45426: Teste e Qualidade de Software

BDD: behavior driven testing

Ilídio Oliveira

v2022-04-05

universidade de aveiro
departamento de eletrónica,
telecomunicações e informática
deti

Learning objectives

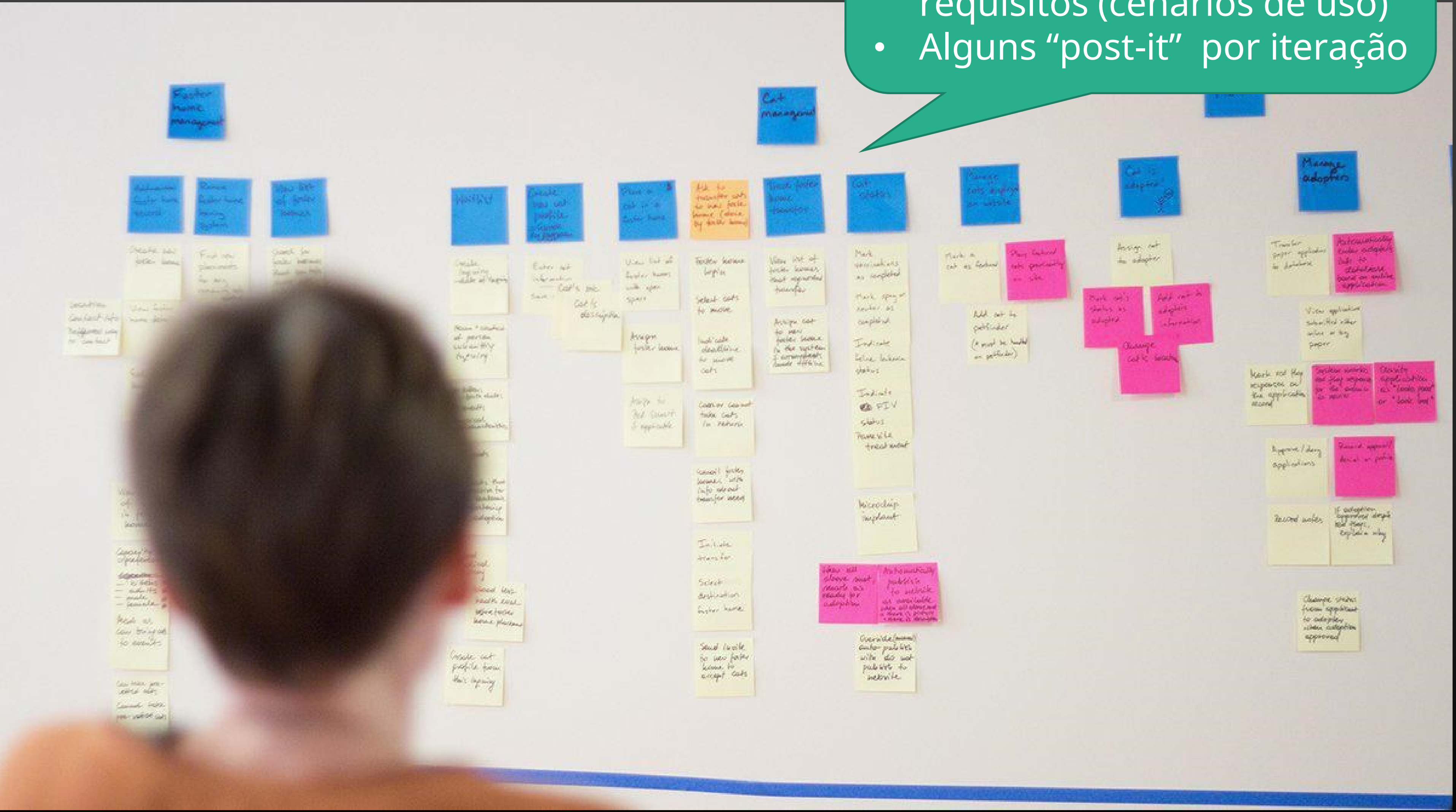
Explain how “features/user-stories” are used as a conversation tool to build functional specifications

Write simple acceptance criteria for a user story in structured text

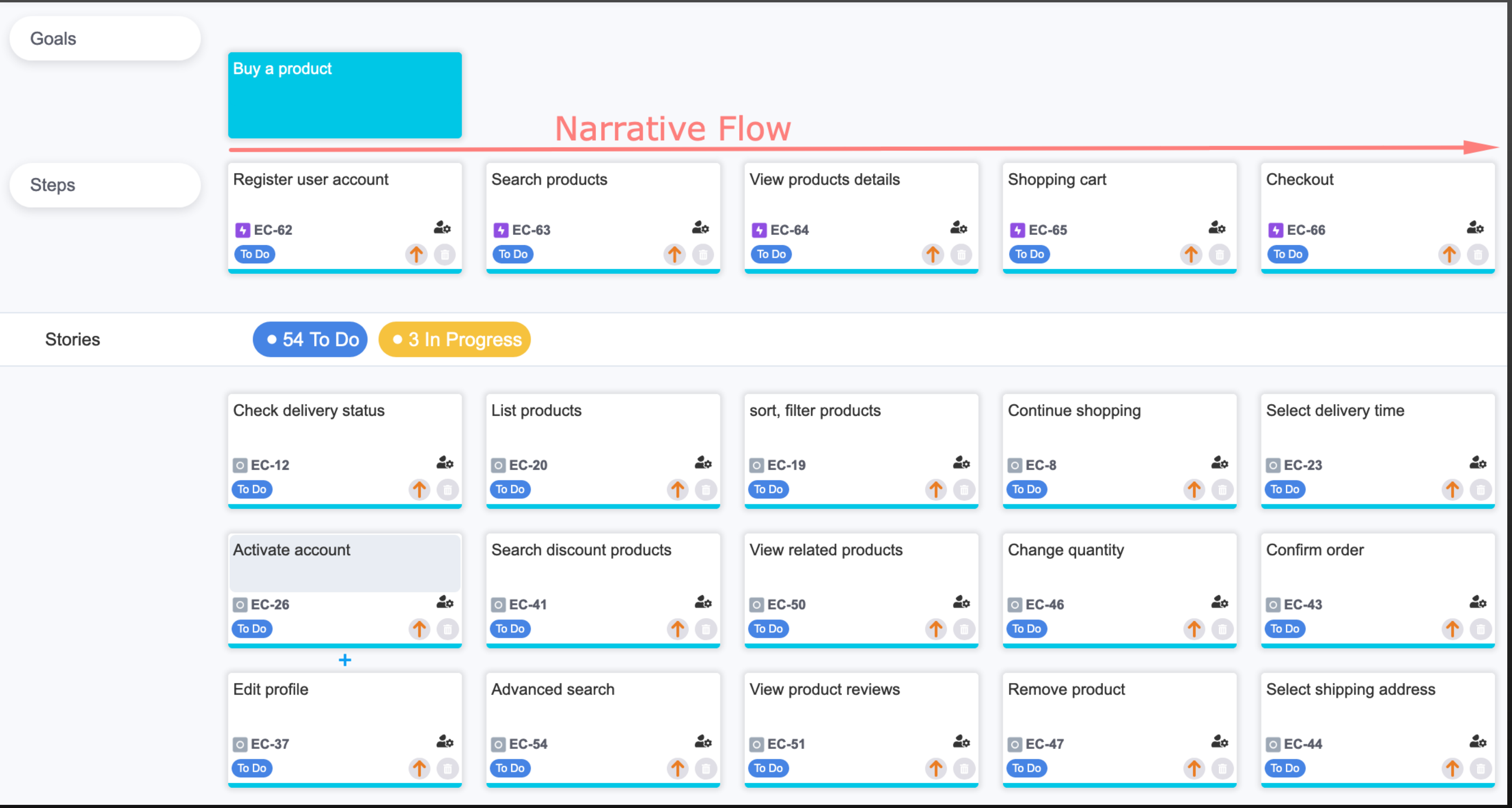
Write acceptance scenarios using the Gerking language

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

A metáfora do "post-it"



- Granularidade adequada para distribuir o trabalho
- Rastreabilidade para os requisitos (cenários de uso)
- Alguns "post-it" por iteração


Goals

Buy a product

Narrative Flow

Steps

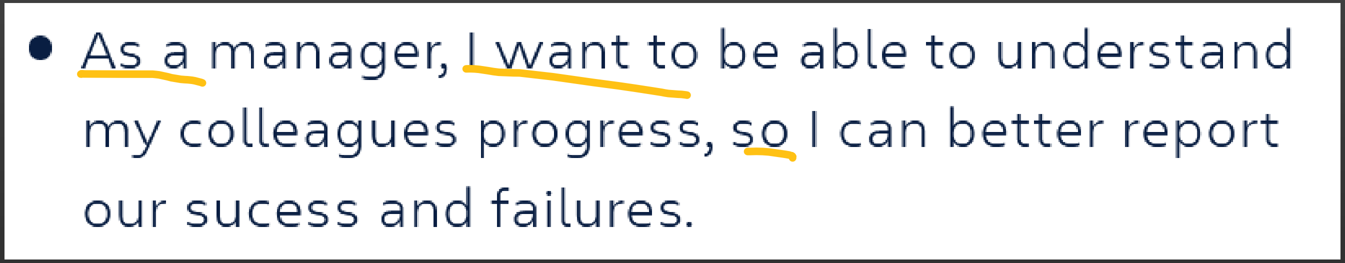
Register user account EC-62 To Do	Search products EC-63 To Do	View products details EC-64 To Do	Shopping cart EC-65 To Do	Checkout EC-66 To Do

