

BEHAVIOR-DRIVEN DEVELOPMENT

TEST

PBA SOFTWAREUDVIKLING /
BSC SOFTWARE DEVELOPMENT

Christian Nielsen cnls@cphbusiness.dk

Tine Marbjerg tm@cphbusiness.dk

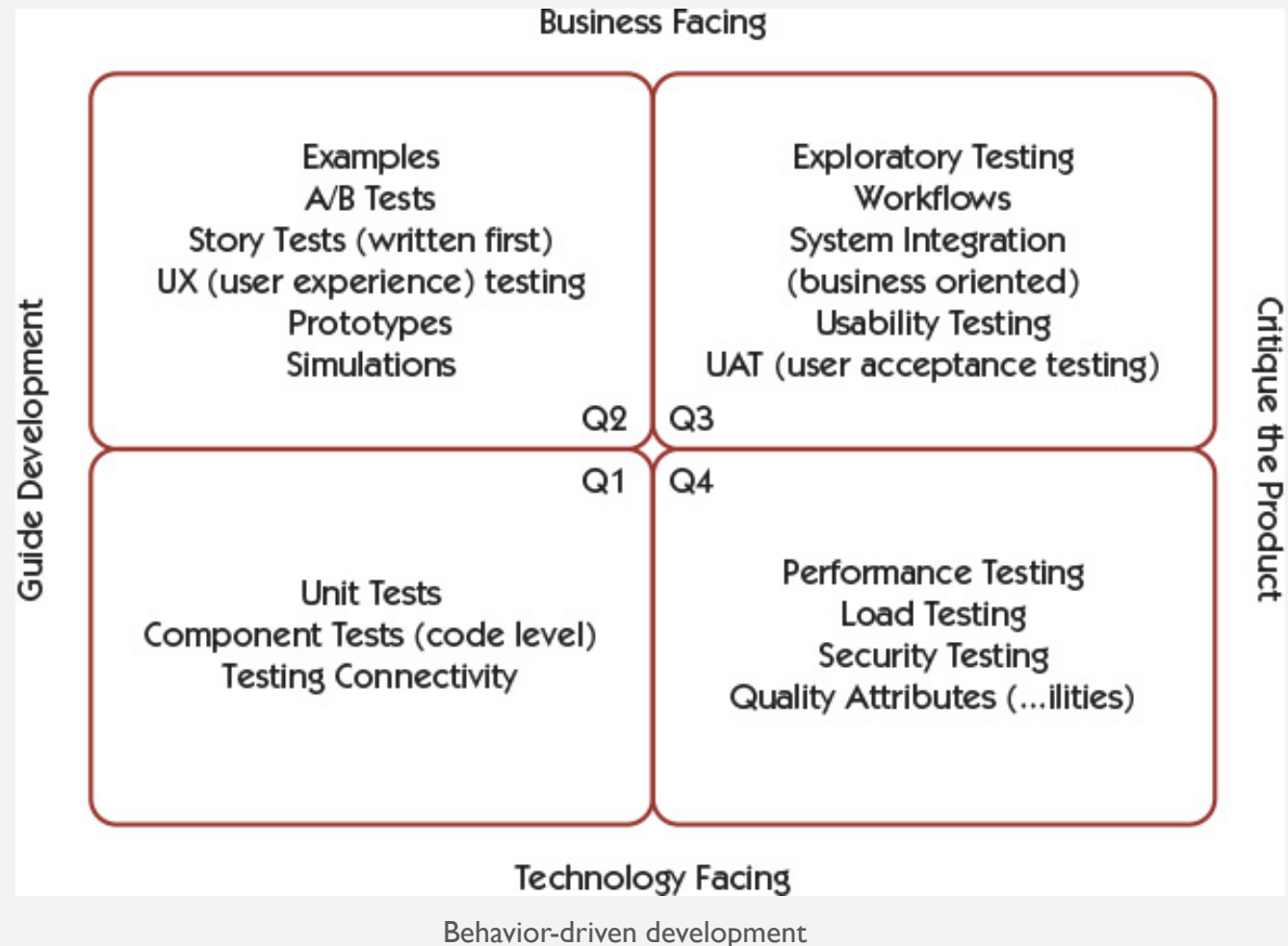
SPRING 2019



TODAY'S TOPIC

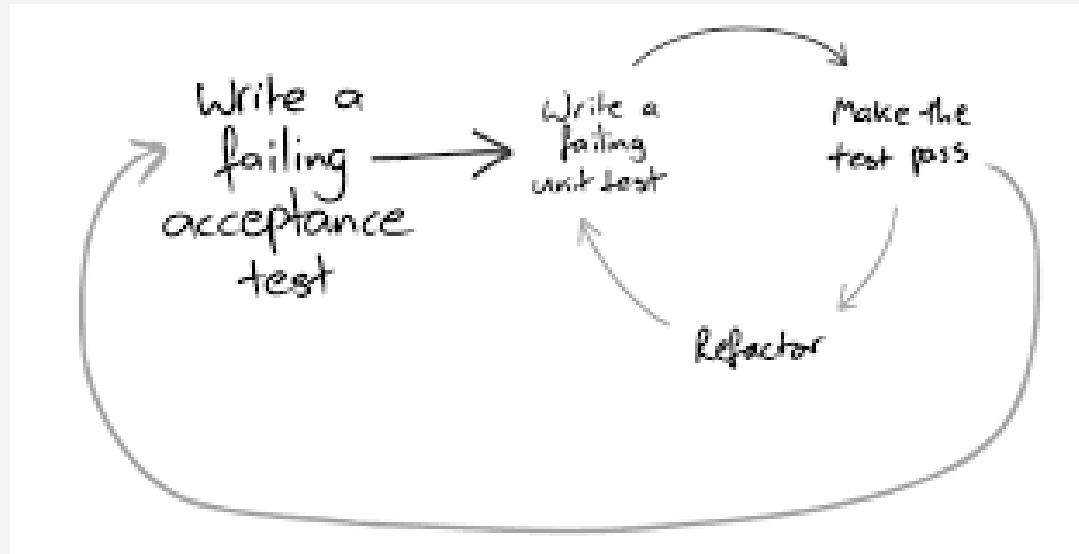
- Develop by examples with Behavior-driven development (BDD)
 - Gherkin & Cucumber
 - Result: Automated functional tests
- Status study point assignments
- Guest lecturer Gitte Ottosen on Agile Testing

BUSINESS FACING TESTS – WE ARE IN Q2 TODAY



ACCEPTANCE TEST-DRIVEN DEVELOPMENT (ATDD)

- Start each feature with an acceptance test
 - Clarifies WHAT to do with no underlying tech focus



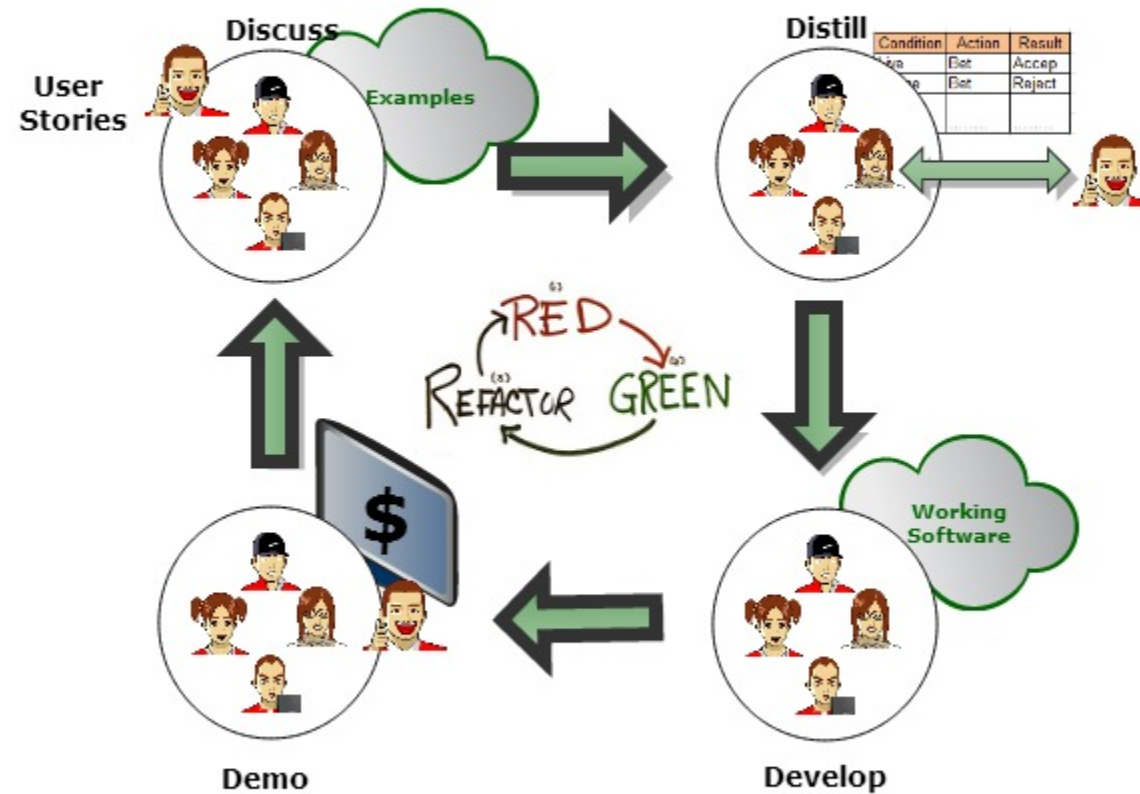
MATURITY MODEL OF CONTINUOUS DELIVERY

- We want automated functional tests as part of Continuous Delivery pipeline to reduce time spent on regression testing
- If we write acceptance tests early by getting examples of desired and undesired system behavior from the customer, we know the right things to build

	Novice	Beginner	Intermediate	Advanced	Expert
Build	Automated builds	Artifacts are managed	Automated release notes	Full traceability	Delivery pipeline
Test	Unit testing, mocks, stubs and proxies	Automated functional tests	Maintain test data	Adaptive test suites	Test in production
Version Control	Commits are tied to tasks	Release train branching strategy	Version numbers matter	Use distributed VCS	Pristine integration branch
DevOps	One Team	Automated deployment	Access to production-like environments	Infrastructure as code	Live monitoring and feedback
Architecture & Design	Code metrics	Testable code	Dependencies are managed	Individually releasable components	Full audit trail in production
Organization & Culture	Agile process	Buy-in from management	Tasks are groomed	Designated roles	Explicit knowledge transfer

ATDD Cycle

from User Stories to Business value



(Based on ATDD cycle model developed by James Shore with changes suggested by Grigori Melnick, Brian Marick, and Elisabeth Hendrickson.)
The Specification by Example concept is taken from Gojko Adzic.

