Testing, An Introduction

Pieter van den Hombergh
Stefan Sobek

Fontys Hogeschool voor Techniek en Logistiek

February 6, 2018

Introduction

- What is testing?

Introduction

- What is testing?

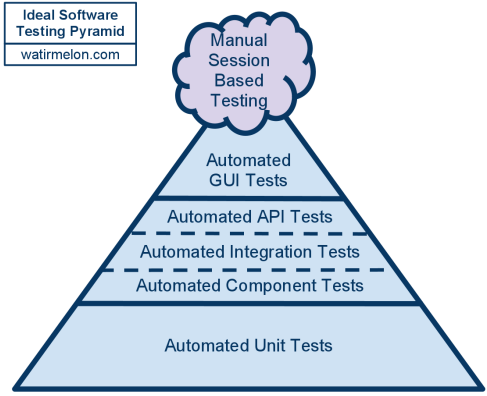
New, New

- You will learn to work with new tools, techniques and processes.
- We will introduce new tools and concepts almost every week. The pace is high.
- Testing itself, however, is really quite simple.

OO and Types Of Tests

In object oriented development, like in Java, we can distinguish several kinds of test. They can be ordered in a layer hierarchy. The top and bottom typically have common names. The three main test types are:

- **End to End Tests** or **UI** tests at the top
- **Unit Tests** at the bottom
- **Integration Tests** in the middle. In the middle layers you see various other names as well

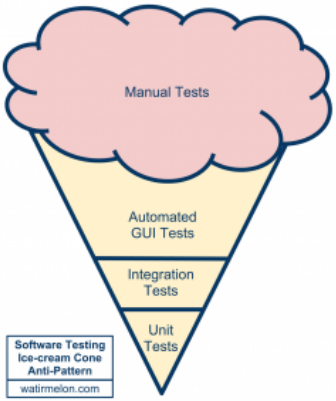


The idealized test pyramid.

The picture describes the ideal world.

Most testing reality

- In reality there is a triangle, but of a different kind.
- Much nicer  . Sometimes a bit messy and may give slippery floors. 
- Which model do you think is best? The A) ice-cone or the B) Pyramid?



The ice cream cone.

The importance of Unit test in the OO world

- If you answered B) Pyramid, you may stay.

When an application is developed in OO style, the application is developed by breaking the whole problem into smaller sub-problems, like classes and objects, and then the application is build up by assembling it from the classes.

The unit tests tests the classes, the building blocks of the application and with it testing of these building blocks forms the foundation on which you rely.

Proper Unit Testing is the foundation of test automation and continuous building and delivery.

What are unit tests?

- Unit tests test a UNIT, duh.
- The UNIT is ONE class, the **S**ystem **U**nder **T**est (SUT)
- Intended to make your code *fail fast*
  - Meaning it should run fast.
  - It should NOT test other classes, the Dependent On Component (DOC) or collaborators.
- and help make it correct
- Often you see JUnit used as the testing framework, That does not make the tests unit tests. For instance it may very well be used in integration and even end to end tests.

Creating tests is easy

- Using Netbeans IDE.
- Letting netbeans guess the SUT code is easy too.
- let's TDD

DEMO TIME

How to Assert

The term is asserting: it is Okay. Test the value you you expect with the value you get.

- We will be using JUnit.
- Looking more precisely to the details.

to assert or not to assert

- `assert` is a java keyword

```
assert args.length > 0;
```

- assert test the expression following the keyword to be true.
  - If not it throws an `AssertionError` exception.
- Such assert tests are off by default but can be turned on by starting the JVM with `-ea` (for enable assertions). Demo.

Although `assert` has its purpose, it is typically not used very often. In particular not for testing.

Order of parameters and message

- Looking at org.junit.Assert.java we see a class with only static methods of the type: (equals as example)
  `static public void assertEquals(String msg, Object expected, Object actual )`.
- Order:
  1. Message
  2. Expected value expression
  3. Actual value (in case of equals, same)
  4. delta in case inexact numbers (double of float values or arrays of) are compared. Margin of error allowed.
- It is good style to always add a meaningful or identifying message, because it helps (you) to find the exact spot where things went wrong. More so if your test method contains more then one assert.
- Let us have a look (demo)

The Zoo of Asserts

True	expression must be true
False	expression must be false
Null	expression must be null
NotNull	expression must be not null
Equals	expected and actual expression must be equals according to `Object.equals( Object )`
Same	**Quiz:** what is the difference with equals?

Arrays can be compared too

- Note that size and order of elements matter.
- If you want to assert that arrays contain the same members, irrespective of order, sort the arrays, then compare, or use a assert framework that has more elegant tests, e.g AssertJ.

Sometimes simple assert is not enough

- Then you can implement your own 'rules' matching the business rule.
- The order is msg, actual, matcher, and not what you would expect from (msg, expected, actual).
  - Reason (I infer) is that the matcher is typically a anonymous inner class, because you implement it on the spot. Having this anonymous inner class in the middle instead of at the end would give even more awkward code.
- Example on next slide.

Cola light; machtes

```
        assertThat( "my tastbuds have been violated! ", taste,
                new BaseMatcher() {
            @Override
            public boolean matches( Object item ) {
                Taste t = (Taste) item;
                switch(t) {
                    case COLA: case COGNAC:
                    case WHISKEY:
                        return true;
                        default:
                    case COLALIGHT:
                        return false;
                }
            }

            @Override
            public void describeTo( Description description ) {
                description.
                        appendText( "HOM hates light stuff, "
                                + "it is for sissies or people "
                                + "without discipline");
            }
        });
```