eviware

Testdriven utveckling av Web Services

Ole Matzura

eviware

Vad är Test-Driven utveckling?

Test Driven Utveckling

- **2 Grundregler (Kent Beck)**
  - **Skriv aldrig kod utan ett fallerande test**
  - **Eliminera duplicering**
- **Tester ska vara automatiserade, enkla och testa av så små enheter som möjligt – unit-tester**
- **Tester skrivs alltid innan eftersträvad funktionalitet / förbättringar implementeras**
- **Refaktorera för att optimera kod, testerna = skyddsnät**

TDD Processen

- **Specifikation / Krav måste vara tydliga och uppfattade!**
  - **T.ex. User-stories, Sekvens-diagram, etc**

- **Följande iterative process används sedan:**

1. **Skapa ett test**
2. **Kör alla tester och se att det nya fallerar (viktigt!)**
3. **Implementera**
4. **Kör alla tester och se att de går igenom**
5. **Refaktorera koden efter behov**
6. **Börja om till specifikationen är uppfylld**

Externa beroenden

- **Externa beroenden som koden har bör alltid abstraheras i gränssnitt som kan mockas/simuleras**
- **Mock-objekt kan både delta i testet genom att validera input och returnera ”förväntat” data**
- **Nackdel: de externa beroenden testas inte:**
  - **tester med ”riktiga” externa beroende; integrations-tester**

TDD Fördelar

- **Fokus på funktionalitet som faktiskt behövs**
- **Utveckling sker i små steg -> fel hittas tidigt**
- **Abstraktion av externa beroenden leder till ökad modularisering**
- **Fokus på test leder till ”ökad kvalitet”**
- **”Refaktorerings-säker” kod**
- **Tester = Utvecklardokumentation**

TDD Myter

- **En lösning måsta testas till 100%**
- **Unit-tester = Specifikation**
- **Det räcker med unit-tester**
- **TDD räcker som test-insats i ett projekt**
- **TDD skalar inte**

TDD i ett större perspektiv : AMDD

- **Agile Model Driven Development hanterar övergripande frågor**

1. **Iteration 0 : System-vision**
   - **Projektets omfattning**
   - **Övergripande krav**
   - **Arkitekturell vision**

2. **Iteration 1..X : Utveckling**
   - **Modellering**
   - **TDD**
   - **Reviews (helst...)**

Vad är Web Services?

Web Services

- **XML-baserade meddelandeutbyten**
- **Vanligtvis SOAP / HTTP som meddelande / transport-protokoll**
- **Tjänstegränssnitt definieras i WSDL (Web Service Description Language)**
- **Meddelanden definieras inom en WSDL med XML Schema**
- **En uppsjö av standarder (kallas för WS-*, ”WS-Star”) finns för t.ex.**
  - **Säkerhet – WS-Security**
  - **Tillförlitlighet – WS-ReliableMessaging**
  - **Etc..**

Web Services och SOA

- **SOA är ett ”arkitektur-mönster”, inte en teknologi**
- **SOA kan realiseras utan Web Services**
- **Web Services kan finnas utan SOA**
- **Web Services passar dock väl i en SOA pga deras**
  - **abstrakta definition**
  - **meddelande-centrering**
  - **lösa kopplingar**
  - **teknikoberoende**

Web Services och REST

- **REST (Representational State Transfer) är i grunden ett design-mönster för informationsöverföring**
  - **Resurser exponeras via URL:ar och ”bearbetas” via HTTP**
  - **HTTP används som ”envelope” istället för t.ex. SOAP**
- **Används lite slarvigt för att beskriva XML / HTTP (utan SOAP/WSDL)**
- **Saknar standardiserat definitions-språk**
- **JSR-311 : JAX-RS**
  - **Redan nu finns dock visst REST stöd i JAX-WS**

Hur sker utveckling av Web Services i Java?