THE POWER OF USING EXAMPLES

Having conversations

Is more important than **capturing** conversations

Is more important than **automating** conversations

Automation becomes a useful side effect of using that tool

Source: More Agile Testing by Janet Gregory & Lisa Crispin

EXPOSE UNCERTAINTY EARLY

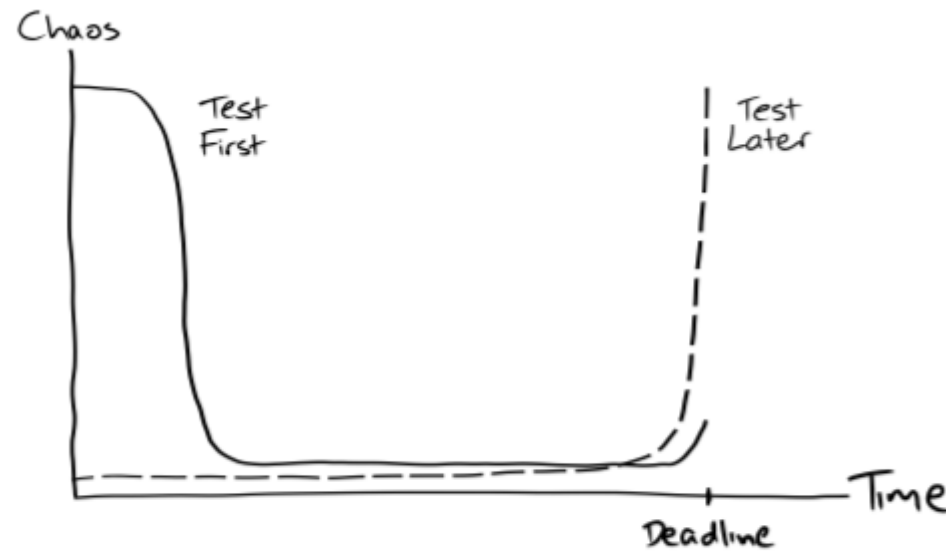


Figure 4.4 *Visible uncertainty in test-first and test-later projects*

BUSINESS REQUIREMENTS

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



BUILD THE RIGHT PRODUCT

Non standard naming for getting examples in agile development:

- Acceptance-test-driven development (ATDD)
- Behavior-driven development (BDD)
- Specification by example (SBE)
- These practices have minor differences between them, but all address the problem of different stakeholders using different vocabularies which in turn result in incorrect interpretations of requirements and discrepancies between code, test and customer expectations

