Behavior Driven Testing with Cucumber demystified

KVIV – 11 juni 2013

Steven Wierckx

Agenda

- Who am I
- Scope
- Behavior Driven Development / Testing (BDD / BDT)
- Domain Specific Languages (DSL)
- Cucumber
- Bringing it all together

Who am I

- Steven Wierckx
- More than 10 years of experience in development and testing
- Specialist in test automation and web site security
- Personal blog: www.ihackforfun.eu
- Steven.wierckx@polteq.com

Scope of the talk

Today we will focus on the testing aspect within BDD

BDD / BDT

2 main problems in software development:

- Documentation
  - Lacking
  - Out of date
- Communication
  - Lacking
  - Different language

RESULT: Time is wasted

Test specific: what and when are we going to test?

BDD / BDT

Behavior Driven Development is a

- ✓ second-generation,
- ✓ outside–in,
- ✓ pull-based,
- ✓ multiple-stakeholder,
- ✓ multiple-scale,
- ✓ high-automation,
- ✓ agile methodology

It describes a cycle of interactions with well-defined outputs, resulting in the delivery of working, tested software

BDD / BDT

This answers two important questions on testing:

- What to test
- When to test

BDD / BDT

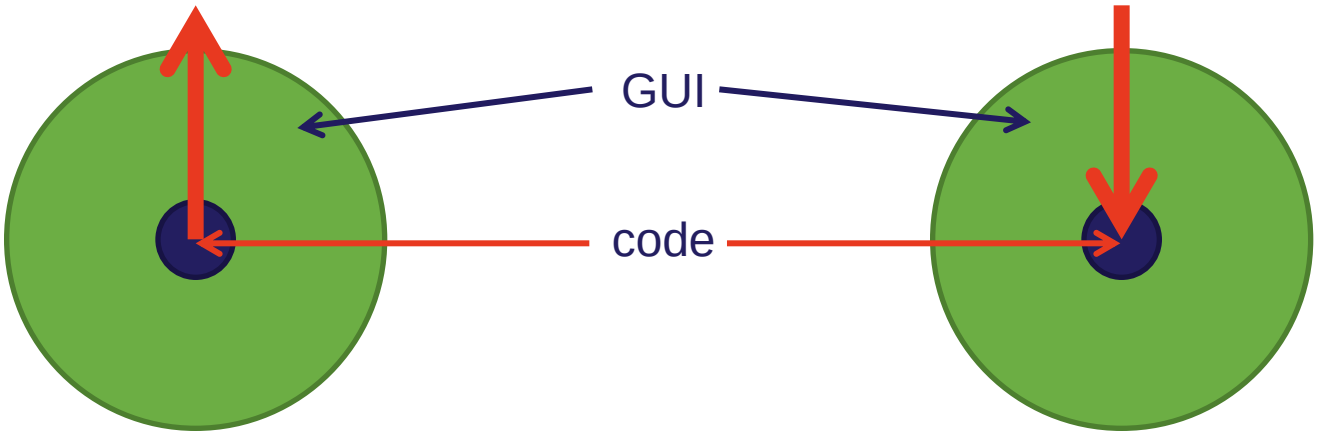
Behavior Driven Development is a

- ✓ second-generation,
- ✓ **outside–in**,
- ✓ pull-based,
- ✓ multiple-stakeholder,
- ✓ multiple-scale,
- ✓ **high-automation**,
- ✓ agile methodology

It describes a cycle of interactions with **well-defined outputs**, resulting in the delivery of working, tested software

BDT – outside-in

- Inside-out
  (unit testing)
  - Based on technical requirements
  - Small testable units
  - Tightly coupled
  - Fast
- Outside-in
  (acceptance testing)
  - Based on business requirements
  - End-user behaviour
  - Loosely coupled
  - Slow



BDT - principles

- Test Driven Development
  - define a test set for the unit first
  - then implement the unit
  - finally verify that the implementation of the unit makes the tests succeed

