TDD & BDD

Test-Driven Developent & Behavior-Driven Development

Software Engineering 2

Don’t know the real source of this figure, anyway thanks !!

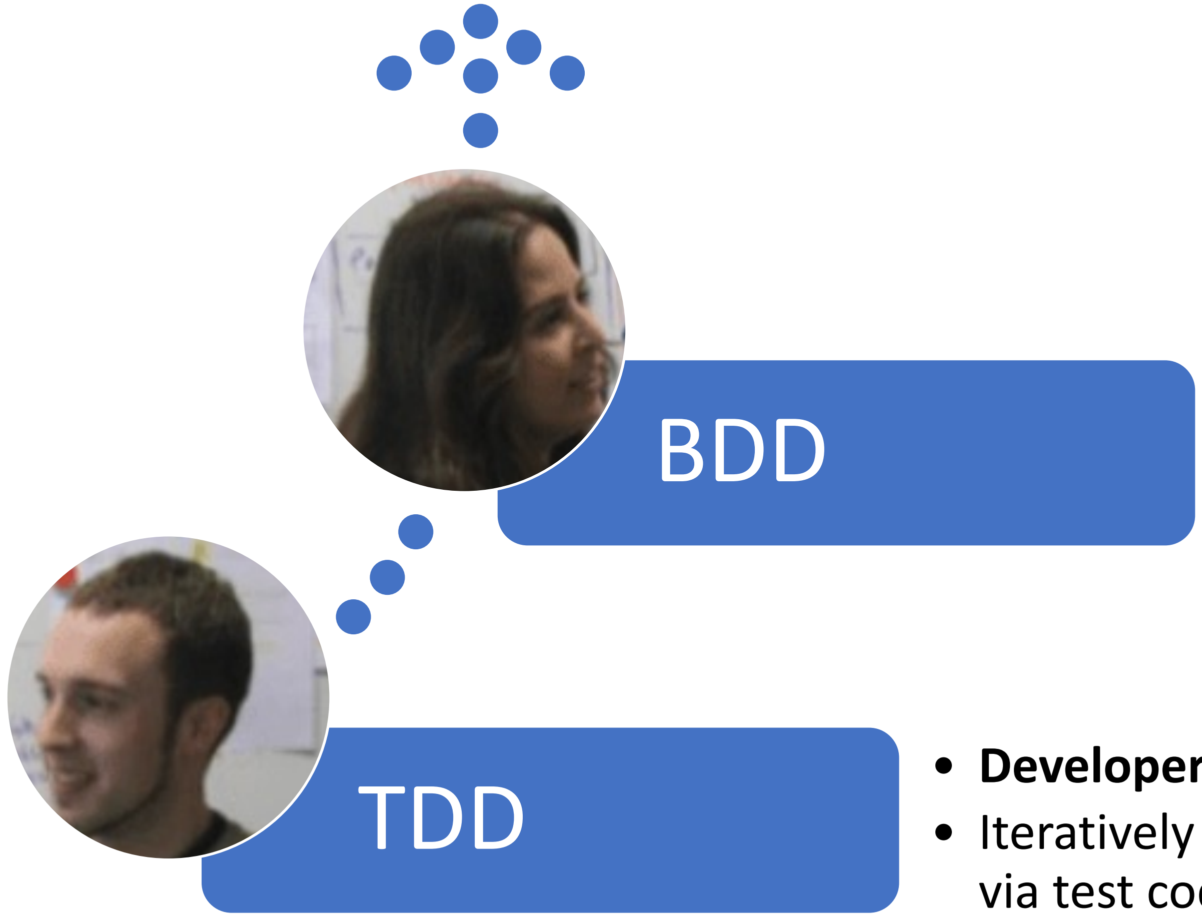
We don’t care about the users! Our ONLY objective is to trick the QA

Me waiting for QA to test my work
When QA didn't find the missing feature
374
001
ProgrammerHumor.io

test-memes, object-memes, feature-memes | ProgrammerHumor.io

https://programmerhumor.io/testing-memes/we-dont-care-about-the-users-our-only-objective-is-to-trick-the-qa/

Main differences between TDD and BDD



- **End user-centric**
- Iteratively improve codes scenarios and behaviors via collaboration and feedbacks!!

- **Developer-centric**
- Iteratively improve codes via test code failures and subsequence code modifications

Main steps in TDD

Developers can ensure that bad bugsand even worse from hot fixes will rarely happen

Create test

Write a test with
(1) Code input
(2) Code output
(3) Preconditions / Dependencies
(4) Assertions

Execute a specific test

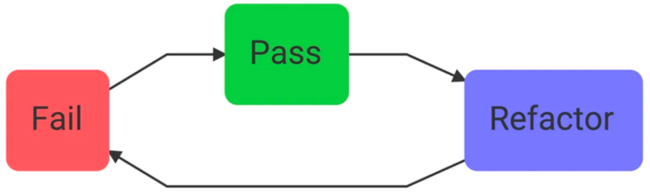
- To check if testing infrastructure is working as expected
- To develop guidelines for future code development activities

Implement the codes

- Developer starts to write just enough codes
- Intuitively promote code minimalism

Run all tests and refactor the codes

- Improve the design, readability, maintainability
- Form a full cycle of **fail-pass-refactor**
- QA then can later develop a comprehensive set of test suites to test codes in different scenarios/envs.



https://katalon.com/resources-center/blog/tdd-vs-bdd

Main steps in BDD

Write scenarios using Gherkin

- Stakeholders, developers, and testers sit together to gather the requirements with expected outcomes.
- Each scenario describes a specific aspect of the system.