Source: Develop Testing by Tarlinder chapter 2



AUTOMATED ACCEPTANCE TESTS

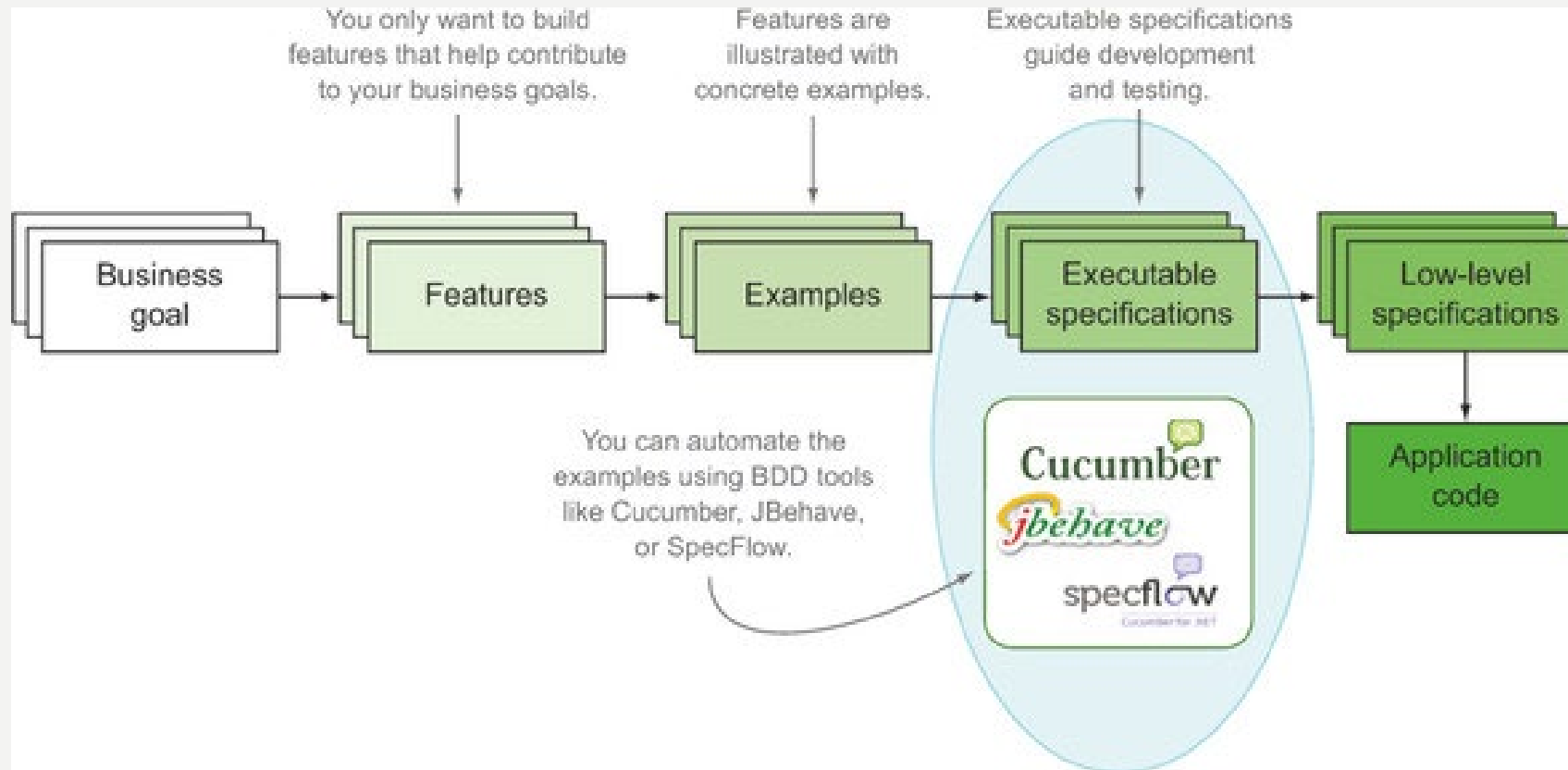
- Examples of desired behavior from conversations with stakeholders are turned into executable tests
- Tools: FitNesse, Cucumber, SpecFlow ...

Textual artifact (scenario or table)

