Bitter TDD

Jan Västernäs

jan.vasternas@callistaenterprise.se

Agenda

- TDD Concepts
- Project Setup
- Experiences
- Summary

CALLISTA

Agenda

- TDD Concepts
- Project Setup
- Experiences
- Summary

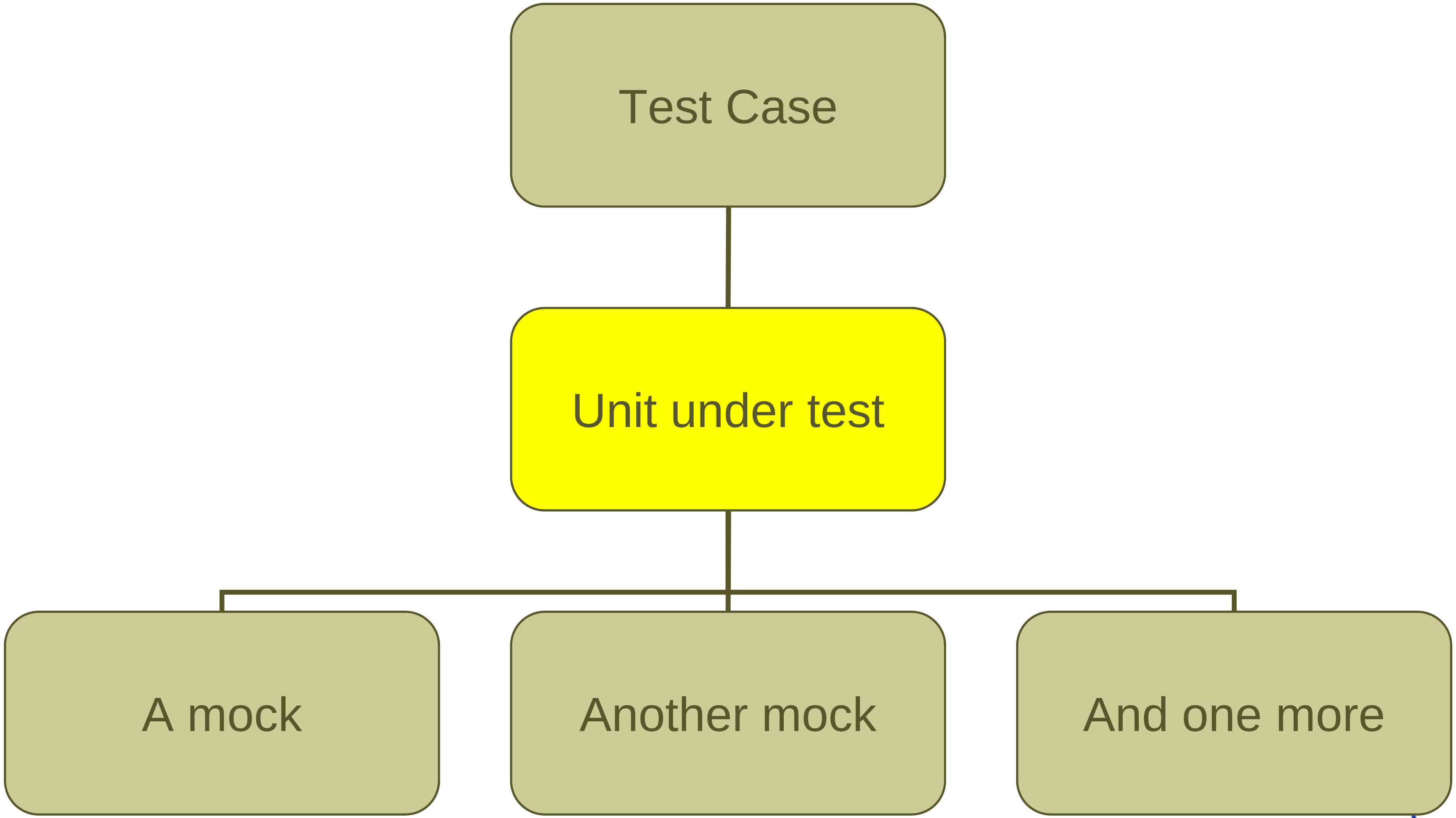
Different kind of tests

- Unit test
  - Mock dependencies
    - Test external interface or
    - Test internal state

