e-tecture 

Erfahrungsbericht Test-Driven Development in einem Web-basierten J2EE Projekt

Sascha Appel
Sascha.Appel@e-tecture.com
069 - 67737 103

Agenda

- Vorstellung
- Projektbeschreibung
- Motivation und Erwartungshaltung bzgl. TDD
- Problemstellungen und Lösungen
- Fazit
- Links

Agenda

- **Vorstellung**
- **Projektbeschreibung**
- **Motivation und Erwartungshaltung bzgl. TDD**
- **Problemstellungen und Lösungen**
- **Fazit**
- **Links**

Das Unternehmen e-tecture

„e-tecture konzipiert und realisiert individuelle Software-Lösungen auf Basis von Internet-Technologien.“

- Architekturen für On- und Offline-Applikationen
- Realisierung von IT-Anwendungen im On- und Offline-Bereich
- Beratungs-, Gestaltungs-, Schulungs-, und Serviceaufgaben
- Vertrieb von IT-Standardprodukten sowie Individual-Software

Agenda

Vorstellung

Projektbeschreibung

Motivation und Erwartungshaltung bzgl. TDD

Problemstellungen und Lösungen

Fazit

Links

Projektbeschreibung

-Kunde: Großer Automobil Finanzdienstleister

-Webapplikation zur Raten-Berechnung für Neufahrzeuge, Motorräder und Gebrauchtwagen

-Im wesentlichen 4 Screens:

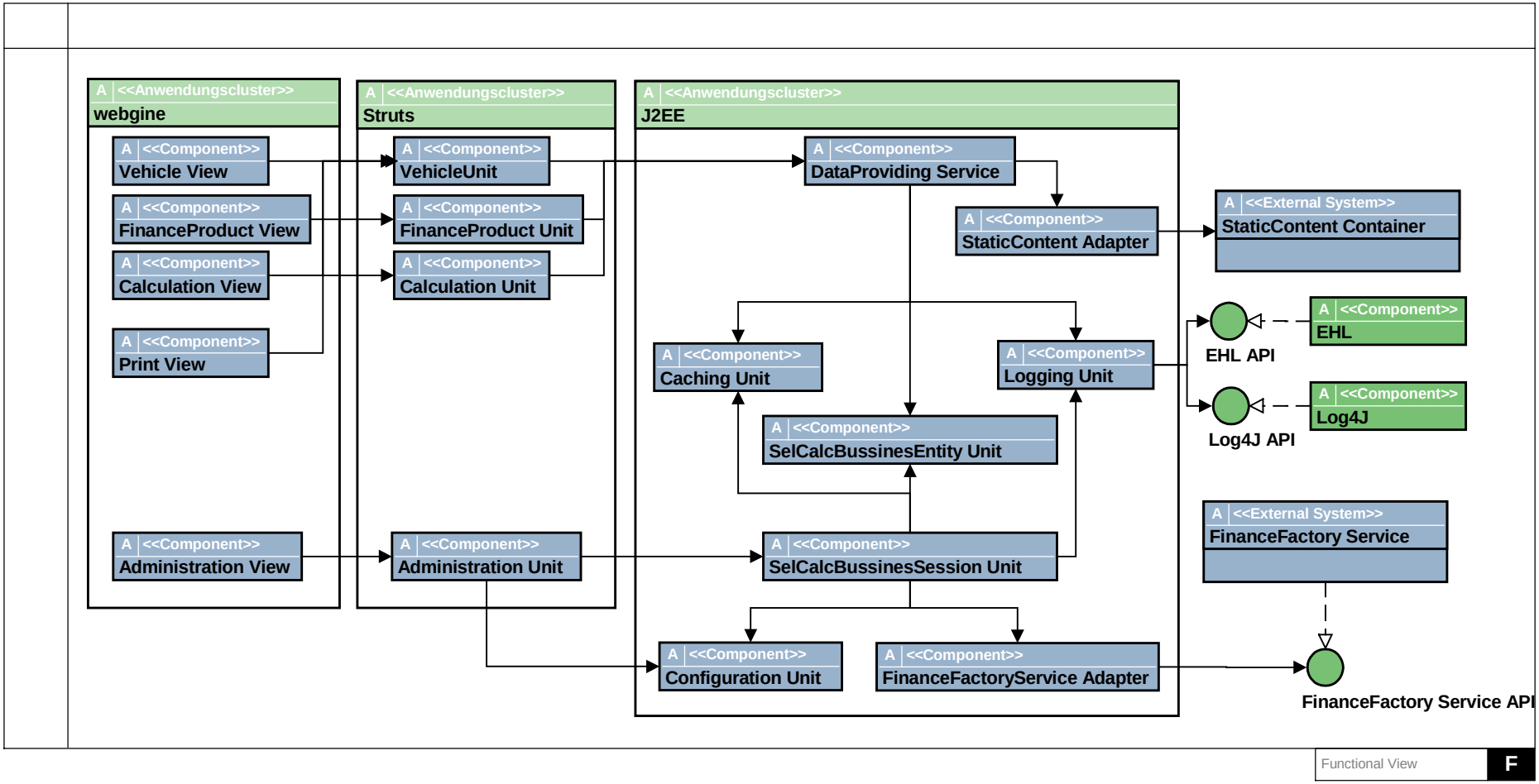
- Auswahl Modellreihe/-variante
- Finanzproduktauswahl
- Persönliche Ratenberechnung (Kalkulator)
- Druckansicht