Stories ● 54 To Do ● 3 In Progress

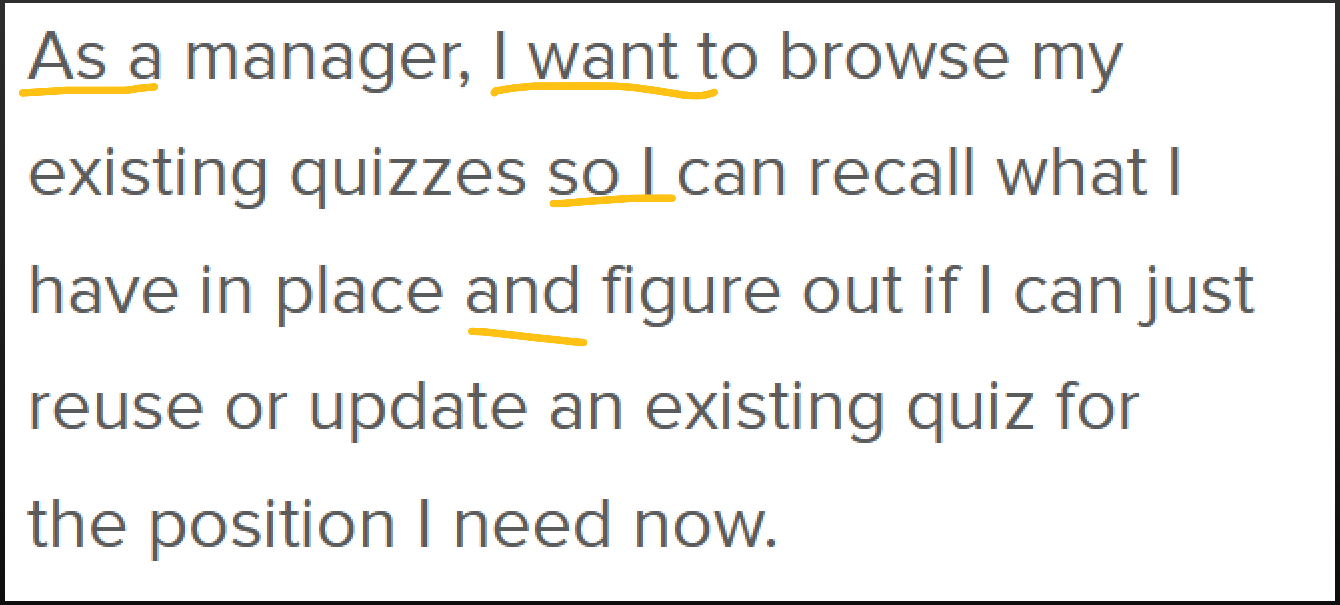
Check delivery status EC-12 To Do	List products EC-20 To Do	sort, filter products EC-19 To Do	Continue shopping EC-8 To Do	Select delivery time EC-23 To Do
Activate account EC-26 To Do	Search discount products EC-41 To Do	View related products EC-50 To Do	Change quantity EC-46 To Do	Confirm order EC-43 To Do
Edit profile EC-37 To Do	Advanced search EC-54 To Do	View product reviews EC-51 To Do	Remove product EC-47 To Do	Select shipping address EC-44 To Do

https://www.devsamurai.com/en/agile-user-story-mapping-for-jira/

User stories: behavior by example



- As a manager, I want to be able to understand my colleagues progress, so I can better report our sucess and failures.



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

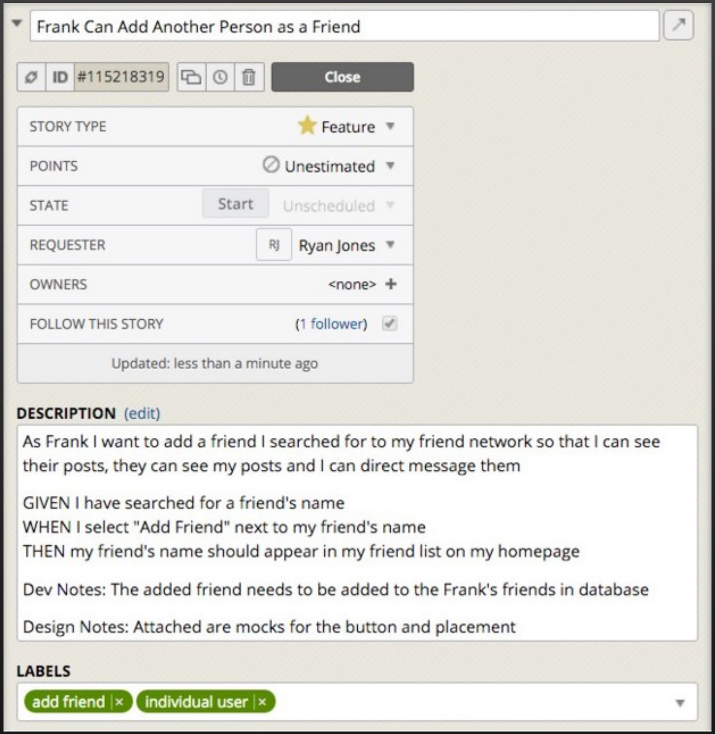
(User) Story as the basic unit of functionality, and therefore of delivery.

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


Frank Can Add Another Person as a Friend

ID #115218319 Close

STORY TYPE	Feature
POINTS	Unestimated
STATE	Start Unscheduled
REQUESTER	RJ Ryan Jones
OWNERS	<none> +
FOLLOW THIS STORY	(1 follower)

Updated: less than a minute ago

DESCRIPTION (edit)

As Frank I want to add a friend I searched for to my friend network so that I can see their posts, they can see my posts and I can direct message them

GIVEN I have searched for a friend's name
WHEN I select "Add Friend" next to my friend's name
THEN my friend's name should appear in my friend list on my homepage