----- bind to -----

executable code

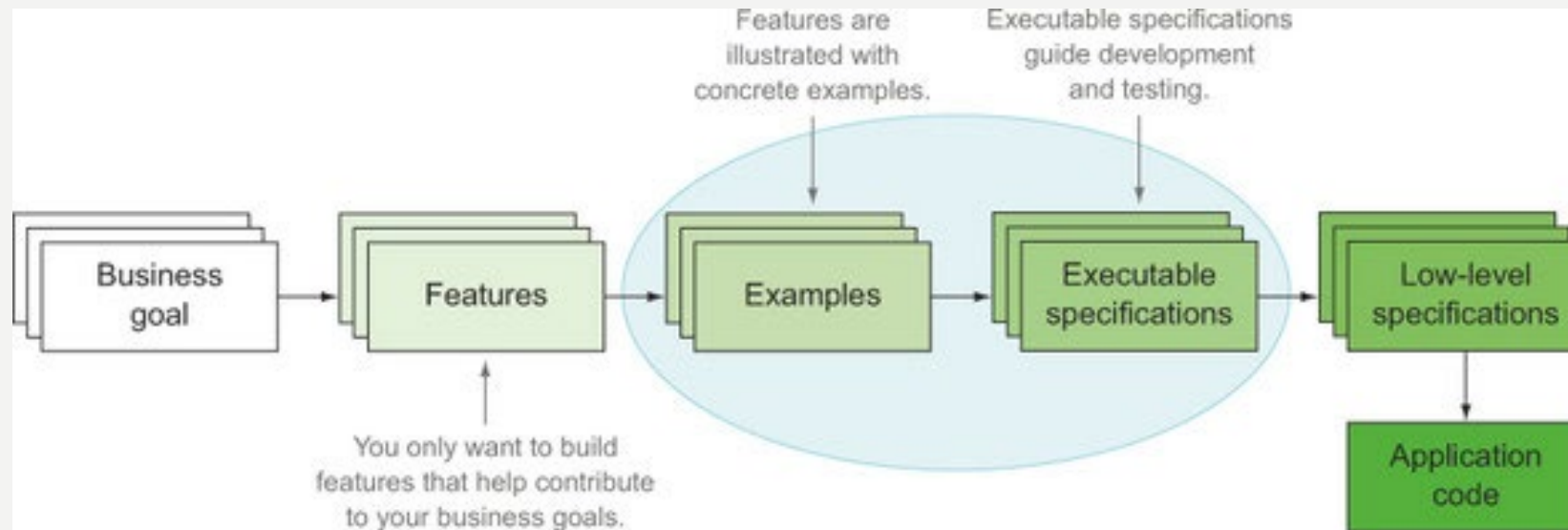
CUCUMBER CAN HELP AUTOMATION



Source: John Ferguson Smart- BDD in Action chap 7

EXAMPLES WRITTEN IN GHERKIN

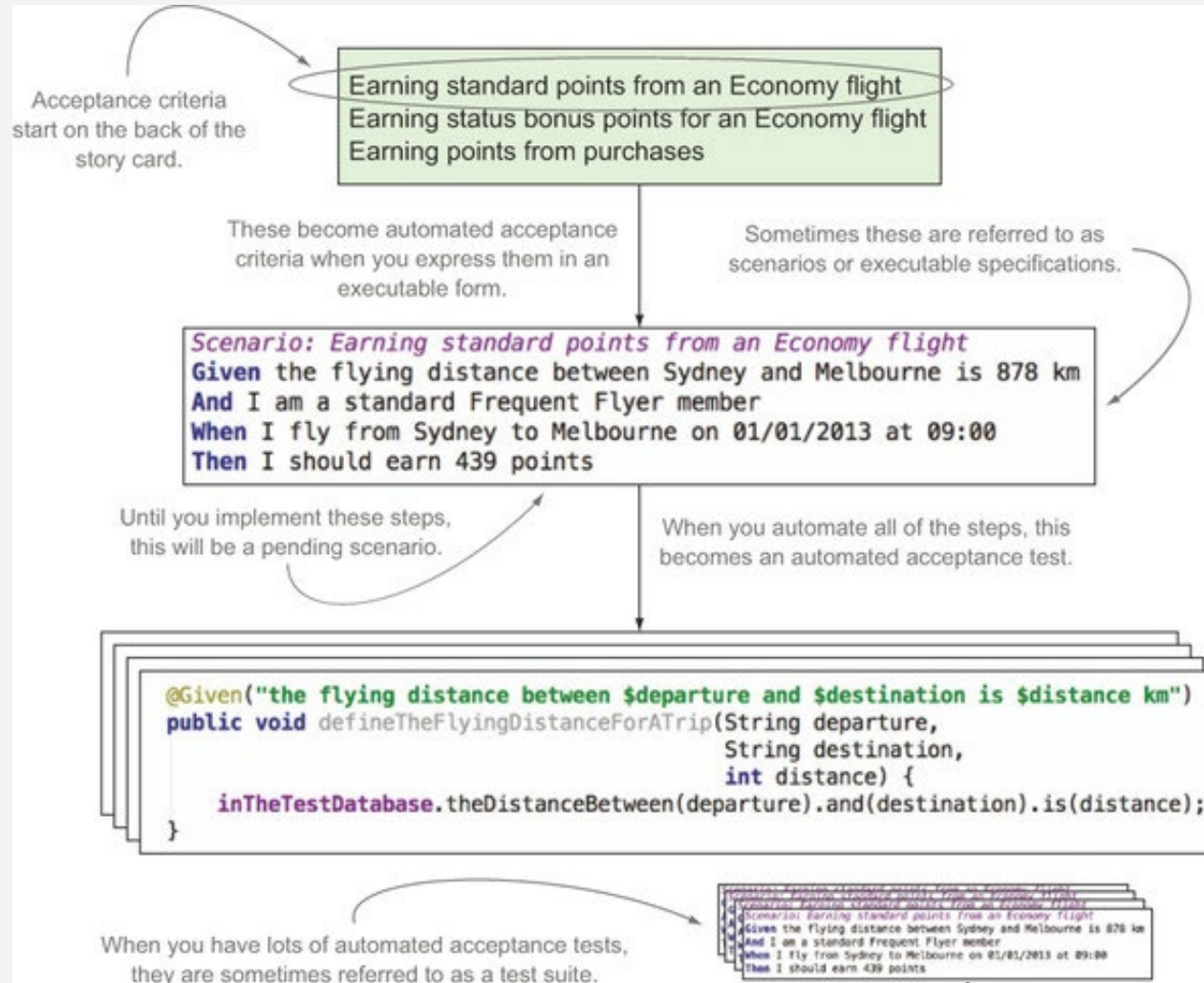
- If examples are expressed in a clear and precise way, they can be transformed into executable specifications and living documentation



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

ACCEPTANCE CRITERIA TURNED INTO EXECUTABLE SPECIFICATIONS

BDD at acceptance test level:
Executable tests written in a
given_when_then
format (Gherkin)



MOST USEFUL DISCUSSION?

What gives you most useful discussion with business stakeholders?

1. “Can you give me a scenario where that happens?” / “Can you give me an example?”

OR

2. “Can you give me acceptance criteria for this?” / “Can you help me work out how to test this?”

