



# Gherkin & BDD

# Gherkin Refresher

- What
- Why
- Syntax

How it fits  
into BDD

Practical  
Example



# Gherkin Refresher

# WHAT IS GHERKIN?

Officially ...

Gherkin is the language that Cucumber understands. It is a Business Readable, Domain Specific Language, that lets you describe software's behaviour without detailing how that behaviour is implemented.

# WHAT IS GHERKIN?

Unofficially it's ...

Gherkin is a business readable language used to express the system's behaviour. The language can be understood by an automation tool called Cucumber

It consists of 10 key words

# WHAT IS GHERKIN?

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

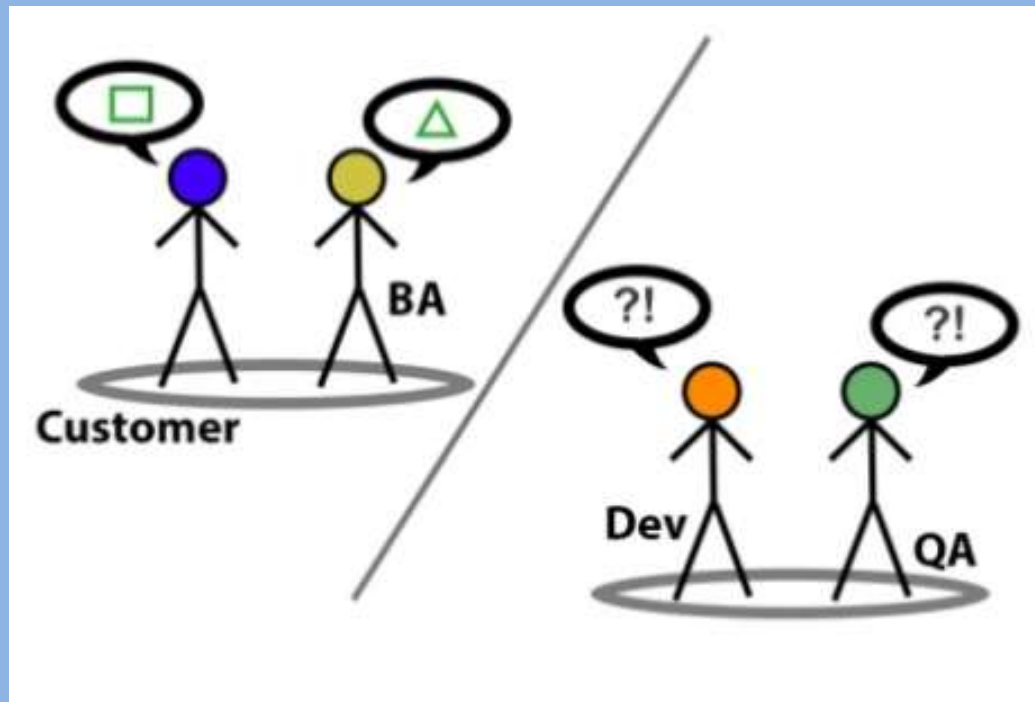
Acceptance test written in Gherkin

# WHAT IS GHERKIN?

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

# WHY USE GHERKIN?

- 1) Gherkin allows us to document acceptance tests in a language devs, QA, BAs & the business can understand. It allows the 3 Amigos to collaborate and understand tests in a common language.





# WHY USE GHERKIN?

2) Gherkin links our acceptance tests (GIVEN/WHEN/THEN) directly to automated tests. This means if we change an acceptance test – the developers underlying tests should fail. We can be confident the system matches the specification.

*As part of BDD we want to write many automated tests to improve our confidence in the product. We want these tests to be understandable + valuable.*

# BASIC SYNTAX

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

**Given** = system in known state

**When** = key action

**Then** = observable outcome

**Scenario** = Title of the acceptance test (e.g. “the one where”)

# BASIC SYNTAX

```
Scenario: Dr Bill posts to his own blog
  Given I am logged in as Dr Bill
  And   I have a premium account
  When  I try to post to my blog
  Then  I should see "Your article was published"
```