Dev Notes: The added friend needs to be added to the Frank's friends in database

Design Notes: Attached are mocks for the button and placement

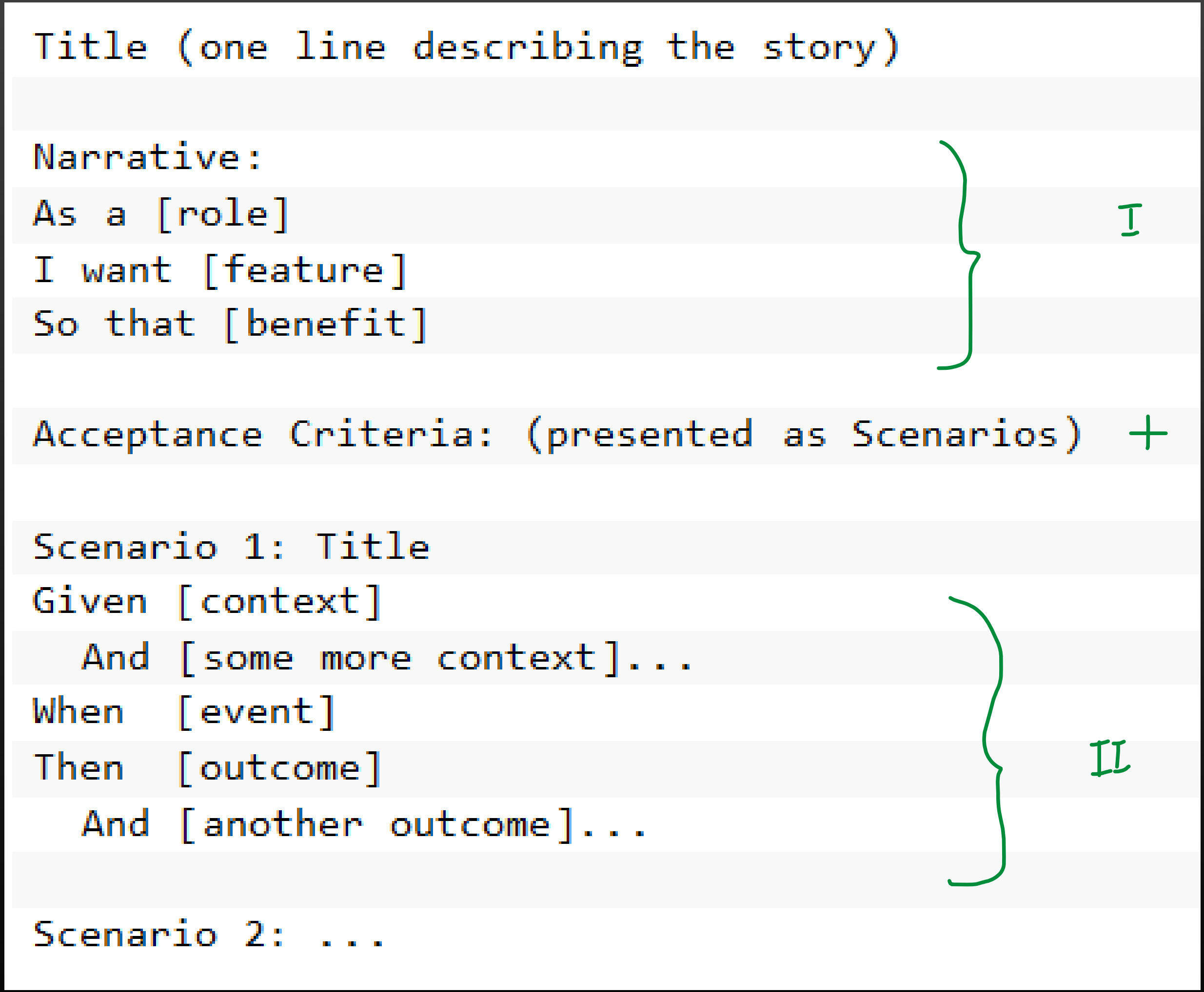
LABELS

add friend × individual user ×

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/

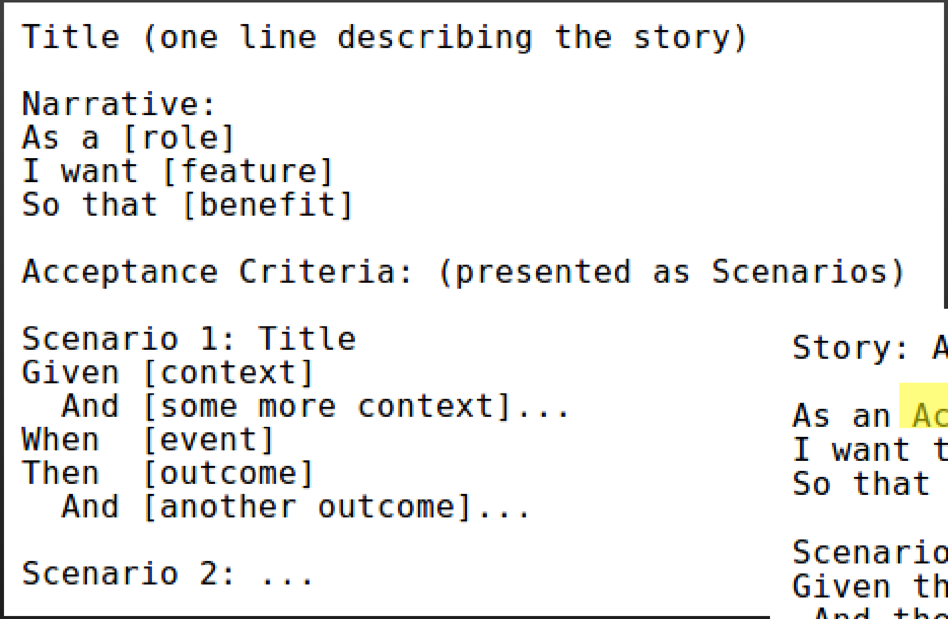
A story and the tests...



```
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: ...
```

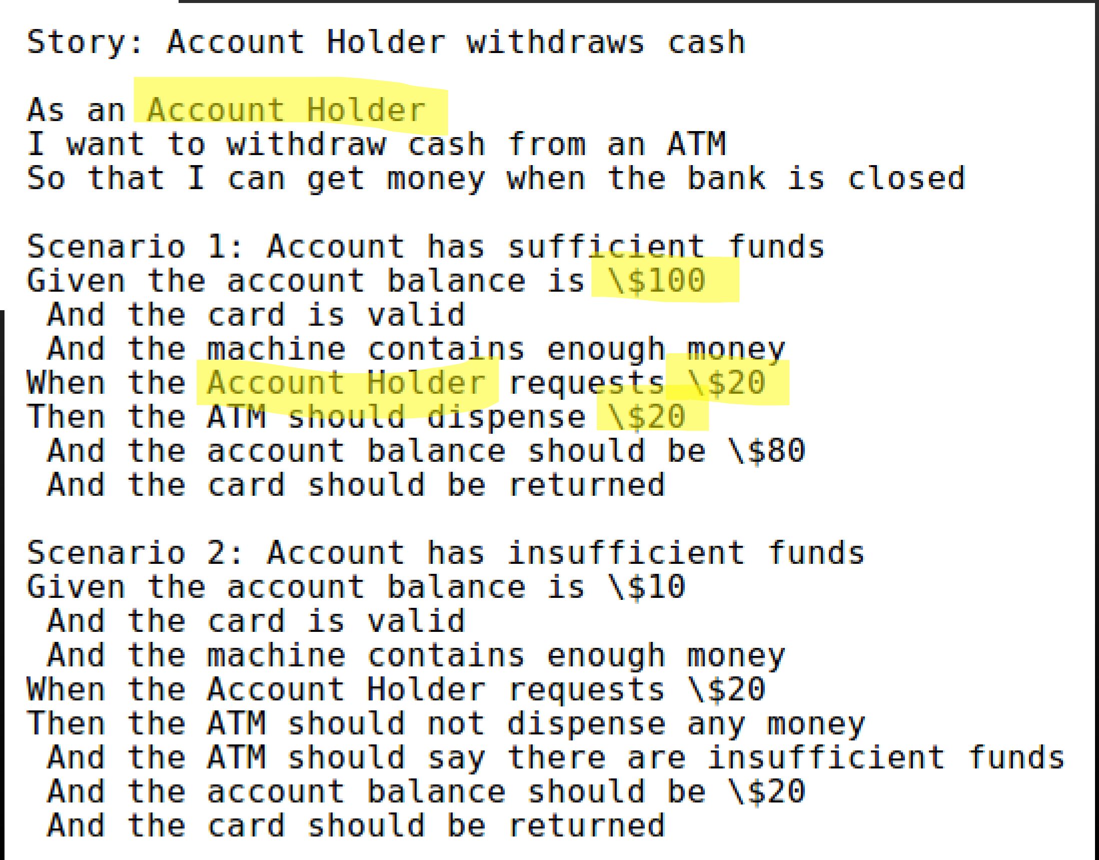
Can we write the acceptance criteria in a way that it is <u>executable</u>?

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: ...
```

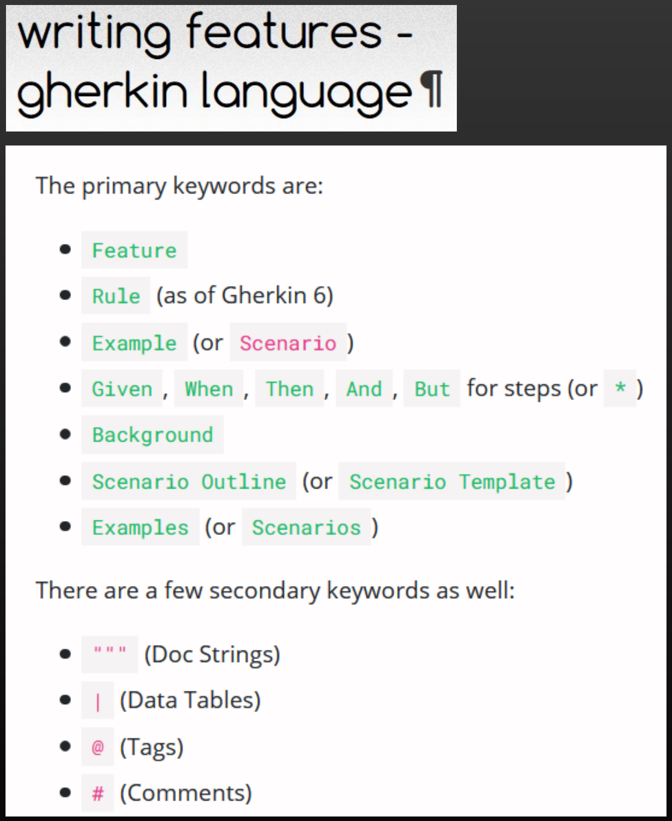
Functional view.
Value for the user.
Specification by examples.



```
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
```

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

Features are described in the Gherkin Language (DSL)



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)

Gherkin to describe a feature (for testing):

Feature: what

Scenario: some determinable business situation

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

- And...

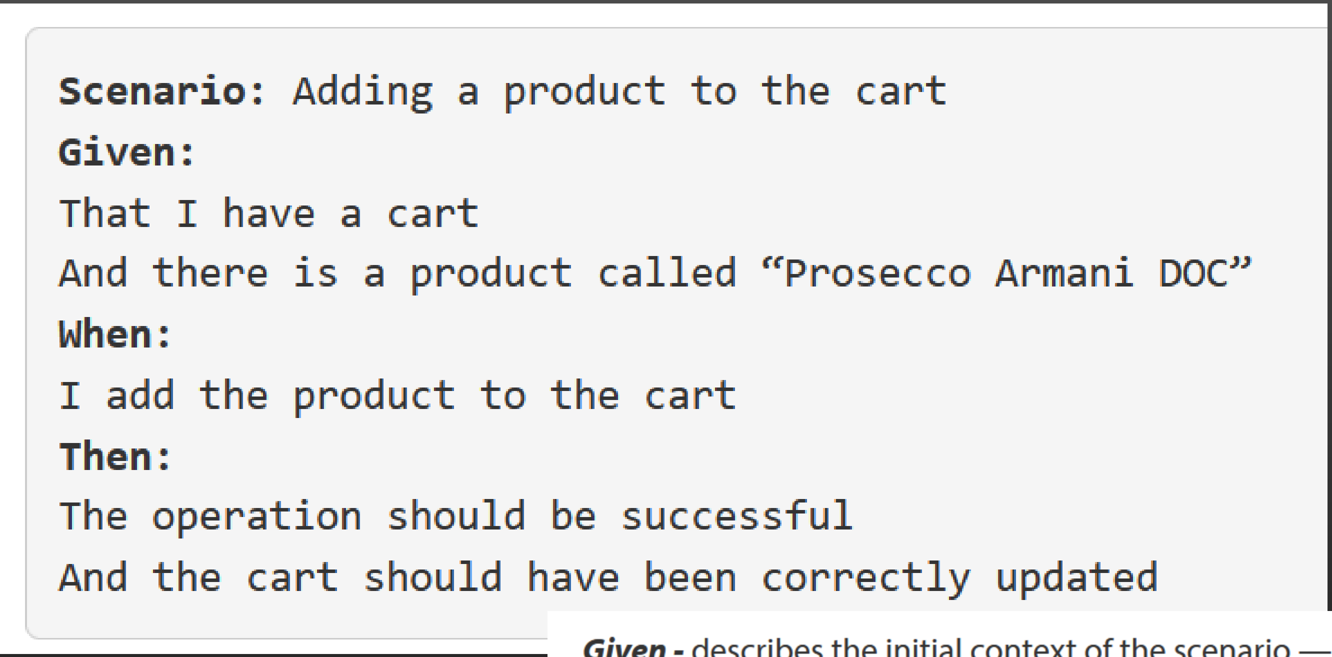
When: the set of actions (execute).

- And...

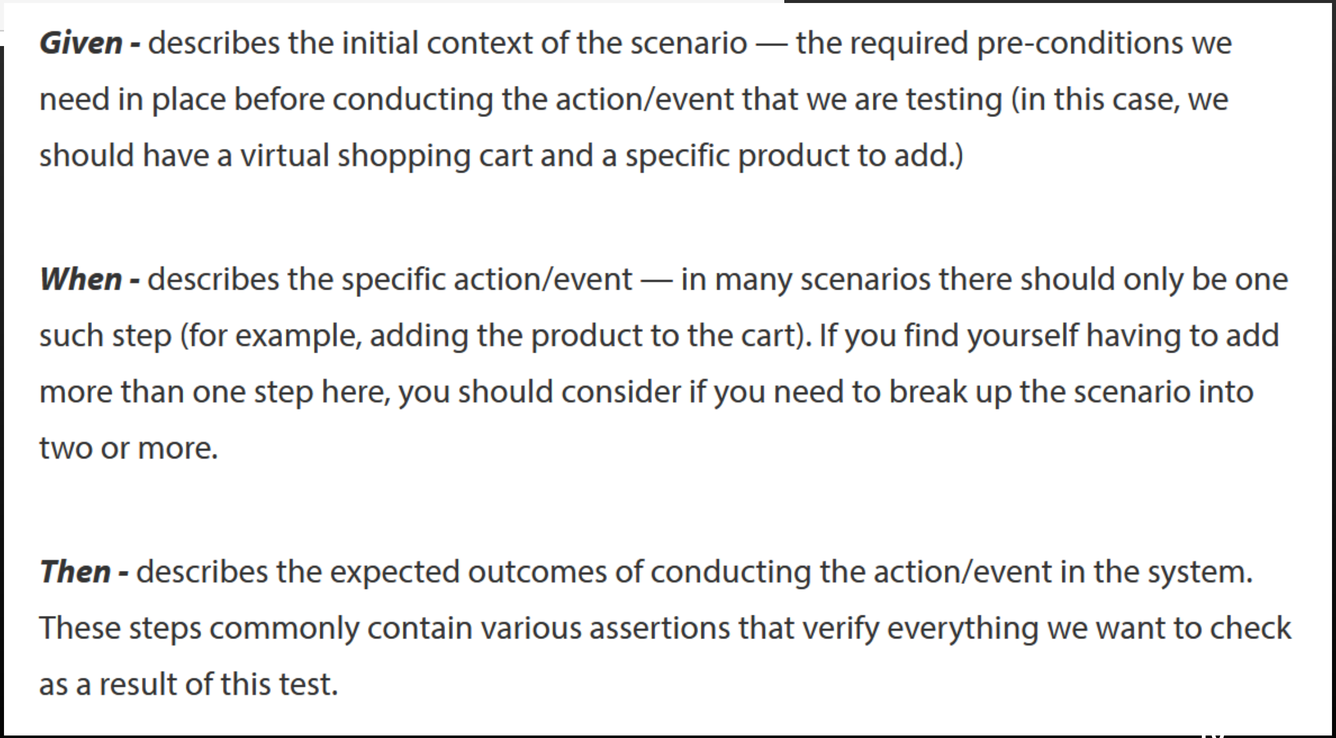
Then: specifies the expected resulting state (assert).

- And...

Sample



