

TESTING

W. Schreier
nov. 2011

Functioneel / niet-functioneel

- Functionele test:
 - ▣ test functionele behoeften (functional requirements)
- Niet-functionele test:
 - ▣ test niet-functionele behoeften (non-functional req.):
 - prestatie
 - schaalbaarheid
 - foutgevoeligheid
 - veiligheid
 - compatibiliteit
 - ...

White box / black box

- Black box testing:
 - ▣ binnenkant (implementatie) niet gekend enkel interface
- White box testing:
 - ▣ binnenkant (implementatie) wel gekend
 - speciale gevallen
- Grey box testing:
 - ▣ mix

Unit / integratie / validatie

1. Unit testing:
 - ▣ apart onderdeel (klasse, methode, ...)
 - ▣ soms mockups (dummy's) voor afhankelijkheden
 - ▣ kan vroeg tijdens ontwikkeling
2. Integratietest:
 - ▣ delen samenbrengen
 - ▣ later tijdens ontwikkeling
3. Verificatie en validatietest:
 - ▣ zijn specificaties voldaan?

Regression testing

- regressie = fout in (getest) systeem geïntroduceerd door wijziging (bv. refactoring, bugfix, patch, uitbreiding, ...)
 - ▣ invloed op andere stukken code?
 - ▣ werkt alles nog?
 - ▣ oude fouten terug verschenen?
- oude tests opnieuw uitvoeren
 - => automated tests, bv. na elke build

Smoke testing

- Oorspronkelijk bij loodgieterij en bij elektronica
- Snelle eenvoudige test
 - ▣ bv. Start het programma? Wordt er een venster geopend? Gebeurt er iets als je op de knop klikt...
- Indien mislukt: geen moeite meer steken in
 - ▣ installatie
 - ▣ verder testen

Boundary testing

- speciale gevallen

Wanneer?

- Zo vroeg mogelijk fouten zoeken
 - hoe later, hoe duurder om te herstellen
 - Vooraf ?
 - Test Driven Development (TDD)
 - Behaviour Driven Development
 - ...
- Test Cases volgen Use Cases / Scenario's
Helpt bij het ontwerpen (design)!

Wie?

- Oorspronkelijk developer zelf
- Later aparte testers
- In moderne methodologieën (Agile, ...)
 - meer geïntegreerd in team
 - en betrokken bij ontwerp en ontwikkeling

Wat?

- Business Logic (modelklassen)
- Datalaag
 - DAO
 - databank
- User interface
 - Desktop GUI
 - Web
- SUT = system under test
- Code coverage: hoeveel % van de code getest?

Uitvoer testprogramma

- Zal iemand anders de uitvoer begrijpen?
- En jijzelf, enkele weken later?
- ⇒ Schrijf geen gegevens uit
- ⇒ Schrijf "ok" of "niet ok"...

Unit testing

- Onderdelen apart testen
 - klasse
 - methode
- Meest bekend: JUnit
 - versies 3.x en 4.x

Fixture en fases

- Fixture
 - = vaste context/begintoestand voor testen
 - = baseline state
 - bv. lege databank, geformatteerde schijf, geen gebruikers, etc.
- Vier fases:
 1. **Set up**: test fixture klaarzetten
 2. **Exercise**: interageer met het te onderzoeken systeem
 3. **Verify**: controleer of verwacht resultaat bereikt is
 4. **Tear down**: test fixture afbreken naar originele toestand
- Test suite: (onafhankelijke) unit tests met zelfde fixture

JUnit 4.x voorbeeld

```

public class Stack<T>
{
    private ArrayList<T> elements;

    public void push(T x)
    {
        elements.add(x);
    }

    public T pop()
    {
        if (elements.isEmpty())
            throw new EmptyStackException();
        return elements.remove(elements.size()-1);
    }

    public boolean isEmpty()
    {
        return elements.isEmpty();
    }

    public int size()
    {
        return elements.size();
    }
}

```

Te testen klasse

Opgelet: deze code bevat fouten!

JUnit 4.x voorbeeld

```

import org.junit.Test;
import static org.junit.Assert.*;

public class StackTest
{
    @Test
    public void testBasicStackUsage()
    {
        Stack<String> stack = new Stack<String>();

        assertTrue(stack.isEmpty());
        assertEquals(0, stack.size());

        stack.push("aap");
        assertFalse(stack.isEmpty());
        assertEquals(1, stack.size());

        String x = stack.pop();
        assertEquals("aap", x);
        assertTrue(stack.isEmpty());
        assertEquals(0, stack.size());
    }
}

```

Annotatie

Resultaat

0,00 %

No test passed, 1 test caused an error (0,072 s)

util.StackTest FAILED

testBasicStackUsage caused an ERROR: java.lang.NullPointerException

java.lang.NullPointerException

at util.Stack.isEmpty(Stack.java:27)

at util.StackTest.testBasicStackUsage(StackTest.java:13)

