

COPENHAGEN BUSINESS ACADEMY



BDD with Cucumber

Exploratory Test

Profiling - JMeter

Today's Goals

- Develop by examples
 - BDD with Gherkin & Cucumber
- Pairwise testing
- Exploratory testing
- Non functional testing
 - JMeter

Requirement Specification in Agile

- Getting Examples

- Eliciting examples from customers is powerful way to guide development with both technology and business-facing tests.
- **Examples** form the heart of **Quadrants 1 and 2**, whether it's a paper prototype, a flow diagram drawn on the whiteboard, or a spreadsheet with inputs and expected outputs.

Question:

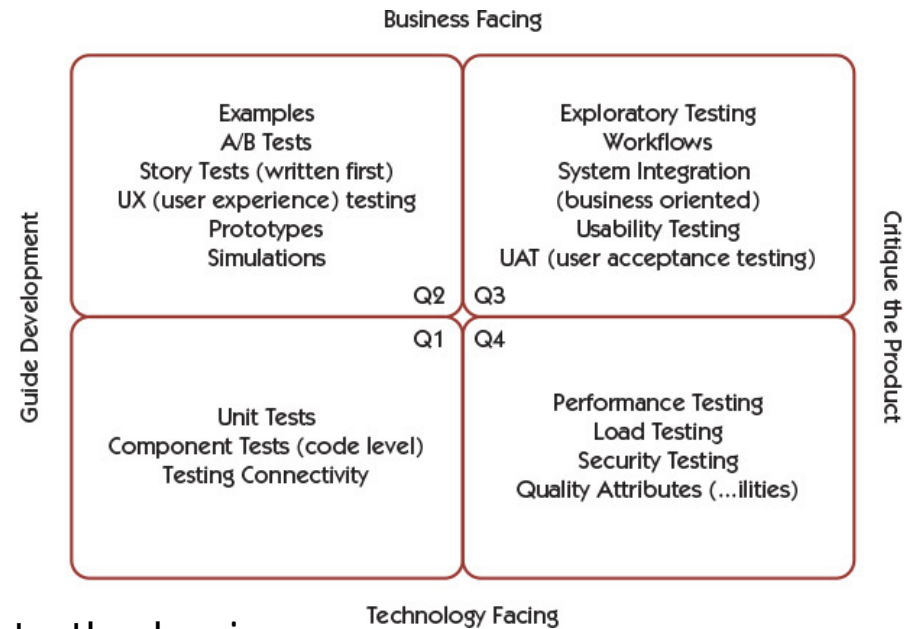
- What gives you most useful discussion with business stakeholders?
 - “Can you give me a scenario where that happens?” / “Can you give me an example?” **OR**
 - “Can you give me acceptance criteria for this?” / “Can you help me work out how to test this?”

Source: Gregory & Crispin- More Agile Testing (chap. 11)



Test Quadrants- taxonomy of testing types

- **The left side** of the quadrant matrix is about **preventing** defects before and during coding.
- **The right side** is about **finding** defects and discovering missing features, but with the understanding that we want to find them as fast as possible.
- **The top half** is about **exposing** tests to the business
- **The bottom half** is about tests that are more **internal** to the team but equally important to the success of the software product.
- “**Facing**” simply refers to the language of the tests—for example, performance tests satisfy a business need, but the business would not be able to read the tests; they are concerned with the results



Does "Getting Examples" Have a Name?

- Confusing and non standard naming for getting examples in agile development
- "Example-driven-development" never caught on, but these names are dominant:
 - Acceptance-test-driven-development (**ATTD**)
 - Specification-driven-development (**SBE**)
 - Behaviour-driven-development (**BDD**)
- BDD
 - Originally a response to TDD, BDD has evolved into both analysis and automated testing at the acceptance level.
 - Today will focus on BDD-style acceptance tests

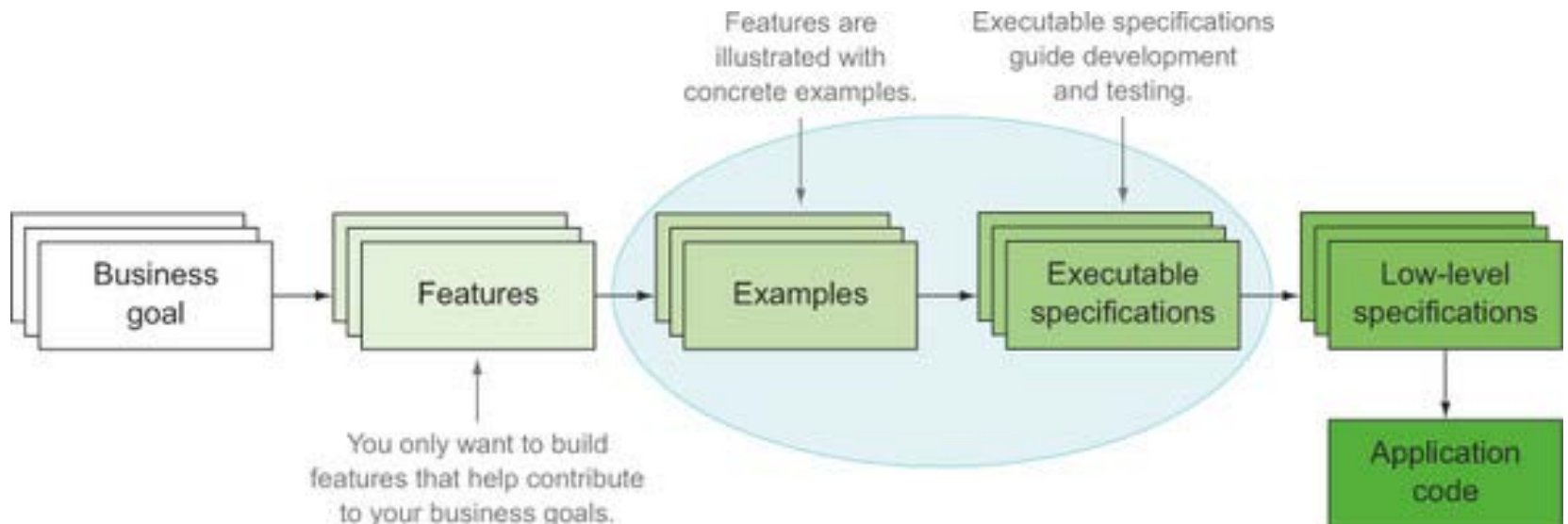
BDD (and TDD) is not about testing

- A common misunderstanding of TDD and BDD is that they are testing techniques. They're not. As the name suggests, TDD and BDD are about software *development*.
- It is the process of approaching your design and forcing you to think about the desired outcome and API before you code.
- **Automated tests are a by-product of TDD and BDD!!!**

Source: <https://docs.cucumber.io/bdd/overview/>

Executable Specifications

- If examples are expressed in a clear and precise way, they can be transformed into executable specifications and living documentation



Source: John Ferguson Smart- BDD in Action (chap. 5)

