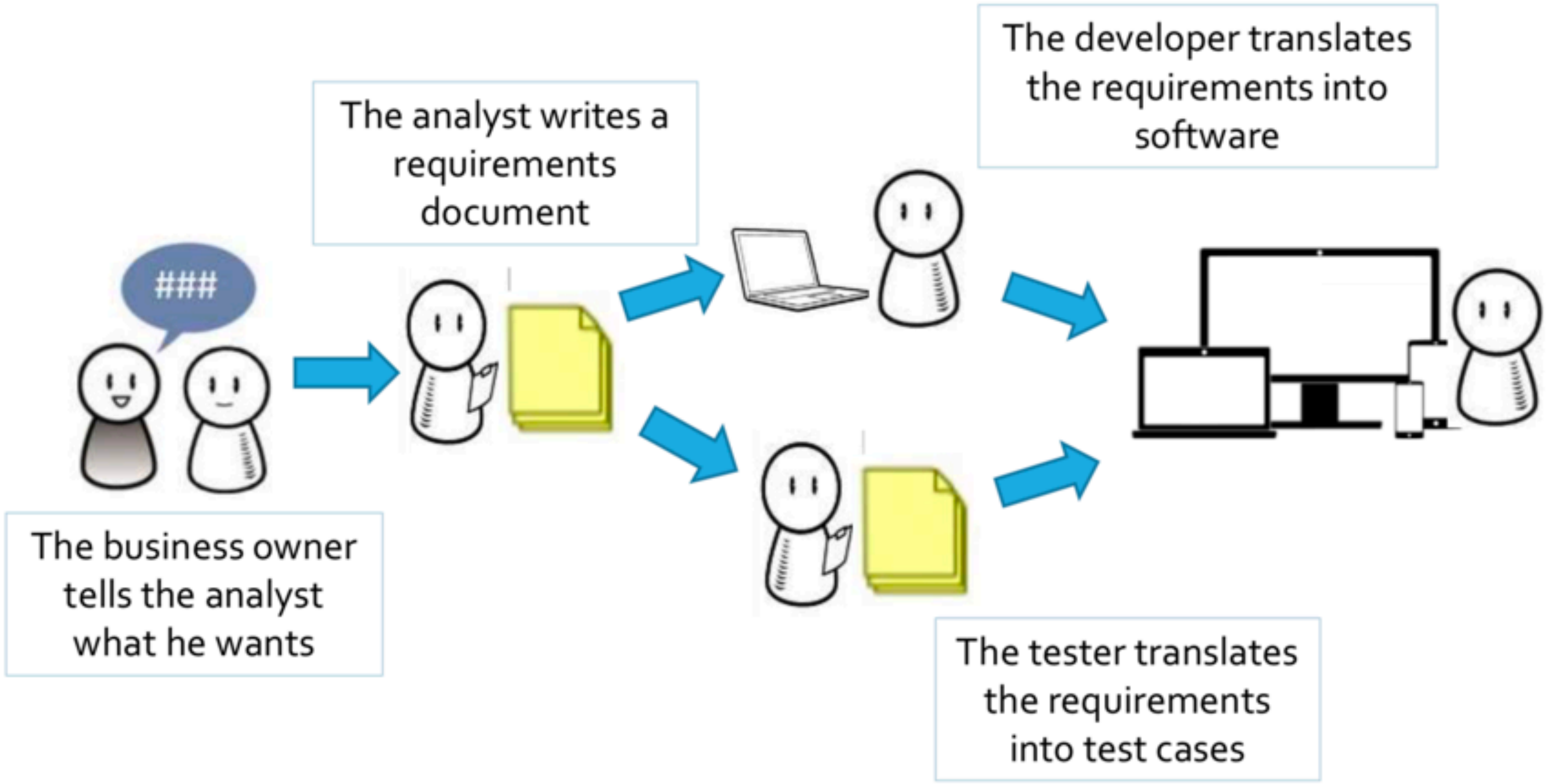
Behavior Driven Development

using Cucumber

Santhosh Kumar
22/04/2018

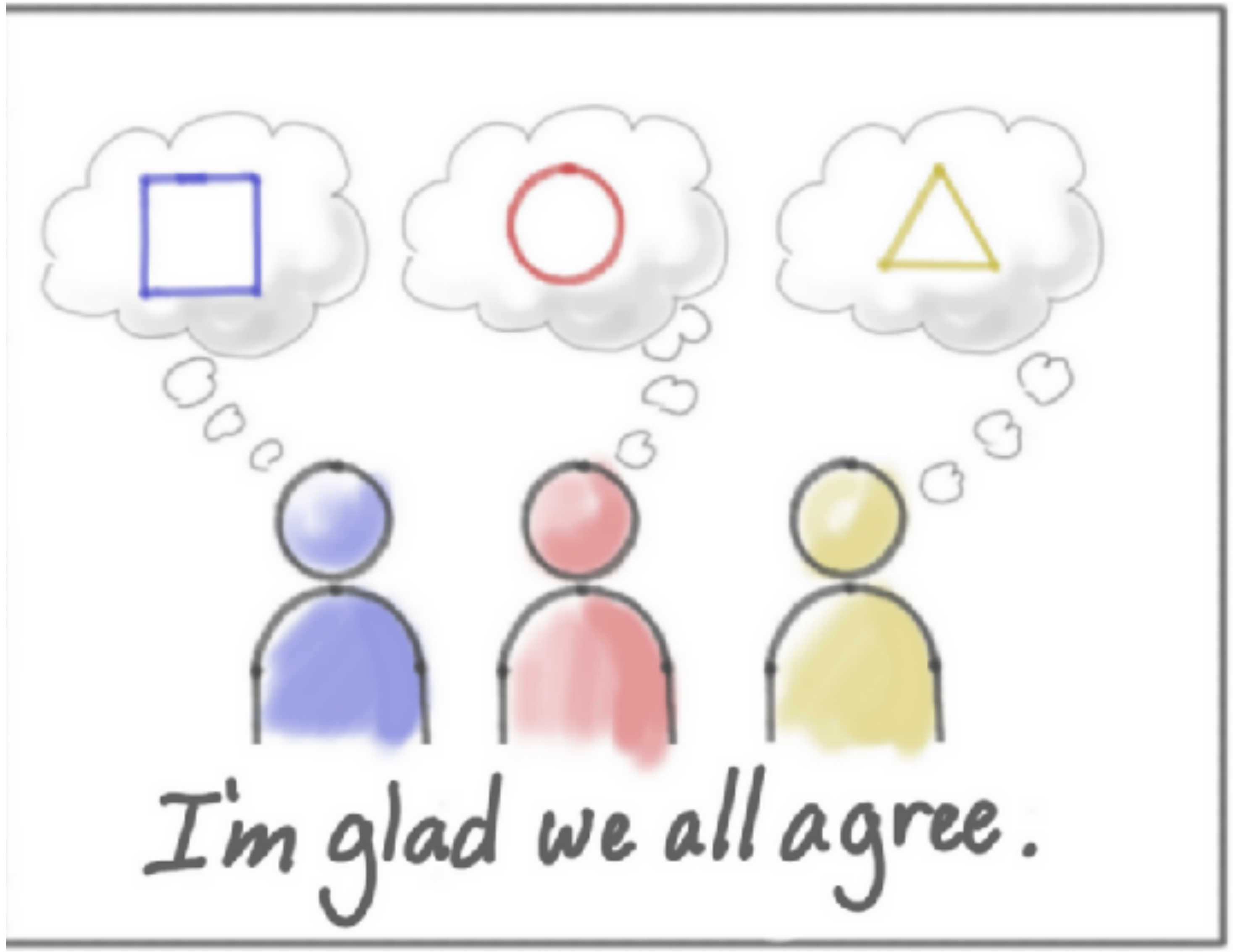
Why BDD ?

Traditional Development Process


###
The business owner tells the analyst what he wants
The analyst writes a requirements document
The developer translates the requirements into software
The tester translates the requirements into test cases

Reference : https://www.slideshare.net/JohnPatterson7/behaviour-driven-development-bdd-closing-the-loop-on-a-great-fiori-ux?from_action=save