Apply TDD

- TDD can be standalone or complement BDD

https://katalon.com/resources-center/blog/tdd-vs-bdd

How BDD and TDD relate

Feature file in Gherkin syntax

```
Feature: Place order

Scenario 1: Approve order
Given a valid customer_id
Given a valid credit card
When I place an order
Then the order should be
APPROVED

Scenario 2: Reject order
Scenario 3:……
```

mapping

Step definition file in specific programming language

```
@Given(”a valid customer” and “a valid credit card”)
public String placeOrder(String c_name, String
card_ID, String card_expire,  String card_CCV2){
   boolean r1 = verifyCustomer(c_name);
   boolean r2 = verifyCreditcard(card_ID,
card_expire, card_CCV2);
    if r1 & r2 {return ”APPROVED”};
    else {return “REJECTED”};
}
```

Example of TDD related to previous slide

```
public class VerificationsTest{
   public void shouldReturnTrue(){
       String card_id_1 = ‘551488219902’; Boolean exp_flag_1 = False; String ccv2_1 = ‘110’;
       String card_id_2 = ‘551488219902’; Boolean exp_flag_2 = True; String ccv2_2 = ‘110’;
       String card_id_3 = ‘551488219902’; Boolean exp_flag_3 = False; String ccv2_2 = ‘008’;
       ….
       assertTrue(verifyCreditcard(card_id_1, exp_flag1, ccv2_1);
       assertTrue(verifyCreditcard(card_id_1, exp_flag1, ccv2_1);
}
```

```
public class Verifications{
    public static void main String[] args{
        verifyCredictcard(cnum, is_expire, ccv2);
     }
     static boolean verifyCredictcard(String cnum, Boolean is_expire, String ccv2){
            // Need to write code here to remove code failures
    }
}
```

Differences between TDD and BDD (in various aspects)

หัวข้อในการพิจารณา	Test-Driven Development (TDD)	Behaviour-Driven Development (BDD)
Focus and perspective	Implementation of code functionality through a test-first approach	Collaboration, shared understanding, and validation of system behavior from a user's perspective
Terminology and readability	Test cases written in programming-centric terminology	Scenarios written in a natural language format, understandable by both technical and non-technical people
Collaboration	Testers & Developers	Testers & Developers & stakeholders to define and validate system behaviours

https://katalon.com/resources-center/blog/tdd-vs-bdd

Differences between TDD and BDD (in various aspects)

หัวข้อในการพิจารณา	Test-Driven Development (TDD)	Behaviour-Driven Development (BDD)
Level of abstraction	Low-level unit test	High level tests that simulate user interactions or end-to-end scenarios
Purpose	Ensure code correctness through automated tests	Promote shared understanding, communication, and validation of system behaviour
Iterative refinement and feedbacks	Iteratively refines codes through code failures and code modifications	Iteratively refines behaviors through feedbacks and collaborations

https://katalon.com/resources-center/blog/tdd-vs-bdd

Gherkin and Cucumber

Software Engineering 2

What is Gherkin?

- **Gherkin** is a business readable language.
- Helps you to describe business behavior without going into details of implementation.
- A domain specific language for defining tests in Cucumber.
- It uses plain language to describe use cases and allows users to remove logic details from behavior tests.

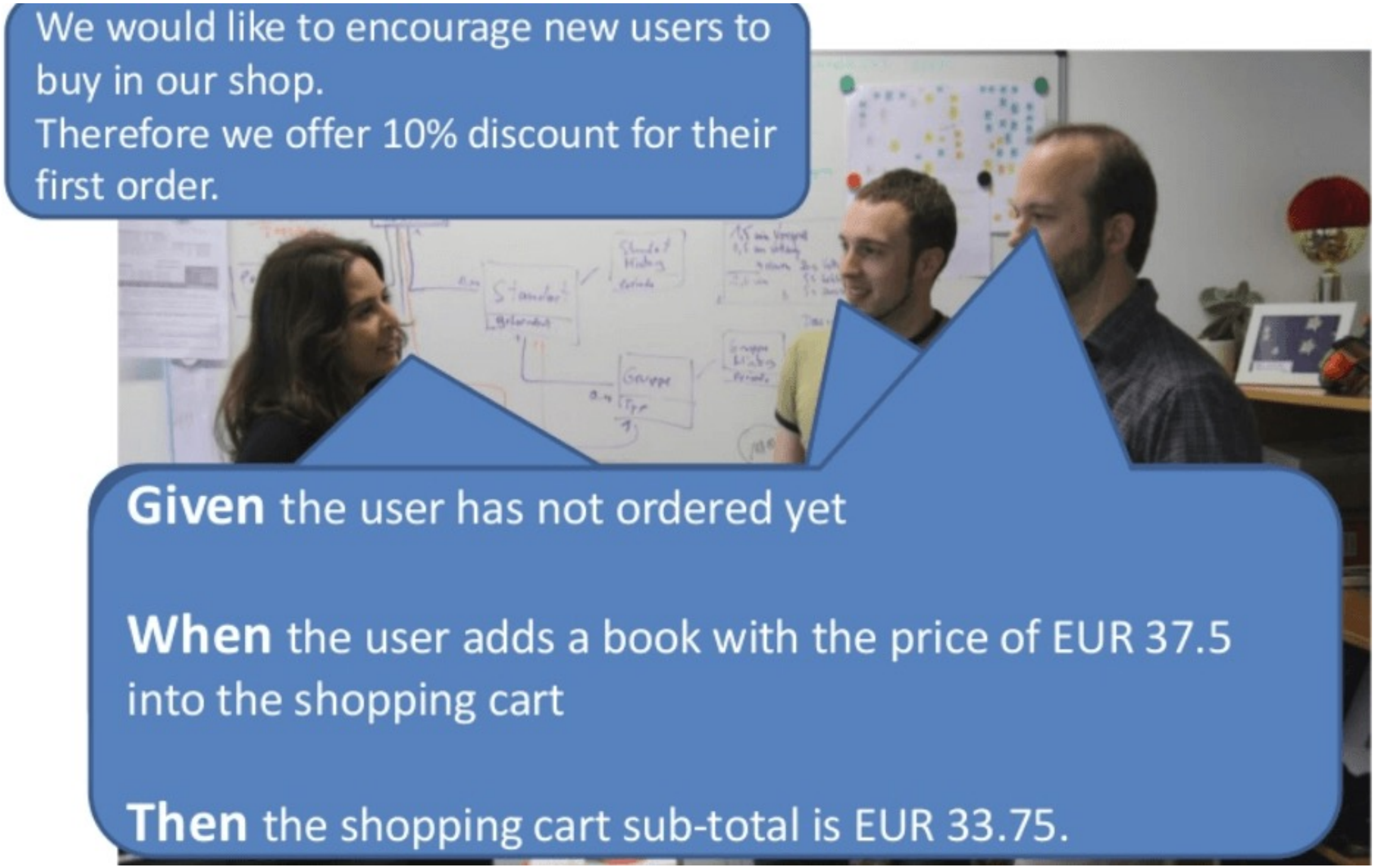
Why Gherkin? -- Before

We would like to encourage new users to buy in our shop.
Therefore we offer 10% discount for their first order.

```
public void CalculateDiscount(Order order)
{
  if (order.Customer.IsNew)
    order.FinalAmount =
      Math.Round(order.Total * 9/10);
}
```

Register as “bart_bookworm”
Go to “/catalog/search”
Enter “ISBN-0955683610”
Click “Search”
Click “Add to Cart”
Click “View Cart”
Verify “Subtotal” is “$33.75”

Why Gherkin? -- After


We would like to encourage new users to buy in our shop.
Therefore we offer 10% discount for their first order.