DEMO



cucumber 

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

MAVEN POM.XML

```
<dependencies>
  <dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>2.3.1</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>2.3.1</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

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"

DEFINE TEST RUNNER

```
import cucumber.api.CucumberOptions;  
import cucumber.api.junit.Cucumber;  
import org.junit.runner.RunWith;  
  
@RunWith(Cucumber.class)  
@CucumberOptions(plugin = {"pretty"})  
public class RunCucumberTest {  
}
```

RUN TEST

```
-----  
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
```

LOOK AT TEST OUTPUT

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 DEFINITIONS

```
public class Stepdefs {  
    private String today;  
    private String actualAnswer;  
  
    @Given("^today is Sunday$")  
    public void today_is_Sunday() {  
        this.today = "Sunday";  
    }  
  
    @When("^I ask whether it's Friday yet$")  
    public void i_ask_whether_is_s_Friday_yet() {  
        this.actualAnswer = IsItFriday.isItFriday(today);  
    }  
  
    @Then("^I should be told \"([^\"]*)\"$")  
    public void i_should_be_told(String expectedAnswer) {  
        assertEquals(expectedAnswer, actualAnswer);  
    }  
}
```

WRITE (SIMPLE!) CODE

```
class IsItFriday {  
    static String isItFriday(String today) {  
        return "Nope";  
    }  
}
```

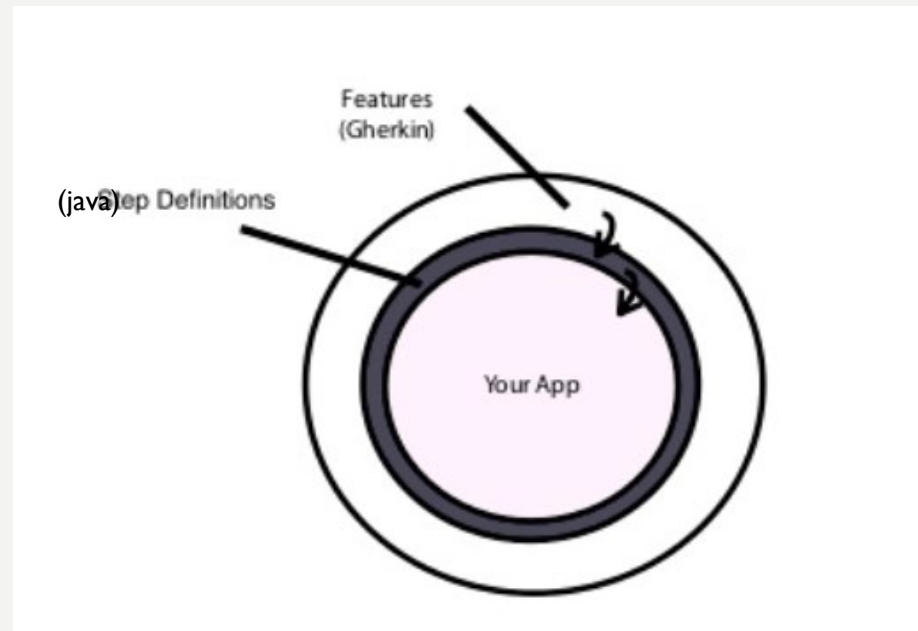
RESOURCES

- Cucumber introduction <https://docs.cucumber.io/guides/overview/>
- Gherkin syntax <https://docs.cucumber.io/gherkin/reference/>

YOUR TURN!



- NB: Must be < Java 9
- Soda machine example: <https://media.pragprog.com/titles/dhwcr/jvm.pdf>



FEATURE FILE

In src/test/resources (text file): SodaMachine.feature

Feature: Soda machine

Scenario: Get soda

Given I have \$2 in my account

When I wave my magic ring at the machine