Info over de plaats van de fout

Na het toevoegen van elements = new ArrayList<T>():

0,00 %

No test passed, 1 test caused an error (0,083 s)

util.StackTest FAILED

testBasicStackUsage caused an ERROR: Index: 1, Size: 1

Index: 1, Size: 1

java.lang.IndexOutOfBoundsException

at java.util.ArrayList.rangeCheck(ArrayList.java:547)

at java.util.ArrayList.remove(ArrayList.java:387)

at util.Stack.pop(Stack.java:22)

at util.StackTest.testBasicStackUsage(StackTest.java:20)

Resultaat (2)

Na de correctie in pop()

elements.size() -> elements.size() - 1 :

100,00 %

The test passed (0,071 s)

JUnit met setUp en tearDown

```

public class DBTest
{
    private Connection conn;

    @Before
    public void setUp() {
        // open connection...
    }

    @After
    public void tearDown() {
        // close connection...
    }

    @Test
    public void test1() {
        // ...
    }

    @Test
    public void test2() {
        // ...
    }
}

```

Excepties testen (1)

```
try
{
    Stack<String> stack = new Stack<String>();

    assertTrue(stack.isEmpty());

    // ... verdere testen ...
}
catch (Exception ex)
{
    fail("Unexpected exception" + ex);
}
```

Mag geen exceptie opwerpen

Excepties testen (2)

```
try
{
    Stack<String> stack = new Stack<String>();
    assertTrue(stack.isEmpty());
    try
    {
        stack.pop();
        fail("Should throw an exception");
    }
    catch (EmptyStackException ex)
    {
        // ok
    }

    // ... verdere testen ...
}
catch (Exception ex)
{
    fail("Unexpected exception" + ex);
}
```

Moet een exceptie opwerpen

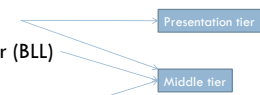
JUnit in NetBeans

- JUnit genereren
 - ▢ leeg (imports...)
 - ▢ test voor bepaalde klasse
 - manueel aanpassen!
- JUnit's uitvoeren
- Testsuites

Tiers en layers

- multitier = n-tier
- meest bekend: 3-tier
 - ▢ presentation tier (bv. browser op laptop)
 - ▢ application/middle tier (bv. J2EE-applicatie)
 - ▢ data tier (bv. Oracle server)
- tier = fysische indeling (proces, machine)
- layer = logische indeling (code)
- tier en layer soms als synoniemen gebruikt
- heel veel varianten, andere naamgeving, discussie, ...

Typische layers

- presentation layer
 - business logic layer (BLL)
 - ▢ business objects
 - data access layer (DAL)
 - ▢ data access objects (DAOs, Entity Beans, ORM-objects)
 - ▢ toegang tot data store (databank, xml, webservice, ...)
 - soms: service layer
 - ▢ e-mail, logging, ...
 - andere indelingen en naamgeving mogelijk!
- 

Principes layers

- Laag: grote cohesie
- Tussen lagen: losse koppeling
- Laag kent enkel lagen erboven en eronder
- Lagen (zeker buitenlagen) moeten vervangbaar zijn
- Ideaal voor IoC / DI
- Niet zelfde als MVC!

Multitier/multilayer en testing

- Lagen apart testen
 - ▣ bv. unit tests voor business logic
 - ▣ vervang andere lagen door mockup
- ▣ bv. lokale i.p.v. remote databank
- ▣ bv. file persistence i.p.v. databank
- ▣ bv. in-memory databank (bv. hsqldb)
- ▣ bv. console-UI i.p.v. GUI
- ▣ bv. GUI i.p.v. web-UI

Database testing

- niet op production database
 - sandboxes
 - precieze toestand moet gekend zijn vóór test
 - ▣ DB opnieuw aanmaken of data verwijderen
 - ▣ Indien nodig initialisatiedata toevoegen
 - bv.
- ! scripts voor elk van deze stappen !**

Database testing

- Testdata: verschillende mogelijkheden:
 - ▣ uit bestanden (tekst, csv, excel, ...) of andere databank of andere tabellen
 - ▣ genereren met script(s)
 - ▣ "self contained" testcase: voegt zelf nodige data toe
 - ▣ combinatie van vorige
- Tear down (opruimen)