Customer. Tester. Developer



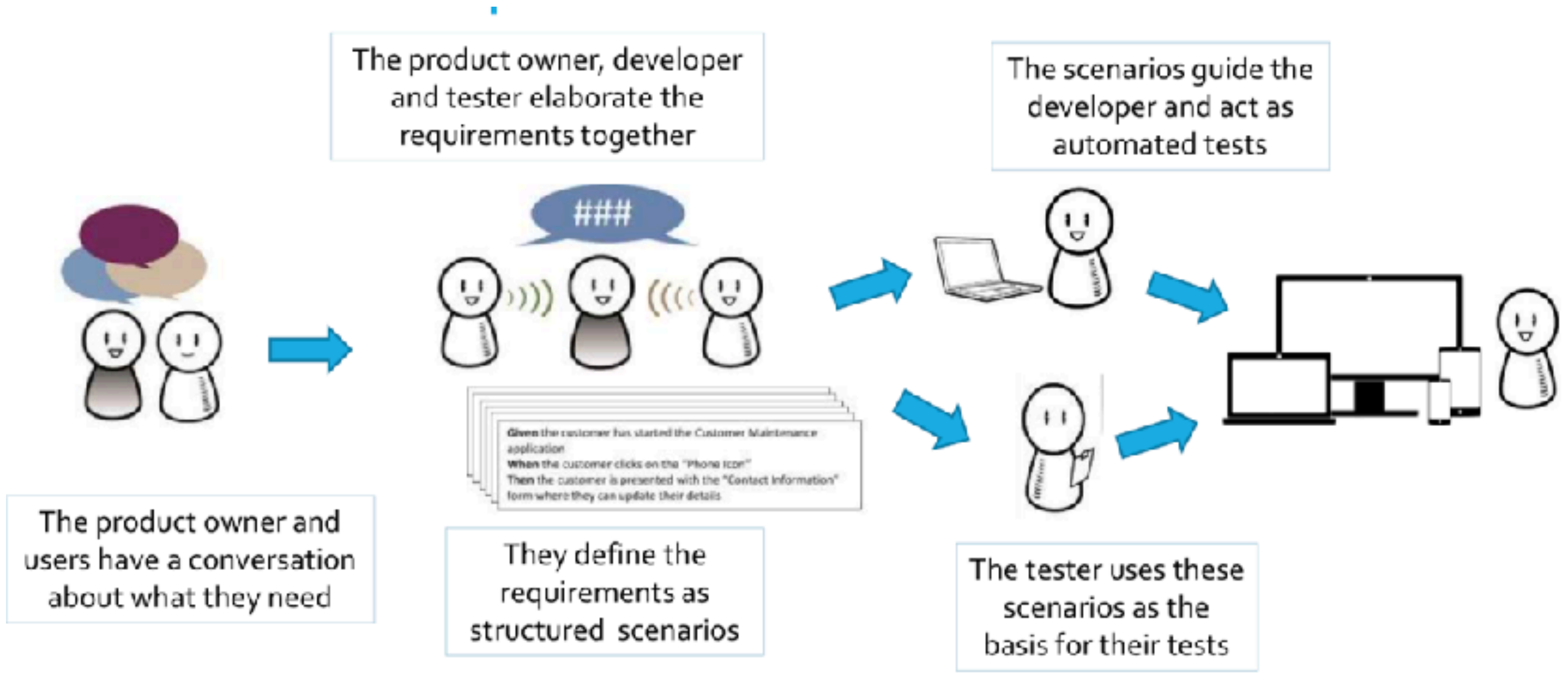
I hope they understand what I need!
Customer
I hope they follow my specifications.
Business Analyst
Requirements Gap
Requirements Documents
I hope this is what the customer wanted.
Developer

reference: https://www.slideshare.net/SumanGuha/an-introduction-to-bdd?next_slideshow=2

- Demerits of the traditional approach
    - Possibility of wastage like defects & extra features which will be uncovered in the last stage.
    - preventive mechanisms are either costly or against the agile methodology
- BDD is a software development **methodology** in which an application is **specified and designed** by how it should appear to an **outside observer**

What is BDD ?

Behavior Driven Development Process


The product owner, developer and tester elaborate the requirements together
The scenarios guide the developer and act as automated tests
###
The product owner and users have a conversation about what they need
They define the requirements as structured scenarios
The tester uses these scenarios as the basis for their tests

Given the customer has started the Customer Maintenance application
When the customer clicks on the "Phone Icon"
Then the customer is presented with the "Contact Information" form where they can update their details

Reference : https://www.slideshare.net/JohnPatterson7/behaviour-driven-development-bdd-closing-the-loop-on-a-great-fiori-ux?from_action=save

Advantages of BDD

- Automated acceptance test
  - Provides fast feedback
- Living documentation
  - Uses a ubiquitous language

How is practiced ?

