cphbusiness

COPENHAGEN BUSINESS ACADEMY

System Testing

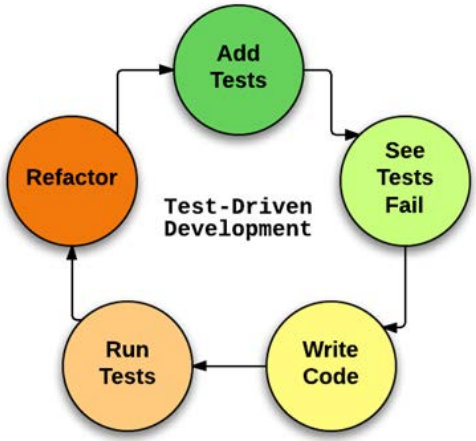
When, why and how

Agenda

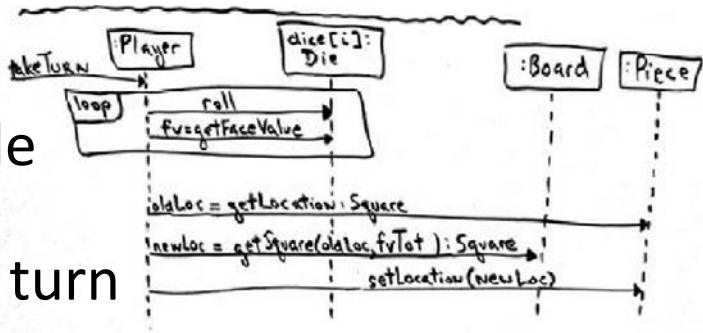
- Recap on TDD workshop (Monday before Easter)
- Follow up on Midterm Assignment
- Introduction to System Testing
- Introduction to Automated System Testing
- Guest lecturer on Test Automation
- More exercises with Automated System Testing (today's focus)

What did we do in TDD Workshop Monday before Easter

- **TDD principles**



- Practice getting into the TDD mindset **by example**
- Use **mock objects** to support TDD cycle
- Workshop result for Monopoly – take turn
  - https://github.com/Tine-m/monopoly-with-mocking



Midterm Assignment Follow Up

- Intention /learning objectives
- How to be used in exam
- Anything needs recapping?

Midterm Assignment project 1

- **Demonstrate your solution to the exercise "Testing Real Life Code".** *You should (as a minimum):*

- *Explain the purpose of the Test (what the original test exposed, and what your test exposes)*
- *Explain about **Parameterized Tests** in JUnit and how you have used it in this exercise.*
- *Explain the topic **Data Driven Testing**, and why it often makes a lot of sense to read test data from a file.*
- *Your answers to the question; whether what you implemented was **a Unit Test or a JUnit Test**, the problems you might have discovered with the test and, your suggestions for ways this could have been fixed.*
- *The steps you took to include **Hamcrest** matchers in the project, and the difference they made for the test*

Midterm Assignment project 2

- ***Demonstrate your solution to the exercise "Unit Testing, Testable Code, Mocking and Code Coverage".***
- *You should (as a minimum):*

- *Explain the necessary steps you did to **make the code testable**, and some of the patterns involved in this step*
- *Execute your test cases*
- *Explain basically about **JUnit, Hamcrest, Mockito and Jacoco**, and what problems they solve for testers*
- *Demonstrate how you used Mockito to mock away **external Dependencies***
- *Demonstrate how/where you did **state-based testing** and how/where you did **behavior based testing***
- *Explain about **Coverage Criterias**, using the results presented by running Jacoco (or a similar tool) against you final test code.*
- *Explain/demonstrate **what was required** to make this project use JUnit (Hamcrest), Mockito and Jacoco*

Midterm Assignment project 3

- **Demonstrate your solution to the exercise "Monopoly".**
  You should (as a minimum):

- *Explain your **solution** with focus on its **design**. Does it have low coupling, interfaces, polymorphism, Inversion of Control and Dependency injection as **central design principles**? If not, why? Otherwise, demonstrate it!*
- *Explain your **test case design activity** and argument for the choices that you have made:*
  - *how to choose the right **test conditions** (i.e. what to test which in principle will be all code when you are a TDD'er ☺)?*
  - *what **test design techniques** to use (both consider black-box techniques and white-box technique)?*
  - *how much effort to put into each test condition (how critical is the item will influence it's test coverage percentage)?*