Business requirements

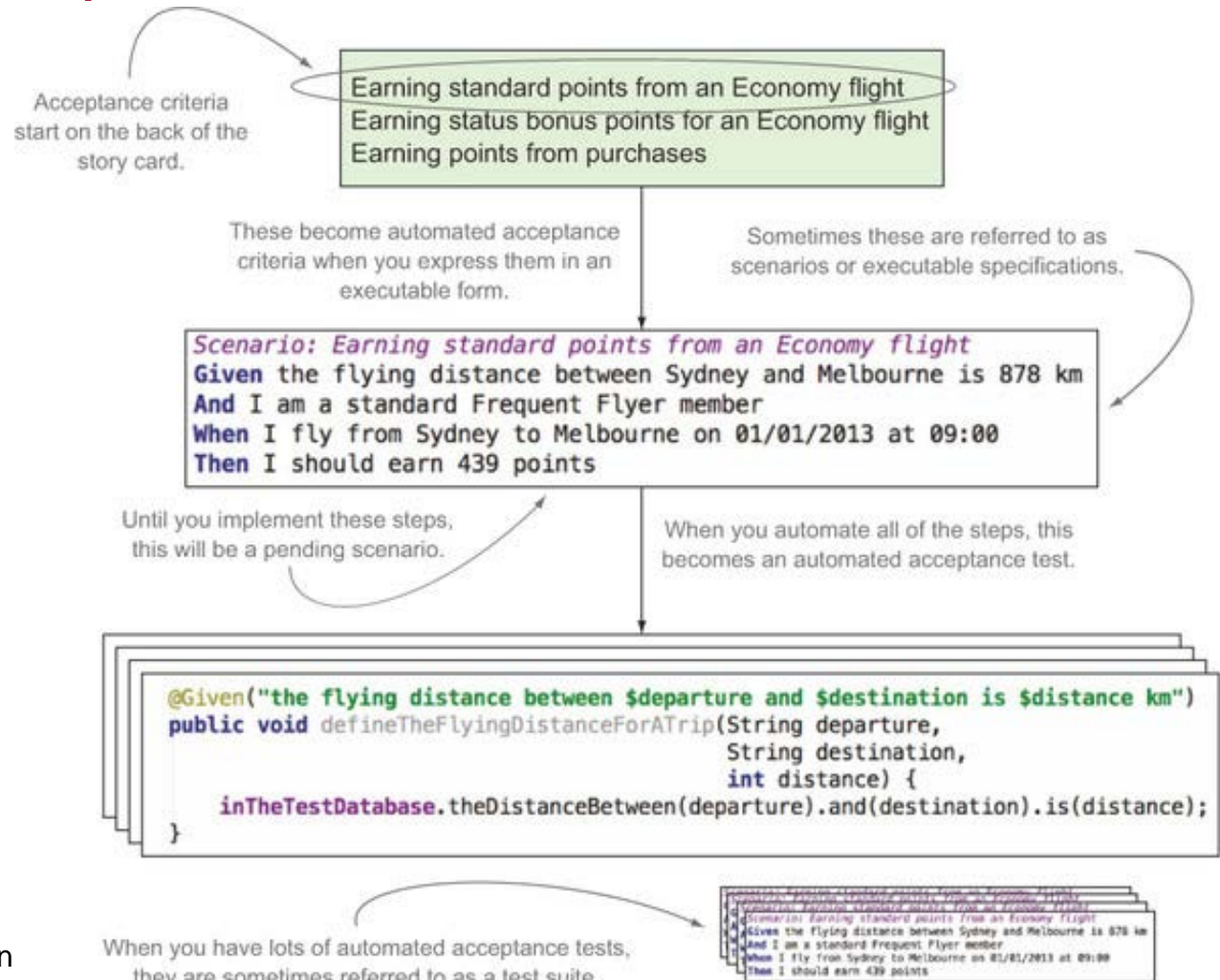
Business requirements are often not as simple as they appear – therefore examples can help:

USER FRIENDLY by J.D. "Illiad" Frazer



How to turn acceptance criteria into executable specifications

Acceptance criteria are those brief notes you write on the back of your story cards that help define when a story or feature is complete.



Source: John Ferguson Smart- BDD in Action

Acceptance Criteria vs. Scenarios 1

- A **scenario** is example of system's behavior from users' perspectives
- **Acceptance criteria** are a set of rules which cover aspects of a system's behavior, and from which scenarios can be derived.

A scenario (*example*) from pet shop:

Given a rabbit called Fluffy who is 1 1/2 months old
When we try to sell Fluffy
Then we should be told Fluffy is too young.

Acceptance criteria from pet shop:

Given a baby animal is younger than its recommended selling age
When we try to sell it
Then we should be told it's too young

Despite Given, When, Then format it is a full specification of this aspect of behavior – phrased in scenario form.

Source: <https://lizkeogh.com/2011/06/20/acceptance-criteria-vs-scenarios/>

Acceptance Criteria vs. Scenarios 2

- It is also possible to phrase acceptance criteria in other ways:

We should be prevented from selling animals younger than the recommended age.

- Now you can talk through the scenarios, and see potential other criteria which you missed:

Customers should be encouraged to return when the animal is old enough to sell

Acceptance Criteria vs. Scenarios 3

- As you're talking through the scenarios, the information we need for automation will emerge, and a better understanding of the domain spreads amongst the people involved in those discussions (usually a dev, BD and tester).

Scenario example:

Given rabbits can't be sold before they're 2 months old

Given Fluffy the rabbit is 1 1/2 months old

When we try to sell Fluffy

Then we should be prompted to tell the customer, "This pet is too young. Please come back in 15 days to collect your pet."

Well-formed user stories 1

- **Convention Over Configuration** is a principle used in many software solutions, e.g. Java JPA, Rails, and delivers enormous value
- Suggestion: **Write user stories in Gherkin** (even though you don't use cucumber)
 - Offers a standard so every new-formed team does not have to re-invent the wheel of "how should we write and format our stories?".

source: <https://content.pivotal.io/blog/how-to-write-well-formed-user-stories>

Well-formed user stories 2

Feature/user story: Shopping Cart
As a Shopper
I want to put items in my shopping cart
Because I want to manage items before I check out

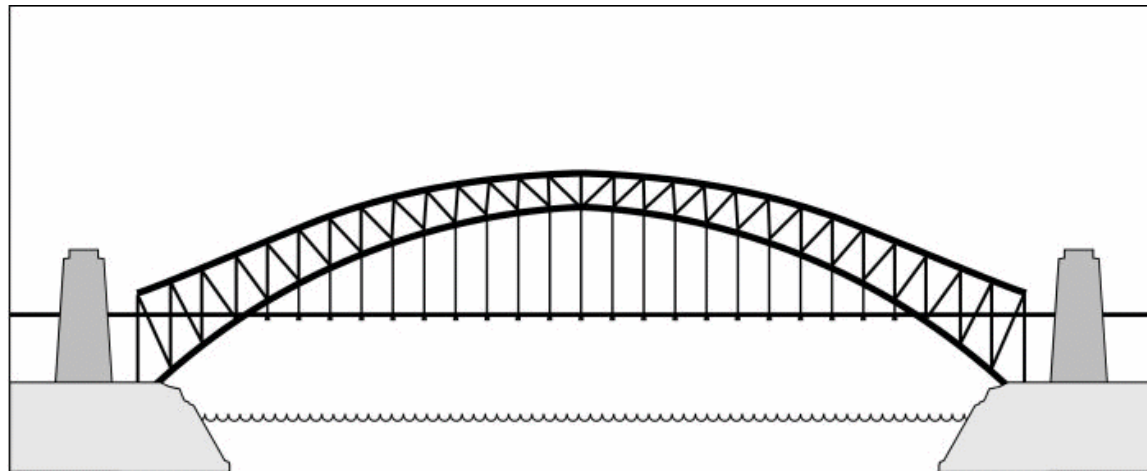
Scenario: User adds item to cart
Given I'm a logged-in User
When I go to the Item page
And I click "Add item to cart"
Then the quantity of items in my cart should go up
And my subtotal should increment
And the warehouse inventory should decrement

Examples from: <https://content.pivotal.io/blog/how-to-write-well-formed-user-stories>

Cucumber

Tool for Communication Collaboration Testing

Non-
technical
people
like
BA, PM



Technical
people

How to use Cucumber – process-wise

- OK not to be test-driven by outside-in, but it helps you stay honest and avoid building functionality nobody asks for at the moment 😊
- Rooted in Behaviour-driven development (BDD)
 - BDD emerged from and extends TDD.
 - Idea: Instead of writing unit tests from specification, then let the specification by a test itself.

```
Feature: Eating too many cucumbers may not be good for you
```

```
Eating too much of anything may not be good for you.
```

```
Scenario: Eating a few is no problem
```