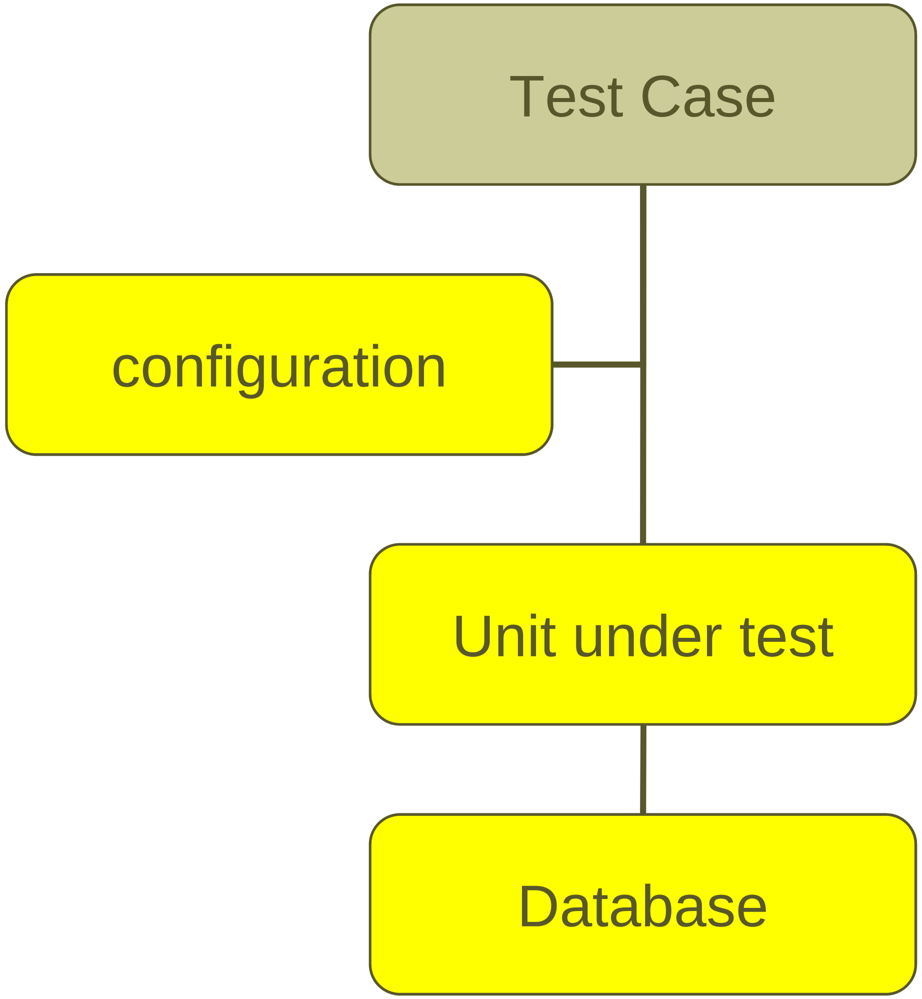
- Integration Test
  - Configuration(spring/java ee), Database, JMS, JCA etc
  - Inside/outside application server (out-of-container)