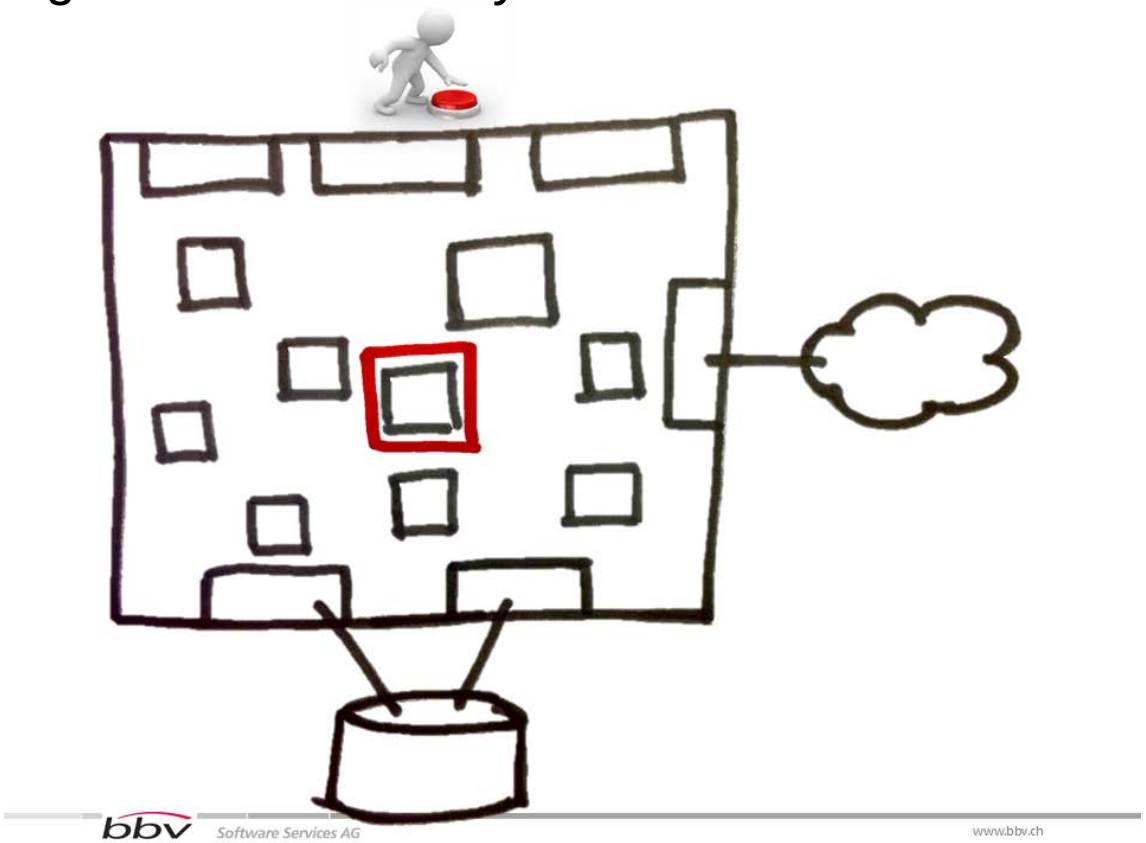
Today's Learning Objectives

When this day/topic is over, including its accompanying exercises you should be able to explain about:

- Define the term **System Testing** and it purpose, both in terms of V-model and in agile context
- Explain the "kind of tests" that often are associated with System Testing and the difference between functional and non-functional testing
- Who is (often) carrying out the System Test (would your answer be the same for an organization following the V-model versus a modern agile organization?
- What is the difference between **System** and **Acceptance** testing
- Discuss Pros and Cons with **manual** versus **automated** tests
- Explain about Mike Cohn's Test Pyramid and why it's relevant for test planning
- Discuss some of the problems with automated GUI tests and what makes such test "vulnerable"
- Explain about the **Selenium**, its purpose, and how it "fits into" the testing world
- Demonstrate, using practical examples, how you have tested Web-UI's using Selenium.
- Explain the difference between a Headless browser, versus a GUI-based browser, and the pros & cons of each
- Explain the problems we face when testing modern AJAX/JavaScript applications and demonstrate, using practical examples, how you have solved them

Unit Testing = Test in Isolation

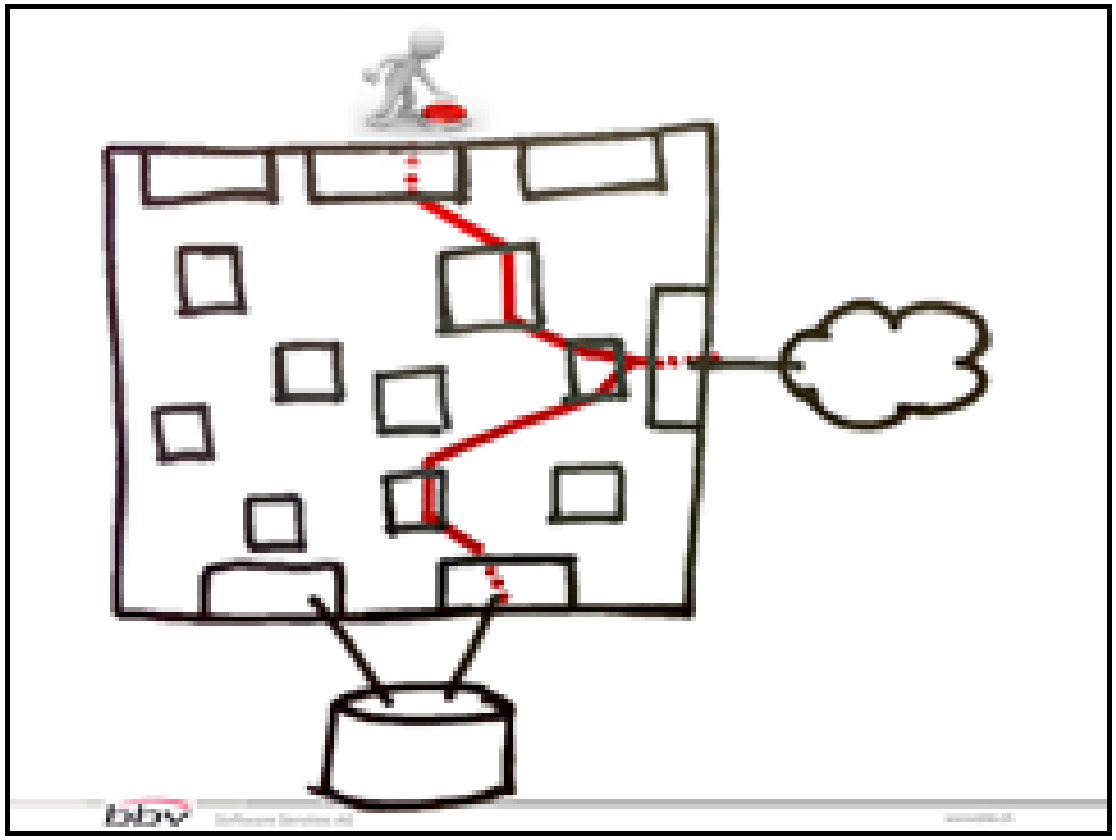
- Tests a single class in the system – isolated from the rest.