- Behavior Driven Testing
  - same steps as TDD
  - but describe the test as the expected behavior from that unit including the business value

BDT - specification

- Behavioral Specification is done through user stories
- User stories follow a template
  - title
  - setting (who and what)
  - scenario(s)
    - initial state
    - actions
    - expectation

BDT - example

Story: I want to log in to the application

In order to make purchases from the online store
As a registered user
I want to log in to the application

Scenario : a correct login
Given I have user credentials
When I enter my user ID
And I enter my password
And I press login
Then I should see the welcome page

BDT - conventions

- Test method names should be sentences
  - Start with “should”
  - Challenge: should it really?
- “Behavior” is more useful than “test”
  - Describe the behavior you are interested in
  - Use a common language, the Domain Specific Language (DSL)

Domain Specific Language

- Domain Specific Language: DSL
- The **syntax** and **vocabulary** we are going to use to write user stories
- The text files become readable by tools that know our DSL
- Several tools do this, we use Cucumber as example

DSL - syntax

- Syntax: the structure of the sentence

- Example: “When I enter my password”
  - When I enter “the password” in the password textbox
  - When I enter “the password” as the password
  - When I enter my password

- Example: “And I press login”
  - The test framework “sees” the verb ‘press’ and knows it will need to simulate a button being pressed

DSL - vocabulary

- Vocabulary: the name of each element of the program under test

- Example: “When I enter my password”
  - ✓The test framework should know where the field password is located

DSL - pitfalls

- Defining too many verbs
  - When I type ...
  - When I enter my password
- Not using the same syntax for all similar actions
  - When I press “name of the button”
  - When I click “name of a link”

- Written in Ruby
- Open Source
- For use in Ruby and Ruby on Rails
- Several extentions are available
  - Selenium for web testing
  - Aruba for command line testing
  - Swinger for GUI testing of JAVA/Swing applications
  - Native Ruby support for XML (web services)

Cucumber - example

Feature: I want to log in to the application

In order to make purchases from the online store
As a registered user
I want to log in to the application

Scenario : a correct login
Given I have user credentials
When I enter my user ID
And I enter my password
And I press login
Then I should see the welcome page

Cucumber - outline

```
Scenario  Outline: blending things
When I put <thing> in a blender
And I switch the blender on
Then it should transform into <other thing>

Examples: Amphibians
      |thing                |other thing       |
      |frog                 |mush              |

Examples: Electronics
      |thing                |other thing       |
      |iPhone               |toxic waste       |
      |Galaxy Nexus         |toxic waste       |
```

Cucumber - tags

@wip @slow

Scenario Outline: blending things

- Tag selection on the command-line:
  - tags = wip, slow
    - Will select all cases tagged *either* with “wip” or “slow”
  - tags = wip –tags = slow
    - Will select all cases tagged *both* “wip” and “slow”
  - tags = -slow
    - Will select all cases *except* the slow ones

Bringing things together

```
1  Feature:
2  In order to explain how all these things work together
3  As an instructor
4  I want run through the code
5
6  Scenario: running through the code
7  When I start the explenation
8  And I complete the explenation
9  Then I should see "understanding" looks from the audience
```

```
1  When /^I start the explenation$/ do
2    #Ruby code to start an explenation
3  end
4
5  When /^I complete the explenation$/ do
6    #Ruby code to complete an explenation
7  end
8
9  Then /^I should see "(.*?)" looks from the audience$/ do |arg1|
10   #Check the argument with the actual result, hopefully they match
11 end
```