- Projektlaufzeit: 4 Monate
- Projektteam: 6 Personen

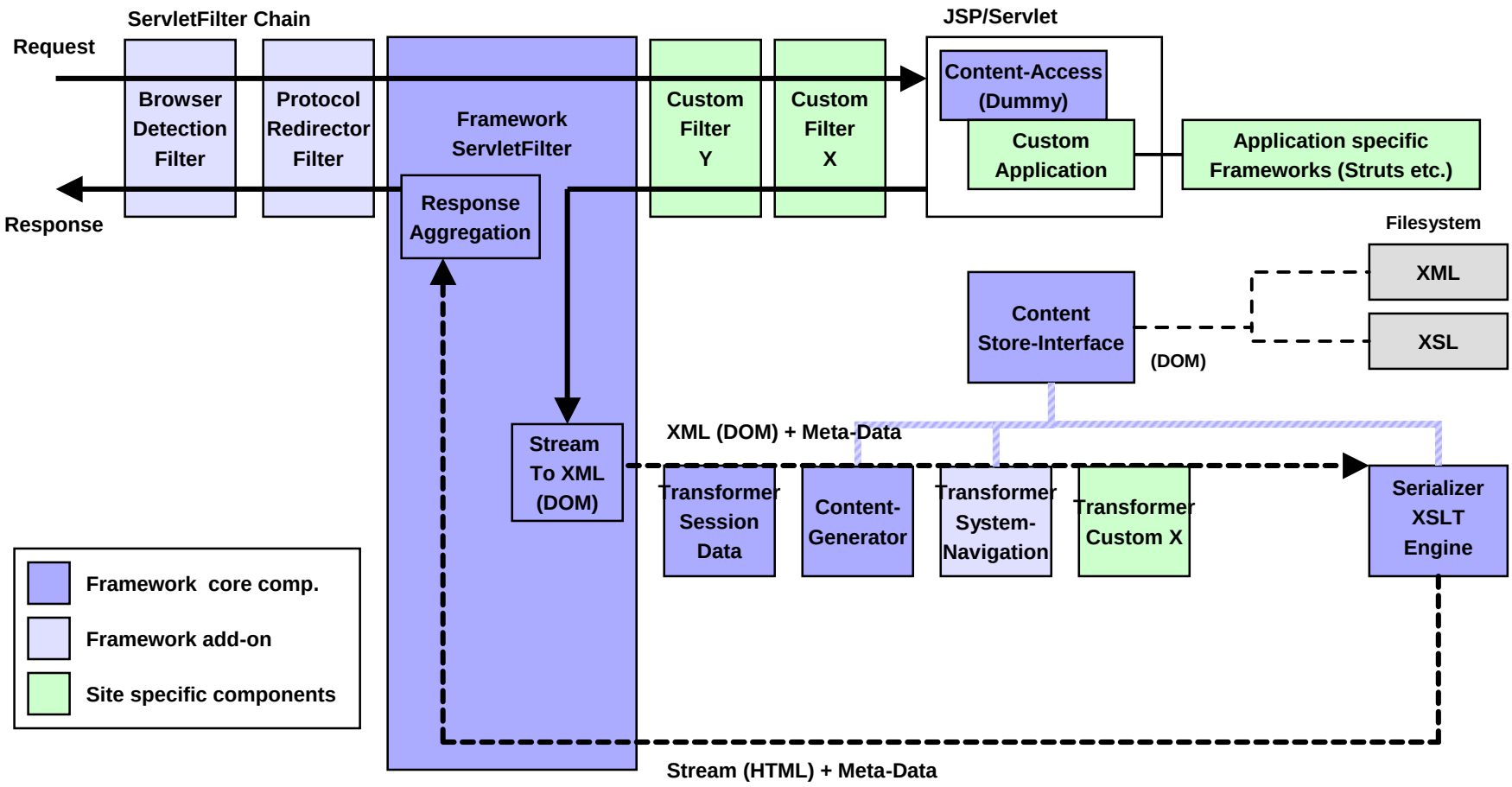
Projektbeschreibung

-J2EE Architektur
-JDK 1.3, BEA Weblogic 6.1 Cluster, Netscape iPlanet, Sun Solaris
-Einbindung der Applikation in bestehendes Web-Framework
-Konsequente Trennung von Layout, Content und Logik
-Einsatz von Struts 1.1
-XML/XSL basiert
-Anbindung an Backendsystem zur Finanzdaten-Berechnung (EJBs)
-Anbindung an bestehendes Logging-Framework

Projektbeschreibung



Das Web-Framework Webgine



Agenda

Vorstellung

Projektbeschreibung

Motivation und Erwartungshaltung bzgl. TDD

Problemstellungen und Lösungen

Fazit

Links

Motivation/Erwartungshaltung TDD

-Steigerung der Qualität der Software
-„Bedarfsorientiertes“ Arbeiten
-Sauberes und schlankes Design
-Reduktion von Abhängigkeiten innerhalb des Teams
-Reduktion von unbemerkten „Nebeneffekten“ bei Code-Änderungen
-Unabhängigkeit von Backend-Systemen
-Verbesserung der Dokumentation
-Beschleunigung des Entwicklungsprozesses

„Clean Code that works – now“
Kent Beck

Agenda

Vorstellung

Projektbeschreibung

Motivation und Erwartungshaltung bzgl. TDD

Problemstellungen und Lösungen

Fazit

Links

Integration mit bestehenden Technologien

-J2EE 1.3

- Servlets und Servlet-Filter
- Keine JSPs
- Keine Eigenentwicklung von EJBs

-Struts 1.1

- Model 2X Ansatz
- Applikation liefert XML, es folgen 2 XSL Transformationen
  - 1. Transformation: Strukturelle Änderungen (Reihenfolge der Elemente etc.)
  - 2. Transformation: Layout-Transformation

-Webgine 1.1 als Web-Framework zur Trennung von Logik, Struktur und Layout (Eigenentwicklung)

Testen von WEB-Applikationen

-Problem:

- Junit allein ist nicht brauchbar für Tests von Web-Applikationen
- Testen von Web-Applikationen ohne Container
- Simulation von J2EE spezifischen Verhalten und Objekten (Request, Response, Session, Servlet Konfiguration etc.)