Source: http://www.planetgeek.ch/2012/06/12/acceptance-test-driven-development/

System Testing = Test of Big Picture

- Tests functionality at system level

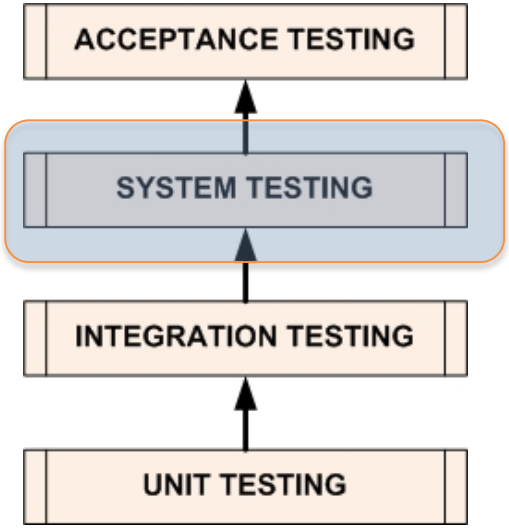


Source: http://www.planetgeek.ch/2012/06/12/acceptance-test-driven-development/

System Testing Definitions

Definition according to ISTQB

A level of the software testing process where a complete, integrated system/software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.



Definition according to Wikipedia

https://en.wikipedia.org/wiki/System_testing

System Testing
What and how are we testing

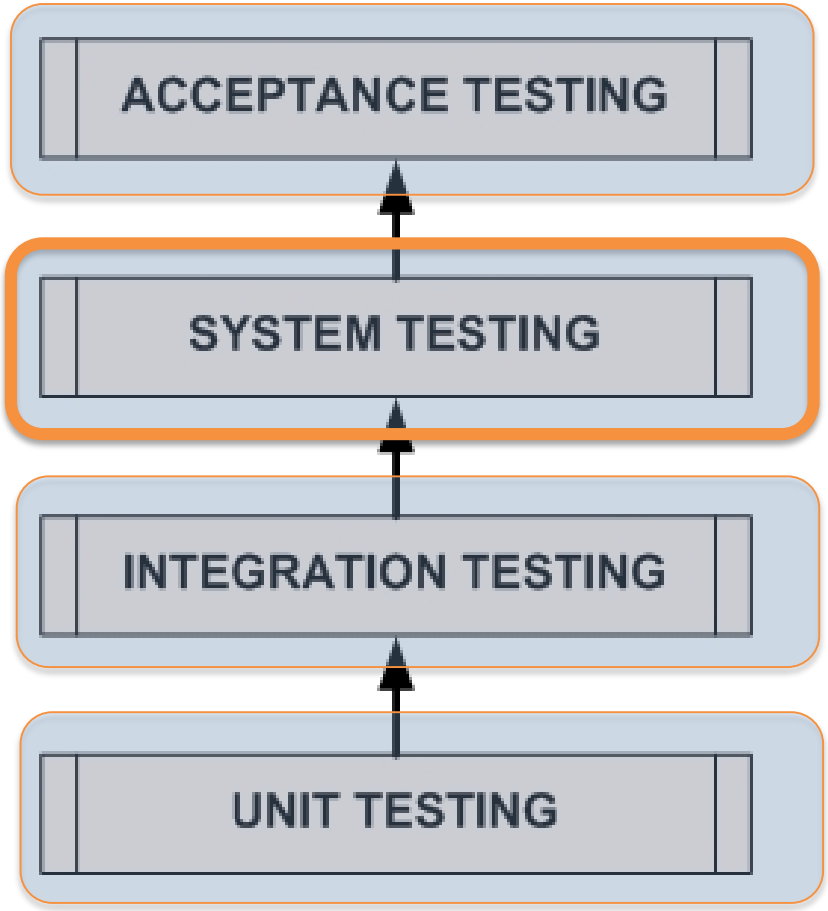
What are we testing:

- Functional Requirements
- Non Functional Requirements
  - Usability Testing
  - Performance
  - Load Testing
  - Security
  - Installation Testing
  - …. See Wikipedia for a much longer list

How are we testing:

- Usually black box testing
- Preferably the test are automated

Software Testing Levels
Class Discussion



What is Acceptance Testing
Who writes the test cases
Who/what executes the test cases

?

What is System Testing
Who writes the test cases
Who/what executes the test cases

?

What is Integration Testing
Who writes the test cases
Who/what executes the tests cases

?

What is Unit Testing
Who writes the test cases
Who/what executes the test cases

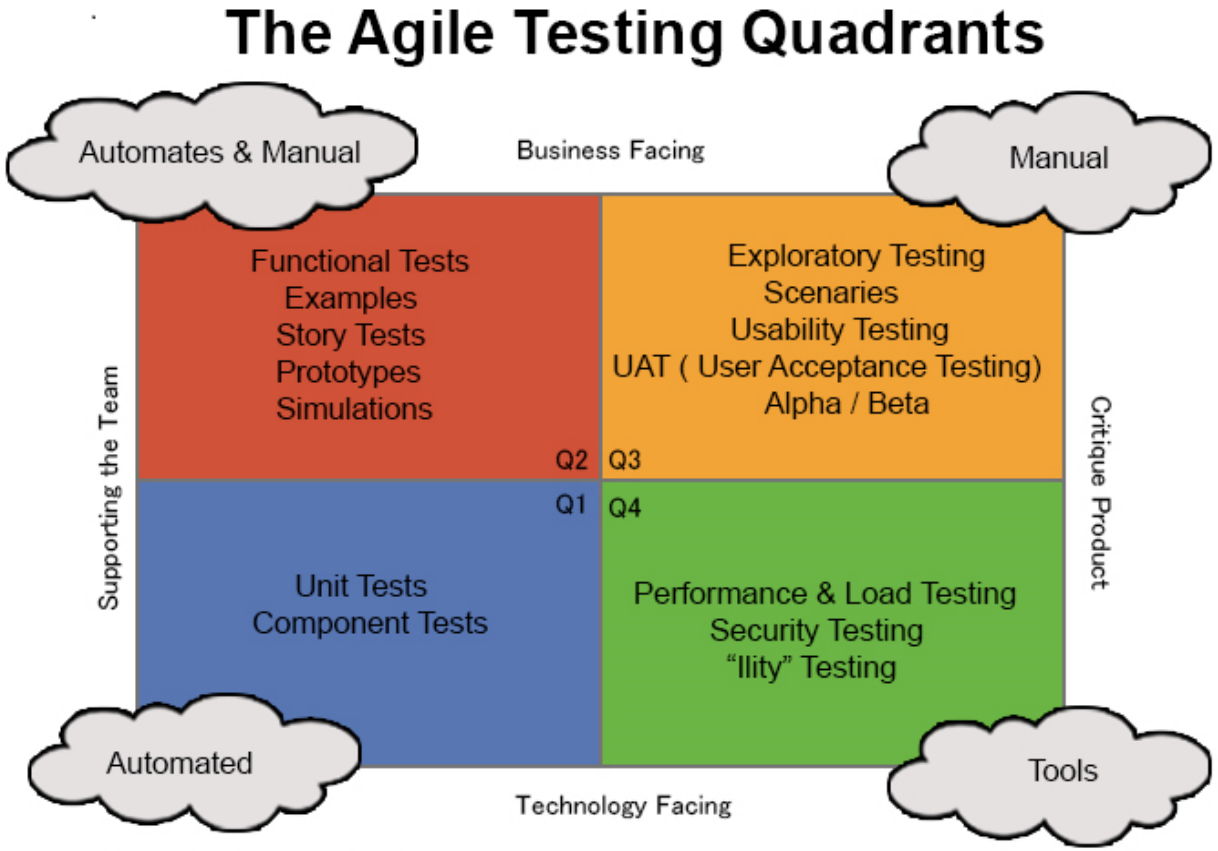
?