Bringing things together in the SDLC

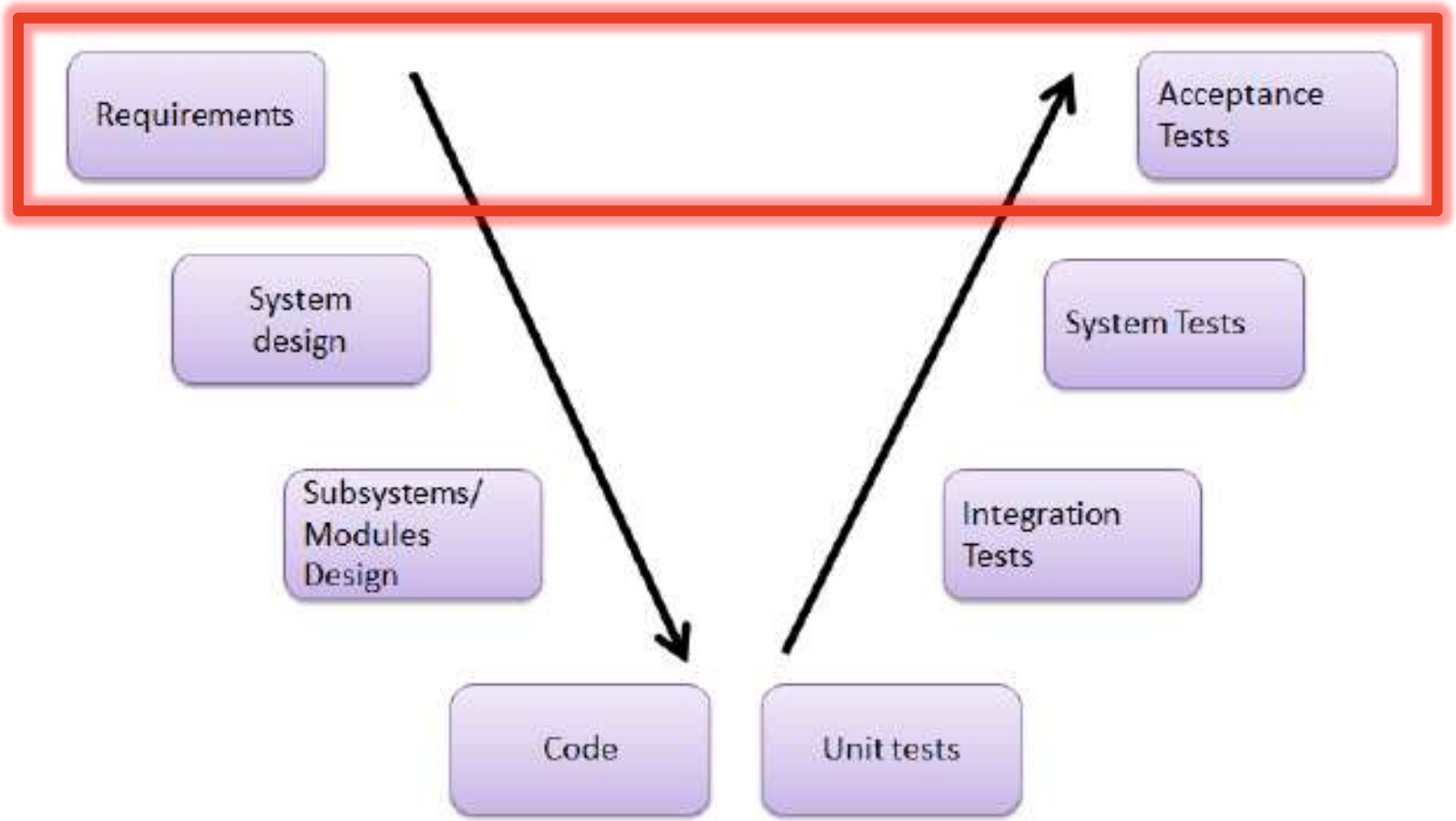


Figure 2 - V Model for software development (Adapted from Pfleeger, 2004)

Bringing things together in a team

- Senior tester(s) with domain knowledge to create the DSL
- Developer(s) to link the DSL with the Cucumber framework
- Analysts and testers to write feature files

What is missing

- Central storing of results
- Reporting