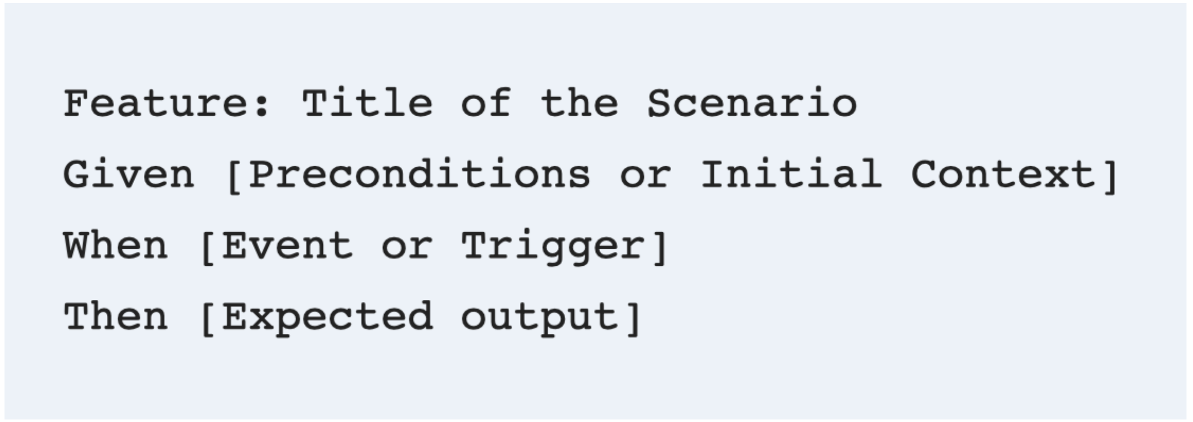
Given the user has not ordered yet

When the user adds a book with the price of EUR 37.5 into the shopping cart

Then the shopping cart sub-total is EUR 33.75.

Gherkin Syntax

- A Gherkin document has an extension .feature and simply just a test file with a fancy extension.
- Cucumber reads Gherkin document and executes a test to validate that the software behaves as per the Gherkin syntax.



```
Feature: Title of the Scenario
Given [Preconditions or Initial Context]
When [Event or Trigger]
Then [Expected output]
```

Keywords

Each line that isn't a blank line has to start with a Gherkin keyword, followed by any text you like.

The only exceptions are the free-form descriptions placed underneath Example/Scenario, Background, Scenario Outline and Rule lines.

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`)

The secondary keywords are:

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

Feature

- The purpose of the Feature keyword is to provide a high-level description of a software feature, and to group related scenarios.
- The first primary keyword in a Gherkin document must always be Feature, followed by a : and a short text that describes the feature.
- You can add free-form text underneath Feature to add more description.
- The free format description for Feature ends when you start a line with the keyword Background, Rule, Example or Scenario Outline (or their alias keywords).
- You can only have a single Feature in a .feature file.

```
Feature: Guess the word

  The word guess game is a turn-based game for two players.
  The Maker makes a word for the Breaker to guess. The game
  is over when the Breaker guesses the Maker's word.

  Example: Maker starts a game
```

Example or Scenario

- The keyword Scenario is a synonym of the keyword Example.
- It consists of a list of steps.
- You can have as many steps as you like, but 3-5 steps per example are recommended.
- Examples follow this same pattern:
  - Describe an initial context (Given steps)
  - Describe an event (When steps)
  - Describe an expected outcome (Then steps)

Steps

- Each step starts with Given, When, Then, And, or But.
- Cucumber executes each step in a scenario one at a time, in the sequence you've written them in.
- You cannot have a Given, When, Then, And, or But step with the same text as another step.
- Cucumber considers the following steps duplicates:

```
Given there is money in my account
Then there is money in my account
```

- This might seem like a limitation, but it forces you to come up with a less ambiguous, more clear domain language:

```
Given my account has a balance of £430
Then my account should have a balance of £430
```

Given

- Given steps are used to describe the initial context of the system - the scene of the scenario. It is typically something that happened in the past.
- The purpose of Given steps is to put the system in a known state before the user (or external system) starts interacting with the system (in the When steps).
- Avoid talking about user interaction in Given's. If you were creating use cases, Given's would be your preconditions.
- It's okay to have several Given steps (use And or But for number 2 and upwards to make it more readable).
- Examples:
  - Mickey and Minnie have started a game
  - I am logged in
  - Joe has a balance of £42

When

- When steps are used to describe an event, or an action.
- This can be a person interacting with the system, or it can be an event triggered by another system.
- Examples:
    - Guess a word
    - Invite a friend
    - Withdraw money

Then

- Then steps are used to describe an expected outcome, or result.
- The step definition of a Then step should use an assertion to compare the actual outcome (what the system actually does) to the expected outcome (what the step says the system is supposed to do).
- An outcome should be on an observable output
  - That is, something that comes out of the system (report, user interface, message)
  - Not a behaviour deeply buried inside the system (like a record in a database)
- Examples:
  - See that the guessed word was wrong
  - Receive an invitation
  - Card should be swallowed

And, But

- If you have successive Given’s or Then’s, you could write:

```
Example: Multiple Givens
  Given one thing
  Given another thing
  Given yet another thing
  When I open my eyes
  Then I should see something
  Then I shouldn't see something else
```

- Or, you could make the example more fluidly structured by replacing the successive Given’s or Then’s with And’s and But’s:

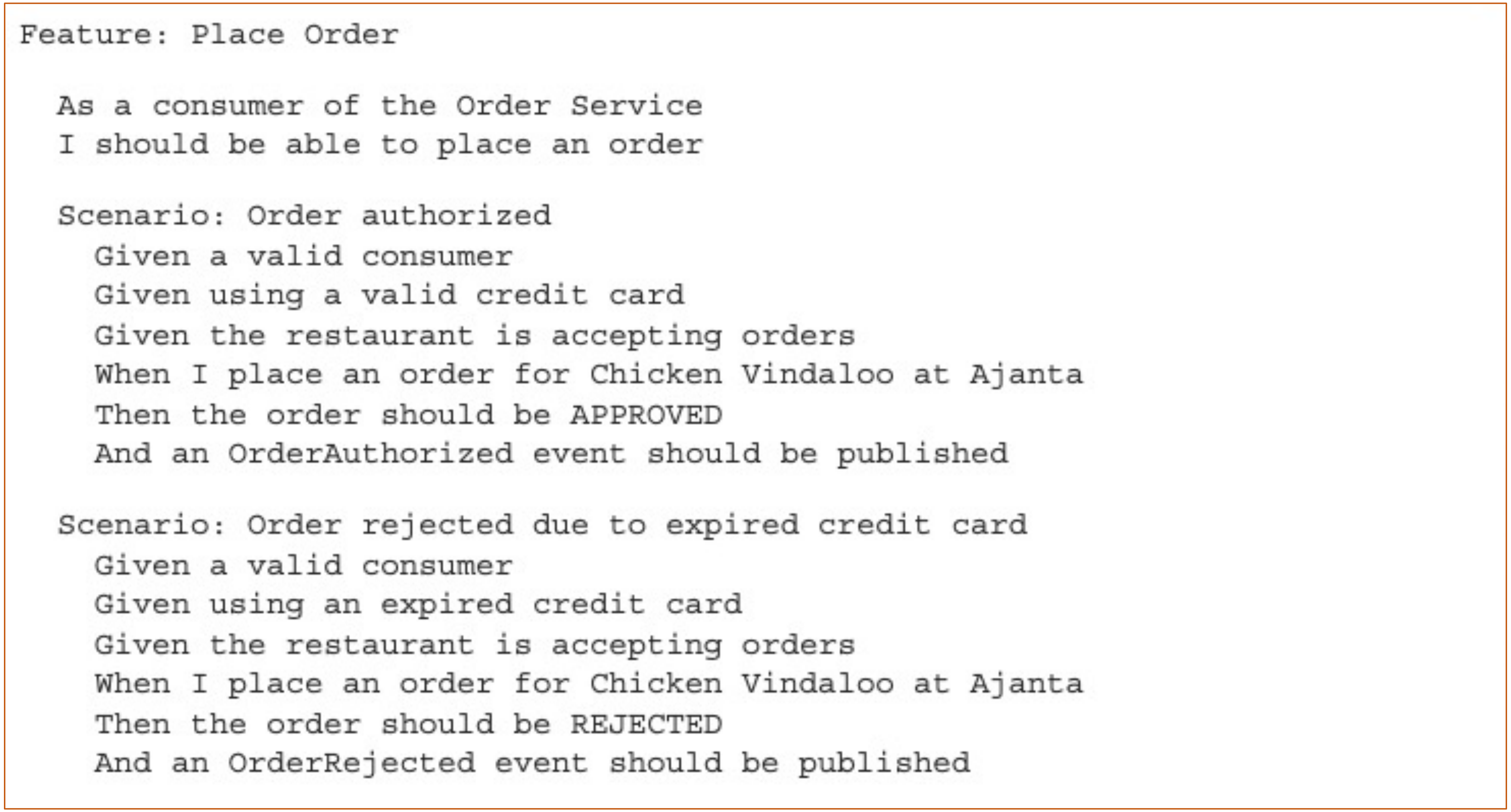
```
Example: Multiple Givens
  Given one thing
  And another thing
  And yet another thing
  When I open my eyes
  Then I should see something
  But I shouldn't see something else
```

Background

- Occasionally you'll find yourself repeating the same Given steps in all of the scenarios in a Feature.
- A Background allows you to add some context to the scenarios that follow it. It can contain one or more Given steps, which are run before each scenario.
- A Background is placed before the first Scenario/Example, at the same level of indentation.

```
Feature: Multiple site support
  Only blog owners can post to a blog, except administrators,
  who can post to all blogs.

  Background:
    Given a global administrator named "Greg"
    And a blog named "Greg's anti-tax rants"
    And a customer named "Dr. Bill"
    And a blog named "Expensive Therapy" owned by "Dr. Bill"

  Scenario: Dr. Bill posts to his own blog
    Given I am logged in as Dr. Bill
    When I try to post to "Expensive Therapy"
    Then I should see "Your article was published."

  Scenario: Dr. Bill tries to post to somebody else's blog, and fails
    Given I am logged in as Dr. Bill
    When I try to post to "Greg's anti-tax rants"
    Then I should see "Hey! That's not your blog!"

  Scenario: Greg posts to a client's blog
    Given I am logged in as Greg
    When I try to post to "Expensive Therapy"
    Then I should see "Your article was published."
```

Gherkin Example



```
Feature: Place Order

  As a consumer of the Order Service
  I should be able to place an order

  Scenario: Order authorized
    Given a valid consumer
    Given using a valid credit card
    Given the restaurant is accepting orders
    When I place an order for Chicken Vindaloo at Ajanta
    Then the order should be APPROVED
    And an OrderAuthorized event should be published

  Scenario: Order rejected due to expired credit card
    Given a valid consumer
    Given using an expired credit card
    Given the restaurant is accepting orders
    When I place an order for Chicken Vindaloo at Ajanta
    Then the order should be REJECTED
    And an OrderRejected event should be published
```