Other Names for System Testing?

???

Where are we in Agile Context?

The quadrant numbering system does not dictate some order of testing you work through as in Waterfall. It is an arbitrary numbering used as a reference when referring to the matrix.



Source: Lisa Crispin, Brian Marick

Why System Testing?

Basic assumption:

- Test of units cannot guarantee a correctly functioning system
  - Some defects can only be revealed at system level

Purpose:

- Demonstrates that SUT implements the requirements
  - Are we building the right system?
  - Can we ship?

Problem with System Tests 1

They are slow:

- Selenium tests: 0.1 / second

Total test count: 63, total duration: 9m:15s

Status	Test	Duration	Order#
OK	Authentication (authentication.test)	6s,205ms	1
OK	Component Library menu testing (ComponentLibrary.test)	18s,390ms	2
OK	Input Templates Menu (ComponentLibrary.test)	1s,792ms	3
OK	Result Templates Menu (ComponentLibrary.test)	1s,154ms	4

- Unit tests: 61 / second

Total test count: 1845 (8 ignored), total duration: 30s

Status	Test	Duration	Order#
OK	TfBrowserCacheStrategyFactory: Create_Event_Does_Not_Override_Bypass (Wizer.UnitTest.dll Wizer.UnitTest.BrowserCache)	321ms	1
OK	TfBrowserCacheStrategyFactory: Create_Event_Overrides (Wizer.UnitTest.dll Wizer.UnitTest.BrowserCache)	13ms	2
OK	TfBrowserCacheStrategyFactory: Create_NoEvent (Wizer.UnitTest.dll Wizer.UnitTest.BrowserCache)	15ms	3
OK	TfBypassStrategy: GetCacheId (Wizer.UnitTest.dll Wizer.UnitTest.BrowserCache)	1ms	4

Source: http://zealake.xpqf.com/2016/03/13/super-fast-end-to-end-tests/

Problem with System Tests 2

They give poor feedback:

- End-to-end tests: "something went wrong"

- Unit tests: the problem is "in this method in this class"

Problem with System Tests 3

They are complicated:

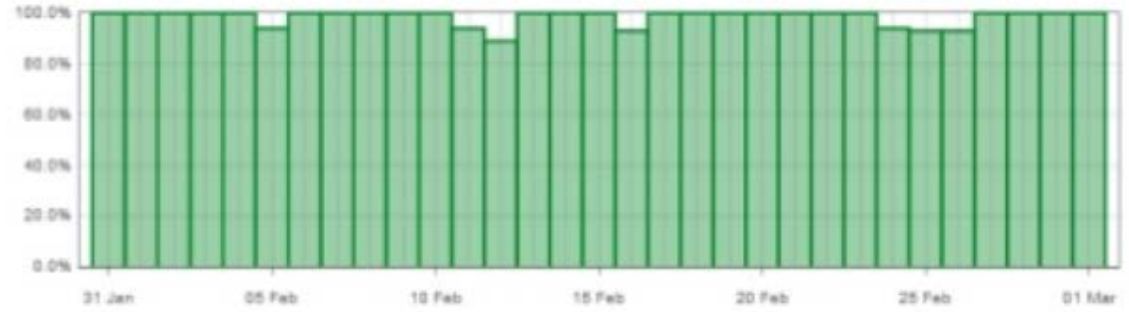
- Load specific set of test data into the database
- Run the application on a web server
- Start a browser
- Writing the test
  - Depend on test data loaded elsewhere
  - Ignore side-effects of other tests

Problem with System Tests 4

They are fragile:

- End-to-end tests fails sporadically:

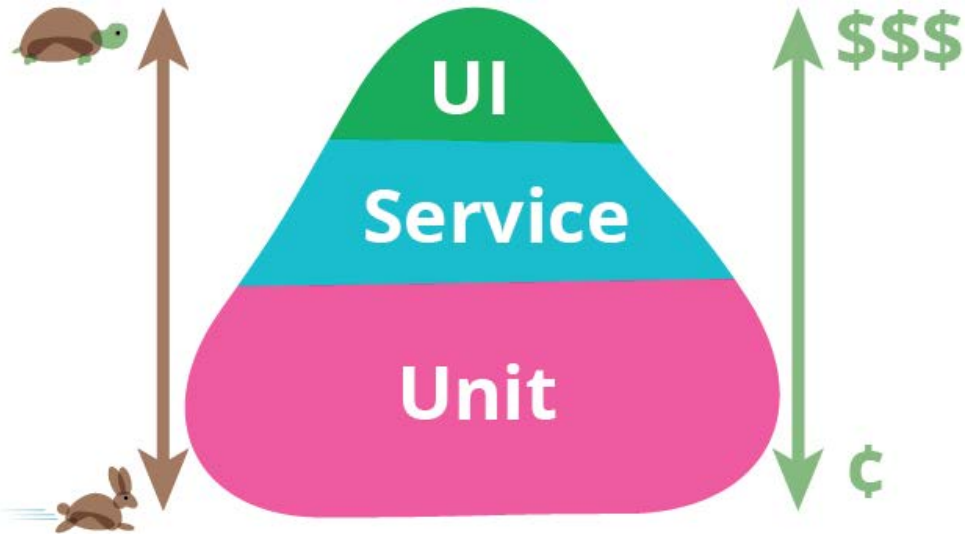
- Unit tests fails for a reason:



The Testing Pyramid (Mike Cohn)

The essential point of the Testing Pyramid is that you should have many more low-level unit tests than high level end-to-end tests running through a GUI since:

- Cost of tests goes up, as you go up in the pyramid
- UI tests are typically slow
- UI test typically requires a much larger setup



https://martinfowler.com/bliki/TestPyramid.html

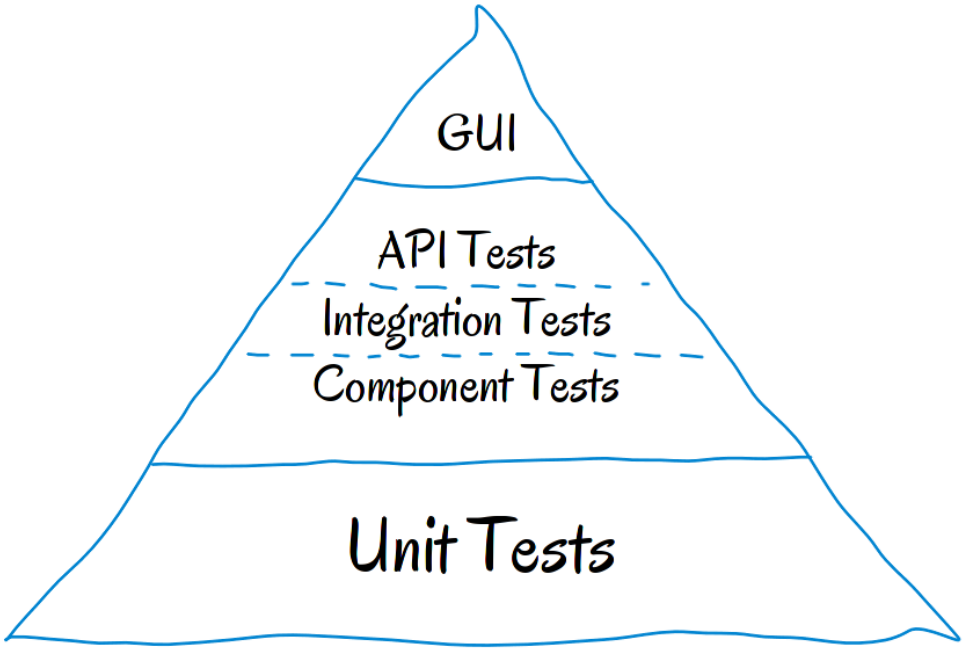
The Testing Pyramid (Mike Cohn)

Martin Fowler's view on high-level tests:

*"high-level tests are there as **a second line of test defense**.*

If you get a failure in a high level test, not just do you have a bug in your functional code, you also have a missing unit test."

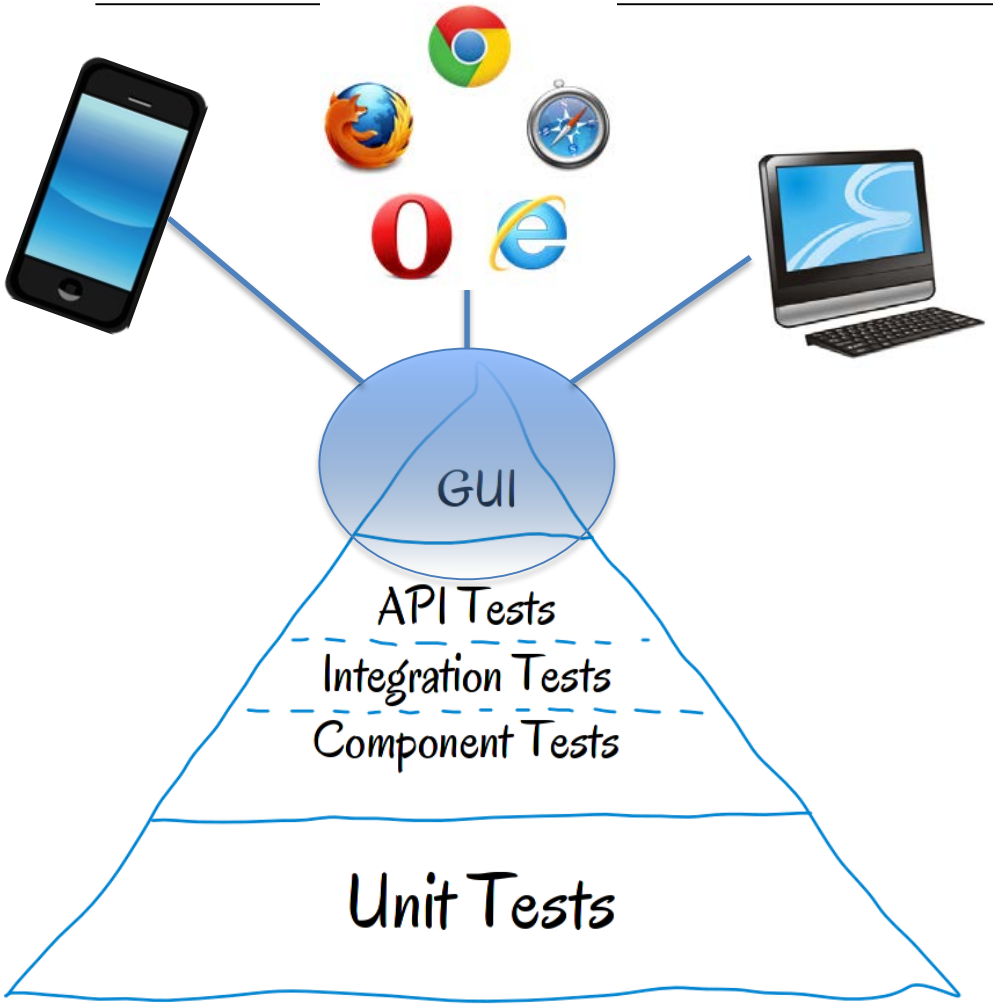
The Testing Pyramid
Detailed (just a little bit ;-)



