45426: Teste e Qualidade de Software

BDD: behavior driven testing

Ilídio Oliveira

v2025-03-11

Learning objectives

Explain how "features/user-stories" are used as a conversation tool to build functional specifications

Write simple acceptance criteria for a user story in structured text

Write acceptance scenarios using the Gerking language

Describe the steps to implement BDD in Java using the Cucumber framework

JOliveira

"post-it" & borads metaphor

- The right granularity to distribute the work
- Traceability to requirements (usage scenarios)
- A few "post-it notes" per iteration

Goals

Steps

Buy a product

Sell a product

Register user account	Search products	View products details	Shopping cart	Checkout	Register seller account	Create produ
EC-62	EC-63	EC-64	EC-65	EC-66	EC-67	EC-68
To Do	To Do	To Do	To Do	To Do	To Do	To Do

⌄ 2020-07-31 📅 Release 1.0 ● 22 To Do ● 2 In Progress ✏ Edit 🗑 De

Edit profile	Filter products	View product photos	Add to cart	Select shipping address	Fill-in retailer form	Regster proc
EC-37	EC-56	EC-52	EC-48	EC-44	EC-28	EC-25
To Do	In Progress	In Progress	To Do	To Do	To Do	To Do

Activate account	List products	sort, filter products	Change quantity	Confirm order	Upload certificates	Product ima
EC-26	EC-20	EC-19	EC-46	EC-43	EC-27	EC-24
To Do	To Do	To Do	To Do	To Do	To Do	To Do

Sort products		Remove product		View seller profile
EC-55		EC-47		EC-22
To Do		To Do		To Do

⌄ 2020-10-30 📅 Release 2.1 ● 16 To Do ✏ Edit 🗑 Delete

Check delivery status	Advanced search	View related products	Continue shopping	Enter payment info	Inventory process	Coupon
EC-12	EC-54	EC-50	EC-8	EC-45	EC-16	EC-4
To Do	To Do	To Do	To Do	To Do	To Do	To Do

Search discount products	View product reviews		Select delivery time
EC-41	EC-51		EC-23
To Do	To Do		To Do

User stories: behavior by example

As a manager, I want to browse my existing quizzes so I can recall what I have in place and figure out if I can just reuse or update an existing quiz for the position I need now.

→ some examples

Stories and scenarios

(User) Story as the basic unit of functionality, and therefore of delivery.

Captures a feature of the system

defines the scope of the feature and its acceptance criteria.

They are also used as the basis for estimation when we come to do our planning

Can be mapped on outcomes, requirements

https://www.pivotaltracker.com/blog/principles-of-effective-story-writing-the-pivotal-labs-way

What's in a Story?
http://dannorth.net/whats-in-a-story/

Stories and testing...

```
Title (one line describing the story)

Narrative:
As a [role]                                        I
I want [feature]
So that [benefit]

Acceptance Criteria: (presented as Scenarios)  +

Scenario 1: Title
Given [context]
  And [some more context]...
When   [event]
Then   [outcome]                                   II
  And [another outcome]...

Scenario 2: ...
```

Can we write the acceptance criteria in a way that it is executable?

Oliveira

Story: the scope of a feature + its acceptance criteria.

```
Title (one line describing the story)

Narrative:
As a [role]
I want [feature]
So that [benefit]

Acceptance Criteria: (presented as Scenarios)

Scenario 1: Title
Given [context]
  And [some more context]...
When  [event]
Then  [outcome]
  And [another outcome]...

Scenario 2: ...
```

Functional view. ✓
Value for the user. ✓
Specification by examples. ✓

Credit: http://dannorth.net/whats-in-a-story/

```
Story: Account Holder withdraws cash

As an Account Holder
I want to withdraw cash from an ATM
So that I can get money when the bank is closed

Scenario 1: Account has sufficient funds
Given the account balance is \$100
 And the card is valid
 And the machine contains enough money
When the Account Holder requests \$20
Then the ATM should dispense \$20
 And the account balance should be \$80
 And the card should be returned

Scenario 2: Account has insufficient funds
Given the account balance is \$10
 And the card is valid
 And the machine contains enough money
When the Account Holder requests \$20
Then the ATM should not dispense any money
 And the ATM should say there are insufficient funds
 And the account balance should be \$20
 And the card should be returned
```

Features are described in the Gherkin Language (DSL)

<u>Gherkin</u> to describe a feature (for testing):

writing features - gherkin language ¶

The primary keywords are:

- `Feature`
- `Rule` (as of Gherkin 6)
- `Example` (or `Scenario`)
- `Given`, `When`, `Then`, `And`, `But` for steps (or `*`)
- `Background`
- `Scenario Outline` (or `Scenario Template`)
- `Examples` (or `Scenarios`)

There are a few secondary keywords as well:

- `"""` (Doc Strings)
- `|` (Data Tables)
- `@` (Tags)
- `#` (Comments)

<u>Feature</u>: what

<u>Scenario</u>: some determinable business situation

<u>Given</u>: preparation/setup (e.g.: required data)

- <u>And</u>...