Advantages of Gherkin

Gherkin is simple enough for non-programmers to understand

Programmers can use it as a very solid base to start their tests

It makes User Stories easier to digest

Gherkin Testing targets the business requirements

You don’t need to be expert to understand the small Gherkin command set

Gherkin Test cases link acceptance tests directly to automated tests

Style of writing tests cases are easier to reuse code in other tests

Disadvantages of Gherkin

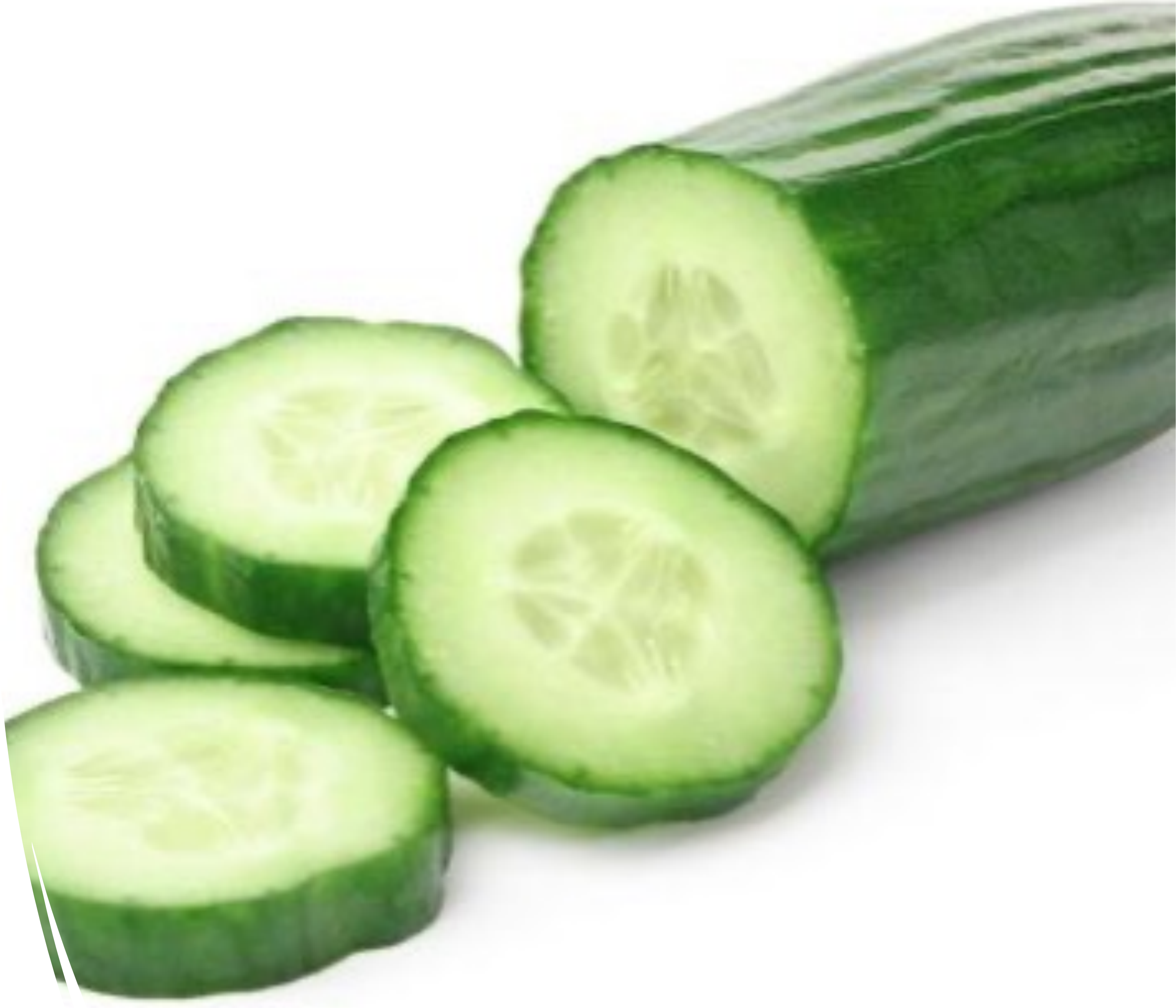
It requires a high level of business engagement and collaborations

May not work well in all scenarios

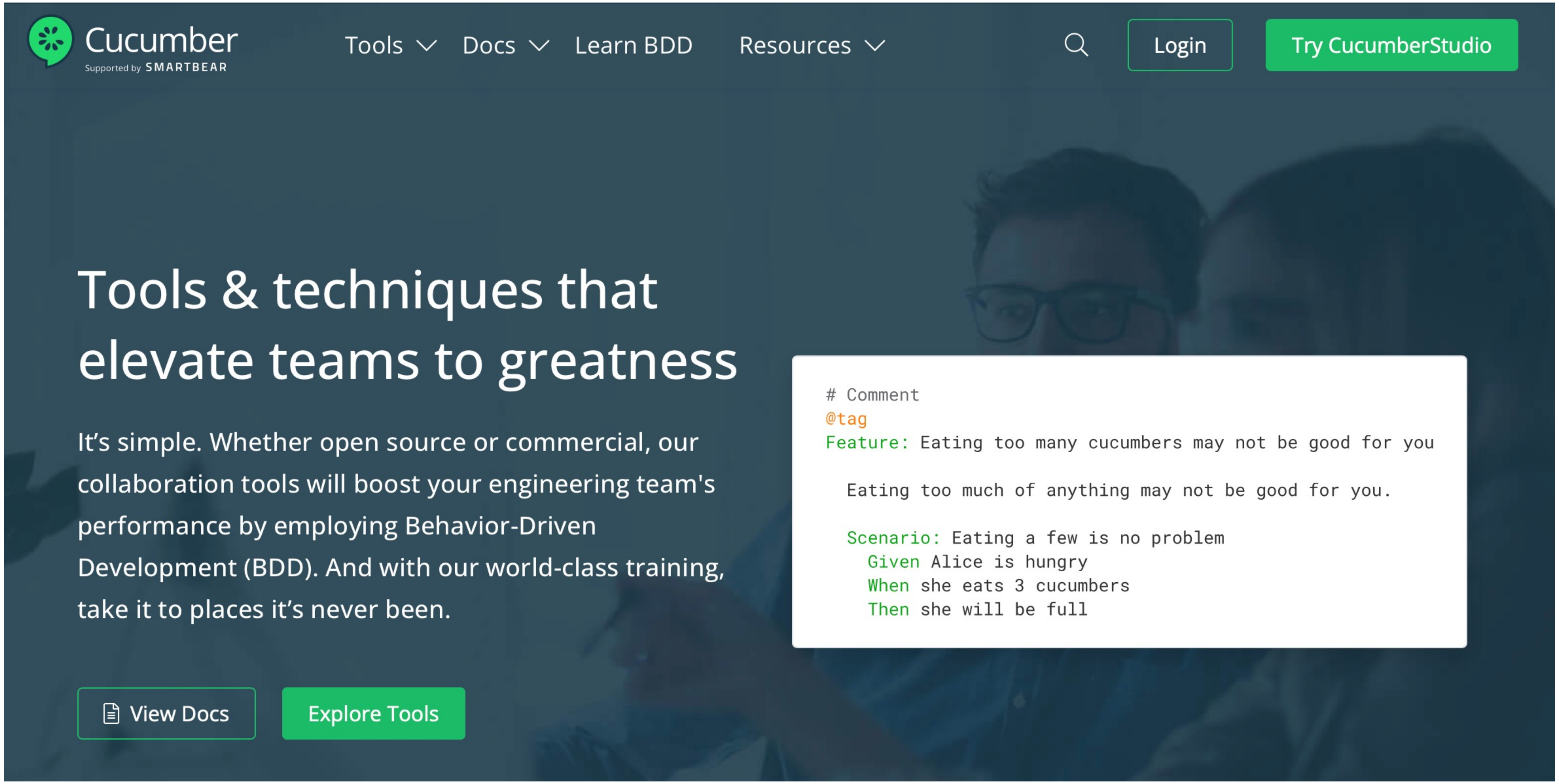
Poorly written tests can easily increase test-maintenance cost