The three main layers can be clearly distinguished, but the center layer is more detailed as sketched in this version of the pyramid

The Testing Pyramid

Year 2017, what is a GUI in a modern software architecture ?



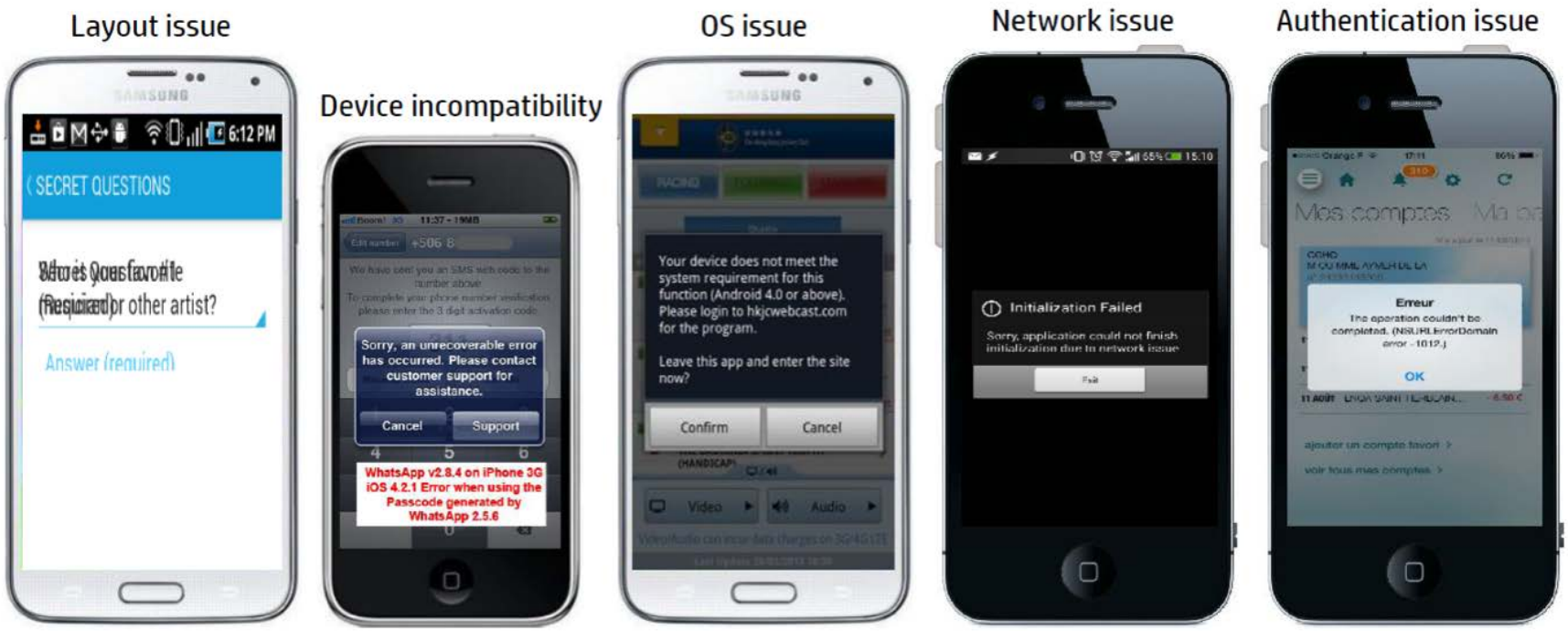
- Make sure sufficient resources have been used on lower (less expensive) layers before testing GUI(s)
- If at all possible, automate the GUI tests

We cannot expect to get one tool that "fits all". Today we will focus on System Testing Tools involving only:

Functional Testing and further limit our scope to **Web applications only**

The Mobile Testing Pyramid

Mobile testing involves more complexity compared to web UI testing.
A few examples of mobile testing challenges:



Selenium
Class Discussion

We have taken the decision to go with Selenium

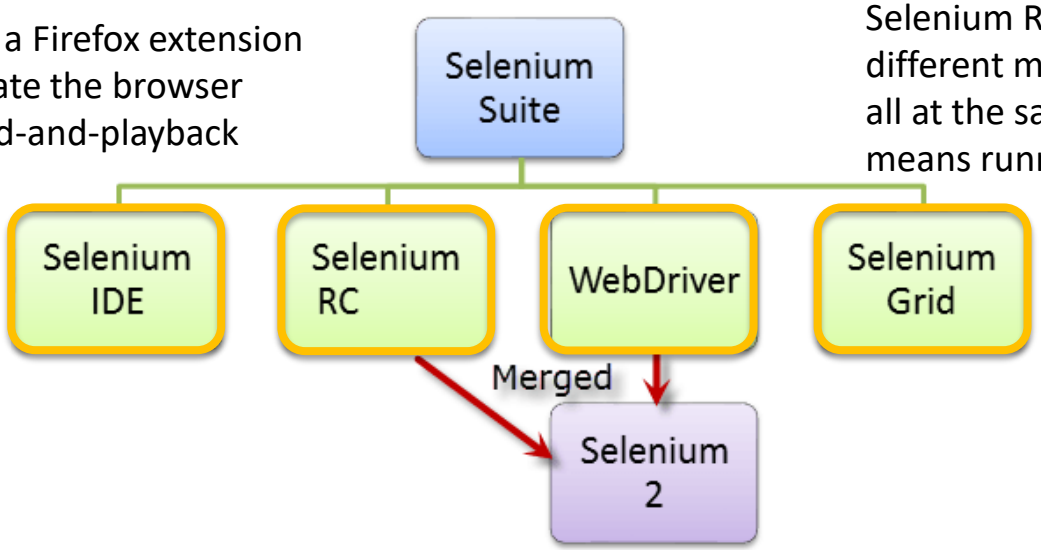
Spend a few minutes on the WEB and come up with answers to the following questions

- What is Selenium?
- What can Selenium do for you?
- Why Selenium?