Then I should get a soda

CUCUMBER

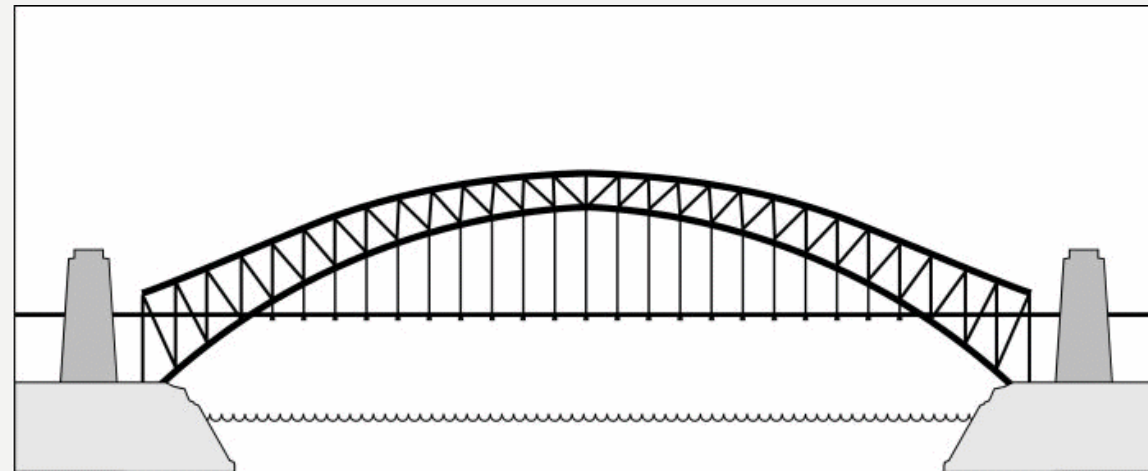
Tool for

Communication

Collaboration

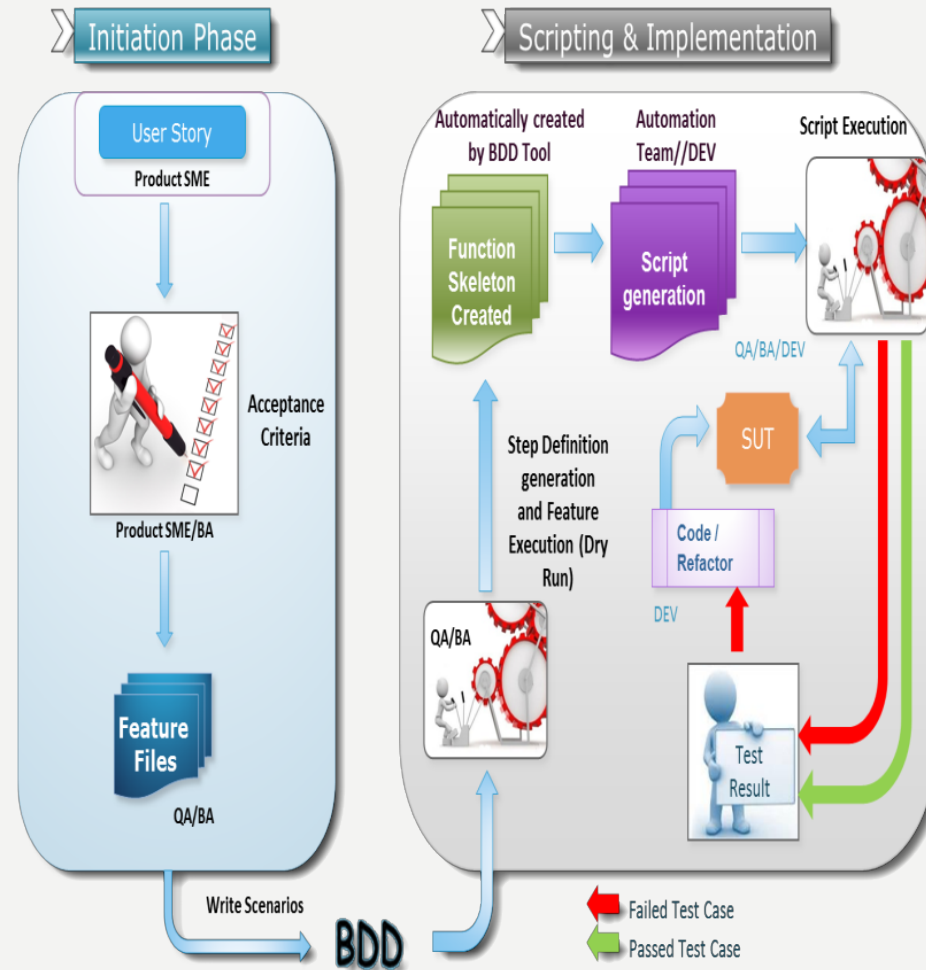
Testing

Non-
technical
people like
BA, PM



Technical
people

TEST APPROACH FOR BDD



Source:

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

GHERKIN SYNTAX

- Gherkin is Domain-Specific-Language
 - Given (Arrange/Context)
 - When (Act)
 - Then (Assert)
- **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 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/

REGULAR EXPRESSION 2 –THE DOT

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

- The dot is a *metacharacter* with magical powers meaning: *match any single character*.
- So, we can try this instead (matchin any three-figure dollar sum):

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

REGULAR EXPRESSION 3 – STAR *

- 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 4 - CHARACTER CLASSES

- Character classes allow you to match one of a range of characters

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

REGULAR EXPRESSION 5 – CHARECTER CLASSES SHORTHAND

- For common patterns of characters like [0-9], there are a *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 

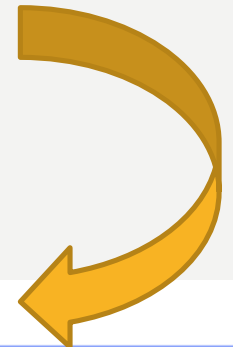
HTML REPORT

Look at /reports/test-report/index.html

Feature: Login Functionality Feature

Scenario: Login Functionality

- * user navigates to the-internet.herokuapp.com/login
- * user logs in using Username as "tomsmith"
- * password as "SuperSecretPassword!"
- * login should be successful
- * Home page should be displayed



▼ **Feature:** Login Functionality Feature

▼ **Scenario:** Login Functionality

- * user navigates to the-internet.herokuapp.com/login
- * user logs in using Username as "tomsmith"
- * password as "SuperSecretPassword!"
- * login should be successful
- * Home page should be displayed