```
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 tool

Goal

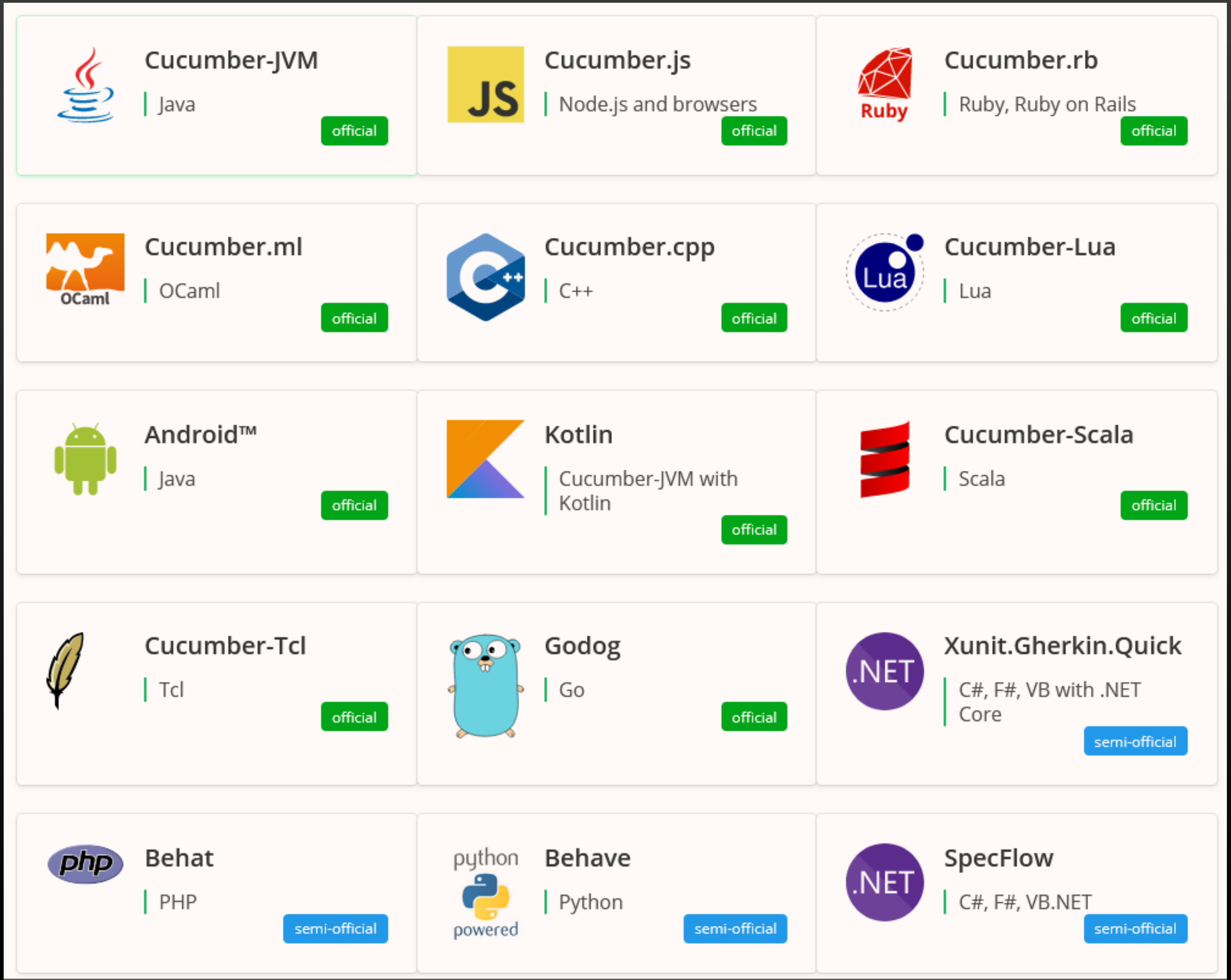
common understanding of the problem ⇒ simplify the communication between all parties

Cucumber way

- express requirements using concrete examples
- 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
| Java
official

Cucumber.js
| Node.js and browsers
official

Cucumber.rb
| Ruby, Ruby on Rails
official

Cucumber.ml
| OCaml
official

Cucumber.cpp
| C++
official

Cucumber-Lua
| Lua
official

Android™
| Java
official

Kotlin
| Cucumber-JVM with Kotlin
official

Cucumber-Scala
| Scala
official

Cucumber-Tcl
| Tcl
official

Godog
| Go
official

Xunit.Gherkin.Quick
| C#, F#, VB with .NET Core
semi-official

Behat
| PHP
semi-official

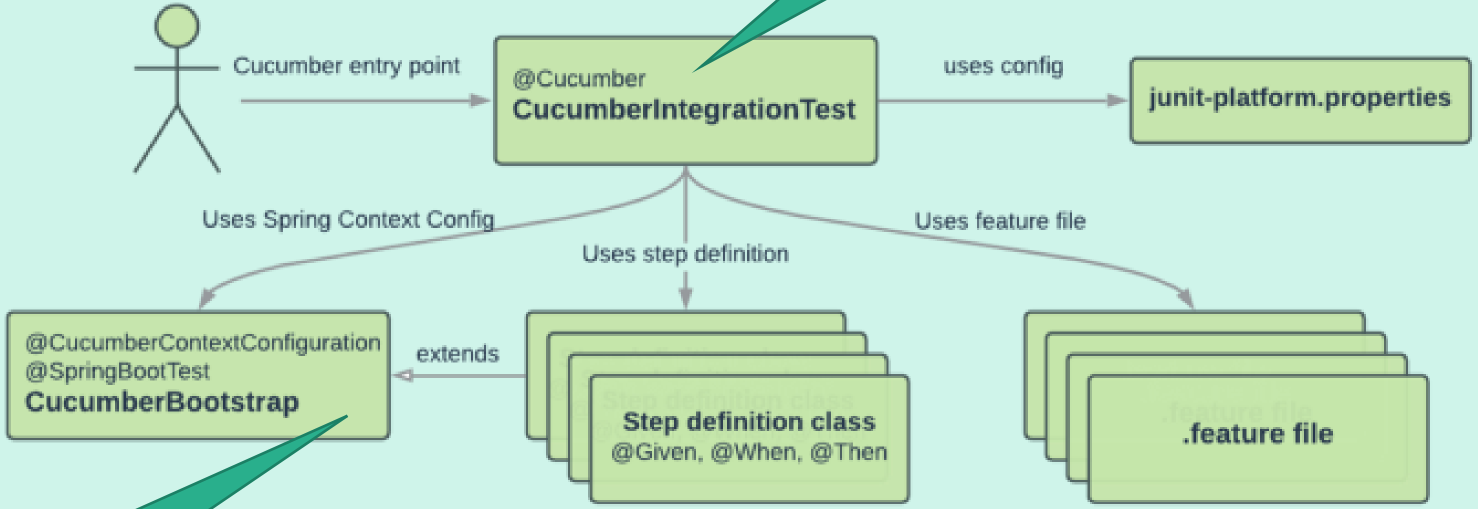
Behave
| Python
semi-official

SpecFlow
| C#, F#, VB.NET
semi-official

I Oliveira

Test elements (example)