What is Selenium ?

Selenium is a free (open source) automated testing **suite** for web applications across different browsers and platforms

Selenium IDE is a Firefox extension that can automate the browser through a record-and-playback feature



Selenium Grid is a tool used together with Selenium RC to run parallel tests across different machines and different browsers all at the same time. Parallel execution means running multiple tests at once.

Selenium RC was the first automated web testing tool that allowed users to use a programming language they prefer. Supports the following languages:

- Java
- C#
- PHP
- Python
- Perl
- Ruby

The **WebDriver** proves itself to be better than both Selenium IDE and Selenium RC in many aspects. It implements a more modern and stable approach in automating the browser's actions.
WebDriver, unlike Selenium RC, does not rely on JavaScript for automation. It controls the browser by **directly communicating to it**.
Supports these languages

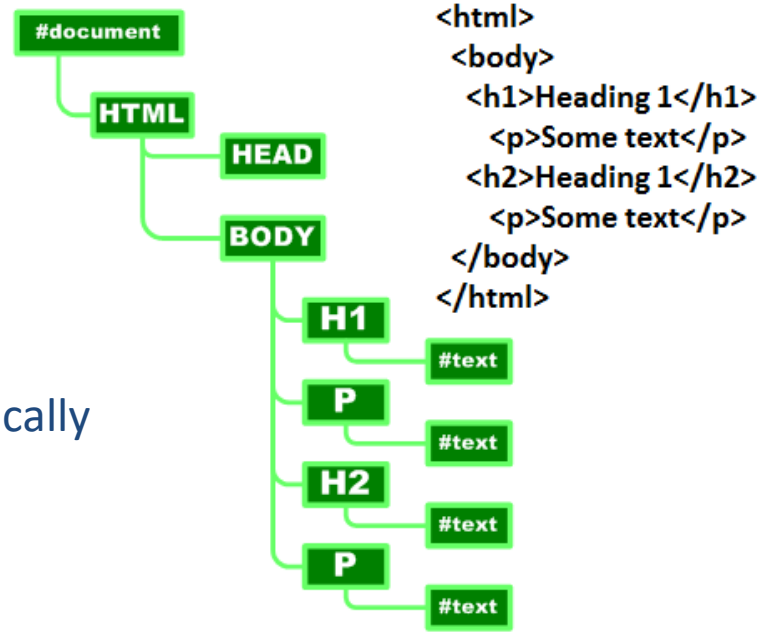
http://www.guru99.com/introduction-to-selenium.html

Using Selenium
What you need to know

What the DOM is

How to inspect DOM elements in your browser (just right click the element)

That the DOM can be manipulated programmatically (which is what you will do over and over in your Selenium tests)



Selenium webdriver Setup - 1

You can use Selenium with/from **Java, C#, JavaScript** etc. The examples in the following, and the class demo, will use Java, but it should not be a big deal to use whatever language you prefer, see here for instructions

Create a new Java-maven project and add these dependencies to your pom-file (check here for the newest version)

```
<dependency>
   <groupId>org.seleniumhq.selenium</groupId>
   <artifactId>selenium-java</artifactId>
   <version>3.3.1</version>
</dependency>
<!-- not really needed for the exercises, see link for
details
<dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-server</artifactId>
      <version>3.0.1</version>
 </dependency>  -->
<dependency>
   <groupId>org.seleniumhq.selenium</groupId>
   <artifactId>htmlunit-driver</artifactId>
   <version>2.25</version>
</dependency>
```

Selenium webdriver Setup - 2

The project we have so far, will be sufficient, if you can live with only the `htmlunit-driver` installed via maven.
This is a driver for the HtmlUnit headless browser, which are the fastest of all drivers, but NOT graphical.

Drivers (graphical) for FireFox, Chrome etc. **cannot** be installed with Maven with Selenium2.
So create a folder **`drivers`** somewhere on you system, download, unpack and copy the drivers for FireFox (take version 14) and Chrome into this folder.
http://docs.seleniumhq.org/download/

Now either set the path to this folder in you OS-environment variables or (my recommendation), add the following lines to the top of all your selenium projects:

```
System.setProperty("webdriver.gecko.driver", "YOURPATH\\drivers\\geckodriver.exe");
System.setProperty("webdriver.chrome.driver","YOURPATH\\drivers\\chromedriver.exe");
```

See here for details (related to the gecko-driver = Firefox):
http://toolsqa.com/selenium-webdriver/how-to-use-geckodriver/

Using Selenium

With all this done, lets start using Selenium. Create a class `Selenium2Example` in your project and copy the example code into this class:

Add-on to Selenium Google example

1. Manually inspecting `System.out.println()` in an output window doesn't seem efficient
   - Change the Google cheese search program into a JUnit test in order to replace S.O.P with Assert method

2. Testing without opening the browser is more lightweight and efficient.
   - Change the test into a non-GUI version using the HTMLDriver instead of Firefox (or whatever graphical driver you have been using so far)

3. Go back to the graphical version of Google cheese program
   - Can you do without the WebDriverWait? What is it doing?

The 7 Basic Steps of Selenium Tests

There are seven basic steps in creating a Selenium test script, which apply to any test case and any system under test (SUT).