- Larger Unit Integration Test
  - More than one class

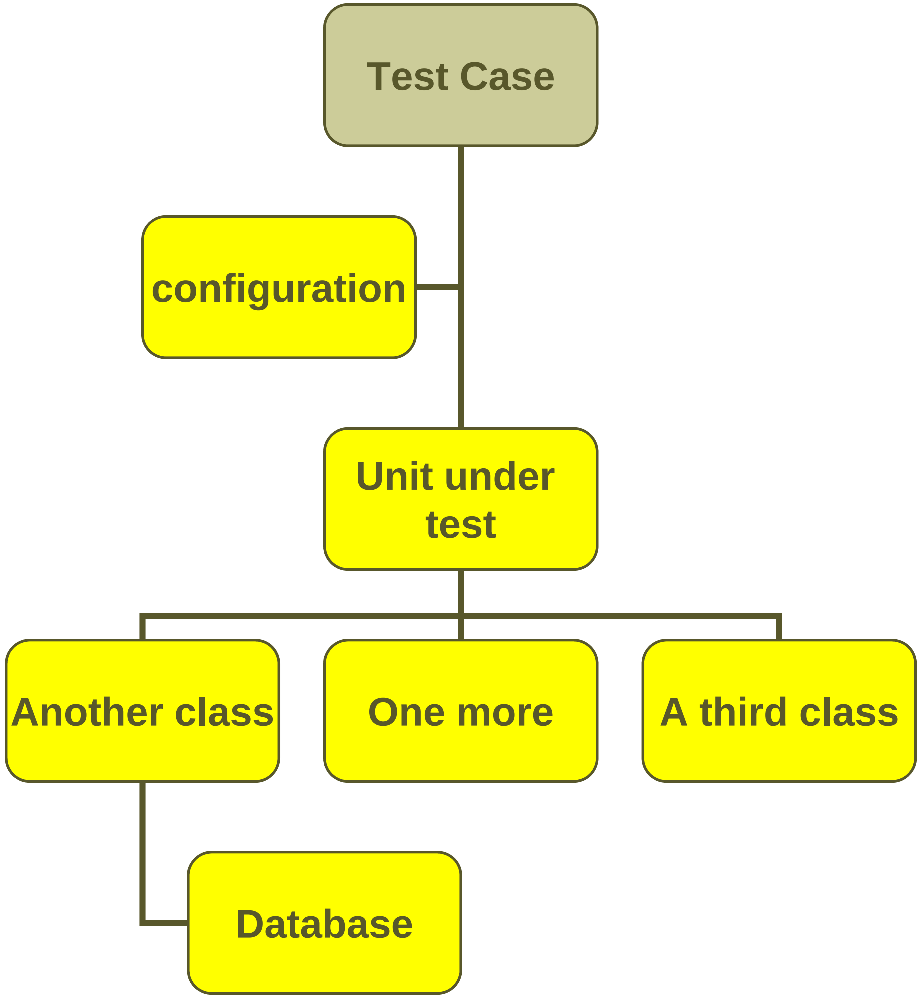
Unit test



Integration test



Integration Test – larger unit



Sample mock unit test using easymock

```
CustomerDao daoMock = createMock(CustomerDao.class);

List<Customer> list = new ArrayList<Customer>()
List.add(new Customer(”Callista”, CustomerType.GOOD, ... );
expect(daoMock.getAll()).andReturn(list);
replay(daoMock);

CustomerEntityImpl customerEntity = new CustomerEntity();
customerEntity.setCustomerDao(daoMock);
List<Customer> list = customerEntity.getAllCustomers();
assertEquals(1, list.size());
verify(daoMock);
```

Sample unit test set/get internal state

```
WorkingCalendarAssembly cal = new WorkingCalendarAssembly();
cal.workingCycleList = new ArrayList<WorkingCycle>();
cal.workingCycleList.add(new WorkingCycle(1,2,1,800,1600));
cal.workingCycleList.add(new WorkingCycle(1,2,2,1600,2400));
......

cal.expand();

List<OpenInterval> intervals = cal.intervals;
assertEquals(36,intervals.size());
..........
```

Sample integration test using DBUNIT

```
String INTITAL_DATA = "<?xml version='1.0'?>“ +
“<CUSTOMER SID='1‘ NAME=‘Callista´ TYPE=‘GOOD’  ....”
“<CUSTOMER SID=‘2‘ NAME=‘Universal Software´ TYPE=‘UGLY’ ...”


setup()
  DatabaseOperation.CLEAN_INSERT ........   INITIAL_DATA
  customerEntity = (CustomerEntityImpl)
        applicationContext.getBean(“customerEntity");


testMetod()
   List<Customer> list = customerEntity.getAllCustomers();
   assertEquals(2, list.size());
```

Agenda

- TDD Concepts
- Project Setup
- Experiences
- Summary

Project setup

- 2 Projects, 3 years
- Java SE 5, J2EE 1.4
- Spring-beans, web or mdb
- Layered, component-based architecture
- Struts/JSP
- Eclipse Rich Client Platform + spring remoting
- *Test : JUnit, DbUnit, EasyMock*
- Maven
- JDBC - Oracle
- Websphere Application Server (target platform), MQ
- Ambitious TDD combined with CI
- Custom libraries, documented best practice for test

Agenda

- TDD Concepts
- Project Setup
- Experiences
- Summary

Test experience in a nutshell

+ Good

- Expensive

Why is it good ?

- Drives interfaces, documents intended functionality
- Enables bit by bit unit testing rather than testing from system boundries.
  - Easier
  - Faster
- Refactoring
  - Shows other needs for change
  - Shows that the rest is OK
- Fast feedback after check in of changed code

- All phases

Why is it expensive ?

- Time to write
- Time to maintain
- Time to run
- Yet another set of frameworks
- Needs training, mentoring, reviews
- One-time investments, running cost

- On the other hand
  - Saves development time
  - Saves error/debugging time

Vital to maximize RoI

- Keep reasonable cost level

- Right-size number of tests
- Maximize value of each test

- How ? Our experience

Detailed Experience

- Test Driven
- Test Coverage
- Test all classes/methods
- Pure unit tests
- Integration test
- Test Data
- Database schemas
- Change implementation
- Out of container testing
- Bug fixes

CALLISTA

Test Driven Development

A) Write all tests first, then write implementation

B) Write implementation first, then write test

C) Test a little/write a little

D) All of the above

OK, depends on

E) Write implementation

- Often not OK, A) and C) minimizes risk of E)