<u>When</u>: the set of actions (execute).

- <u>And</u>...

<u>Then</u>: specifies the expected resulting state (assert).

- <u>And</u>...

→ <u>Sample</u>

```
Scenario: Adding a product to the cart
Given:
That I have a cart
And there is a product called "Prosecco Armani DOC"
When:
I add the product to the cart
Then:
The operation should be successful
And the cart should have been correctly updated
```

Given - describes the initial context of the scenario — the required pre-conditions we need in place before conducting the action/event that we are testing (in this case, we should have a virtual shopping cart and a specific product to add.)

When - describes the specific action/event — in many scenarios there should only be one such step (for example, adding the product to the cart). If you find yourself having to add more than one step here, you should consider if you need to break up the scenario into two or more.

Then - describes the expected outcomes of conducting the action/event in the system. These steps commonly contain various assertions that verify everything we want to check as a result of this test.

Cucumber framework for tests

Goal

common understanding of the problem ⇒ simplify the communication between all parties

Simple, human collaboration

Cucumber way

- express requirements using concrete examples (Gherkin syntax)

- create examples of behavior that are executable

- examples are found in a collaborative way (business analysts, testers and developers)

- examples can be used as acceptance tests (with additional preparation steps)

The same approach, many frameworks

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

Cucumber-JVM	Cucumber.js	Cucumber.rb
Java — official	Node.js and browsers — official	Ruby, Ruby on Rails — official
Cucumber.ml	Cucumber.cpp	Cucumber-Lua
OCaml — official	C++ — official	Lua — official
Android™	Kotlin	Cucumber-Scala
Java — official	Cucumber-JVM with Kotlin — official	Scala — official
Cucumber-Tcl	Godog	Xunit.Gherkin.Quick
Tcl — official	Go — official	C#, F#, VB with .NET Core — semi-official
Behat	Behave	SpecFlow
PHP — semi-official	Python — semi-official	C#, F#, VB.NET — semi-official

Test elements (example)

Note: For Cucumber JUnit 5, feature files need to be located under the same package structure as their corresponding step definition's steps package

@Cucumber is now deprecated...

