Automatic test anti-patterns
Agenda

► Motivation
► What is automatic test
► Anti-patterns catalog
► Basic mistakes
► Best practices
► Summary
► Questions
Motivation

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.
~Martin Fowler

- Detect **REAL** bugs early
- Avoid boring regression tests
- Better design, api documentation
- Code quality
// Getting the model we just read from file:
importDefinitionTO def = null; // adapter.getDefinition;

// Set the date where this import should be valid from
def.setKeyDate(this.validFromDate);
def.setTargetCatalogCode(this.targetCatalog);
def.setTargetFolderCode(this.targetFolder);
Agenda

- Motivation
- What is automatic test
- Anti-patterns catalog
- Basic mistakes
- Best practices
- Summary
- Questions
What is automatic test

► No definition in wiki ;-(
► ..... example in eclipse
Agenda

- Motivation
- What is automatic test
- Anti-patterns catalog
- Basic mistakes
- Best practices
- Summary
- Questions
Anti-patterns catalog

Anti-patterns catalog (2)

http://blog.james-carr.org/2006/11/03/tdd-anti-patterns/

http://stackoverflow.com/questions/333682/tdd-anti-patterns-catalogue
Agenda

- Motivation
- What is automatic test
- Anti-patterns catalog
- **Basic mistakes**
- Best practices
- Summary
- Questions
Basic mistakes

- No tests
- Manual (missing) Assertions
- Redundant Assertions
- Using the Wrong Assert
- Only Easy/Happy Path Tests
- Complex Tests
- External Dependencies
- Catching Unexpected Exceptions
- Mixing Production and Test Code
No tests

► Still common in many companies
► “It is too expensive to write tests.”
► “We will write tests later”. (never)
► The Liar, Success Against All Odds
No tests

// TODO: This tests need to be activated when nightly db-reset is done!
@Ignore
@Test
public void testXXX() {
    ...
}

@Test
public void testYYY() {
    // TODO implement me
}

// TODO implement me
Manual (missing) Assertions

► Sysout instead of assert
► Why not just use a debugger?
► The Secret Catcher

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."
Brian W. Kernighan
Manual (missing) Assertions

public void testCalculation() {
    deepThought.calculate();
    System.out.println(deepThought.getResult());
}

public void testCalculation() {
    deepThought.calculate();
    assertEquals(42, deepThought.getResult());
}
Redundant Assertions

- Always true or always false assertions
- Introduced by mistake or for debugging purposes
Redundant Assertions

```java
public void testSomething() {
    ....
    assertEquals(10.0, updatedEntity.getQuantity(), Double.MAX_VALUE);
    assertTrue(true);
}
```

```java
public void testSomething() {
    ....
    assertTrue(existingServiceEntryTOs.size() == 0);
    assertNotNull(existingServiceEntryTOs);
}
```
Using the Wrong Assert

► There is more than assertTrue
► java assert keyword
Using the Wrong Assert

assertTrue("Objects must be the same", expected == actual);
assertTrue("Objects must be equal", expected.equals(actual));
assertTrue("Object must be null", actual == null);
assertTrue("Object must not be null", actual != null);

assertSame("Objects must be the same", expected, actual);
assertEquals("Objects must be equal", expected, actual);
assertNull("Object must be null", actual);
assertNotNull("Object must not be null", actual);
Only Easy/Happy Path Tests

► Only expected behavior is tested
► Only easy to verify things are tested
► The Liar, The Dodger, Success Against All Odds
Only Easy/Happy Path Tests

```java
@Test
public void testGoodCase()
{
    Assert.assertEquals(2.0, Math.sqrt(4.0), 0.0);
}

@Test
public void testInteger()
{
    Integer intToTest = new Integer(10);
    Assert.assertEquals(intToTest.intValue(), 10);
}
```
Complex Tests

► Same rules like for production code
Complex Tests

In general, you should refactor a test until it can be easily recognised to follow the general structure:

► Set up
► Declare the expected results
► Excercise the unit under test
► Get the action results
► Assert that the actual results match the expected results
(External) Dependencies

- External dependencies that code may need to rely on to work correctly.
- Dependencies between tests – one test prepares data for other one.
- Dependencies to object internal state
Catching Unexpected Exceptions