Utveckling av Web Services i Java

- **Utmaningar**
  - **Översätta meddelanden till/från java-objekt**
  - **Meddelandecentrering vs RPC**
  - **Hantera WS-* standarder**
  - **Välja Ramverk (Axis, CXF, SpringWS, JWSDP, JBossWS, etc..)**
- **Standarder**
  - **JAX-WS – annotations-baserat**
  - **JAX-RPC – konfigurations-baserat**
- **Verktyg för generering av kod <-> WSDL**

Kod-driven utveckling av Web Services

- **Utgå från kod, generera WSDL via verktyg**
  - **Fördelar :**
    - **absolut enklast**
    - **kräver mindre WSDL kunskap**
    - **vanliga POJOs kan exponeras**
    - **standardiserat i Java (JAX-WS)**
  - **Nackdelar :**
    - **lämpar sig ffa för RPC**
    - **genererad WSDL inte alltid ”som man vill”**
    - **Ramverk kan sätta begränsningar**

Kontraktsdriven utveckling med WSDL / XML Schema

- **Utgå ifrån WSDL / XML Schema, generera kod via verktyg**
  - **Fördelar :**
    - **Lättare att skapa ”meddelande-centrerade” gränssnitt**
    - **WSDL kan dela XML-scheman, versioneras, stödja standarder etc. efter behov.**
    - **Ej bundet till funktionalitet i ramverk och kan ”lätt” byta ramverk**
    - **WSDL kan ses som en ”design-specifikation”**
  - **Nackdelar:**
    - **Kräver WSDL kunskaper**
    - **Kan vara mer tidskrävande**
    - **Inte alla ramverk stödjer alls WSDL konstruktioner**

Test och Mockning av Web Services

- **Funktionella Tester mot WSDL kontraktet – ”Black Box tester”**
  - **Kan köras mot olika miljöer / implementationer**
  - **Kan köras kontinuerligt för t.ex. Övervakning**
  - **Kan mha mock skapas utan befintlig implementation**
- **Bör kunna köras via verktyg för en CI/CT miljö**
- **Mockning / Simulering för**
  - **klient-tester**
  - **Skapande av tester innan implementation finns på plats**
  - **Mockning av externa beroenden**
- **soapUI!**

Hur applicera TDD på Web Service utveckling ?

Hur Applicera TDD på WS Utveckling?

- **Vid kod-driven utveckling**
  - **tester kan skrivas i java direkt mot web-service objekten innan de publiceras som web services = ”standard unit-tester”**
- **Vid både kod och kontraktsdriven utveckling**
  - **tester kan skrivas mot ”tomma” web-services som endera publiceras eller genereras från befintlig kod/WSDL**
- **Vid kontraktsdriven utveckling**
  - **Tester kan skapas mot mockade web services redan innan kod genereras/implementeras**

När man tänker efter...

Så passar TDD alldeles utmärkt till

Kontrakts-driven utveckling av Web Services!

TDD vid kontraktsdriven utveckling

- **WSDL:en fungerar som specifikation och utökas för varje iteration**
  - **Operationer/Meddelanden skapas via prototyper**

- **Sen:**
  1. **Skapa test(er) mot tillägget i WSDL:en**
  2. **Kör alla tester och se att det nya fallerar**
  3. **Generera kod och implementera**
  4. **Kör alla tester och se att de går igenom**
  5. **Refaktorera koden efter behov**
  6. **Börja om**

- **Bra på pappret, men hur är det i praktiken?**

eviware

Demo – Skapa specifikation/WSDL inför TDD iteration

- **Vår demo-tjänst: OrderService**
  - **addOrder**

```
<addOrder>
    <productId></productId>
    <quantity></quantity>
</addOrder>
```

```
<addOrderReceipt>
    <orderId></orderId>
</addOrderReceipt>
```

Demo!