**And** = used instead of multiple Givens / for complicated scenarios

# BASIC SYNTAX

```
Scenario: Dr Bill posts to his own blog
  Given I am logged in as Dr Bill
  And   my account is active
  When  I try to post to my blog
  Then  I should see "Your article was published"
  But   I should not be charged
```

**But** = used in association with Then

# BASIC SYNTAX

Scenario: Dr Bill posts to his own blog

Given I am logged in as Dr Bill

And my account is active

When I try to post to my blog

Then I should see "Your article was published"

But I should not be charged

Steps

# ADVANCED SYNTAX

## Feature: Post a blog

As a customer

I want to post informative blogs

So that that I can share what I'm doing and raise my profile

## Acceptance Criteria

- \* Any logged in user can create a blog
- \* Premium accounts can publish 5 free blogs
- \* Non premium accounts can publish 0 free blogs
- \* Blogs costs £0.50

## Scenario: Dr Bill posts to his own blog

Given I am logged in as Dr Bill

And I have a premium account

And I have published 4 blogs

When I try to post to my blog

Then I should see "Your article was published"

## Scenario: Dr Bill hits max number of free blogs

Given I am logged in as Dr Bill

And I have a premium account

And I have published 5 blogs

When I try to post to my blog

Then I should see a payment page

## Scenario: Dr Bill pays for a blog

Given I am logged in as Dr Bill

And I have a premium account

And I have paid £0.50 for a blog

When I try to post to my blog

Then I should see "Your article was published"

And my account should be updated

Feature = title for the feature

Feature file = a file that contains AC's and Scenarios usually stored in GitHub

# ADVANCED SYNTAX

## Feature: Post a blog

As a customer  
I want to post informative blogs  
So that that I can share what I'm doing and raise my profile

## Acceptance Criteria

- \* Any logged in user can create a blog
- \* Premium accounts can publish 5 free blogs
- \* Non premium accounts can publish 0 free blogs
- \* Blogs costs £0.50

## Background:

Given I am logged in as Dr Bill  
And I have a premium account

## Scenario: Dr Bill posts to his own blog

Given I have published 4 blogs  
When I try to post to my blog  
Then I should see "Your article was published"


## Scenario: Dr Bill hits max number of free blogs

Given I have published 5 blogs  
When I try to post to my blog  
Then I should see a payment page

## Scenario: Dr Bill pays for a blog

Given I have paid £0.50 for a blog  
When I try to post to my blog  
Then I should see "Your article was published"  
And my account should be updated

**Background** = set the context for all scenarios



# ADVANCED SYNTAX

## Feature: Post a blog

As a customer  
I want to post informative blogs  
So that that I can share what I'm doing and raise my profile

### Acceptance Criteria

- \* Any logged in user can create a blog
- \* Premium accounts can publish 5 free blogs
- \* Non premium accounts can publish 0 free blogs
- \* Blogs costs £0.50

### Background:

Given I am logged in as Dr Bill  
And I have a premium account

### Scenario Outline: Dr Bill posts to his own blog

Given I have published <number of blogs>  
When I try to post to my blog  
Then I should see <message>

### Examples:

number of blogs	message
4	"Your article was published"
5	"Please pay £0.50 to publish this blog"

### @manual

#### Scenario: Dr Bill pays for a blog

Given I have paid £0.50 for a blog  
When I try to post to my blog  
Then I should see "Your article was published"  
And my account should be updated