► Test succeeds even if an exception is thrown
► The Liar, The Greedy Catcher

```java
public void testCalculation() {
    try {
        deepThought.calculate();
        assertEquals("Calculation wrong", 42, deepThought.getResult());
    } catch(CalculationException ex) {
        Log.error("Calculation caused exception", ex);
    }
}
```
Catching Unexpected Exceptions

```java
public void testCalculation() {
    try {
        deepThought.calculate();
        assertEquals("Calculation wrong", 42, deepThought.getResult());
    } catch(CalculationException ex) {
        fail("Calculation caused exception");
    }
}
```
private List<NewServiceEntryTO> startCreateDefaultServiceEntriesWithoutParameter(
    final ExistingSessionTO existingSessionTO)
{
    try {
        return createDefaultServiceEntriesWithoutParameter(existingSessionTO);
    } catch (RuntimeException e) {
        // it's not a fail, it's an epic fail!
        fail("Creation must not fail.");
        logger.error(e.getMessage());
    }
    return new ArrayList<NewServiceEntryTO>();
}
Mixing Production and Test Code

- More complex packaging
- Ability to test package private methods
Programming is like sex. One mistake and you have to support it for the rest of your life.
~Michael Sinz
DatabaseFactory.java

/**
 * Attempts to extract the DBMS driver name from the given data source. Because the DS does not provide any API supporting such an endeavor we
 * resort to fugly babysitting of particular implementations. the resulting lowercase string will hopefully contain "sql" or "oracle" which allows
 * detection of the dialect. Be prepared to patch this method when the container environment changes.
 *
 * In case of emergency procure a sixpack of beer and dial ++41 76 348 00 01 ;)
 *
 * @param ds
 * the DS to look at.
 *
 * @return a lowercase string reflecting the DBMS "driver" used by the DS.
 */

private static String getDriverName(@NonNull DataSource ds)
{
    DBC.PRE.assertNot∧null(ds, "The data source to obtain the driver for must not be null.");
    if ("WrapperDataSource".equals(ds.getClass().get SimpleName()))
    {
        try
        {
            // Case 1: JBoss uses its own funny data source wrapper thingumbob.
Agenda

- Motivation
- What is automatic test
- Anti-patterns catalog
- Basic mistakes
- Best practices
- Summary
- Questions
Best Practices

► Same quality as production code
► Common sense
► No logger errors even from tests (The Loudmouth)
► Use test coverage tools
► Do NOT be afraid to rewrite/delete badly written complex tests
► TDD
Agenda

- Motivation
- What is automatic test
- Anti-patterns catalog
- Basic mistakes
- Best practices
- Summary
- Questions
Do NOT repeat mistakes

Manual testing is boring but false alarms are pain as well

A good programmer is someone who always looks both ways before crossing a one-way street. ~Doug Linder
Agenda

► Motivation
► What is automatic test
► Anti-patterns catalog
► Examples
► Best practices
► Summary
► Questions
Links

► http://en.wikipedia.org/wiki/Software_testing
► http://www.exubero.com/junit/antipatterns.html
► http://blog.james-carr.org/2006/11/03/tdd-anti-patterns/
Questions ... 

... and maybe answers
Thank you

Ing. Jiří Kiml,
21/11/2013, ZCU
What is automatic test

► No definition in wiki ;-((
► ..... example in eclipse
Basic mistakes

► No tests
► Manual (missing) Assertions
► Redundant Assertions
► Using the Wrong Assert
► Only Easy/Happy Path Tests
► Complex Tests
► External Dependencies
► Catching Unexpected Exceptions
► Mixing Production and Test Code
No tests

► Still common in many companies
► “It is too expensive to write tests.”
► “We will write tests later”. (never)
► The Liar, Success Against All Odds
No tests

// TODO: This tests need to be activated when nightly db-reset is done!
@Ignore
@Test
public void testXXX() {
    ...
}