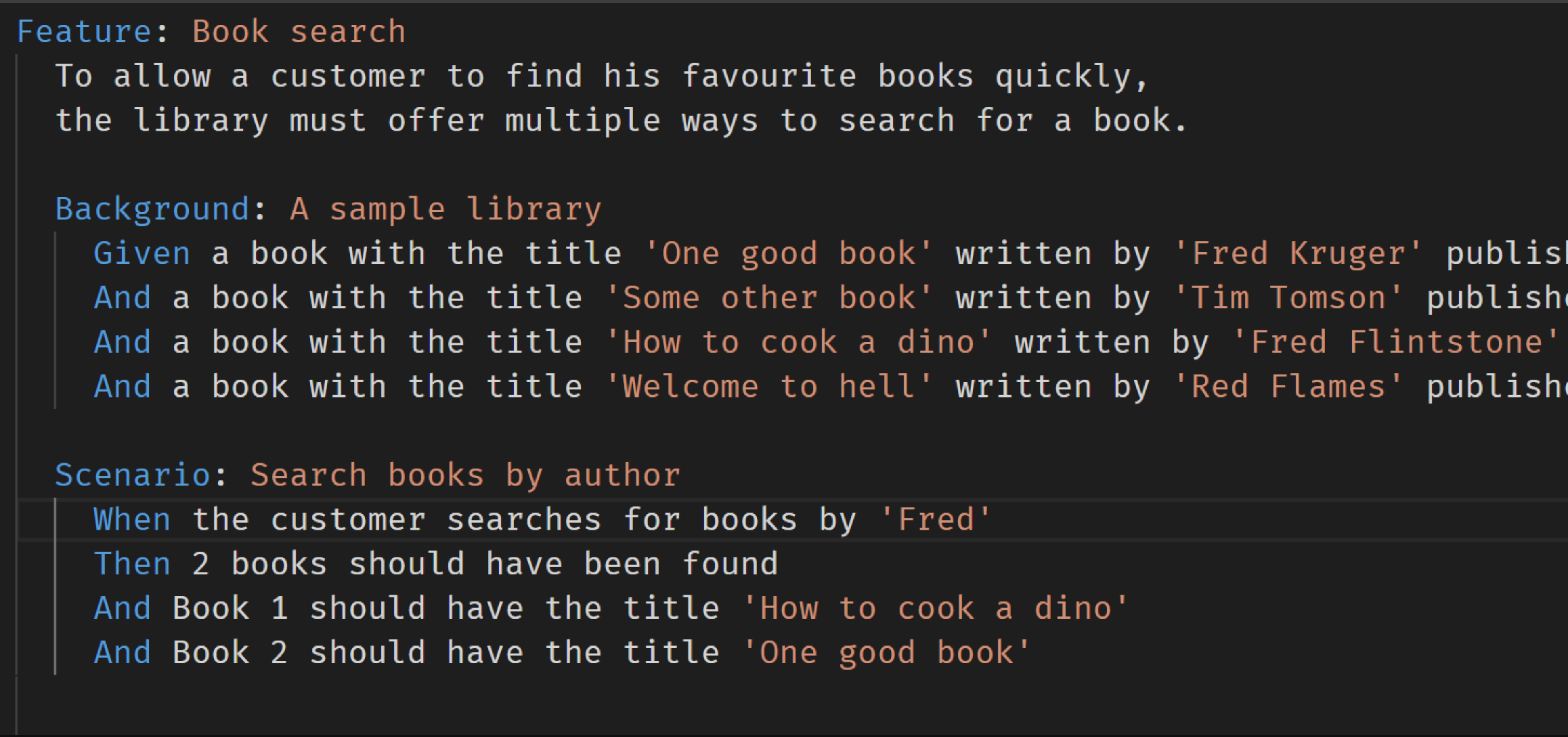
Note: For Cucumber JUnit 5, feature files need to be located under the same package structure as their corresponding step definition's steps package



@Cucumber is now deprecated...

This example uses Spring Boot. Not required though.

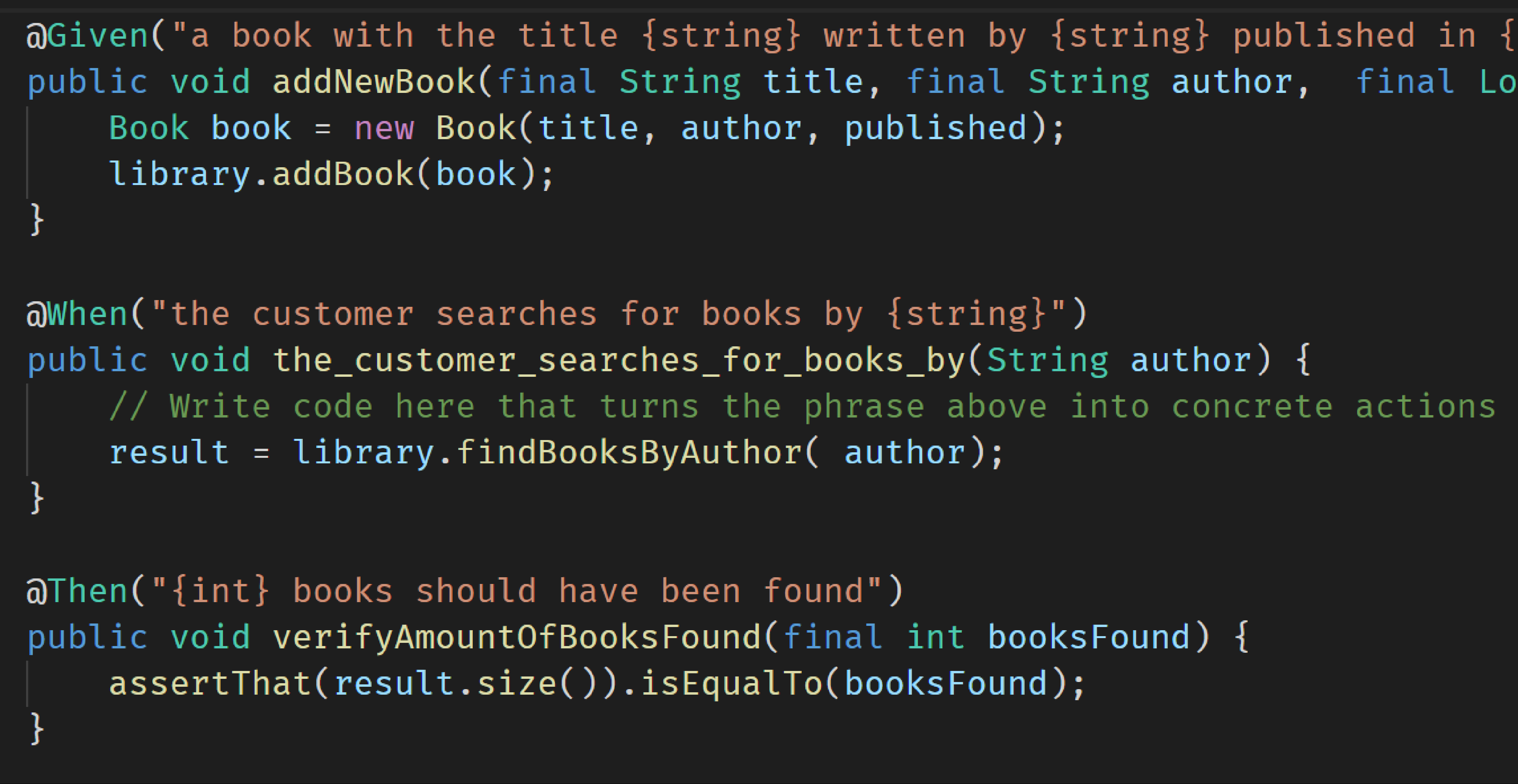
.feature



```
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' publis
    And a book with the title 'Some other book' written by 'Tim Tomson' publish
    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' publish

  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.