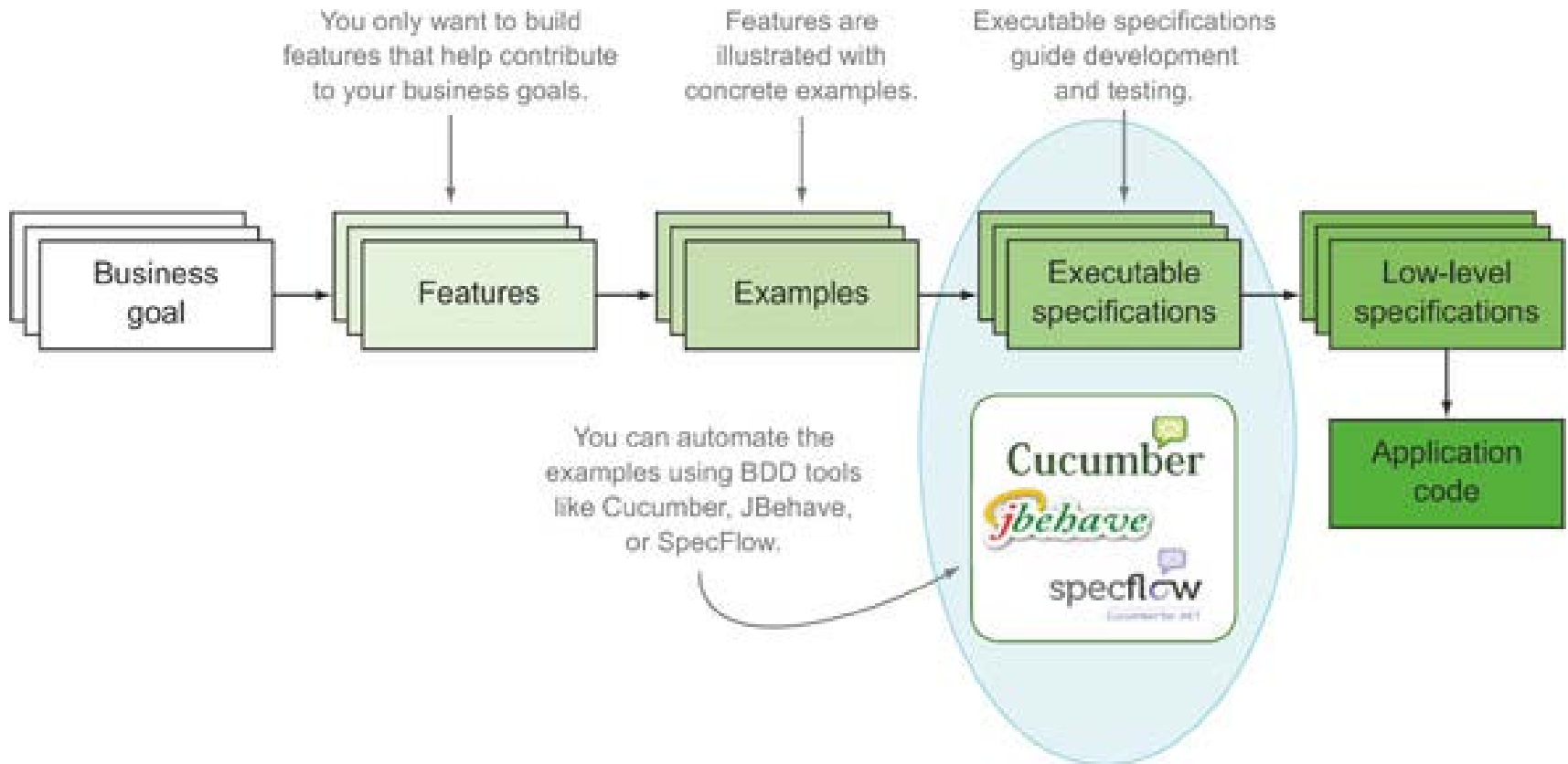
```
  Given Alice is hungry
```

```
  When she eats 3 cucumbers
```

```
  Then she will be full
```

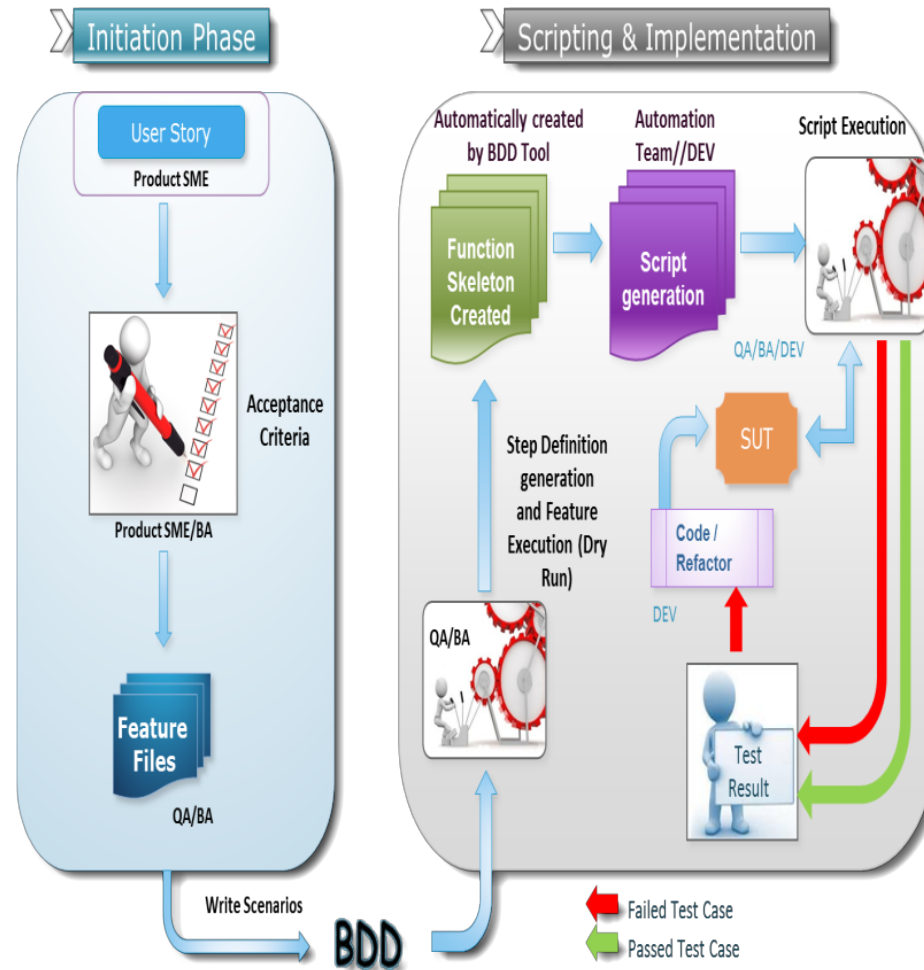
Gherkin
syntax

Cucumber (and other tools) can help automation



Source: John Ferguson Smart- BDD in Action chap 7

Test Approach for BDD



Source:

<http://toolsqa.com/blogs/test-approach-and-comparisons-between-atdd-tdd-and-bdd/>

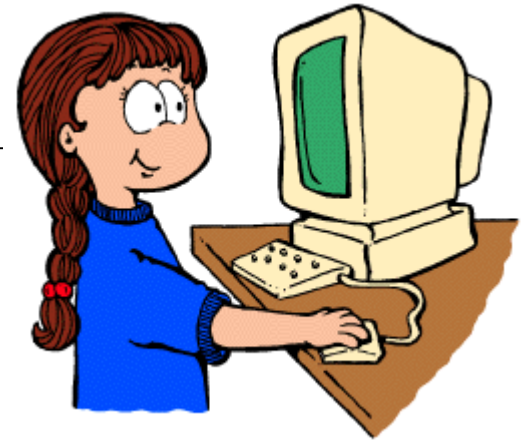
Demo



cucumber 

Cucumber Tutorial

Now it is your turn!



Cucumber tutorial can be found here:

<https://docs.cucumber.io/guides/10-minute-tutorial/>

Cucumber Installation

Java and other JVM languages

<https://github.com/cucumber/cucumber-jvm>

Language overview:

<https://docs.cucumber.io/installation/>

Help for Getting Started

Documentation here:

<https://docs.cucumber.io/>

Tutorial here:

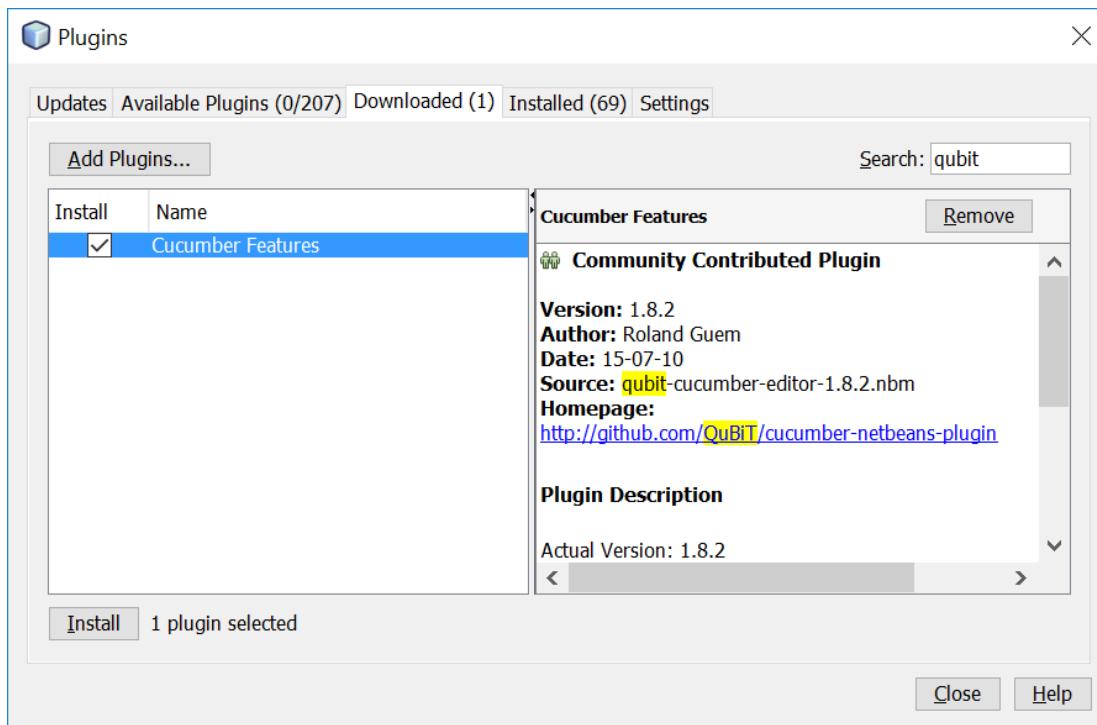
<https://docs.cucumber.io/guides/10-minute-tutorial/>

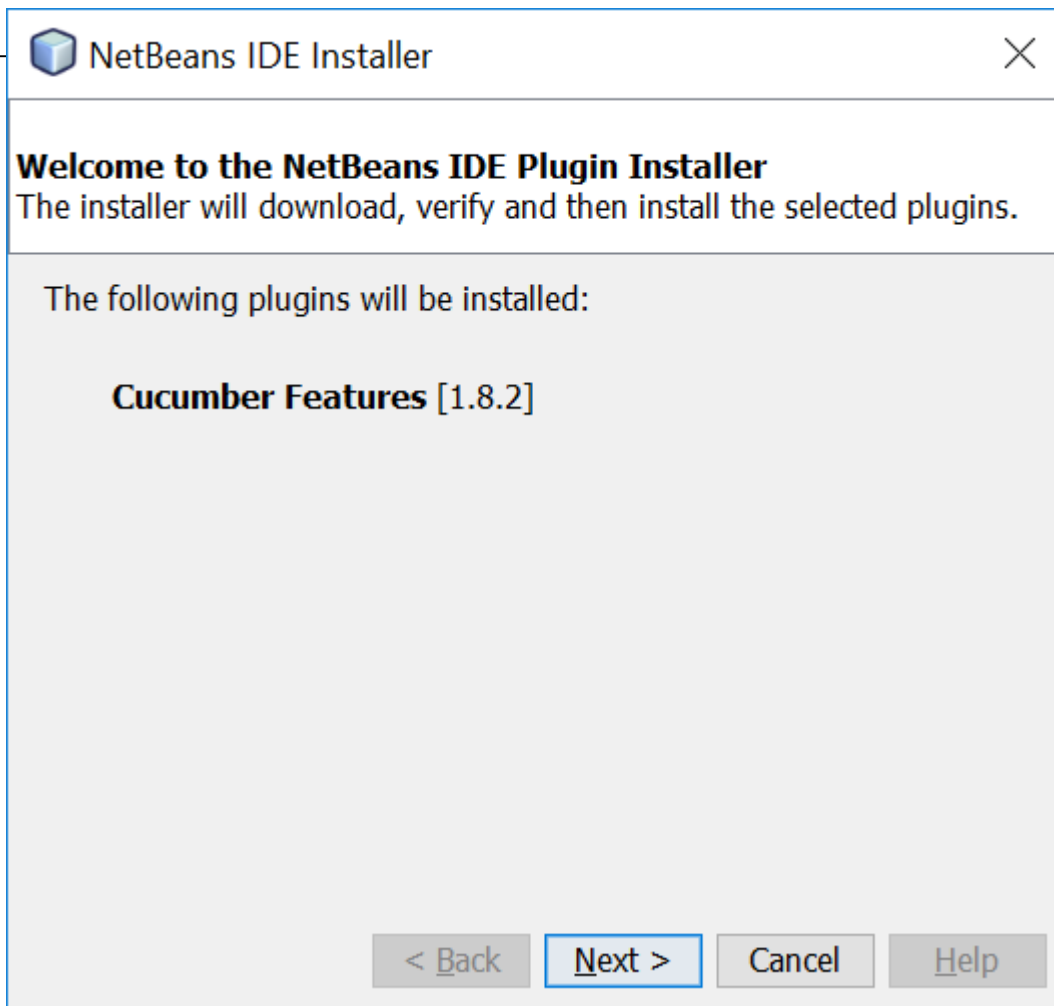
You need plugin for your tool

Tutorial here: <https://docs.cucumber.io/guides/10-minute-tutorial/>

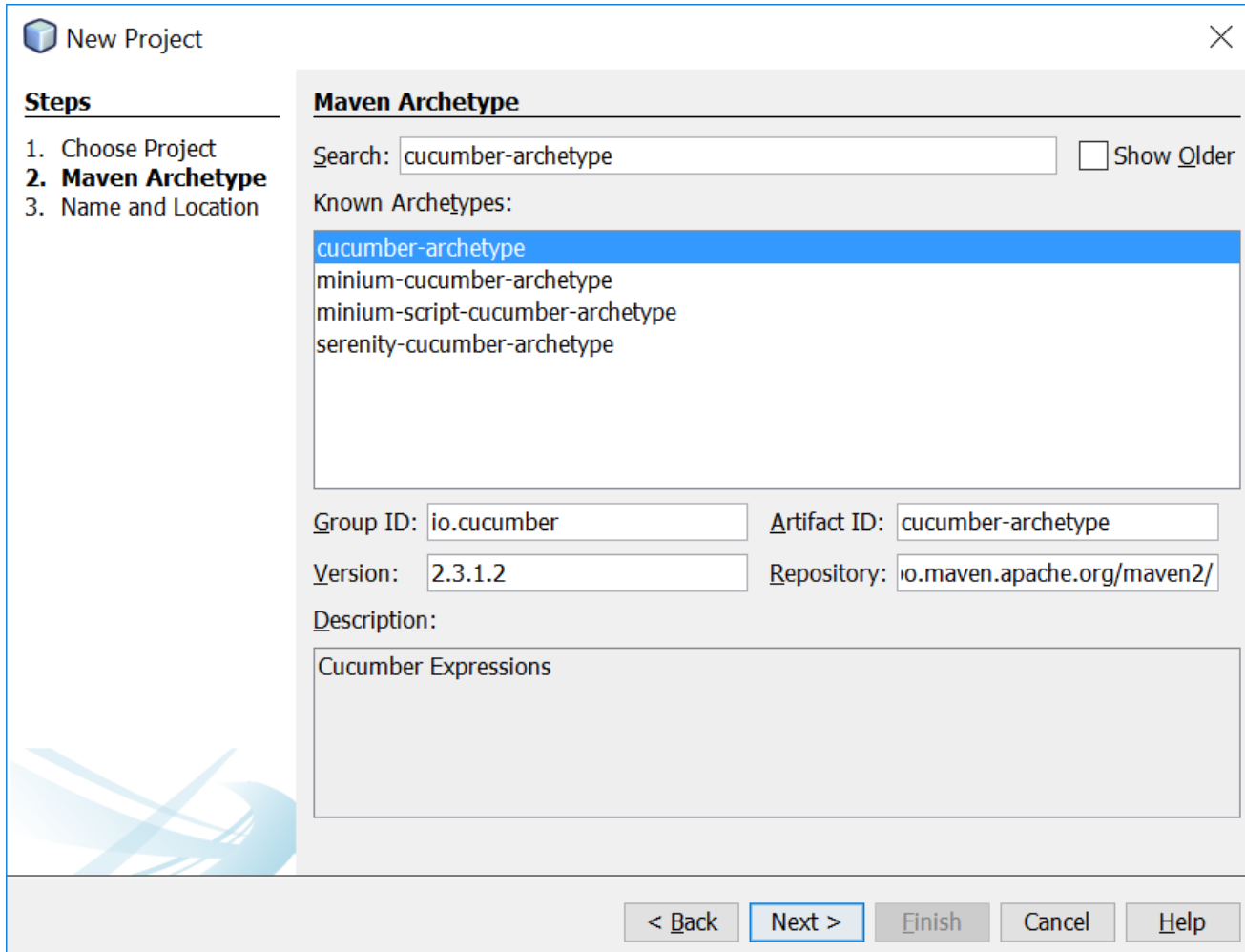
Netbeans plugin: <http://plugins.netbeans.org/plugin/17939/cucumber-features>