-Lösung:

- Einsatz einer Junit Erweiterung für Web-Applikationen
  - Cactus (Apache Jakarta Projekt)
  - HttpUnit (für Frontend-Tests) und ServletUnit (für Controller Tests)

HttpUnit/ServletUnit

-Open Source Library

-HttpUnit

- Streng genommen kein Test-Tool sondern ein programmierbarer Browser
- Java API zum Testen von Web-Sites

-ServletUnit

- Simulation eines Servlet-Containers
- Alle nötigen Containerobjekte abbildbar (Request, Response, Session etc.)
- Einfaches Testen von Servlets ohne Container

ServletUnit am Beispiel

- Erstellen einer ServletConfiguration

```
ServletConfigSimulator config = new ServletConfigSimulator();
config.setInitParameter("xmlfactory",
"org.apache.xerces.jaxp.DocumentBuilderFactoryImpl");
```

- Erstellen eines ServletContext

```
ServletContextSimulator ctxs = (ServletContextSimulator)
(config.getServletContext());
ctxs.setContextDirectory(new File("web"));
```

- Initialisieren des zu testenden Servlets

```
XSLServlet servlet = new XSLServlet();
servlet.init(config);
```

- Initialisieren von Request und Response

```
HttpServletRequestSimulator request = new HttpServletRequestSimulator(ctxs);
HttpServletResponseSimulator response = new HttpServletResponseSimulator();
request.setRequestURI("/xsl/test.xsl");
request.setAttribute("info", "Hello World"); // this will make it into the doc
request.addParameter("into2", "Hello World!"); // this won‘t
```


ServletUnit am Beispiel

- Initialisieren der Session

```
HttpSessionSimulator session = (HttpSessionSimulator) (request.getSession(true));
session.setAttribute("test", "Hello World, too"); // this will not make it in the doc
```

- Starten des Servlet Requests

```
servlet.service(request, response);
```

- Der Test

```
String output = response.getWriterBuffer().toString();
String referenceString = "<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>\r\n"
      + "<root><data><data-container><data xsi:type=\"java:java.lang.String\" "
      + "xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">Hello World</data>"
      + "<ref-id>info</ref-id></data-container></data></root>";
// we expect the strings to match
assertEquals(referenceString, output);
```


Testen der Struts Komponenten

- Problem:
  - Testen der Struts Komponenten ausserhalb eines Containers
  - Nicht nur Actions, sondern auch Mappings, Form-Beans, Forward Deklarationen etc. sollen getestet werden

- Lösung:
  - StrutsTestCase for Junit
  - Kompatibel zu Servlet Spezifikationen 2.2, 2.3, 2.4
  - Kompatibel zu Struts 1.1
  - Verwendet den ActionServlet Controller zum Test

StrutsTestCase Beispiel

-Ableiten der Test-Klasse von MockStrutsTestCase

```
public class TestViewCars extends servletunit.struts.MockStrutsTestCase {
```

- Konfigurieren des TestCase

```
public void testSuccessfulLogin() {
        setContextDirectory(new java.io.File("web"));
        setConfigFile("web/WEB-INF/struts-config.xml");
        setRequestPathInfo("/viewCars.do");
```