DBUnit

- JUnit voor DB-testing
- Omzetting DB <-> XML
- Kan inhoud tabellen controleren

DBUnit voorbeeldfragment

```
public void testMe() throws Exception {
    // Execute the tested code that modify the database here ...

    // Fetch database data after executing your code
    IDatabaseDataSet databaseDataSet = getConnection().createDataSet();
    ITable actualTable = databaseDataSet.getTable("TABLE_NAME");

    // Load expected data from an XML dataset
    IDatabaseDataSet expectedDataSet = new FlatXmlDataSetBuilder().build(
        new File("expectedDataSet.xml"));
    ITable expectedTable = expectedDataSet.getTable("TABLE_NAME");

    // Assert actual database table match expected table
    Assertion.assertEquals(expectedTable, actualTable); }
```

Data access layer testen

- Analooq aan databank testen:
 - ▣ data in gekende initiële toestand brengen
 - ▣ test-DB versus production-DB

Dummy's

- relatie tussen componenten
- vervang een of meerdere componenten door dummy:
 - zelfde "interface"
 - eenvoudiger implementatie, werking
 - eenvoudiger deployment
- Ideaal voor Inversion of Control (interfaces, abstract factory, ...)

Mockup

- = gegenereerde dummy-klasse
- implementeert interface
- heel eenvoudige werking
 - bv. bepaalde vaste waarden teruggeven
- Mockito, EasyMock, JMock, PowerMock

Voorbeeld EasyMockup

```
@Test
public void testAddAndChangeDocument() {
    ClassUnderTest classUnderTest = new ClassUnderTest();
    DocListener mock = createMock(DocListener.class);
    classUnderTest.addListener(mock);

    mock.documentAdded("Document");
    mock.documentChanged("Document");
    expectLastCall().times(3);

    replay(mock);

    classUnderTest.addDocument("Document", new byte[8]);
    classUnderTest.addDocument("Document", new byte[8]);
    classUnderTest.addDocument("Document", new byte[8]);
    classUnderTest.addDocument("Document", new byte[8]);

    verify(mock);
}
```

Diagram labels:

- specifieer "verwachting" (points to `expectLastCall().times(3);`)
- einde specificatie / begin echte test (points to `replay(mock);`)
- verifieer "verwachting" (points to `verify(mock);`)

GUI testing

- Streef naar scheiding van presentatie en logica bv. MVC, MVP, n-tier
- GUI zelf testen
 - simuleer gebruiker die op knoppen klikt etc.
 - eventueel door "macro-recording"
 - controleer eigenschappen van componenten (bv. windowpositie, labeltekst, ...)
- bv. voor Java Swing:
 - UISpec4J, FEST, Abbott, Jemmy, jfcUnit, windowlicker, ... [Cucumber + Swinger]

Voorbeeld UJSpec4J

```
@Test
public void testCreatingAContact() {
    Window window = getMainWindow();
    Table table = window.getTable();
    Button newContactButton = window.getButton("New contact");

    assertTrue(table.getHeader().contentEquals(
        new String[]{"First name", "Last name", "E-mail"}));
    assertTrue(table.isEmpty());

    newContactButton.click();
    assertTrue(table.contentEquals(new String[][]{ {"", "", "", ""} }));
    assertTrue(table.rowsSelected(0));

    window.getTextBox("first").setText("Homer");
    window.getTextBox("last").setText("Simpson");
    window.getTextBox("email").setText("homer@simpson.com");
    assertTrue(table.contentEquals(
        new String[][]{ {"Homer", "Simpson", "homer@simpson.com"} }));
}
```

Web-app testing

- principe vergelijkbaar met GUI-testing
- imiteer gebruikersacties in browser (klikken etc.)
- controleer geproduceerde HTML
- Tools:
 - Selenium, HTMLUnit, JWebUnit, Canoo WebTest, JSFUnit, ...

seleniumhq.org

- Selenium **IDE**
 - ▣ Firefox extension
 - ▣ gebruikersacties opnemen (klikken, tekst ingeven, link volgen, ...) en terug afspelen
 - ▣ assertions toevoegen (bv. `assertTextPresent(...)`)
 - ▣ ideaal voor eenvoudige gevallen
- Selenium **RC** (Remote Control)
 - ▣ deprecated
- Selenium **WebDriver**

Continuous integration

Principles of continuous integration

- 1 Maintain a code repository (**CVS**, **SVN**, **git**, ...)
- 2 Automate the build (**ant**, **maven**, ...)
- 3 Make the build self-testing (**JUnit**, ...)
- 4 Everyone commits to the baseline every day
- 5 Every commit (to baseline) should be built
- 6 Keep the build fast
- 7 Test in a clone of the production environment
- 8 Make it easy to get the latest deliverables
- 9 Everyone can see the results of the latest build
- 10 Automate deployment

Tools: Cruise Control, Jenkins, Hudson, ...

TDD en BDD

- TDD = test driven development
 - ▣ tests eerst schrijven
 - ▣ helpt bij modelleren (bv. keuze interface)

- BDD = behavior driven development
 - ▣ abstracter, meer high level
 - ▣ use cases, scenario's
 - ▣ zie bv. Cucumber