Scenario Outline / Examples =  
used to combine a set of similar  
scenarios

Use tags to identify test  
suites/group scenarios

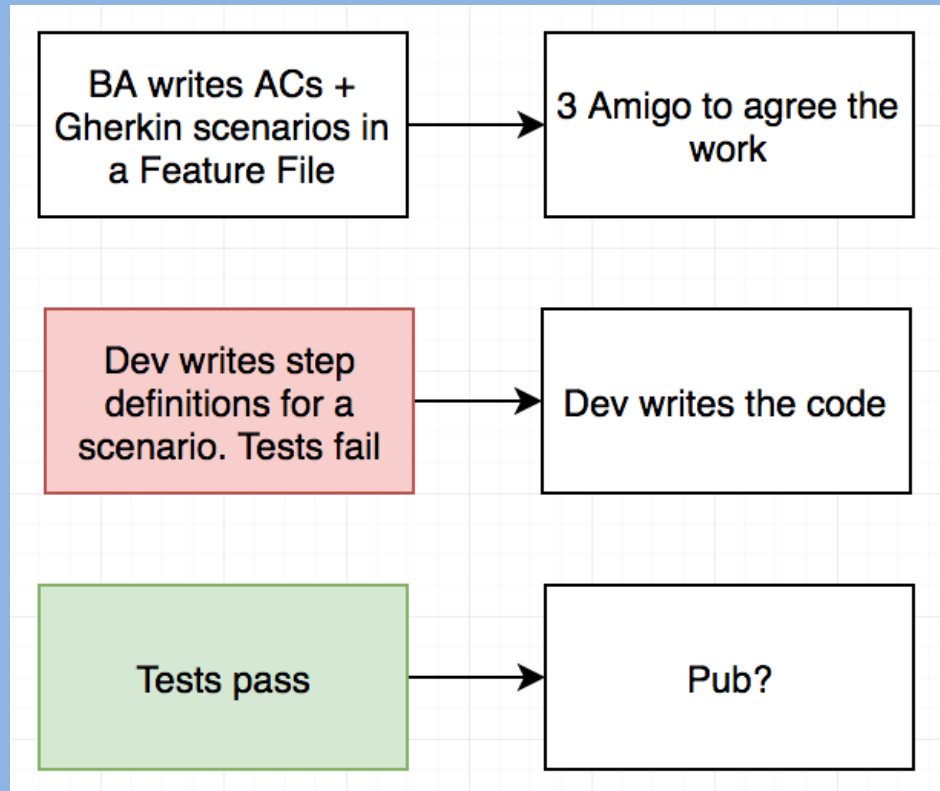


# BDD IN A NUTSHELL

# BASIC PRINCIPLES

- 1) Developer writes the test before the code
- 2) It's behavioural/high level tests (e.g. not unit tests)

