen VMs
  - Remote Debugging erforderlich

Container Types

- Embedded
- Managed
- Remote

Container

- Embedded
  - Pro:
    - Keine Server-Installation notwendig
    - Debugging kein Problem
  - Con:
    - Gleiche JVM → Keine Classloader Isolation
    - Nicht jeder Container hat volle Funktionalität

Container

- Managed
  - Pro:
    - Server wird automatisch gestartet
    - Gleicher Server wie in Produktion
  - Con:
    - Stop/Start braucht Zeit
    - Debugging schwieriger

Container

- Remote
  - Pro:
    - Server Startup ausserhalb von Test-Laufzeit
    - Gleicher Server wie in Produktion
  - Con:
    - Server muss separat verwaltet werden (start/stop)
    - Debugging schwieriger

ShrinkWrap Resolver (Maven)

- Maven Artefacts zu Archiv hinzufügen
  - Inklusive transitive Abhängigkeiten
  - Excludes
  - Versionen aus pom.xml
  - Sämtliche runtime dependencies aus pom.xml

Arquillian Extensions

- Für Erweiterbarkeit ausgelegt
  - Persistence (verwendet DBUnit)
  - Transaction
  - Code Coverage (JaCoCo)
  - Drone & Warp (für Selenium)

Multiple Deployments / Containers

- Deployment mehrerer Applikationen
  - Testen der Kommunikation zwischen den Applikationen

Mögliche Probleme

- Arquillian eingesetzt wo gar nicht nötig
  - Tests werden langsamer
- Zu viele Artefakte in Arquillian-Deployment
  - Tests werden langsamer
  - Wartung wird komplexer
- Verwaltung/Konfiguration bei grossen Projekten kann schnell unübersichtlich werden
- Verschieben von Klassen/Packages kann Deployment brechen

Wann Arquillian?

- Was kann mit Arquillian getestet werden?
  - Security, Persistence, ClassLoader, Bean Lifecycle (z.B. @PostConstruct), Dependency Injection, Interceptors...
    + alles, was auch ausserhalb des Containers getestet werden könnte

- Was sollte NICHT mit Arquillian getestet werden?
  - gewöhnlicher Unit Test sollte immer bevorzugt werden (Arquillian = Overhead)
  - t(unitTest) < t(arquillianTest) < t(fullDeploymentTest)

Referenzen

- JavaMagazin: Einführung in Arquillian [pdf]
- Guides
- arquillian.org Blog
- Arquillian Reference Guide
- Github: kessltho/arquillian