- Three amigos. Business Analyst. Developer. Tester

- Three amigos* collaborate to identify the user stories
- From user stories scenarios are defined
- Scenarios are defined in a ubiquitous language
- All related scenarios are grouped in a feature file
- Development team implements the scenarios in the feature file
- Feature files serves as an automated acceptance test

* The three amigos need not be three individuals

Let’s learn a new language - Gherkin

- Keywords
    - Feature
    - Background
    - Scenario
    - Given
    - When
    - Then
    - And
    - But
    - *
    - Scenario Outline
    - Examples

Feature: *This is the feature title*
This is the description of the feature, which can span multiple lines.
You can even include empty lines, like this one:

In fact, everything until the next gherkin keyword is included in the description.

Feature: *Feedback while entering invalid credit card details*

In user testing we have seen a lot of people who made mistakes entering their credit card details. We need to be as helpful as possible to avoid losing users at this crucial stage of the transaction.

Scenarios follow this pattern

1. Get the system into a particular state
2. Poke it (or tickle it or …)
3. Examine the new state

Scenarios follow this pattern

1. Get the system into a particular state **(Given)**
2. Poke it (or tickle it or …). **(When)**
3. Examine the new state **(Then)**

Scenario: *Tickle a happy robot*
  Given *I am in good mood*
  When *you tickle me*
  Then *I will giggle*

Scenario: *Tickle a happy robot set to low power mode*
   Given *I am in good mood*
   Given *I am set to low power mode*
   When *you tickle me*
   Then *I will giggle in low volume*

Scenario: *Tickle a happy robot set to low power mode*
   Given *I am in good mood*
   And *I am set to low power mode*
   When *you tickle me*
   Then *I will giggle in low volume*

Scenario: *Tickle a happy robot set to low power mode*
   Given *I am in good mood*
   But *I am set to low power mode*
   When *you tickle me*
   Then *I will giggle in low volume*

Scenario: *Tickle a happy robot set to low power mode*
   ** I am in good mood*
   ** I am set to low power mode*
   ** you tickle me*
   ** I will giggle in low volume*

- Each scenario must make sense and be able to be executed independently of any other scenario

Data tables

Given *a user “Michael Jackson” born on August 29, 1958*
And *a user “Elvis” born on January 8, 1935*
And *a user “John Lennon” born on October 9, 1940*

Given *these users*

name	*date of birth.*
Michael Jackson	*August 29, 1958*
Elvis	*January 8, 1935*
John Lennon	*October 9, 1940*

Scenario Outline

```
Scenario: eat 5 out of 12
  Given there are 12 cucumbers
  When I eat 5 cucumbers
  Then I should have 7 cucumbers

Scenario: eat 5 out of 20
  Given there are 20 cucumbers
  When I eat 5 cucumbers
  Then I should have 15 cucumbers
```