- Ausführen der Action

```
        actionPerform();
```

- Testen von Request-Attributen

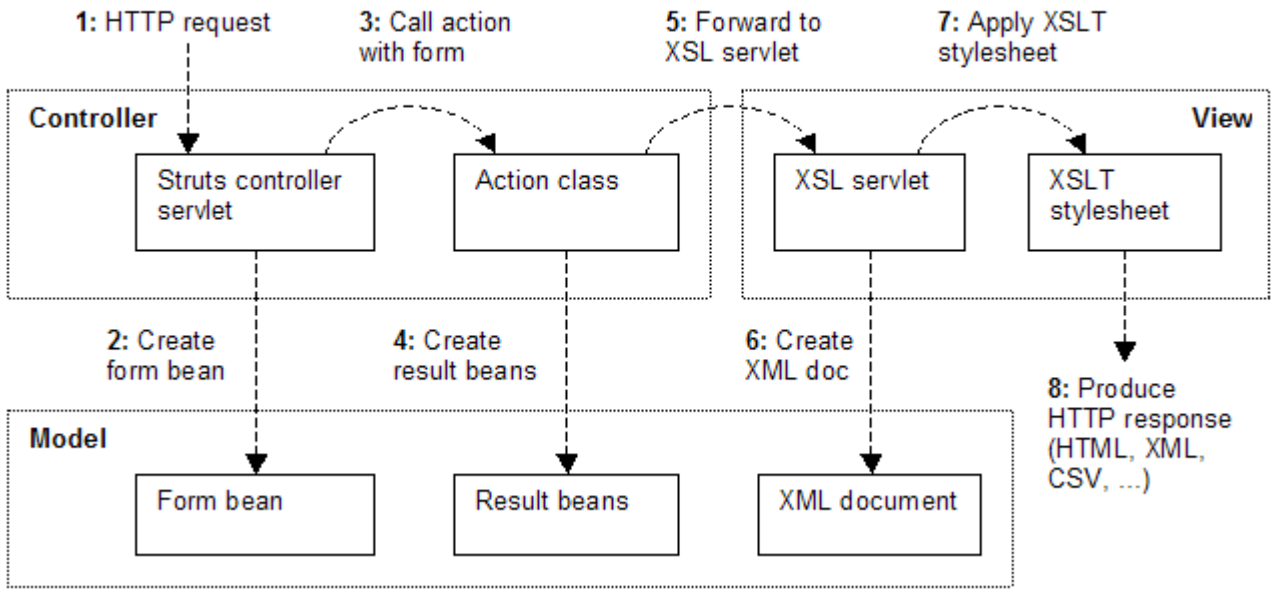
```
        HttpServletRequest request = getRequest();
        assertNotNull(request.getAttribute(com.etecture.struts.taglib.ht
        ml.Constants.TOKEN_KEY));
        assertNotNull(request.getAttribute(„VEHICLES"));
        assertTrue(request.getAttribute("VEHICLES") instanceof
                        VehicleGroupDisplay);
```

- Testen des Forwards

```
        assertEquals("/xsl/viewCars.xsl", getActualForward());
```


Testen des Model 2x Prinzips

- Model 2x ist die Verbindung von Struts (MVC) und XSLT
- Die „View“-Komponente benutzt nach XML serialisierte Beans und XSLT



Testen des Model 2x Prinzips

Vorgehen:

1. Aufruf des XSL-Servlets wie bereits erläutert und „Abfangen“ des generierten Outputs (in unserem Falle XML)
2. Syntaktische Validierung des generierten XML durch ein DTD
3. Inhaltliche Validierung des XML in folgenden Schritten:
    1. erzeugen eines DOM-Objektes des abgefangenen XML Streams
    2. Validieren von Inhalten des DOM mittels XPath Ausdrücken

Testen der Backend Integration

- Zu Projektbeginn war das zu Nutzende Backend zur Kalkulation der Finanzprodukte nocht nicht verfügbar

- Zur Verfügung standen ledigliche Javadocs und ein „finaler“ Interface-Contract

-Die Backend Funktionalitäten wurden durch EJBs zur Verfügung gestellt

=> Vorgehen in 2 Schritten

Testen der Backend Integration, Schritt 1

- Definition eines Interface für den Backend-Adapter
- Implementierung des Factory Patterns für den Adapter
- 2 Implementierungen des Interfaces:
  - 1. Implementierung eines Dummy-Adapters mit definierten Dummy-Werten für alle Methoden
  - 2. Implementierung eines „echten“ Adapters
- Konfiguration der Factory mittels System-Property zur Auswahl des zu instanzieerenden Adapters

=> Junit-testbare Business Logik inkl. emulierter Backend-Anbindung ohne Verfügbarkeit des Backends, ABER: Kein Test des eigentlichen Adapters

Testen der Backend Integration, Schritt 2

- Wie konnten wir den „echten“ Adapter testen ?
- Implementieren eines Mock Object für die spezifizierten EJBs
- Injection dieses Objektes in den „echten“ Adapter

- Ergebnis:
  - Bei Verfügbarkeit des Backend Systemes war der gesamte Applikations- und Schnittstellenteil bereits getestet
  - Ein Umstellen auf das nun existierende Backend verlief völlig problemlos, nach Umstellung der Testcases auf reale Daten liefen alle Tests auf Anhieb fehlerfrei durch.