What is Cucumber?

- Cucumber is a testing tool that supports Behavior Driven Development (BDD).
- It offers a way to write tests that anybody can understand, regardless of their technical knowledge.



Cucumber - https://cucumber.io

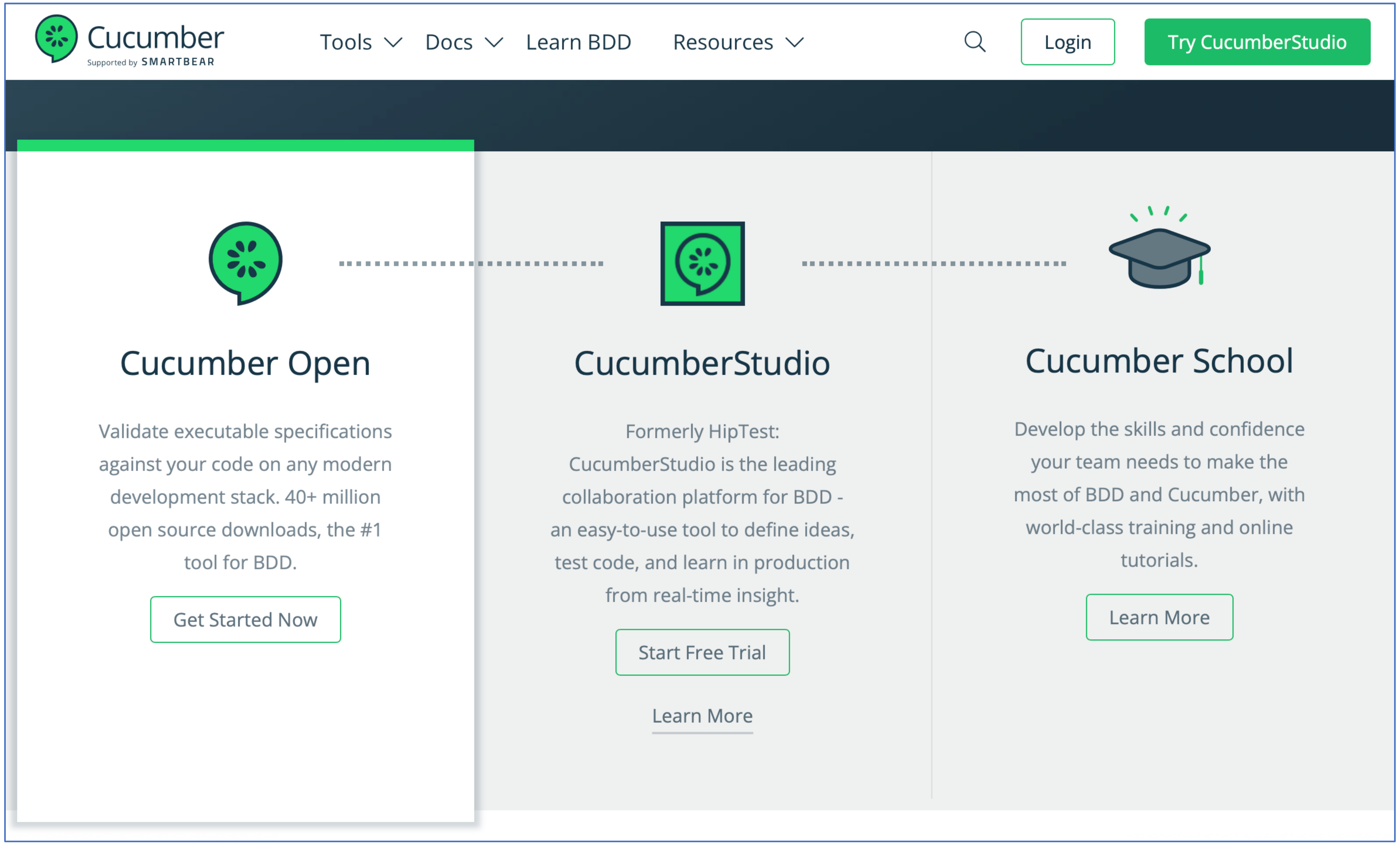


Tools & techniques that elevate teams to greatness

It's simple. Whether open source or commercial, our collaboration tools will boost your engineering team's performance by employing Behavior-Driven Development (BDD). And with our world-class training, take it to places it's never been.

```
# Comment
@tag
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
```

Cucumber Open - https://cucumber.io (cont.)



Cucumber Open

Validate executable specifications against your code on any modern development stack. 40+ million open source downloads, the #1 tool for BDD.

Get Started Now

CucumberStudio

Formerly HipTest: CucumberStudio is the leading collaboration platform for BDD - an easy-to-use tool to define ideas, test code, and learn in production from real-time insight.