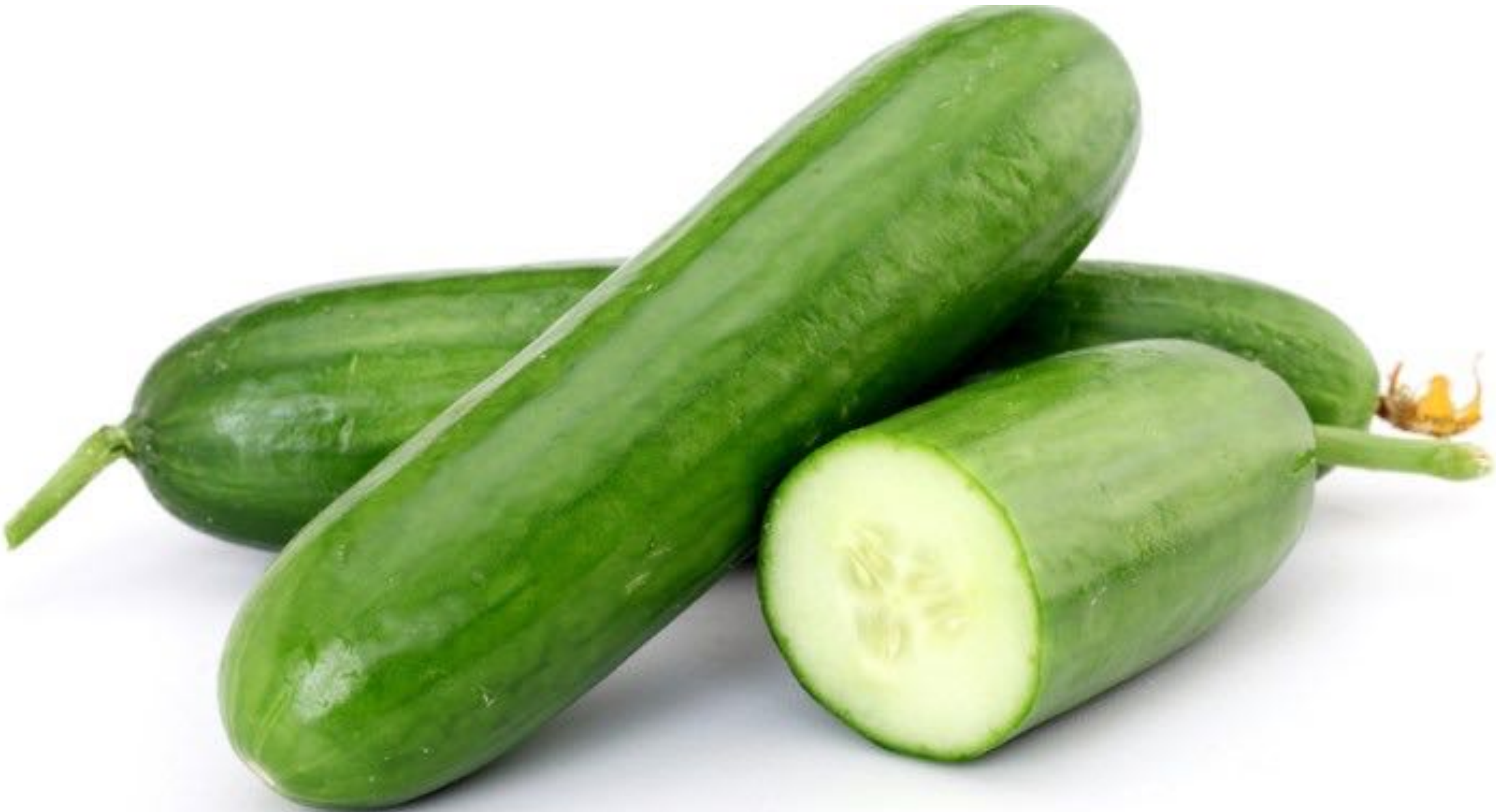
```
Scenario Outline: eating
  Given there are <start> cucumbers
  When I eat <eat> cucumbers
  Then I should have <left> cucumbers

  Examples:
    | start | eat | left |
    |  12   |  5  |  7   |
    |  20   |  5  |  15  |
```

Integrating scenarios to application

- Congratulations!!! Now you know a new language
- All good !!! How do you glue this to your application code *?

The automated test suite



Cucumber

- Cucumber is a test suite for executing your BDDs
    - It scans through the feature file to identify scenarios
    - Matches steps to step definitions
    - Execute steps , scenarios and features
    - Generate a reprot
- Originally developed as a command line tool for Rubi

Step definitions

```
Feature: Checkout
    Scenario: Checkout bananas and apples
        Given the price of the "banana" is 40rs
        And the price of the "apple" is 25rs
        When I checkout 1 "banana"
        And I checkout 1 "apple"
        Then the total price should be 65rs
```

```
public class CheckoutSteps {
    Map<String, Integer> itemPrices = new HashMap<String, Integer>();
    Checkout checkout = new Checkout();

    @Given("^the price of the \"([^\"]*)\" is (\\d+)rs$")
    public void thePriceOfTheIsRs(String itemName, int price) throws Throwable {
        itemPrices.put(itemName, price);
    }

    @When("^I checkout (\\d+) \"([^\"]*)\"$")
    public void iCheckout(int itemCount, String itemName) throws Throwable {
        checkout.add(itemCount, itemPrices.get(itemName));
    }

    @Then("^the total price should be (\\d+)rs$")
    public void theTotalPriceShouldBeRs(int total) throws Throwable {
        assertEquals(total, checkout.total());
    }
}
```

A working example

- https://github.com/santkk/learncucumber

References

- https://www.slideshare.net/JohnPatterson7/behaviour-driven-development-bdd-closing-the-loop-on-a-great-fiori-ux
- https://www.slideshare.net/SumanGuha/an-introduction-to-bdd?next_slideshow=2
- https://pragprog.com/book/hwcuc/the-cucumber-book

Questions

Thank You