<https://github.com/QuBiT/cucumber-netbeans-plugin/downloads>





Create Maven project with archetype



New Project

Steps

1. Choose Project
- 2. Maven Archetype**
3. Name and Location

Maven Archetype

Search: ☐ Show Older

Known Archetypes:

- cucumber-archetype**
- minium-cucumber-archetype
- minium-script-cucumber-archetype
- serenity-cucumber-archetype

Group ID: Artifact ID:

Version: Repository:

Description:

Cucumber Expressions

< Back Next > Finish Cancel Help

Run project

mvn test (right-click in NB project)

```
-----  
T E S T S  
-----
```

```
Running dk.cphbusiness.firstcucumberproject.RunCucumberTest  
No features found at [classpath:dk/cphbusiness/firstcucumberproject]
```

```
0 Scenarios  
0 Steps  
0m0,002s
```

```
Tests run: 0, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.281 sec
```

```
Results :
```

```
Tests run: 0, Failures: 0, Errors: 0, Skipped: 0
```

Write Scenario

New Empty Feature File

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

File Name: is_it_friday_yet

Project: FirstCucumberProject

Folder: rces\dk\cphbusiness\firstcucumberproject **Browse...**

Created File: phbusiness\firstcucumberproject\is_it_friday_yet.feature

< **B**ack Next > **F**inish Cancel **H**elp

Scenario Example

Feature: Is it Friday yet?

Everybody wants to know when it's Friday

Scenario: Sunday isn't Friday

Given today is Sunday

When I ask whether it's Friday yet

Then I should be told "Nope"

Test Scenario

mvn test (right-click in NB project)

```
-----  
T E S T S  
-----
```

```
Running dk.cphbusiness.firstcucumberproject.RunCucumberTest
```

```
Feature: Is it Friday yet?
```

```
  Everybody wants to know when it's Friday
```

```
    Scenario: Sunday isn't Friday          # dk/cphbusiness/firstcucumberproject/is_it_friday_yet.feature:4  
      Given today is Sunday                # null  
      When I ask whether is's Friday yet   # null  
      Then I should be told "Nope"         # null
```

```
1 Scenarios (1 undefined)
```

```
3 Steps (3 undefined)
```

```
0m0,023s
```

Test output cont.

You can implement missing steps with the snippets below:

```
@Given("^today is Sunday$")
public void today_is_Sunday() throws Exception {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```

```
@When("^I ask whether is's Friday yet$")
public void i_ask_whether_is_s_Friday_yet() throws Exception {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```

```
@Then("^I should be told \"([^\"]*)\"$")
public void i_should_be_told(String arg1) throws Exception {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```

Tests run: 3, Failures: 0, Errors: 0, Skipped: 3, Time elapsed: 0.454 sec

Write step definition

Write step definitions

```
@Given("^today is Sunday$")
public void today_is_Sunday() {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^I ask whether is's Friday yet$")
public void i_ask_whether_is_s_Friday_yet() {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Then("^I should be told \"([^\"]*)\"$")
public void i_should_be_told(String arg1) {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```

Test Scenario with step def.

mvn test (right-click in NB project)

```
-----
T E S T S
-----
Running dk.cphbusiness.firstcucumberproject.RunCucumberTest
Feature: Is it Friday yet?
  Everybody wants to know when it's Friday

  Scenario: Sunday isn't Friday          # dk/cphbusiness/firstcucumberproject/is_it_friday_yet.feature:4
    Given today is Sunday                # Stepdefs.today_is_Sunday()
      cucumber.api.PendingException: TODO: implement me
        at dk.cphbusiness.firstcucumberproject.Stepdefs.today_is_Sunday(Stepdefs.java:14)
        at ?.today is Sunday(dk/cphbusiness/firstcucumberproject/is_it_friday_yet.feature:5)

    When I ask whether is's Friday yet # Stepdefs.i_ask_whether_is_s_Friday_yet()
    Then I should be told "Nope"       # Stepdefs.i_should_be_told(String)

1 Scenarios (1 pending)
3 Steps (2 skipped, 1 pending)
0m0,117s

cucumber.api.PendingException: TODO: implement me
  at dk.cphbusiness.firstcucumberproject.Stepdefs.today_is_Sunday(Stepdefs.java:14)
  at ?.today is Sunday(dk/cphbusiness/firstcucumberproject/is_it_friday_yet.feature:5)

Tests run: 1, Failures: 0, Errors: 0, Skipped: 1, Time elapsed: 0.517 sec
```


Gherkin syntax

- Gherkin is Domain-Specific-Language
 - Given (Arrange/Context)
 - When (Act)
 - Then (Assert)
 - See more: <https://github.com/cucumber/cucumber/wiki/Gherkin>
- **Gherkin** steps are expressed in **plain text**.
- **Cucumber** scans the text of each step for patterns that it recognizes, which you define using a ***regular expression***

Regular Expression Example 1

Feature: Cash withdrawal

Scenario: Successful withdrawal from an account in credit

Given I have \$100 in my account

When I request \$20

Then \$20 should be dispensed

- A simple regular expression that will match this step would look like this

/I have \\$100 in my Account/

Source: Wynne & Hellesøy: The cucumber book

Regular Expression Example 2 – Alternations – the dot

```
Given(/I have deposited \$(100|250) in my Account/) do |amount|  
  # TODO: code goes here  
end
```

- The dot is a *metacharacter*, meaning it has magical powers in a regular expression. Literally, a dot means *match any single character*. So, we can try this instead:

```
Given(/I have deposited \$(...) in my Account/) do |amount|  
  # TODO: code goes here  
end
```

Regular Expression Example 3 – Alternations – the start modifier

- That will now match a step with any three-figure dollar sum

```
Given(/I have deposited \$(...) in my Account/) do |amount|
```

```
# TODO: code goes here
```

```
end
```

The star modifier means *any number of times*. So, with `.*` we're capturing *any character, any number of times*.

```
Given(/I have deposited \$(.*) in my Account/) do |amount|
```

```
# TODO: code goes here
```

```
end
```

Regular Expression Example 4 – Alternations – character classes

- Character classes allow you to tell the regular expression engine to match one of a range of characters

```
Given(/I have deposited \$$([0123456789]*) in my Account/) do |amount|  
  # TODO: code goes here  
end
```

For a continuous range of characters, you can use a hyphen:

```
Given(/I have deposited \$$([0-9]*) in my Account/) do |amount|  
  # TODO: code goes here  
end
```

Regular Expression Example 5 – character classes shorthand

- For common patterns of characters like [0-9], there are a few *shorthand character classes*, e.g. digits

```
Given(/I have deposited \$(\d*) in my Account/) do |amount|
```

```
  # TODO: code goes here
```

```
end
```

Useful Shorthand Character Classes

Here are the most useful shorthand character classes:

`\d` stands for *digit*, or `[0-9]`.

`\w` stands for *word character*, specifically `[A-Za-z0-9_]`. Notice that underscores and digits are included but not hyphens.

`\s` stands for *whitespace character*, specifically `[\t\r\n]`. That means a space, a tab, or a line break.

`\b` anchors a match to a *word boundary*, anything that is not a word character is a word boundary. It works a little like `\s`, except it doesn't match a character. Its useful when you want to match whole words or the beginning or end of a word.