Start Free Trial

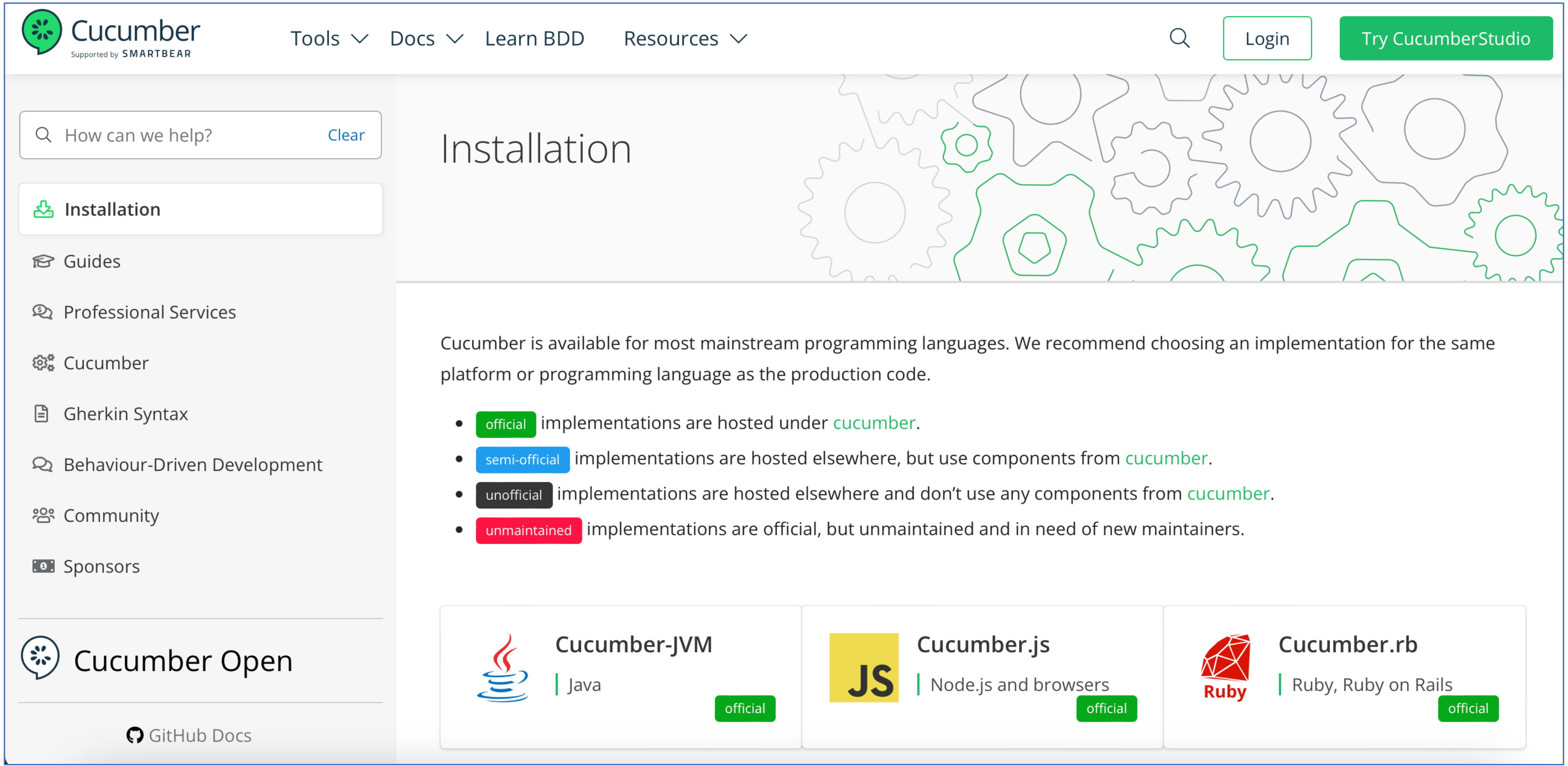
Learn More

Cucumber School

Develop the skills and confidence your team needs to make the most of BDD and Cucumber, with world-class training and online tutorials.