Cucumber entry point

@Cucumber
CucumberIntegrationTest

uses config

junit-platform.properties

Uses Spring Context Config

Uses step definition

Uses feature file

@CucumberContextConfiguration
@SpringBootTest
CucumberBootstrap

extends

Step definition class
@Given, @When, @Then
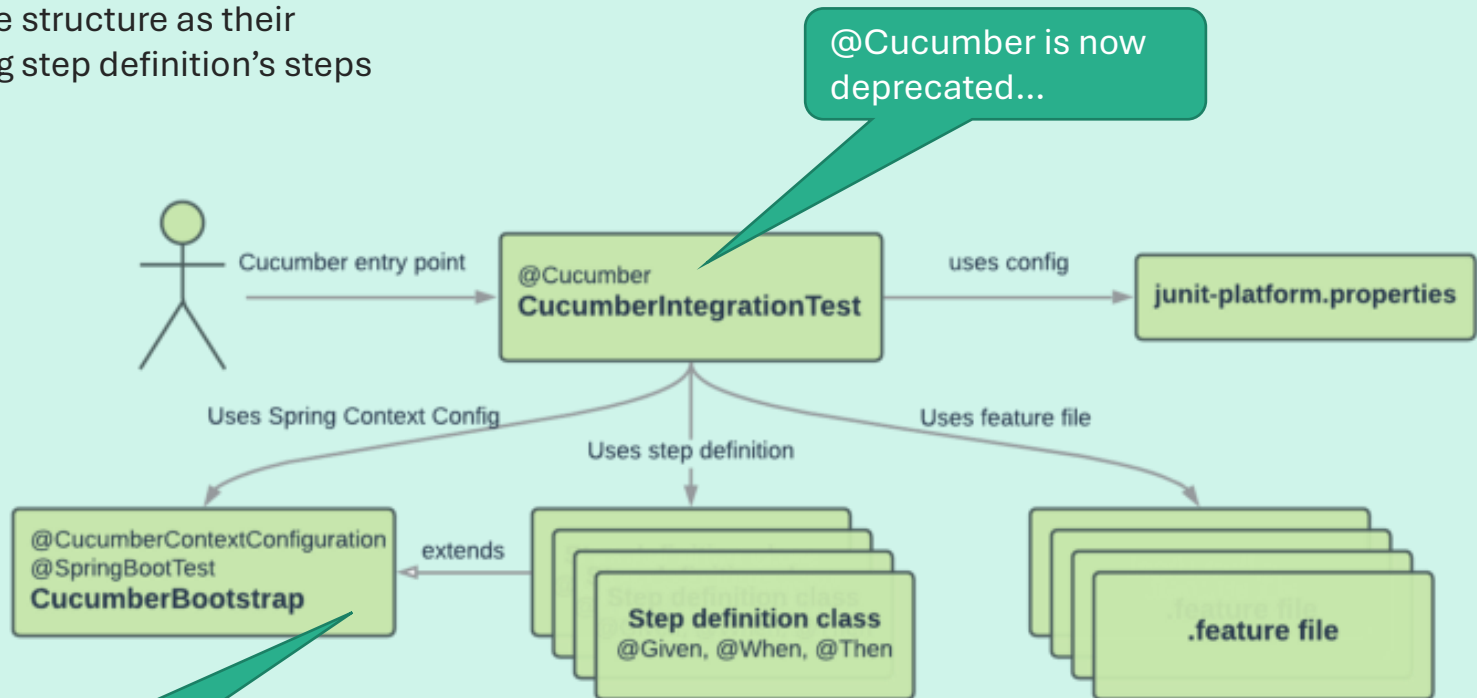
.feature file

This example uses Spring Boot. Not required though.

.feature

```
Feature: Book search
  To allow a customer to find his favourite books quickly,
  the library must offer multiple ways to search for a book.

  Background: A sample library
    Given a book with the title 'One good book' written by 'Fred Kruger' publish
    And a book with the title 'Some other book' written by 'Tim Tomson' publishe
    And a book with the title 'How to cook a dino' written by 'Fred Flintstone'
    And a book with the title 'Welcome to hell' written by 'Red Flames' publishe

  Scenario: Search books by author
    When the customer searches for books by 'Fred'
    Then 2 books should have been found
    And Book 1 should have the title 'How to cook a dino'
    And Book 2 should have the title 'One good book'
```

steps mapping

> The steps in the feature are mapped to test methods, using annotations.

```java
@Given("a book with the title {string} written by {string} published in {
public void addNewBook(final String title, final String author,  final Lo
    Book book = new Book(title, author, published);
    library.addBook(book);
}


@When("the customer searches for books by {string}")
public void the_customer_searches_for_books_by(String author) {
    // Write code here that turns the phrase above into concrete actions
    result = library.findBooksByAuthor( author);
}


@Then("{int} books should have been found")
public void verifyAmountOfBooksFound(final int booksFound) {
    assertThat(result.size()).isEqualTo(booksFound);
}
```

Cucumber reads specifications from plain-language text files called _features_, examines them for _scenarios_ to test.

Each scenario is a list of _steps_ for Cucumber to work through.

Along with the features, you give Cucumber a set of _step definitions_, which map the business-readable language of each step into code to carry out whatever action is being described by the step.

The step definition itself will probably just be one or two lines of code that delegate to a library of _support code_, specific to the domain of your application.