# BDD PROCESS



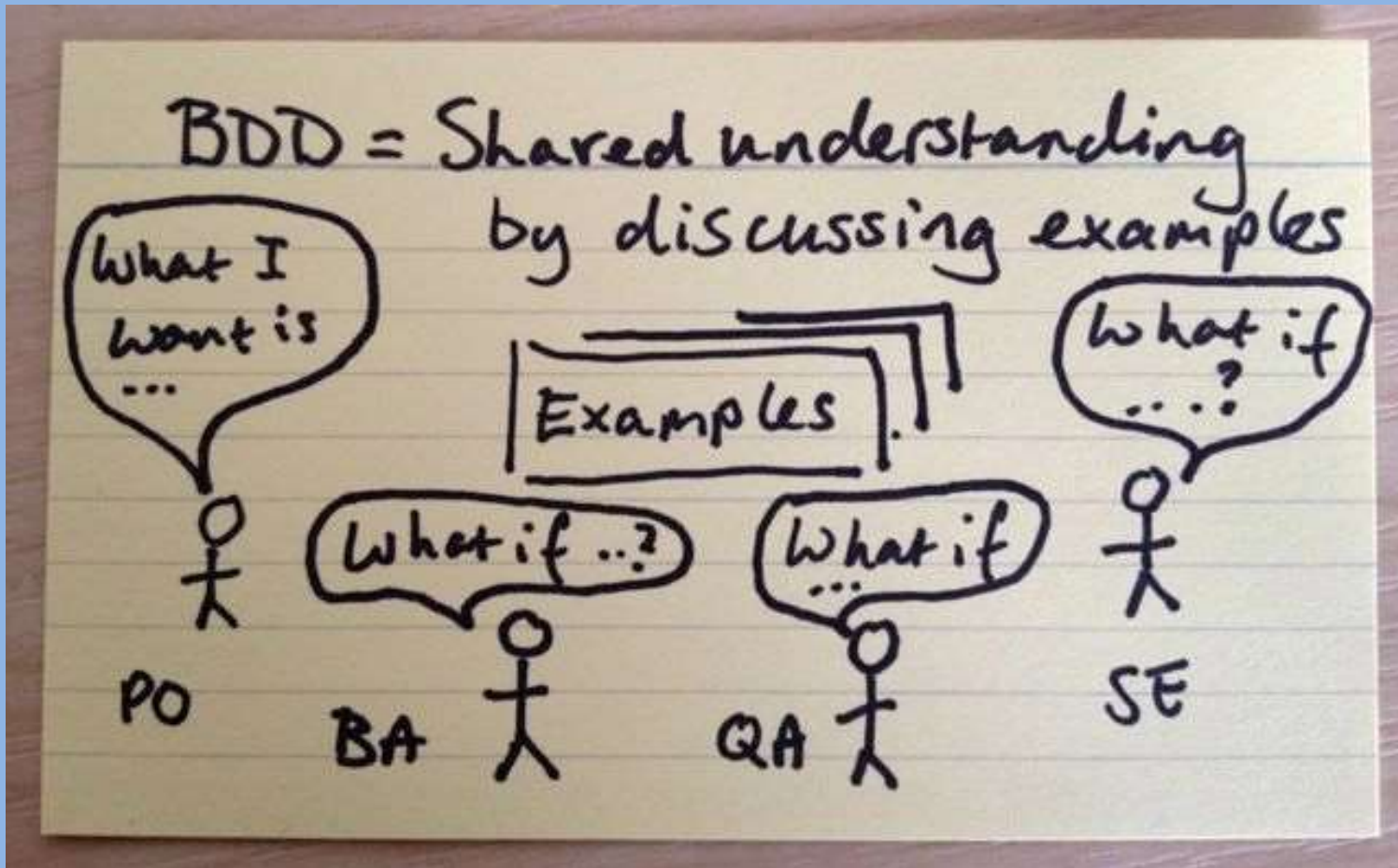
Step definition = written by the devs for each step

```
# features/step_definitions/coffee_steps.rb

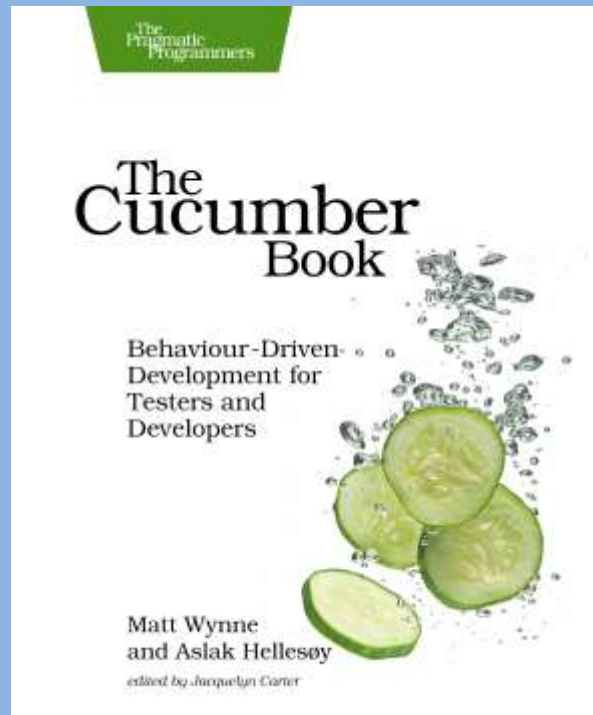
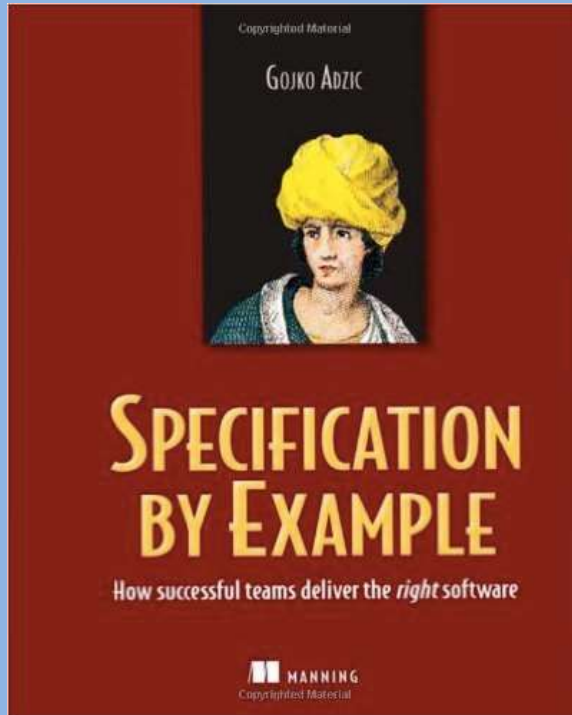
Then "I should be served coffee" do
  @machine.dispensed_drink.should == "coffee"
end
```