Demo – Cucumber & Selenium

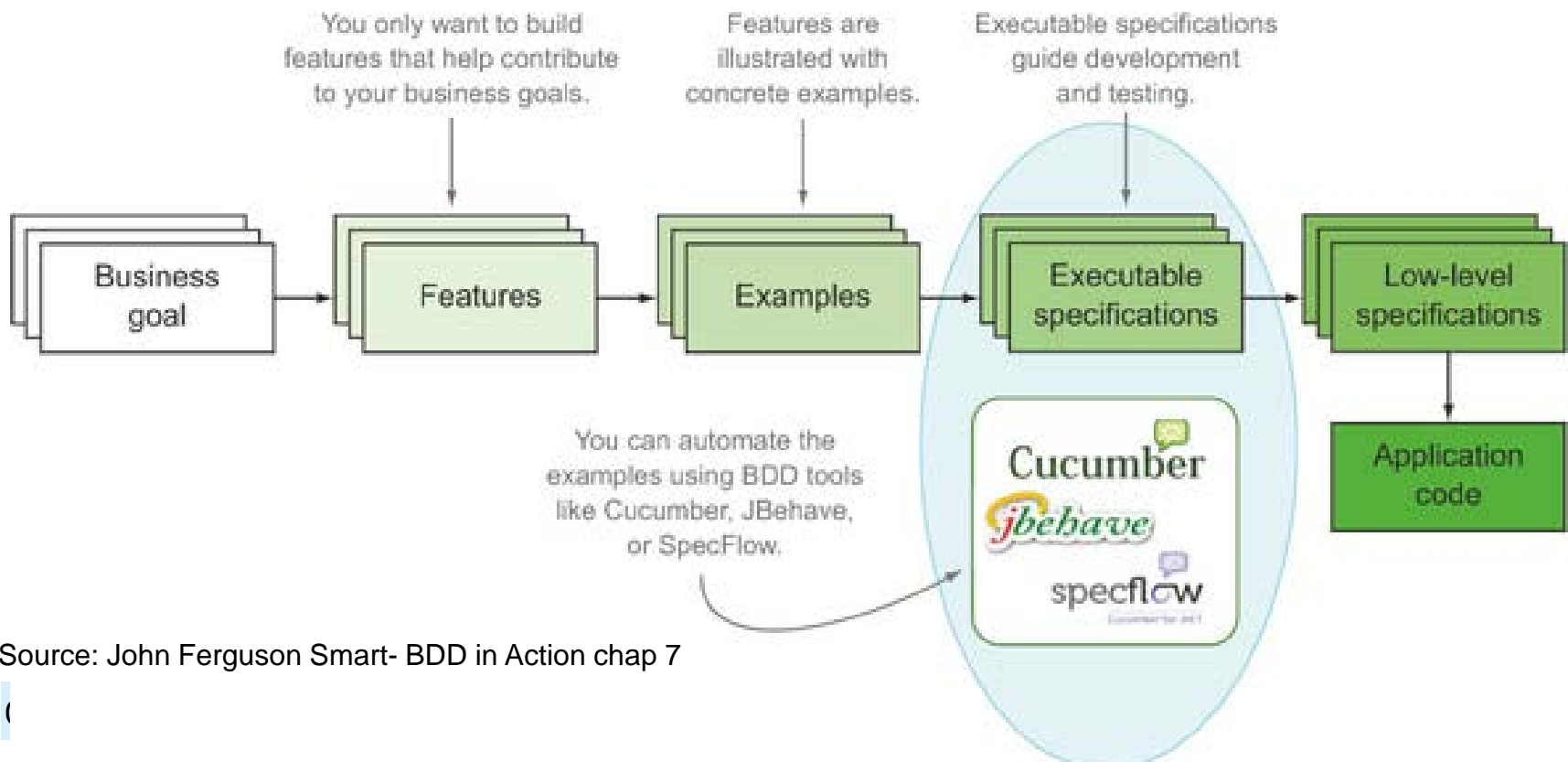


cucumber 

Project: CucumberSelenium

Cucumber Overview - again

- Doesn't have to be web application
- Could be REST API, message queues, database etc.



Source: John Ferguson Smart- BDD in Action chap 7

Features != User Stories

- Example of too big user story -- > split into two stories:

```
In order to pay for the items in my shopping cart,  
As a customer,  
I want a credit card checkout
```

- Split into 2 new stories:

```
In order to pay for the items in my shopping cart,  
As a customer with a FooCorp credit card,  
I want a credit card checkout that accepts my card
```

```
In order to pay for the items in my shopping cart,  
As a customer with a BarCorp credit card,  
I want a credit card checkout that accepts my card
```

Source: <http://blog.mattwynne.net/2010/10/22/features-user-stories/>

Acceptance criteria for first user story (FooCorp credit card)

- Let's assume that we've done some analysis on the story, but let's make sure we understand the acceptance criteria. We have a little sit-down with our product owner, and come up with this list:
 - when a user attempts to pay with a BarCorp card, we show them a "Sorry" message
 - when a user pays with a FooCorp card, we send them an email with their receipt
 - when a user attempts to pay with a FooCorp card but gets their details wrong, we bounce the page and show them an error message, so they can correct their mistake.
 - when a user attempts to pay with a FooCorp card but the card is declined, we bounce the page and show them an error message, so they can try another card.
- Note: those are the acceptance criteria for this story. We didn't do anything fancy with them like put them in a Microsoft Word® document or a Jira ticket. We just wrote them down on a piece of paper



Features != User Stories (or how to organize Cucumber features)

- Don't organise features like below. Why not?
 - Give it a few months, and your features directory is going to look like a history log of the development of your project.

```
features/  
  pay_with_foocorp_card.feature  
  pay_with_barcorp_card.feature  
  pay_with_bazcorp_card.feature
```

- What then? `features/pay_with_credit_card.feature`
 - User Stories are a planning tool. They exist until they're implemented, and then they disappear, absorbed into the code.
 - Cucumber features are a communication tool. They describe how the system behaves today so that if you need to check how it works, you don't need to read code or go punching buttons on the live system.
 - The User Story is absorbed into our features and becomes invisible, leaving the features as a live document of what the software does now, not as a record of how it was constructed.

Combination of many input values

Conventional Test Cases

Example:

- If we have three variables (A,B,C), each can have 3 values say (Red, Green, and Blue).
- The possible combinations in conventional test cases would be 27 i.e. 3^3

All Combinations for Three Variables of Three Levels Each Clip slide

	A	B	C
1	Red	Red	Red
2	Red	Red	Green
3	Red	Red	Blue
4	Red	Green	Red
5	Red	Green	Green
6	Red	Green	Blue
7	Red	Blue	Red
8	Red	Blue	Green
9	Red	Blue	Blue
10	Blue	Red	Red
11	Blue	Red	Green
12	Blue	Red	Blue
13	Blue	Green	Red
14	Blue	Green	Green
15	Blue	Green	Blue
16	Blue	Blue	Red
17	Blue	Blue	Green
18	Blue	Blue	Blue
19	Green	Red	Red
20	Green	Red	Green
21	Green	Red	Blue
22	Green	Green	Red
23	Green	Green	Green
24	Green	Green	Blue
25	Green	Blue	Red
26	Green	Blue	Green
27	Green	Blue	Blue

Pairwise testing

- Systematic way to test pairwise interaction

Example:

- If we have three variables (A,B,C), each can have 3 values say (Red, Green, and Blue).
- The possible combinations in OATS test cases would be 9.

All-Pairs Array, Three Variables of Three Levels Each			
	A	B	C
2	Red	Red	Green
4	Red	Green	Red
9	Red	Blue	Blue
12	Blue	Red	Blue
14	Blue	Green	Green
16	Blue	Blue	Red
19	Green	Red	Red
24	Green	Green	Blue
26	Green	Blue	Green

OAT = Orthogonal Array Testing

Pairwise testing advantages

Guarantees testing the pair-wise combinations of all the selected variables.

Creates an efficient and concise test set with many fewer test cases than testing all combinations of all variables.

Creates a test set that has an even distribution of all pair-wise combinations.

Exercises some of the complex combinations of all the variables.

Is simpler to generate and less error prone than test sets created by hand.

Data Tables in Cucumber

- Data tables can be listed as examples

Scenario Outline: Withdraw fixed amount

Given I have <Balance> in my account

When I choose to withdraw the fixed amount of <Withdrawal>

Then I should <Outcome>

And the balance of my account should be <Remaining>

Examples:

Balance	Withdrawal	Remaining	Outcome
\$500	\$50	\$450	receive \$50 cash
\$500	\$100	\$400	receive \$100 cash
\$500	\$200	\$300	receive \$200 cash
\$100	\$200	\$100	see an error message

Pairwise Tool

- Look here:
<http://pairwisetesting.com/tools.html>
- Or try this one made by former colleague of Gitte Ottosen
- <http://www.hanshenrikolesen.com/testpairs/>



Exploratory Testing

Definition

- An interactive process of **simultaneous** learning, test design, and test execution
- *Exploratory testing is not against the idea of scripting. In some contexts, you will achieve your testing mission better through a more scripted approach; in other contexts, your mission will benefit more from the ability to create and improve tests as you execute them. I find that most situations benefit from a mix of scripted and exploratory approaches.*

James Bach



Who?