Sometimes that may involve using an _automation library_, like the browser automation library Selenium.
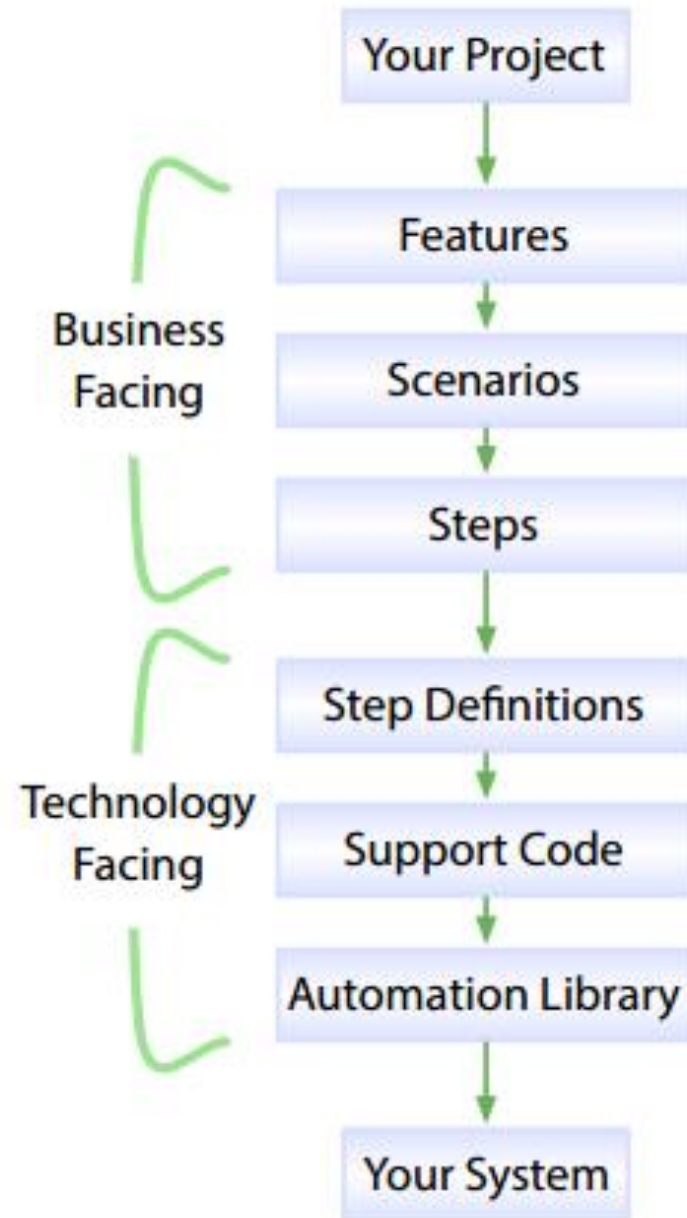
```
Scenario Outline: Several additions
  When I add <a> and <b>
  Then the result is <c>
  Examples: Single digits
    | a | b | c  |
    | 1 | 2 | 3  |
    | 3 | 7 | 10 |
```

```java
@When("I add {int} and {int}")
public void iAddAnd(int arg0, int arg1) {
    calc.push(arg0);
    calc.push(arg1);
    calc.push( arg: "+");
}

@Then("the result is {int}")
public void theResultIs(int arg0) {
    assertEquals( arg0, calc.value().intValue() );
}
```

```gherkin
Feature: Book search with table
  To allow a customer to find his favourite books quickly, t

  Background: A sample library
    Given the following books
      | title           | author          | published   |
      | One good book    |Fred Kruger| 2013-03-14|
      | Some other book  | Tim Tomson  | 2014-08-23 |
      | How to cook a dino  | Fred Flintstone | 2012-01-01 |
      | Welcome to hell     | Red Flames    | 2021-02-01 |

  Scenario: Search books by author
    When the customer searches for books by 'Fred'
    Then 2 books should have been found
    And Book 1 should have the title 'How to cook a dino'
    And Book 2 should have the title 'One good book'
```

Specifications can include data as tables to feed the tests.

```java
@DataTableType
public Book bookEntry(Map<String, String> entry){
    return new Book(
            entry.get("title"),
            entry.get("author"),
            textToDate( entry.get("published") ) );
}


@Given("the following books")
public void theFollowingBooks( List<Book> someBooks ) {
    for (Book book0:someBooks) {
        library.addBook(book0);
    }
}
```

I Oliveira

```
Funcionalidade: Exemplo de uso de tabela de dados

Cenário: Exemplo de uso de tabela de dados
    Dado Que a minha biblioteca esta inicializada vazia
    E a seguinte tabela de livros:
        | titulo    | Numero de Paginas | Topico     | Data de Publicacao | Autores |
        | LivroUm   | 42                | COMPUTACAO | 2020               | Nilton  |
        | LivroDois | 150               | ROMANCE    | 2021               | Santos  |
    Quando Eu pesquiso o livro "LivroUm" e COMPUTACAO
    Entao Eu encontro o livro com "LivroUm"
    Quando Eu pesquiso o livro "LivroDois" e ROMANCE
    Entao Eu encontro o livro com "LivroDois"
```

https://cucumber.io/docs/gherkin/languages/

Keywords available in different languages (localization)

BDD as a work process

A simplified Workflow (Suggestion)

- Get users, developers, testers, product-owners etc.. together
- They describe the behaviour of a new feature in plain text and using the Gherkin syntax
- Developers add the derived feature-files to the project and integrate the cucumber-junit-testrunner
- Run the tests and watch them fail – cucumber prints snippets for the glue code that can be used for writing the step/glue-classes.
- Write the code to make the first test (step) pass
- Repeat until everything is green

BDD: Behavior-driven development

BDD is a test-centric software development process that grew out of Test-Driven Development (TDD). BDD focuses on clearly identifying the desired behavior of a feature from the very start.