```
@RunWith(Cucumber.class)
```

```
@CucumberOptions(plugin = {"pretty", "html:reports/test-report"})
```

```
public class RunCucumberTest {}
```

DATA DRIVEN TESTING

Data tables can be used with **Scenario Outline**

Feature: Adding

Scenario Outline: Add by one

Given the input <input>

When the calculator is run

Then the output should be <output>

Examples:

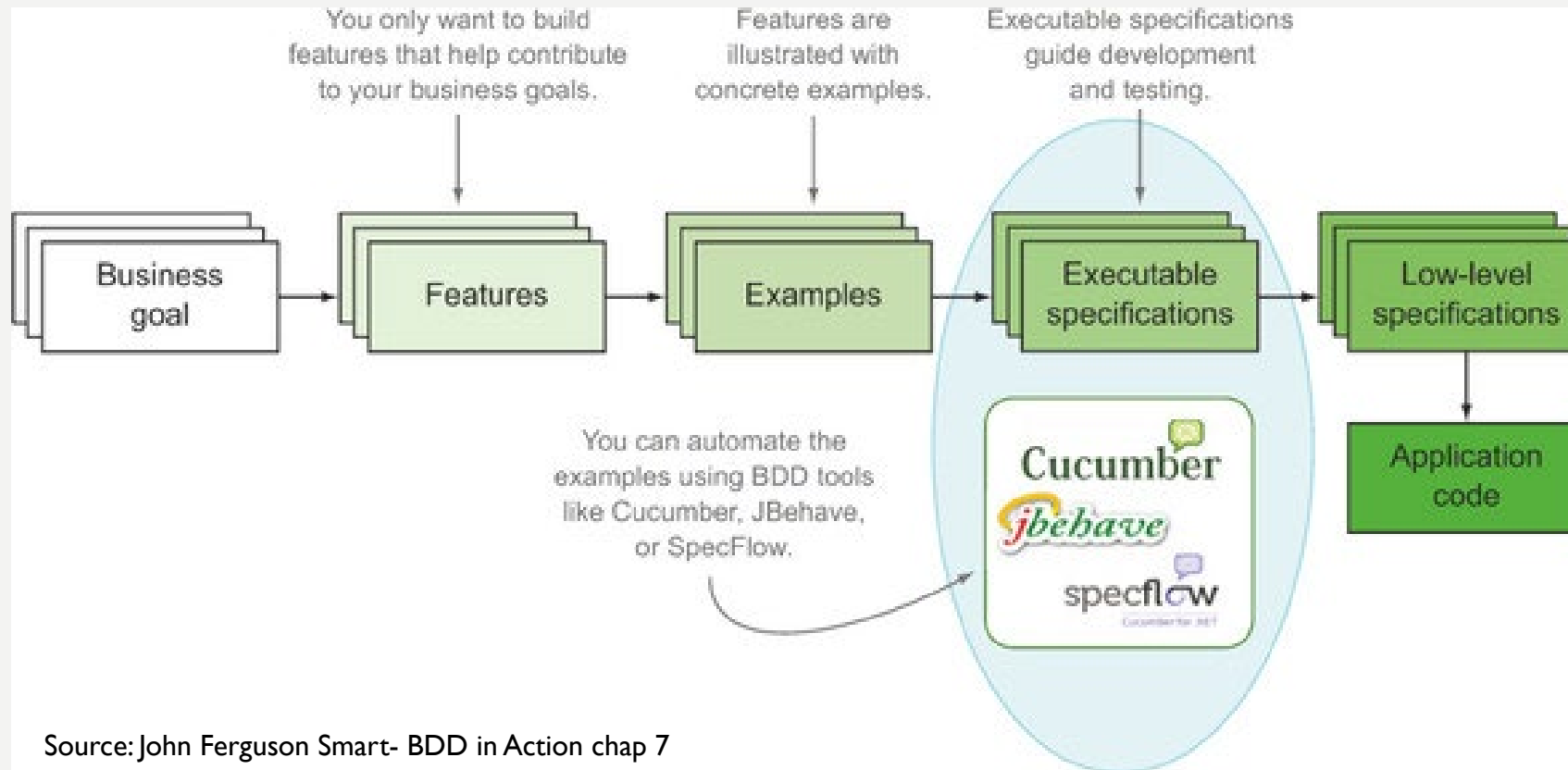
input		output
4		5
14		15

DEMO DATA TABLE

- CucumberCalculator
- CucumberCalculator2

CUCUMBER OVERVIEW - AGAIN

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



WELL-FORMED USER STORIES



Story:

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

Example:

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>

ACCEPTANCE CRITERIA VS. SCENARIOS

- 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 the 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/>

GHERKIN/CUCUMBER EXERCISE

Company X sells merchandise to wholesale and retail outlets. Wholesale customers receive a two percent discount on all orders. The company also encourages both wholesale and retail customers to pay cash on delivery by offering a two percent discount for this method of payment. Another two percent discount is given on orders of 50 or more units. Each column represents a certain type of order.

DECISION TABLE SAMPLE

Less than 50 Units Ordered	Y	Y	Y	Y	N	N	N	N
Cash on Delivery	Y	Y	N	N	Y	Y	N	N
Wholesale Outlet	Y	N	Y	N	Y	N	Y	N
Discount Rate 0%				X				
2%		X	X					X
4%	X					X	X	
6%					X			

Express the process sale function by example in feature file with Gherkin syntax and write Cucumber step definitions.

Conditions are captured in the decision table. Implement the process sale and run Cucumber tests.

Make screen dumps of test results.