```
@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);
}
```

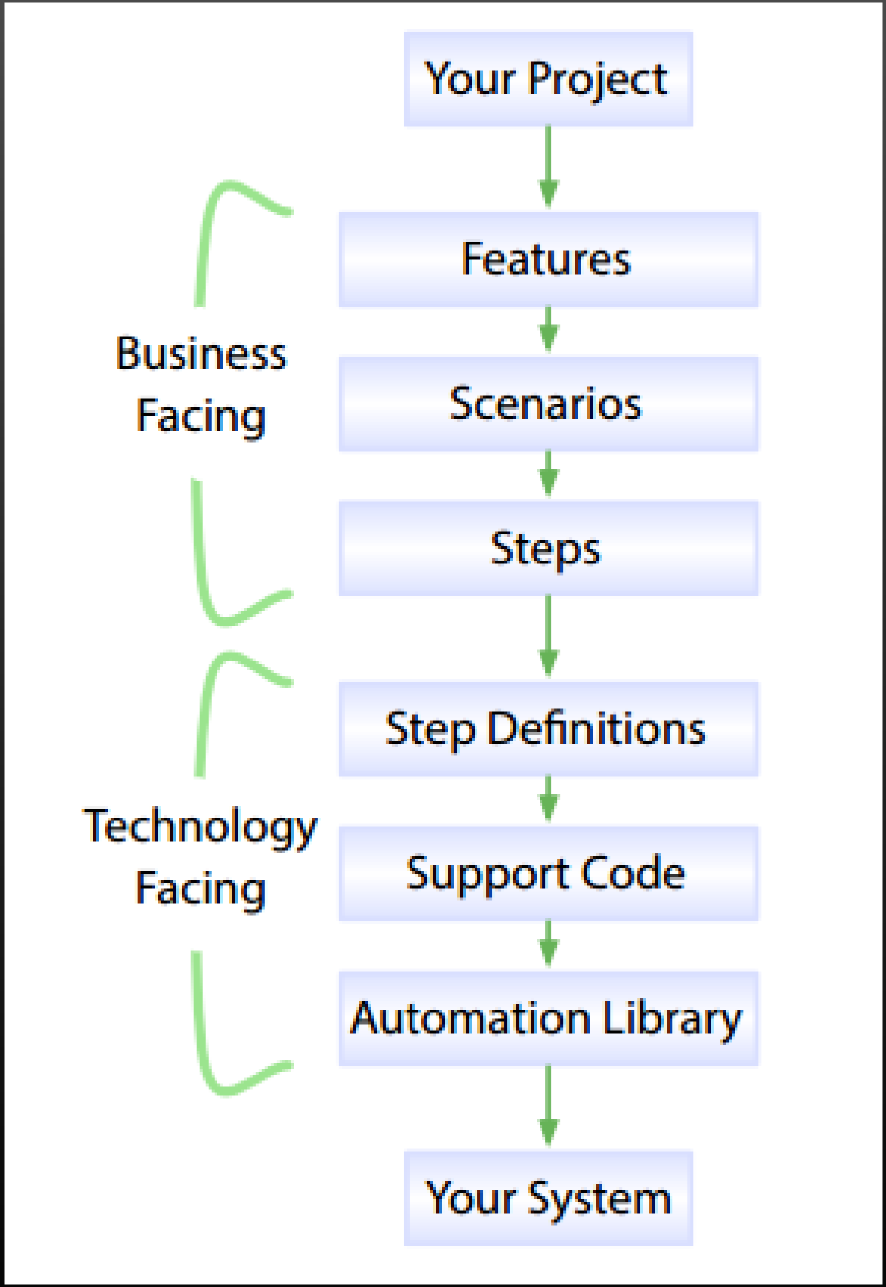
Cucumber reads specifications from plain-language text files called *features*, examines them for *scenarios* to test.

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

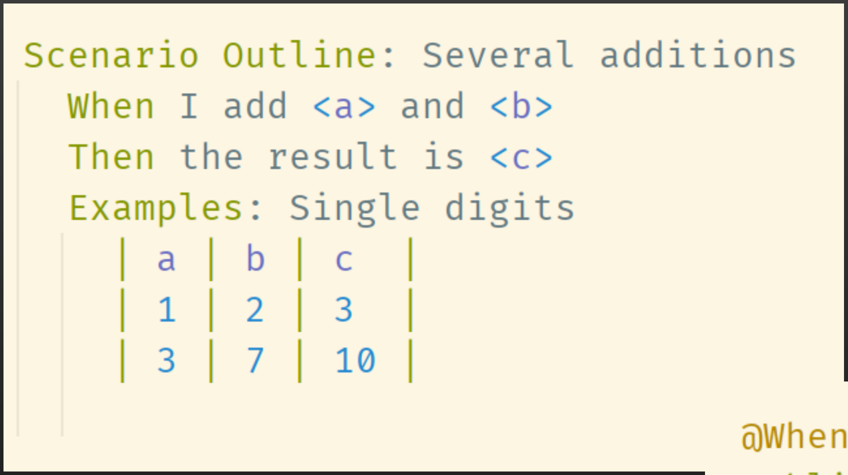
Along with the features, you give Cucumber a set of *step definitions*, which map the business-readable language of each step into code to carry out whatever action is being described by the step.

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.

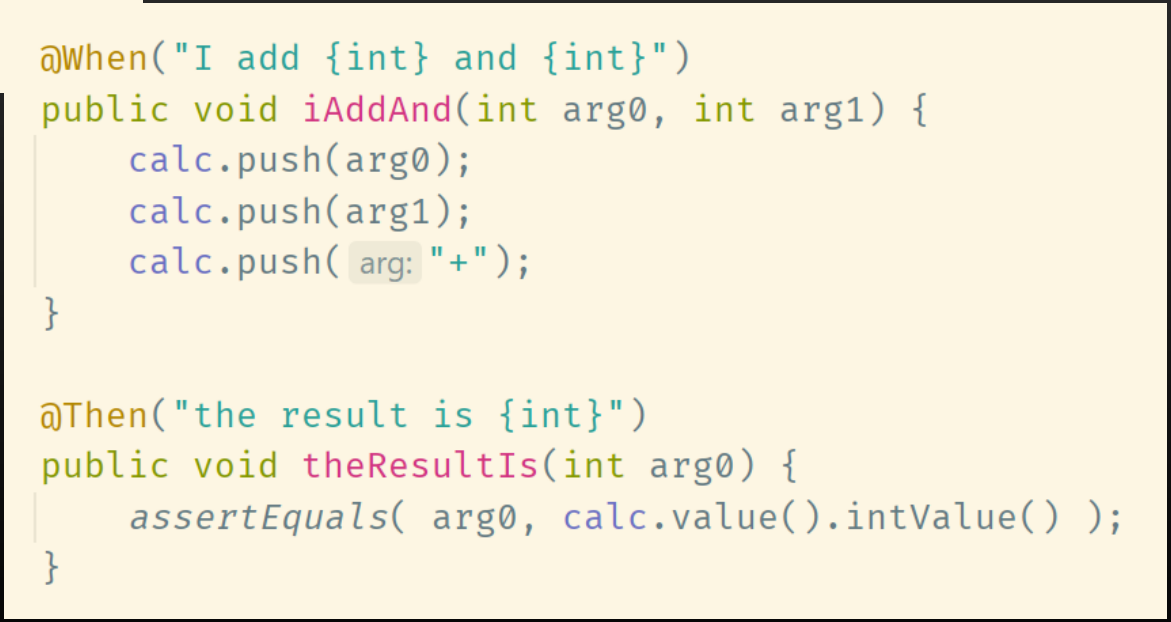
Sometimes that may involve using an *automation library*, like the browser automation library Selenium.



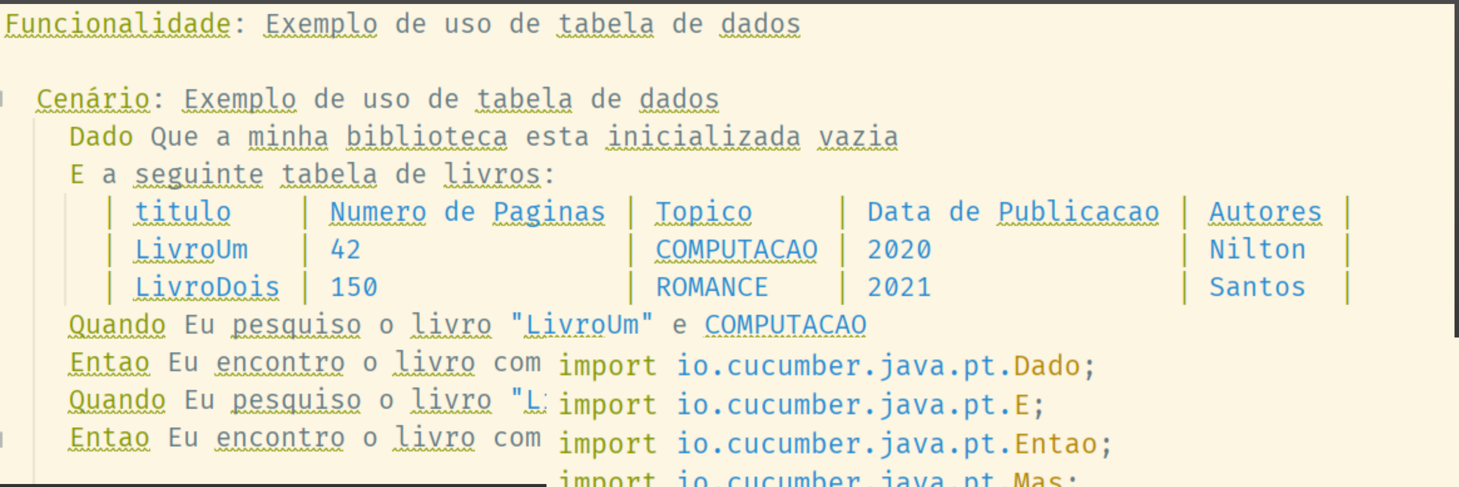
Helpful format: define an outline to be run against a few examples.



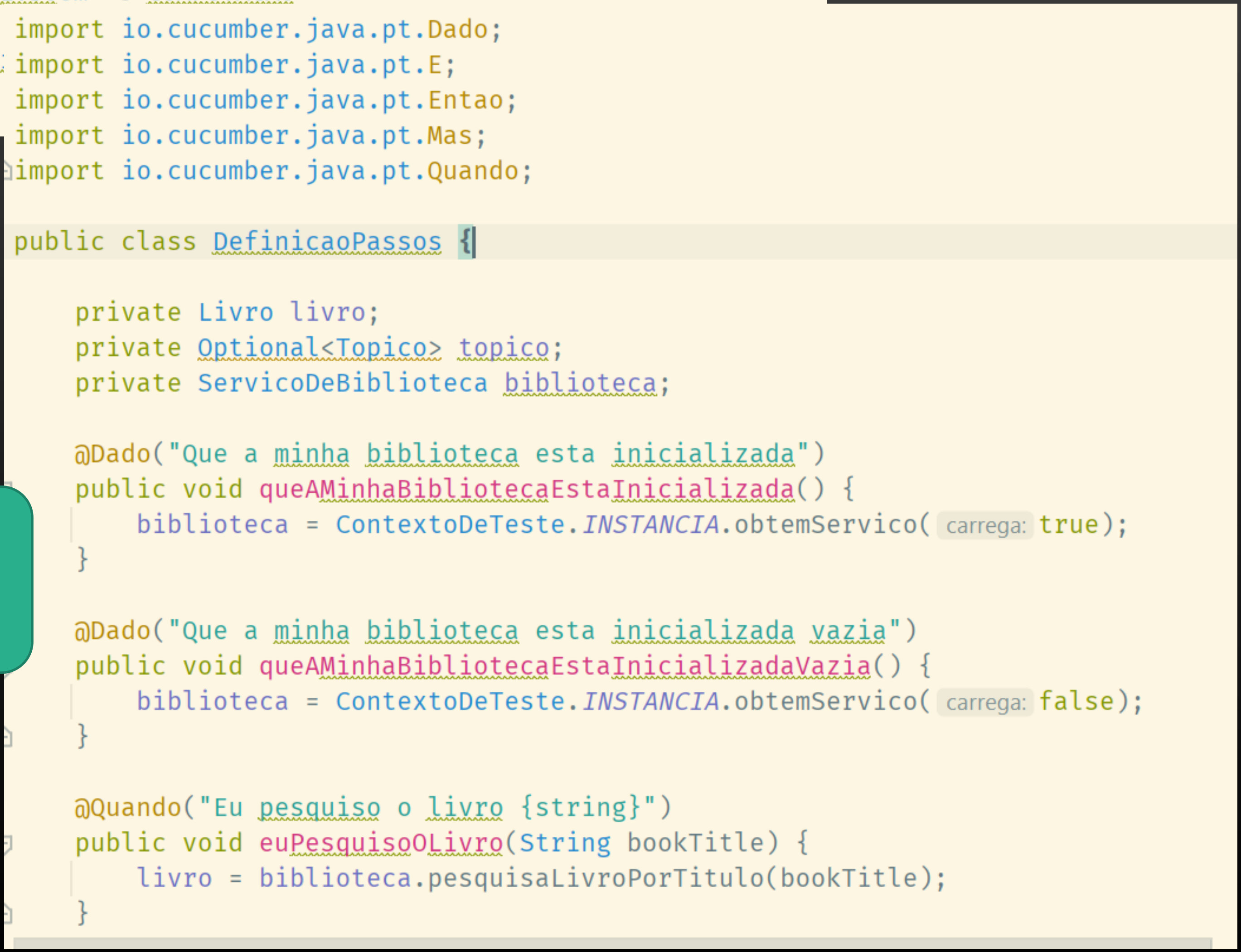
```
Scenario Outline: Several additions
  When I add <a> and <b>
  Then the result is <c>
  Examples: Single digits
    | a | b | c  |
    | 1 | 2 | 3  |
    | 3 | 7 | 10 |
```