@Test
public void testYYY() {
    // TODO implement me
}

// TODO implement me
Manual (missing) Assertions

► Sysout instead of assert
► Why not just use a debugger?
► The Secret Catcher

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."
Brian W. Kernighan
Manual (missing) Assertions

public void testCalculation() {
    deepThought.calculate();
    System.out.println(deepThought.getResult());
}

public void testCalculation() {
    deepThought.calculate();
    assertEquals(42, deepThought.getResult());
}
Redundant Assertions

- Always true or always false assertions
- Introduced by mistake or for debugging purposes
Redundant Assertions

```java
public void testSomething() {
    ....
    assertEquals(10.0, updatedEntity.getQuantity(), Double.MAX_VALUE);
    assertTrue(true);
}

public void testSomething() {
    ....
    assertTrue(existingServiceEntryTOs.size() == 0);
    assertNotNull(existingServiceEntryTOs);
}
```
Using the Wrong Assert

- There is more than `assertTrue`
- `java assert keyword`
Using the Wrong Assert

assertTrue("Objects must be the same", expected == actual);
assertTrue("Objects must be equal", expected.equals(actual));
assertTrue("Object must be null", actual == null);
assertTrue("Object must not be null", actual != null);

assertSame("Objects must be the same", expected, actual);
equals("Objects must be equal", expected, actual);
assertNull("Object must be null", actual);
assertNotNull("Object must not be null", actual);
Only Easy/Happy Path Tests

- Only expected behavior is tested
- Only easy to verify things are tested
- The Liar, The Dodger, Success Against All Odds
Only Easy/Happy Path Tests

```java
@Test
public void testGoodCase()
{
    Assert.assertEquals(2.0, Math.sqrt(4.0), 0.0);
}

@Test
public void testInteger()
{
    Integer intToTest = new Integer(10);
    Assert.assertEquals(intToTest.intValue(), 10);
}
```
Complex Tests

► Same rules like for production code
Complex Tests

In general, you should refactor a test until it can be easily recognised to follow the general structure:

- Set up
- Declare the expected results
- Excercise the unit under test
- Get the action results
- Assert that the actual results match the expected results
(External) Dependencies

► External dependencies that code may need to rely on to work correctly.
► Dependencies between tests – one test prepares data for other one.
► Dependencies to object internal state
Catching Unexpected Exceptions

► Test succeeds even if an exception is thrown
► The Liar, The Greedy Catcher

```java
public void testCalculation() {
    try {
        deepThought.calculate();
        assertEquals("Calculation wrong", 42, deepThought.getResult());
    }
    catch(CalculationException ex) {
        Log.error("Calculation caused exception", ex);
    }
}
```
Catching Unexpected Exceptions

```java
public void testCalculation() {
    try {
        deepThought.calculate();
        assertEquals("Calculation wrong", 42, deepThought.getResult());
    } catch(CalculationException ex) {
        fail("Calculation caused exception");
    }
}
```
private List<NewServiceEntryTO> 
startCreateDefaultServiceEntriesWithoutParameter(
    final ExistingSessionTO existingSessionTO)
{
    try
    {
        return createDefaultServiceEntriesWithoutParameter(existingSessionTO);
    }
    catch (RuntimeException e)
    {
        // it's not a fail, it's an epic fail!
        fail("Creation must not fail.");
        logger.error(e.getMessage());
    }
    return new ArrayList<NewServiceEntryTO>();
Mixing Production and Test Code

- More complex packaging
- Ability to test package private methods
/**
 * Attempts to extract the DBMS driver name from the given data source. Because the
 * DS does not provide any API supporting such an endeavor we
 * resort to fudy babysitting of particular implementations. the resulting
 * lowercase string will hopefully contain "sql" or "oracle" which allows
 * detection of the dialect. Be prepared to patch this method when the container
 * environment changes.
 * In case of emergency procure a sixpack of beer and dial ++41 76 134 80 85 ;)
 * @param ds
 * @return a lowercase string reflecting the DBMS "driver" used by the DS.
 */
private static String getDriverName(@NonNull DataSource ds)
{
    DBC.PRE.assertNonNull(ds, "The data source to obtain the driver for must not be
    null.");
    if ("WrapperDataSource".equals(ds.getClass().getSimpleName()))
    {
        try
        {
            // Case 3: JBoss uses its own funny data source wrapper thingomug
        }
    }