Clusterfähigkeit

-Problem:

- Was in einem einzelnen (simulierten) Container funktioniert kann im Clusterbetrieb scheitern:
  - Statische Variablen als Applikations-Status
  - Nicht serialisierbare Objekte
  - I/O Operationen innerhalb von J2EE Kompenten
  - Applikationsgesteuertes Caching

- Welche Möglichkeiten des Testings dieser speziellen Problemstellungen gibt es ?

Clusterfähigkeit

- Einzig gelöstes Problem: Serialisierbarkeit
- Erweiterung der ServletUnit Library
- Implementierung eines eigenen SessionSimulators
- Jedes in die simulierte Session geschriebene Objekt wird auf seine Serialisierbarkeit geprüft:

```
public void setAttribute(String s, Object obj)
       throws IllegalStateException, IOException
    {
         [...]
         ByteArrayOutputStream bos = new ByteArrayOutputStream() ;
         out = new ObjectOutputStream(bos) ;
         out.writeObject(obj);        // will possibly throw IOException
         out.close();
         [...]
}
```


Ungelöste Probleme/Ungetane Dinge

- Es wurden keine In-Container Tests implementiert

- Es wurden keine Automatisierte Frontend Tests implementiert

- Clusterfähigheit wird nur im Bezug auf Serialisierbarkeit getestet

- Kein Einsatz einer automatisierten zentralen Überwachung der Coverage

Der Faktor Mensch

„Ich brauche keine Tests, ich teste selbst !“

Menschliche Probleme:

- TDD bedeutet einen starken Eingriff in gewohnte Herangehensweisen
- TDD erfordert ein starkes Umdenken und die Bereitschaft dazu
- TDD erfordert ggf. Neudefinition der Prozesse
- Vor allem: TDD erfordert Disziplin und Überwachung

=> Es empfiehlt sich der Einsatz von unterstützenden Tools wie z.B.

- Coverage Analyse: jCoverage
- Nächtliche Analyse, Tests und Builds : Ant Hill
- Tests der Testcases: Jester

Agenda

Vorstellung

Projektbeschreibung

Motivation und Erwartungshaltung bzgl. TDD

Problemstellungen und Lösungen

Fazit

Links

Fazit

+ Deutliche Steigerung der Code-Qualität
+ Schlankes Design
+ Übersichtlicher Code
+ Extrem niedrige Bugzahl nach QS
+ Reibungsloses Arbeiten im Team
- Einarbeitungsaufwand
- Höhere Aufwände bei Projektmanagement durch Überwachungs-Bedarf

Zufriedener Kunde !!!

Ausblick

- J2EE Architekturen werden zunehmend komplexer
- Ein Umsetzen dieser Architekturen ohne den verstärkten Einsatz von Tools wird zunehmend schwieriger
- Beispiele für solche Tools: Bea Weblogic Workshop, IBM WebSphere Studio Application Developer
- Die Entwicklung wird (ähnlich wie im .NET Umfeld) zunehmend visueller
- Die Anzahl der automatisch generierten Codezeilen oder ganzer Komponenten wird dadurch steigen

=> Es müssen Konzepte entwickelt werden um diese Entwicklung mit XP bzw. Test Driven Development in Einklang zu bringen

Agenda

Vorstellung

Projektbeschreibung

Motivation und Erwartungshaltung bzgl. TDD

Problemstellungen und Lösungen

Fazit

Links

Links/Tools

- jUnit http://junit.sourceforge.net/

- jUnitServlet http://sourceforge.net/projects/junitservlet (Retired)

- Cactus http://jakarta.apache.org/commons/cactus/

- StrutsTestCase for Junit http://strutstestcase.sourceforge.net/

- AntHill http://www.urbancode.com/projects/anthill/default.jsp

- Jester http://jester.sourceforge.net/