Why Test Coverage is a blunt instrument

- Team leader: “We need 93 % coverage”
- Unnecessary or expensive tests
- To easy to get
- May result in a test
  - new myClass().callVeryComplicatedMethod(null,0,” ”);
- Or a little better
  - assertNotNull(uut.method());
- Or in worst case
  - if (parameter == null) {return null;}
- Fine-grained goals
- Quality vs quantity

Fine-grained coverage approach

	Component A	Component B Basic Data	Component B Integration	Component B Calculation
Presentation/MDB	**0**	**0**	**80**	**0**
Façade	**20**	**20**	**-**	**-**
Service	**20**	**60**	**100**	**100**
Entity	**20**	**40**	**100**	**80**
Dao	**0**	**40**	**60**	**80**

Tests for all classes ?

- Default answer: Yes
- Combine with minimum coverage
- Good for inexperienced TDD-ers
- Different in a test-mature organization
- ”Too simple to test”
- Complex algorithms etc

Avoid overlapping tests

- Simple delegation

Unit test risk: copy of implementation

- Many calls to other classes
- Set up transfer-object
- Implementation changes ?
- Hard to understand
- Often needed but declining in my mind

Easymock

- Simple to use
- No need to write mocks
- Support for classes
- ”Program to an interface not an implementation” revisited
- Default - equals
  - Base class for all transfer objects
  - Equals compares all attributes with reflection
  - Timestamps ?

Data base integration test risk: Schema changes

- NEW_ATTRIBUTE VARCHAR(10) NOT NULL

- String jadajada = ” VERSION=’1’ CHANGED_BY=’robban” CHA...
- ”<CUSTOMER ID=’1’ NAME=Callista’ ” + jadajada + ” />

- Avoid mixing java and DbUnit

”<CUSTOMER ID=’” + getId(customerMetaData) + “’ VERSION=‘“ +
i++

DbUnit

- Efficient
- Simple

Fixed test data ?

- Data model develops
- Changes cost and introduce risk
- Introduces dependencies
- Better to let all tests setup their own
- Well-documented exceptions

CALLISTA

Schemas

- Original DB-provider / different light-weight ?
- We choose Oracle
  - Syntax, tooling
- Build schema
- Personal schema
- Other schemas
- Different load scripts
  - Minimum - 3 tables (required for tests)
  - Medium - data setup by installation
  - Full/Demo - business-data in all tables
- Need to control schema access at all times

When to run tests ?

- Project rebuild in Eclipse
  - Too often
- Just build – forget about the tests
  - Checkout
  - Small change component A, test component B
  - Command file mvnn

Code changes to simplify/enable testing

- YES
- Refactor private methods
  - Extract logic to test
  - Isolate things that are hard to mock
  - Should private methods be tested ?
- Package visibility (test in same package)
  - Methods – to be able to call
  - Attributes – to be able to set/get

Out of container testing

- Invaluable
  - Datasource and transaction handling pluggable with alternate spring-config and spring-config-factory

- Even for GUI when using Java EE independent technology

Bug fixing strategy

- Recreate with test (red)
- Fix implementation
- Check that test is green

Smells of a less valuable test

```
List<CompositeTO> result = uut.theVeryComplicatedMethod();
System.out.println(result);
```

4 kilometer console output

```
If ( x.equals(null) ){
    throw new NullPointerException();
}
public void test() {
    . . . .
}
```

“OK -- enhetstesterna var kanske inte riktigt heltäckande .....”

Signs of a valuable test ?

```
assertEquals(expected, actual);

expect(getAllCustomers()).andReturn(customerList);

public void testMyFirstMethod_manyCustomersPerCountry()
public void testMyFirstMethod_oneCustomersPerCountry()
public void testMyFirstMethod_noCustomersForCountry()
public void testMyFirstMethod_daoThrowsPKNFException()

public void testSmoke()
```

How to become a certified tester ?

Works on
my
Machine

Technical Certification Requirements

- Compile your application code. **Note:** Getting the latest version of any recent code changes from other developers is purely optional and **not** a requirement for certification.
- Launch the application that has just been compiled.
- Cause one code-path in the code you're checking in to be executed. **Note:** the preferable way to do this is with very ad-hoc manual testing of the simplest case for the feature in question. The Stovell Institute for Application Assurance suggests that it is possible to omit this step if the code change was less than five lines, or if (in the developer's professional opinion) the code change *could not possibly result in an error*.
- Check the code changes into your version control system.

Agenda

- TDD Concepts
- Project Setup
- Experiences
- Summary

Conclusion

- How did we survive in the pre-TDD era ?
  - Great improvement, right on spot
- Focus test effort on complicated implementations !
- Quality before quantity
- RoI ? Hard to prove, but still.

Questions ?

CALLISTA