Learn More

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



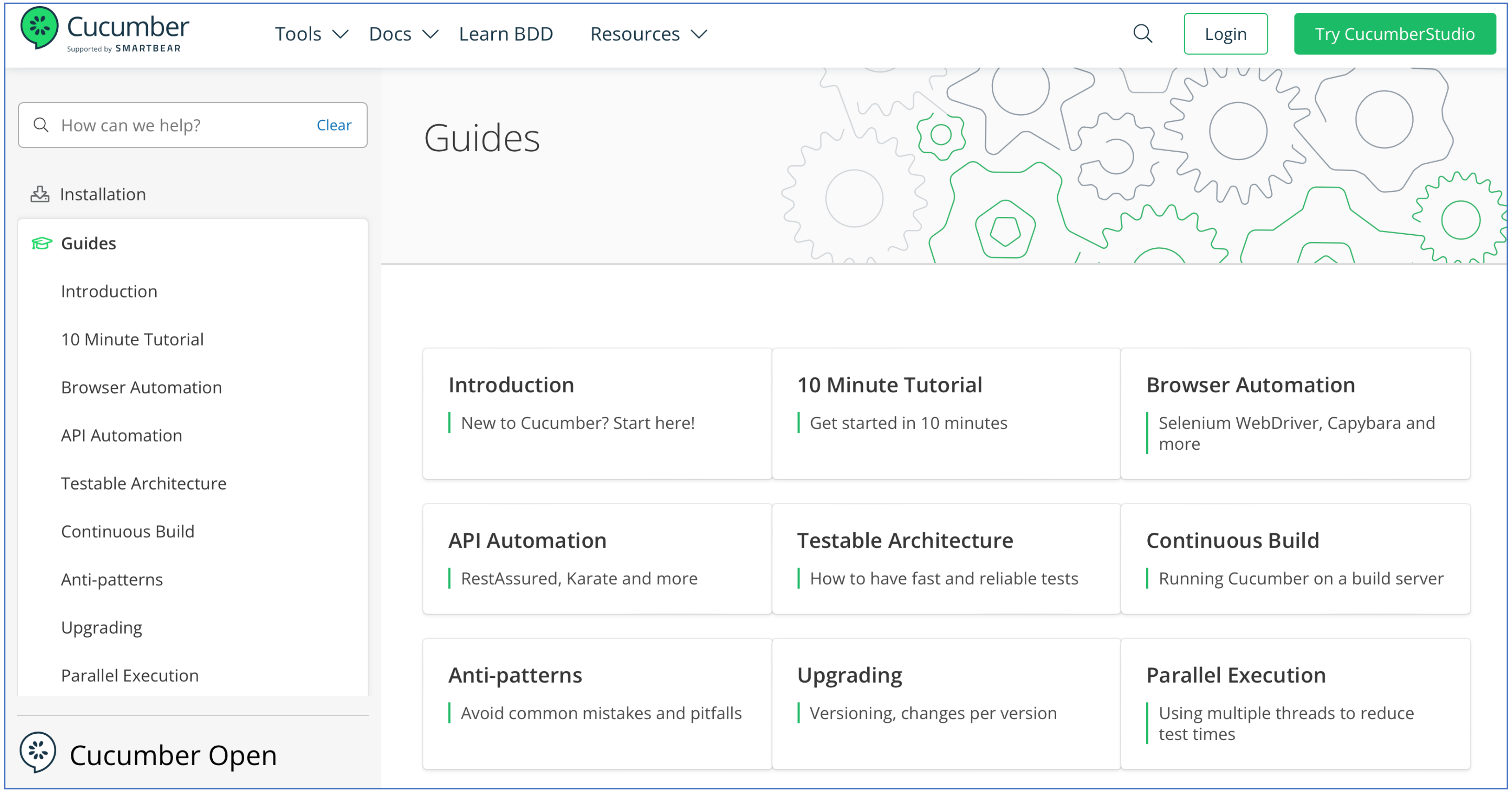
Installation

Cucumber is available for most mainstream programming languages. We recommend choosing an implementation for the same platform or programming language as the production code.

- official implementations are hosted under cucumber.
- semi-official implementations are hosted elsewhere, but use components from cucumber.
- unofficial implementations are hosted elsewhere and don't use any components from cucumber.
- unmaintained implementations are official, but unmaintained and in need of new maintainers.

Cucumber-JVM Java official	Cucumber.js Node.js and browsers official	Cucumber.rb Ruby, Ruby on Rails official

Guides - https://cucumber.io/docs/guides/



Guides

Introduction
New to Cucumber? Start here!

10 Minute Tutorial
Get started in 10 minutes

Browser Automation
Selenium WebDriver, Capybara and more

API Automation
RestAssured, Karate and more

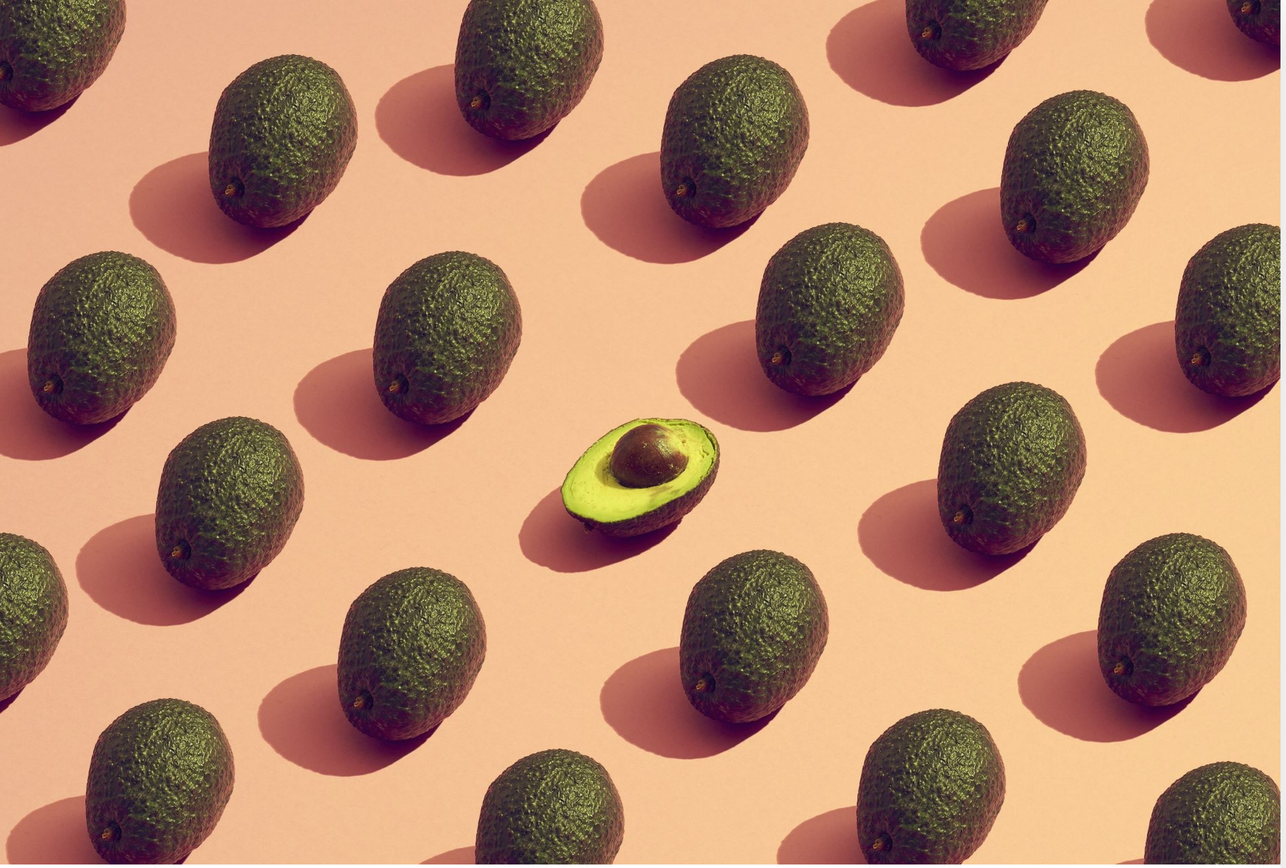
Testable Architecture
How to have fast and reliable tests

Continuous Build
Running Cucumber on a build server

Anti-patterns
Avoid common mistakes and pitfalls

Upgrading
Versioning, changes per version

Parallel Execution
Using multiple threads to reduce test times



References

- https://cucumber.io
- https://cucumber.io/docs/gherkin/reference/
- https://www.guru99.com/gherkin-test-cucumber.html
- https://www.guru99.com/introduction-to-cucumber.html