1. Create a WebDriver instance.
2. Navigate to a Web page.
3. Locate an HTML element on the Web page.
4. Perform an action on an HTML element.
5. Anticipate the browser response to the action.
6. Run tests and record test results using a test framework.
7. Conclude the test.

Selenium-WebDriver API Commands and Operations

If you know how to investigate the DOM (you should ;-) almost all you need can be found here

Waiting with WebDriver:
http://www.seleniumhq.org/docs/04_webdriver_advanced.jsp#explicit-and-implicit-waits

Selenium-WebDriver API Commands
Using the findElement(s) method

Locate UI element (called `WebElement`)

- Use `findElement` method

Locator strategies (using `By`):

- By ID (most efficient and preferred if unique id's exist on page)
- By class name
- By tag name
- By name
- By link text
- By partial link text
- By CSS
- By XPath (uses browser's native XPath capabilities if supported)

Having this HTML code:

```
<input type="text" name="passwd" id="passwd-id" />
```

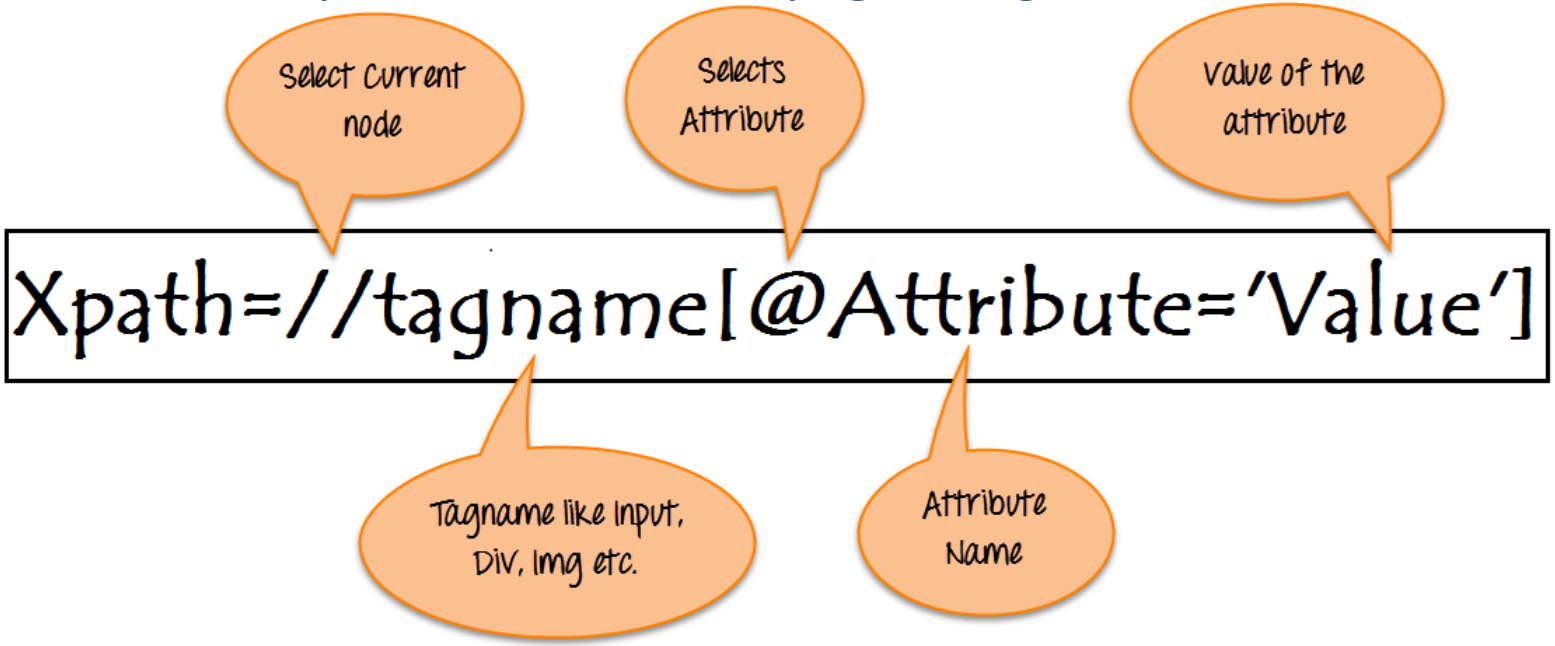
you could find it using any of:

```
WebElement element;
element = driver.findElement(By.id("passwd-id"));
element = driver.findElement(By.name("passwd"));
element = driver.findElement(By.xpath("//input[@id='passwd-id']"));
```

XPath

XPath

XPath (XML path) is a syntax, or language, for finding elements on a web page using XML path expressions. XPath is used to find the location of any element on a webpage using HTML DOM structure



Use one of these as reference:
XPath Syntax: https://www.w3schools.com/xml/xpath_syntax.asp
XPath with Selenium, tutorial: http://www.guru99.com/xpath-selenium.html

XPath

A few Xpath examples to get you started:

```
<table>
  <thead>
    <tr>
      <th>A</th><th>B</th><th>C</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>aa-1</td>
      <td>bb-1</td>
      <td><a href="#">Go</a></td>
    </tr>
    …
  </tbody>
</table>
```

A	B	C
aa-1	bb-1	Go
aa-2	bb-2	Go
aa-3	bb-3	Go

Does the `tbody` contain n-elements?

```
driver.findElements(By.xpath("//tbody/tr")).size();
```

Does the `first row` contain a column with the value 'bb-1' ?

```
driver.findElements(By.xpath("//tbody/tr[1 and td ='bb-1']").size()
```

Does a `row` with a column with the value 'bb-2' contain an anchor-tag we can click ?

```
WebElement row = d.findElement(By.xpath("//tbody/tr[td='bb-2']"));
WebElement a = row.findElements(By.tagName("a")).get(0);
a.click()  //Click the tag
```

Exercises

Do these exercises

Bootcamp 2 (use of css as element locator)

Bootcamp 3