```
@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() );
}
```



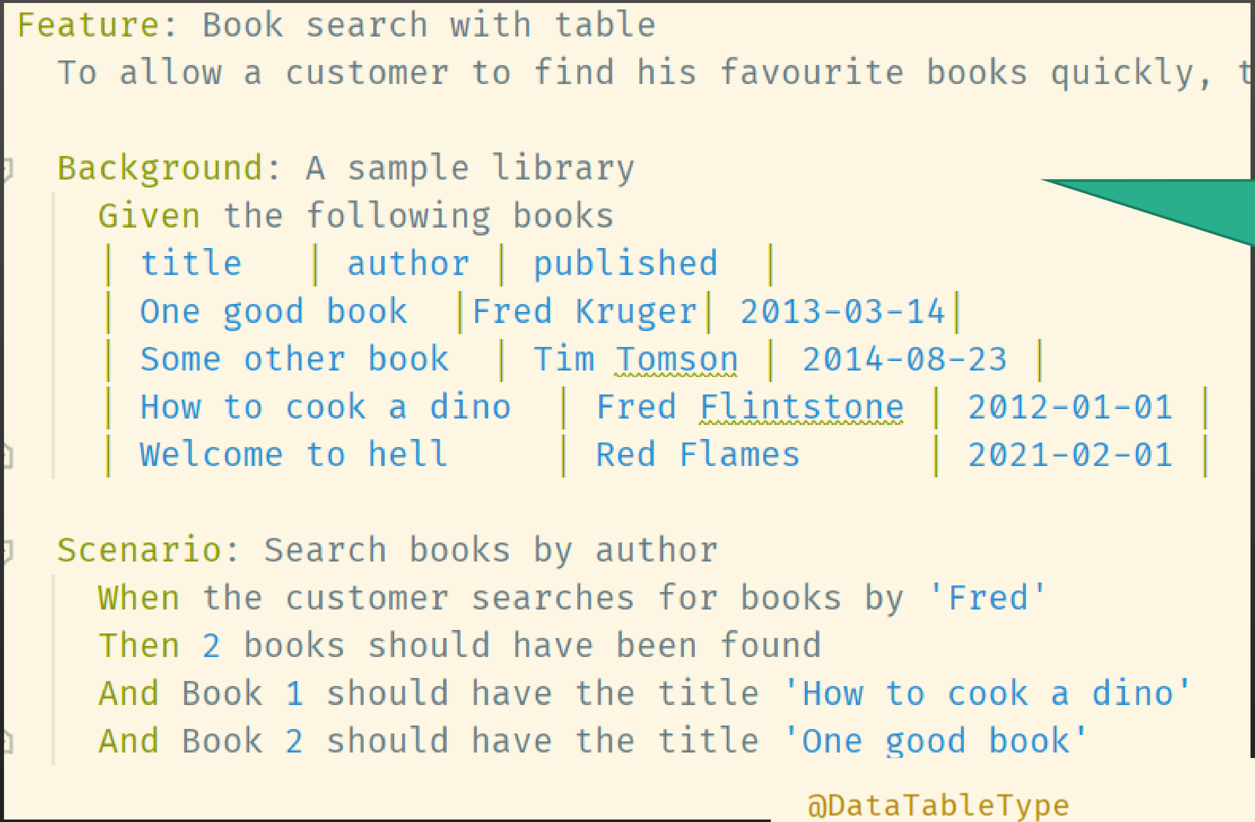
```
Funcionalidade: Exemplo de uso de tabela de dados

  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
    Quando Eu pesquiso o livro "L
    Entao Eu encontro o livro com
```



```
import io.cucumber.java.pt.Dado;
import io.cucumber.java.pt.E;
import io.cucumber.java.pt.Entao;
import io.cucumber.java.pt.Mas;
import io.cucumber.java.pt.Quando;

public class DefinicaoPassos {

    private Livro livro;
    private Optional<Topico> topico;
    private ServicoDeBiblioteca biblioteca;

    @Dado("Que a minha biblioteca esta inicializada")
    public void queAMinhaBibliotecaEstaInicializada() {
        biblioteca = ContextoDeTeste.INSTANCIA.obtemServico( carrega: true);
    }

    @Dado("Que a minha biblioteca esta inicializada vazia")
    public void queAMinhaBibliotecaEstaInicializadaVazia() {
        biblioteca = ContextoDeTeste.INSTANCIA.obtemServico( carrega: false);
    }

    @Quando("Eu pesquiso o livro {string}")
    public void euPesquisoOLivro(String bookTitle) {
        livro = biblioteca.pesquisaLivroPorTitulo(bookTitle);
    }
```

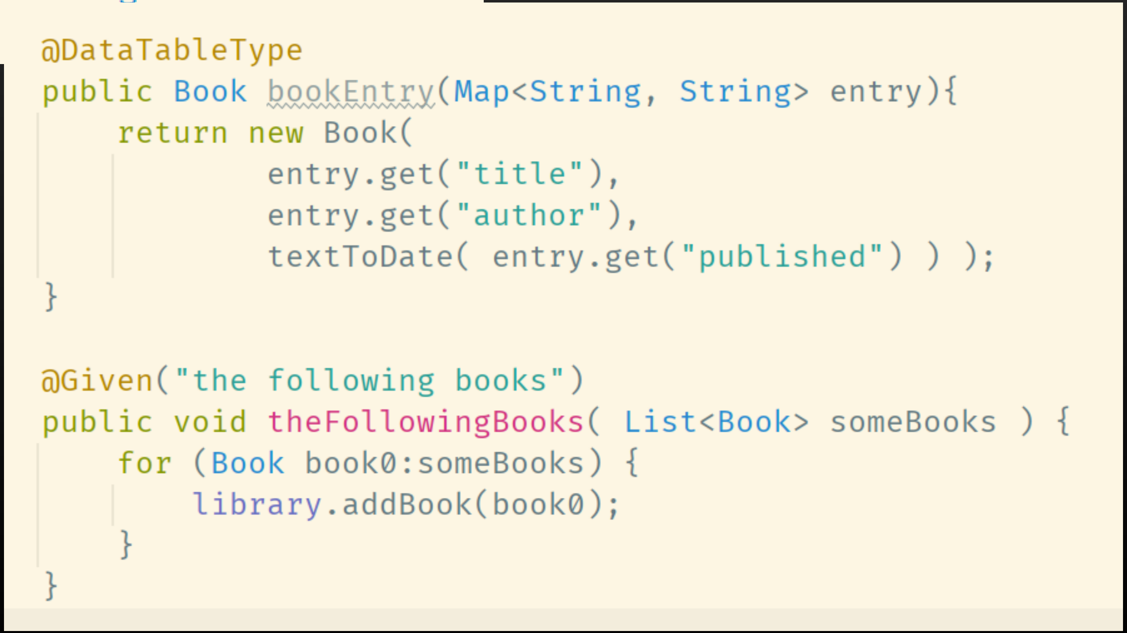
Keywords available in different languages (localization)

I Oliveira



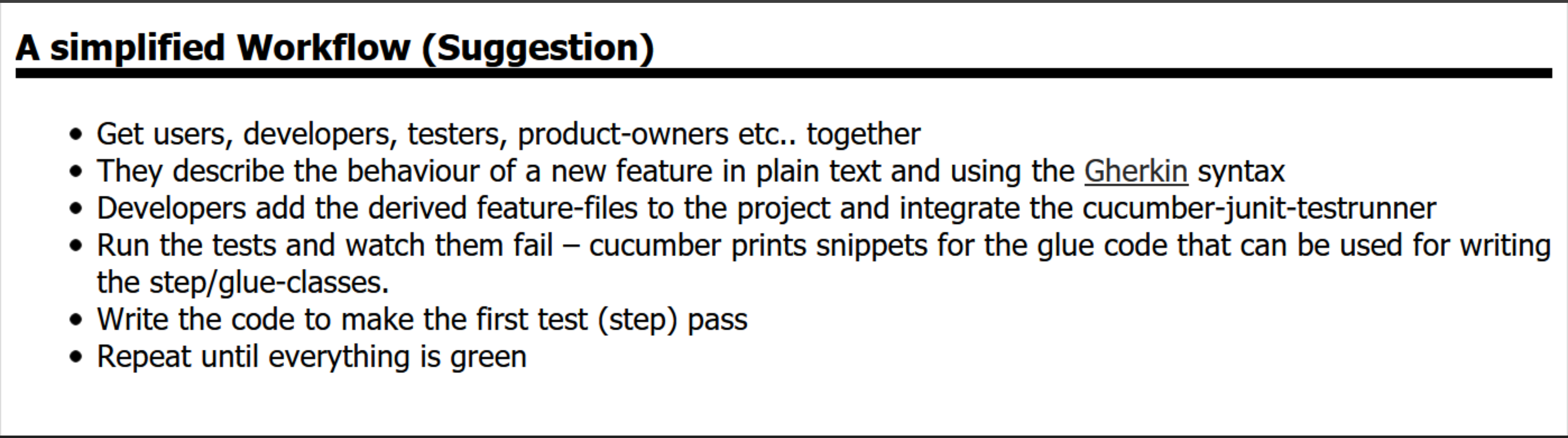
```
Feature: Book search with table
  To allow a customer to find his favourite books quickly, t