Technical (e.g., developers) and non-technical (e.g., business analysts) participants in a software project collaborate to define a common understanding of expected behaviors.

Behaviors are identified using specification by example: behavior specs are written to illustrate the desired behavior with realistic examples. They serve as both the product's requirements/acceptance criteria (before development) and its test cases (after development).

Gherkin is one of the most popular languages for writing formal behavior specifications. With automation tools, scenarios can be turned into automated test cases
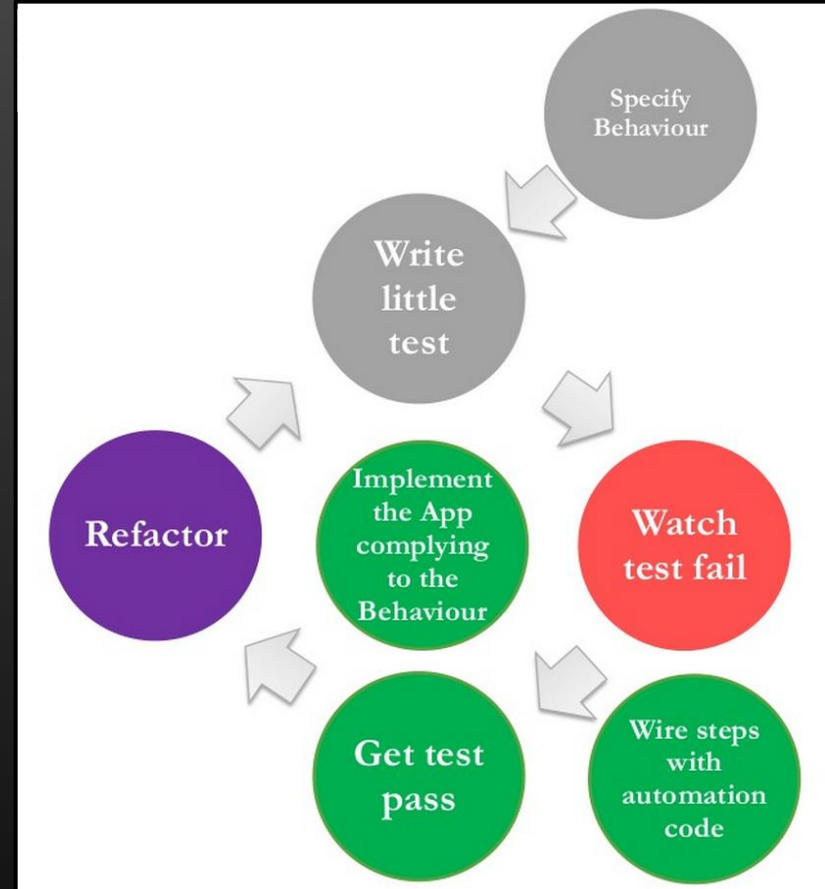
Credit: Nalin Goonawardana

Views from Robert C. Martin

BDD is a variation on TDD.

Whereas in TDD we drive the development of a module by "first" stating the requirements as unit tests, in BDD we drive that development by first stating the requirements as *requirements*.

The form of those requirements is fairly rigid, allowing them to be interpreted by a tool that can execute them in a manner that is similar to unit tests.

https://sites.google.com/site/unclebobconsultingllc/the-truth-about-bdd

BDD vs TDD (xUnit Level)

BDD

Top-down

Human readable

Sharable to all the team

Business-facing

Tangible requirements.

TDD + Unit testing

Bottom-up

Programming language

☹

Developer-facing

Module contracts.

O outsystems | Blog   All Articles   Perspectives   Dev Zone   🔍   ⊕ EN

Dev Zone

Your Complete Guide To BDD Testing In OutSystems

João Proença - September 08, 2020 - 23 min read

The primary purpose of BDD frameworks is to support Behavior-Driven Development, where all technical (e.g., developers) and non-technical (e.g., business analysts) participants in a software project collaborate to define a common understanding of how the software should behave.

24

Resources and readings

Sundberd, T., "Where should you use Behaviour Driven Development, BDD?"

https://smartbear.com/learn/automated-testing/is-bdd-right-for-you/

Kops, "**BDD Testing with Cucumber, Java and Junit**"