# WHY USE BDD?

1. Ensures automated code coverage
2. Valuable tests



# FURTHER READING



## Cucumber Notes

Cucumber is designed to build bridges between the technical and non-technical members of a software team by the use of automated acceptance testing. They are sometimes called *executable specifications*. It's vitally important that feature tests can be easily read and understood by any member of the team.

Developers and stakeholders (POs / BAs / etc) work together with QAs and developers to create automated tests that reflect the outcomes the stakeholders want.

During this process the team will develop a *ubiquitous language* to describe things and actions referenced in the service such as "customers", "service agents", "payments out" or "account balance".

Feature tests are different to *unit tests* which are written by developers to ensure that each component they build functions as expected from a basic functionality point of view. Feature tests can do this from a business point of view but they can also span multiple components.

Tests are written *first* before any code so will they fail when originally written as there is no code to test at that point! This is called *Test Driven Development* or *Outside-In Development*.

Tests are written as *scenarios* which are examples of how we want to use the system (n.b. this includes showing us appropriate errors if we or the system do something wrong).

Cucumber can produce an HTML file describing all the tests that can be executed for a system. This can be used to automatically created *living documentation* of how the system will function.

Cucumber tests are written in a language called *Gherkin*. They consist of:

- *Features*: Groups of tests
- *Tags*: Another way to group tests
- *Scenarios*: Individual tests
- *Steps*: The component parts of each test

Comments in Gherkin must take up a whole line of text and must start with a # (hash).

Developers will create *step definition code* that will map from the humanly readable steps to the code that can interface with the system.

Cucumber can be run by the developers on the command line before they submit their code to git and also as part of a continuous integration test service such as *Jenkins*. The output from Jenkins can be an HTML page that can be read by anyone in the team.

# FURTHER READING

<https://github.com/SkillsFundingAgency/das-paymentsacceptancetesting/tree/master/features>

<https://watirmelon.blog/2015/11/20/the-10-dos-and-500-donts-of-automated-acceptance-testing/>

<https://johnfergusonsmart.com/feature-mapping-a-simpler-path-from-stories-to-executable-acceptance-criteria/>

# KEYWORDS QUIZ – PART 1

- |                        |  |
|------------------------|--|
| 1. Gherkin             | A) These put the system in a state, specify the action & the outcome.<br>GIVEN/WHEN/THEN/AND/BUT |
| 2. Steps               | B) A tool used for testing software  |
| 3. Cucumber            | C) A language used to write acceptance tests.<br>It consists of <u>10 key words</u>              |
| 4. Scenarios           | D) Bullet pointed list of rules. These can be tested using scenarios                             |
| 5. Acceptance criteria | E) Consist of steps + the scenario title. These are your <u>acceptance tests</u>                 |

# KEYWORDS QUIZ – PART 2

- |                            |   |
|----------------------------|---|
| 6. Feature file            | F) Written by the developer. Every step has one of these  |
| 7. BDD                     | G) A repository used to store code + feature files. Think of it like a shared drive                             |
| 8. Step definition         | H) Allow us to write acceptance tests in a common language & for these tests to be automated with Cucumber      |
| 9. Github                  | I) A document/file that contains the user story, acceptance criteria & scenarios. It's saved as <i>.feature</i> |
| 10. Reasons to use Gherkin | J) Behaviour driven development. Write the expected behaviour & tests first – test fails – then write the code  |





**Given** you follow me to the  
dark side  
**When** you have to maintain  
scenarios  
**Then** humm.....

# STARTER FOR 10

- 1) How many people know what Gherkin is?
- 2) How many people use Gherkin on their project?
- 3) How comfortable are people with terms like Gherkin/Cucumber/BDD/Scenario Outline/Step Definitions/Github/Feature Files?