eviware

Demo – Skapa specifikation/WSDL inför TDD iteration

- **Vår demo-tjänst: OrderService**
  - **addOrder**
  - **getOrders**

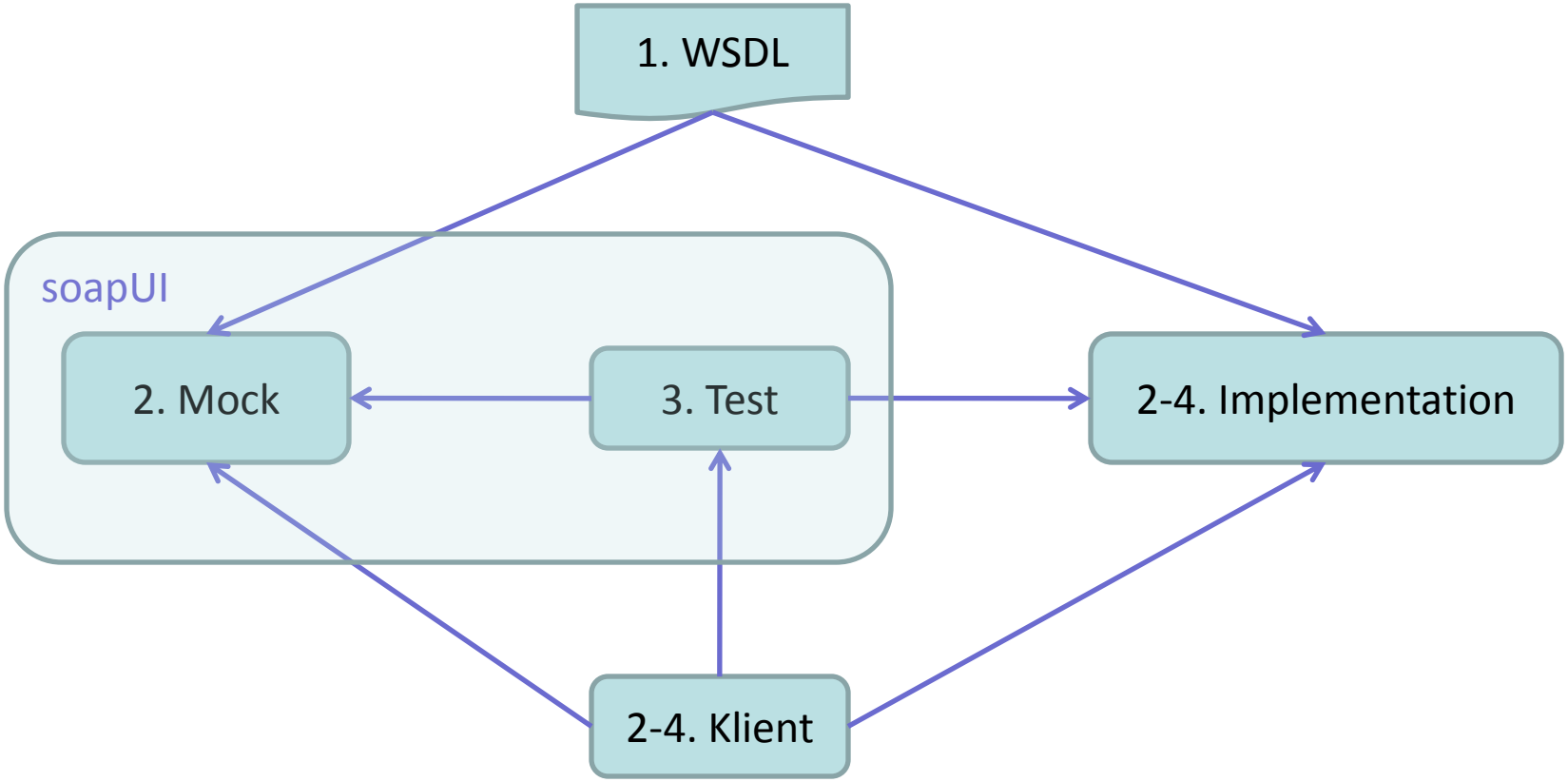
- **getOrders:**

```
<getOrders>
    <productId></productId>
</addOrder>
```

```
<orderList>
    <order>
        <orderId></orderId>
        <productId></productId>
        <quantity></quantity>
    </order>
</orderList>
```

Demo?

Utökat användande av Mockning vid WS-utveckling



eviware

Tack!

ole@eviware.com
http://www.eviware.com