- An exploratory tester is primarily a test designer!
- Everybody can design a test "by accident", but the skilled exploratory tester can produce tests which systematically explore the product
- It requires skills!
- Be able to analyze the product
- Evaluate risks
- Use tools
- Critical thinking
- Is systematic, but can pursue smells (anomalies, pieces that aren't consistent)

Exploratory Testing ≠ Ad Hoc Testing



Why?

- An automated test won't tell you that
 - the system's slow, unless you tell it to look in advance.
 - the window leaves a persistent shadow,
 - that every other record in the database has been trashed,
 - that even the false are returning true,
- unless it knows where to look, and what to look for.



Who?

- An exploratory tester is primarily a test designer!
- Everybody can design a test "by accident", but the skilled exploratory tester can produce tests which systematically explore the product
- It requires skills!
- Be able to analyze the product
- Evaluate risks
- Use tools
- Critical thinking
- Is systematic, but can pursue smells (anomalies, pieces that aren't consistent)

Exploratory Testing ≠ Ad Hoc Testing



How?

- Exploratory testers do not enter into a test session with predefined, expected results. Instead, they compare the behavior of the system against what they might expect, based on experience, heuristics, and perhaps oracles
- Freestyle (original style)
- Bug hunt

Freestyle

- Most often starts with a charter which defines mission



Examples

- Test all fields that allow data input (function, stress, boundaries)
- Identify work tasks through application and do each task. They must represent realistic usage scenarios
- Test integration with external applications, focus on Microsoft Word.

Freestyle – using personae



Tester is the "bad customer" doing actions like

- Invalid parts of input—characters, values, combinations
- Time changes
- Unusual uses
- Too much—long strings, large numbers, many instances
- Stop halfway/jump in halfway
- Wrong assumptions
- Making lots of mistakes, compounding mistakes
- Using the same information for different entities
- Triggering error messages
- Going too fast



Bughunt

Executed as a competition

- All participants are paired up, mixed profiles preferred
- Charters defined in advance
- Test case are done as exploratory tests based on charter
- Each team has a bell which is used when is bug is found
- Timeboxing: Each charter is done within 1 hour

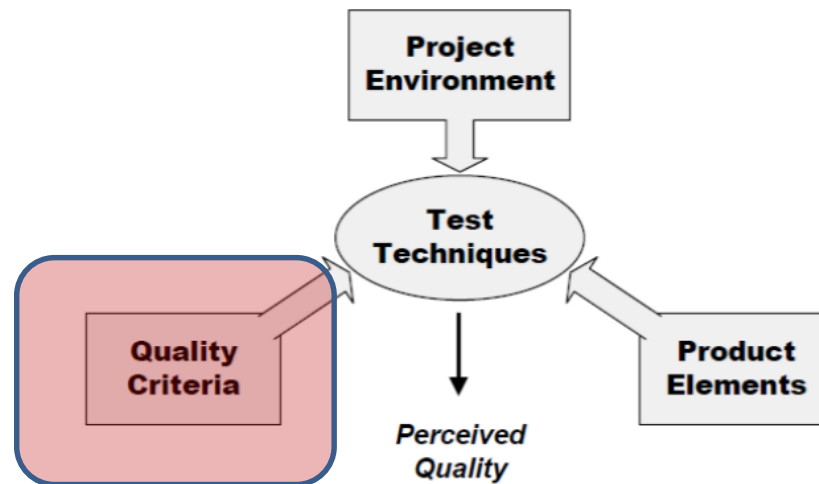
Prize:

- Worst bug (highest severity)
- Most bugs

Non functional testing

- Is test of software product characteristics, i.e. quality criteria.

Heuristic Test Strategy Model



Source: <http://www.satisfice.com/tools/htsm.pdf>

Who does non functional testing?

- Security testing – specialized skill set
- Performance testing – testers & programmers – (Stress testing) typically performed in separate environment that duplicates accurate production-level load for a test
- Larger organizations may have database experts, security groups etc. to support team
- The more diverse skills team have, the less likely you need help from outside consultants.

Performance and Load Testing Tools

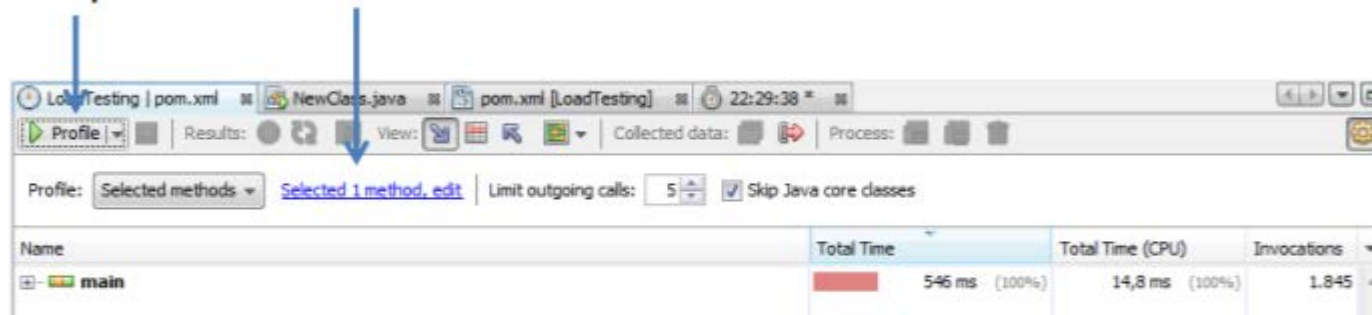
- [Profiling](#): dynamic program analysis
 - Memory
 - Time
 - Frequency and duration of method calls.
- A profiler can look for bottlenecks in application
- Most commonly, profiling information serves to aid program optimization.
- Tools: homegrown, open source, e.g.: JUnitPerf, httpperf, ftptt, The Grinder, Apache Jmeter
- JMeter: unit level + many server types: SOAP, LDAP, POP3

JMeter in Action

Choose Profile → Profile Project:

Set and
start profile

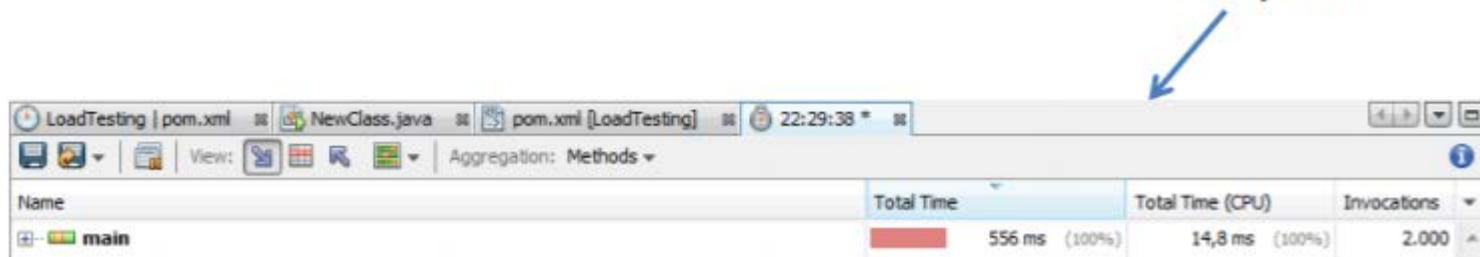
Choose method



Profile: Selected methods ▾ [Selected 1 method, edit](#) Limit outgoing calls: 5 ☒ Skip Java core classes