  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.



```
@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);
    }
}
```

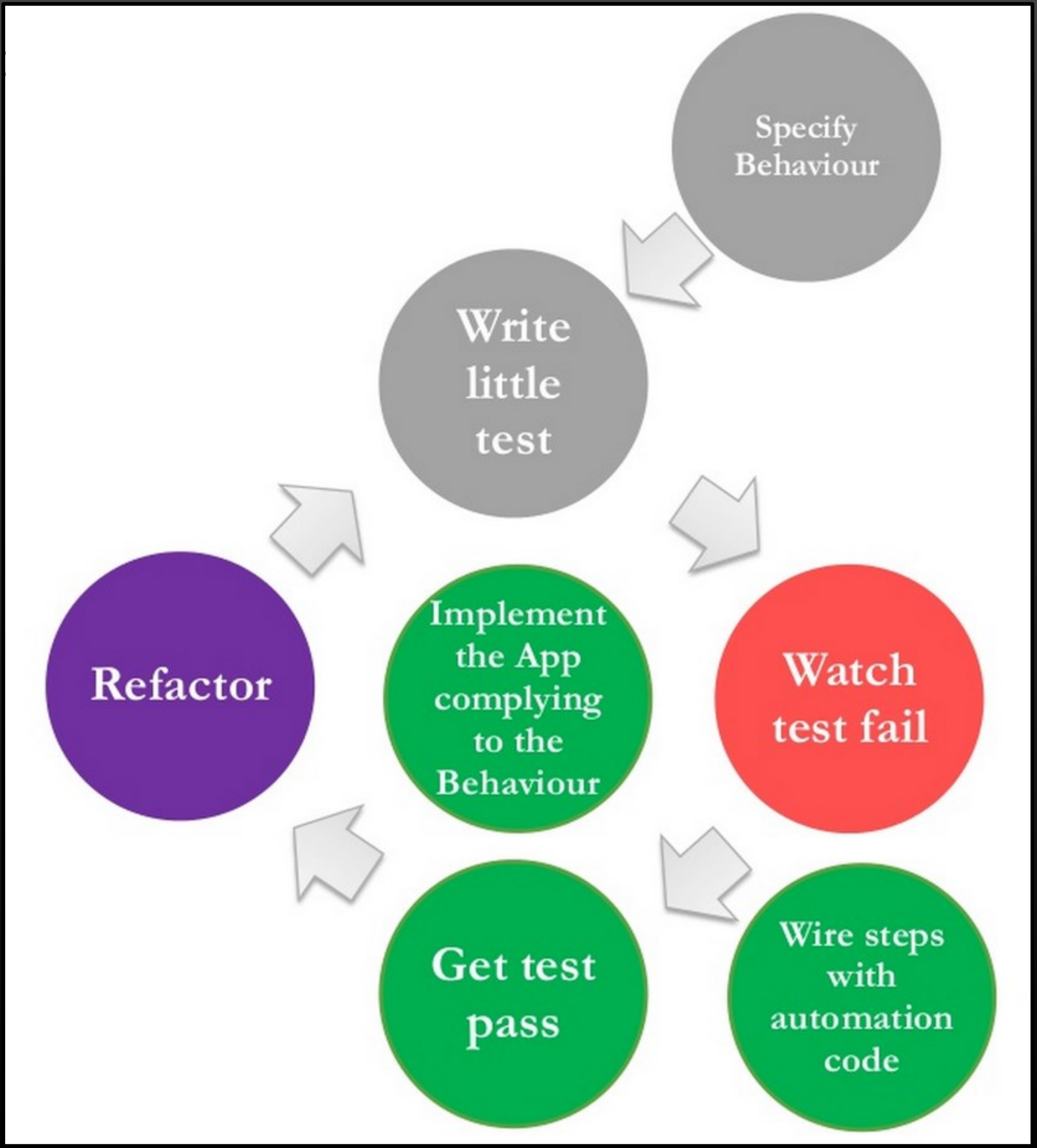
BDD as a work process



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: Behaviour-driven development


Specify Behaviour
Write little test
Watch test fail
Wire steps with automation code
Implement the App complying to the Behaviour
Get test pass
Refactor

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

True requirements.

TDD + Unit testing

Bottom-up

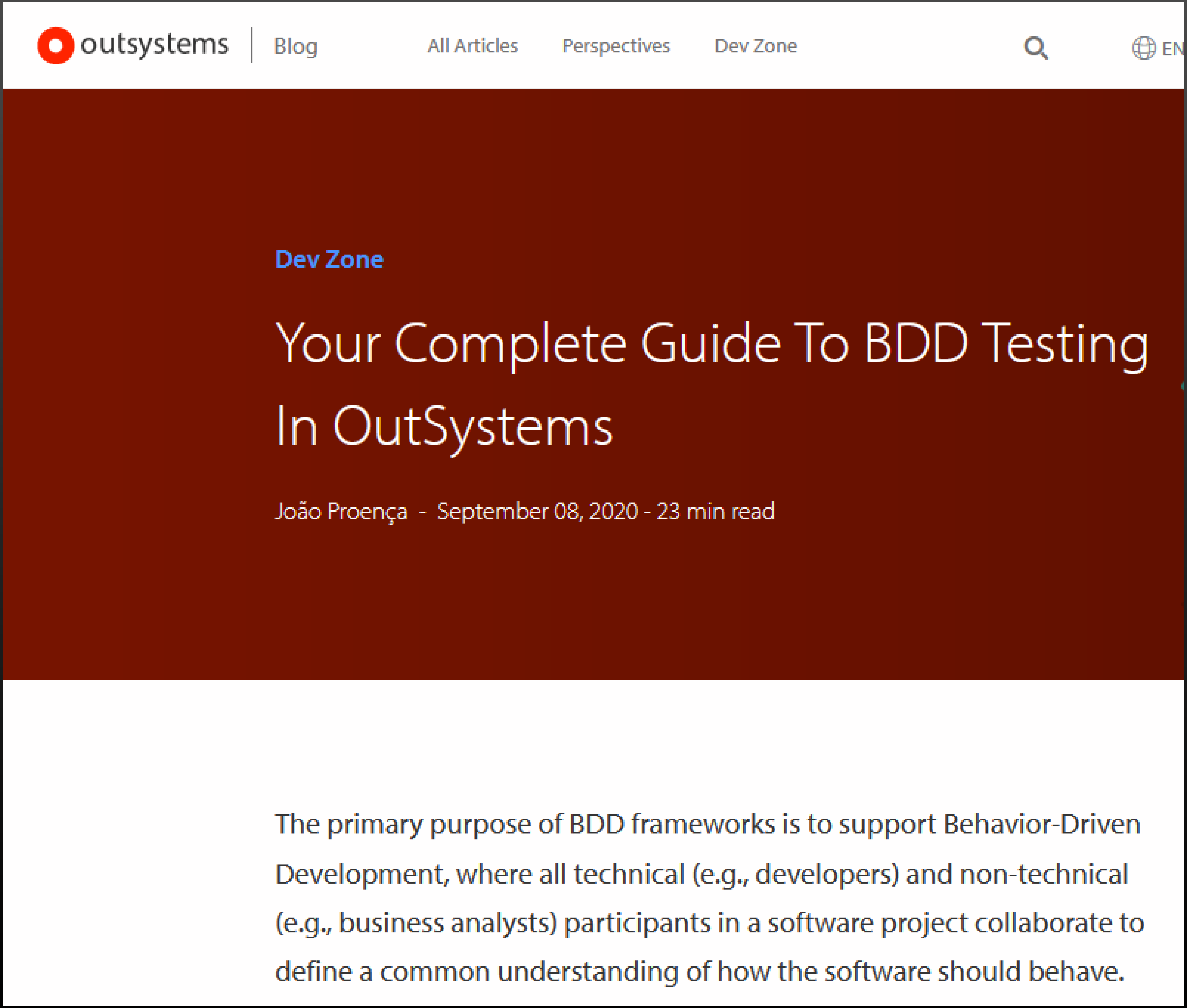
Programming language

☹

Developer-facing

Module contracts.

https://www.outsystems.com/blog/posts/bdd-testing/



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.

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**"