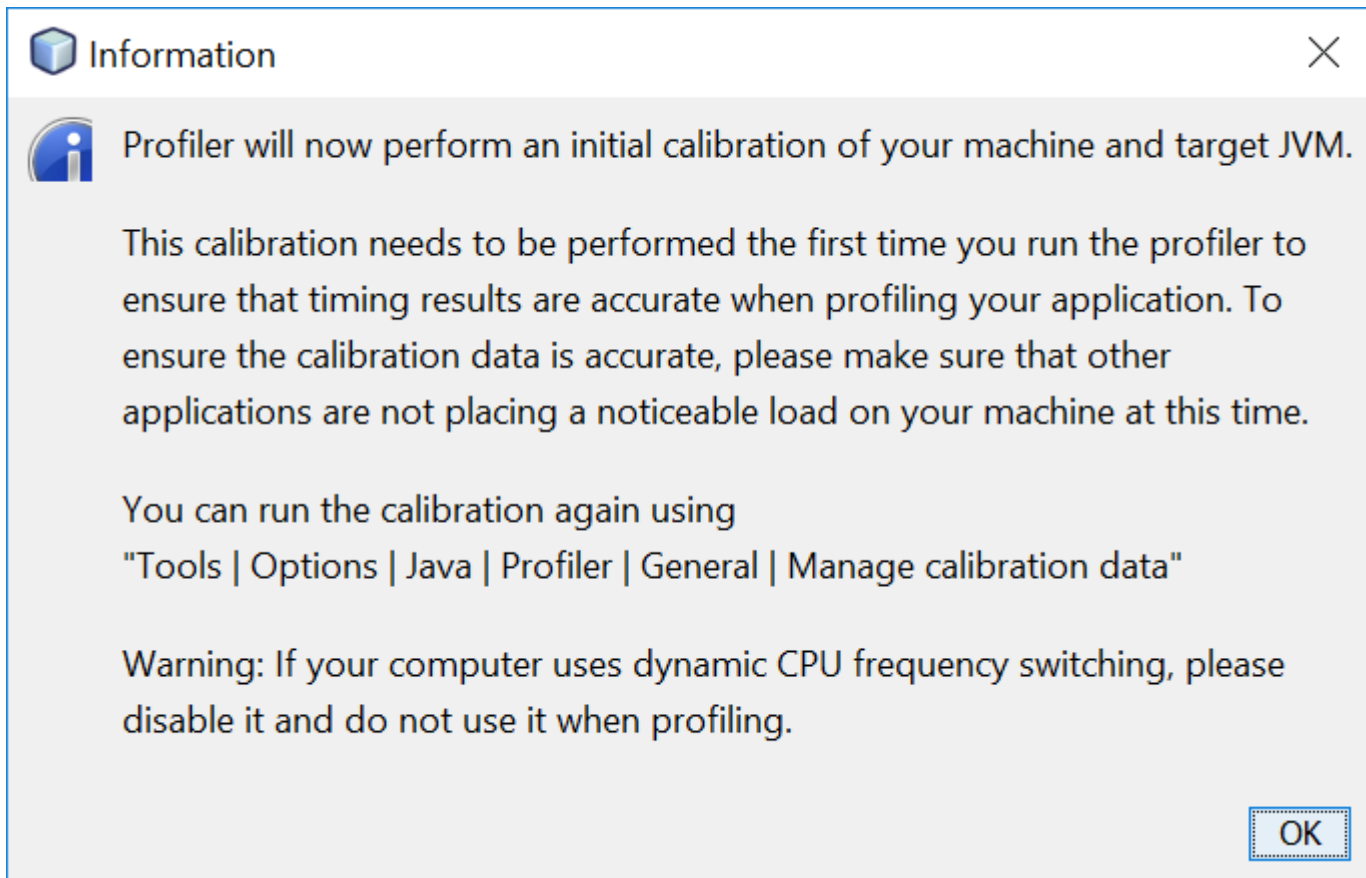
Name	Total Time	Total Time (CPU)	Invocations
main	546 ms (100%)	14,8 ms (100%)	1.845

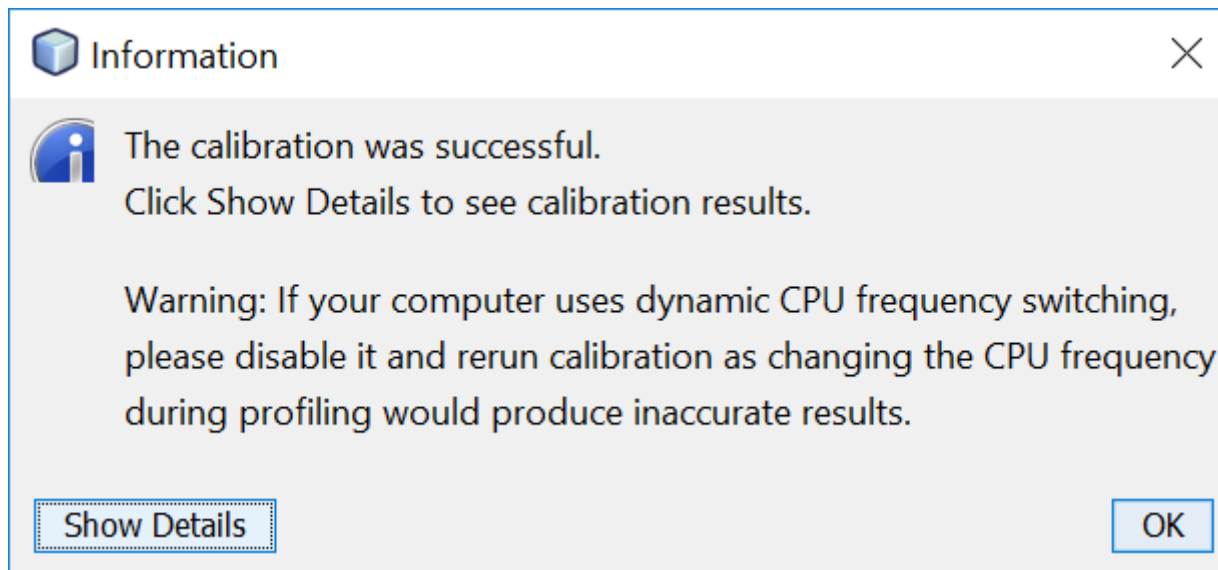
Snapshot

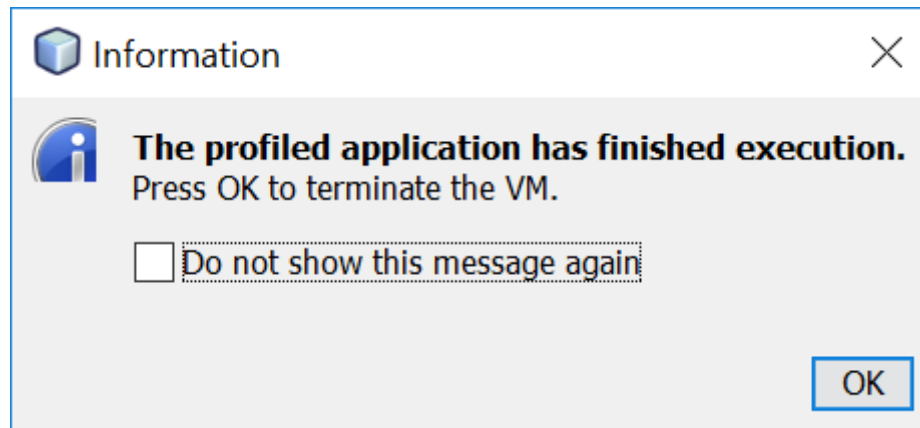


View: Aggregation: Methods ▾

Name	Total Time	Total Time (CPU)	Invocations
main	556 ms (100%)	14,8 ms (100%)	2.000







Study Point Assignment (10 SP)

SPIKE: Technical proof of concept work

This work will help you in your Gutenberg project – it will help you getting wiser at some framework and tools.

1. Examine Cucumber: Set up small Cucumber project with Feature, Scenario(s), and step definitions
 - a) Make sure your project has a number of input values that need to be tested in combination.
 - b) It doesn't have to be app with (G)UI
2. Use pairwise testing technique to create a reduced number of test cases based on a combination of criteria for project above
 - a) You might want to use tool to generate pairwise or threewise combinations
3. Examine JMeter: Profile an application
 1. Methods, objects, telemetry (CPU, memory, threads etc.), SQL queries etc.

Study Point Assignment (10 SP)

Technical proof of concept documentation

1. Write a one page note about the topics with highlights of what you have learned about Cucumber and Profiling and can be used in the Gutenberg project
 1. Also hand-in code used for your experiments, including data files for pairwise testing. If you have used TestPairs, then hand-in
 - xml file with input data
 - cvs file with test data
- **Hand-in on Moodle: Link to code etc. on Github**
 - **Deadline: May 13th at noon.**