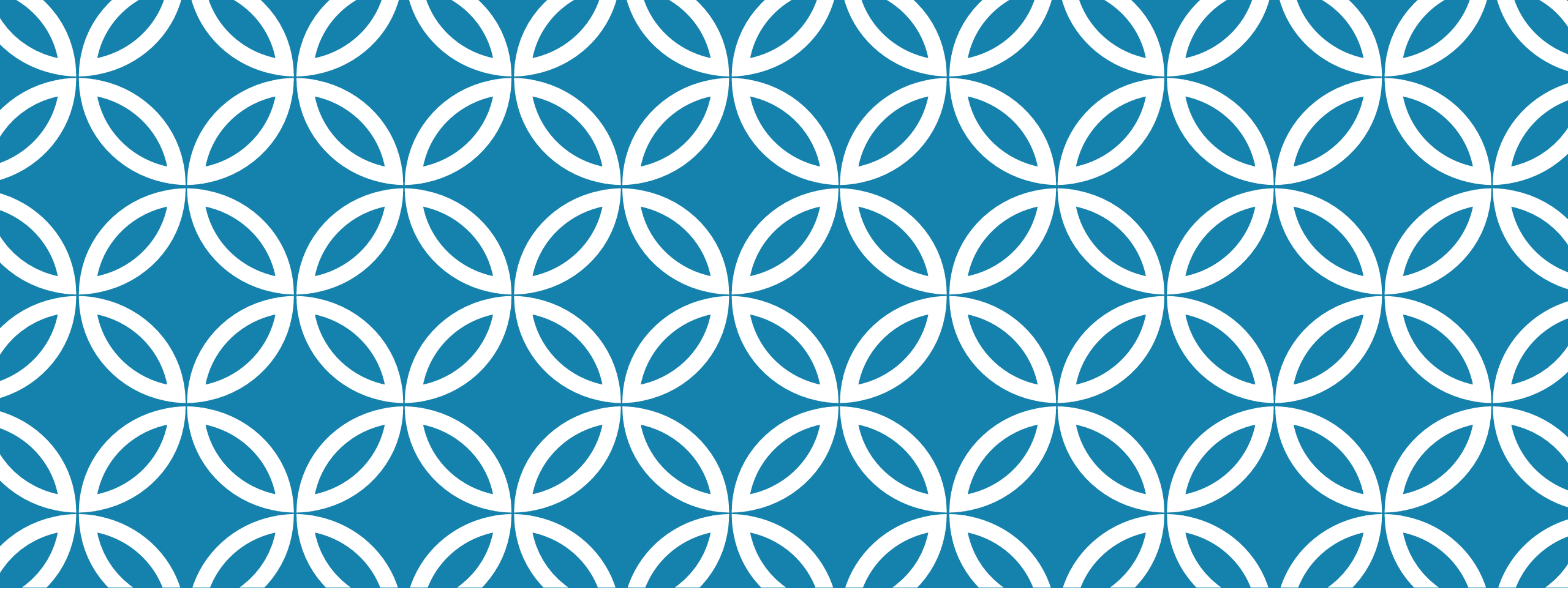


BDD-CUCUMBER

Sumit Ghosh

BDD OVERVIEW

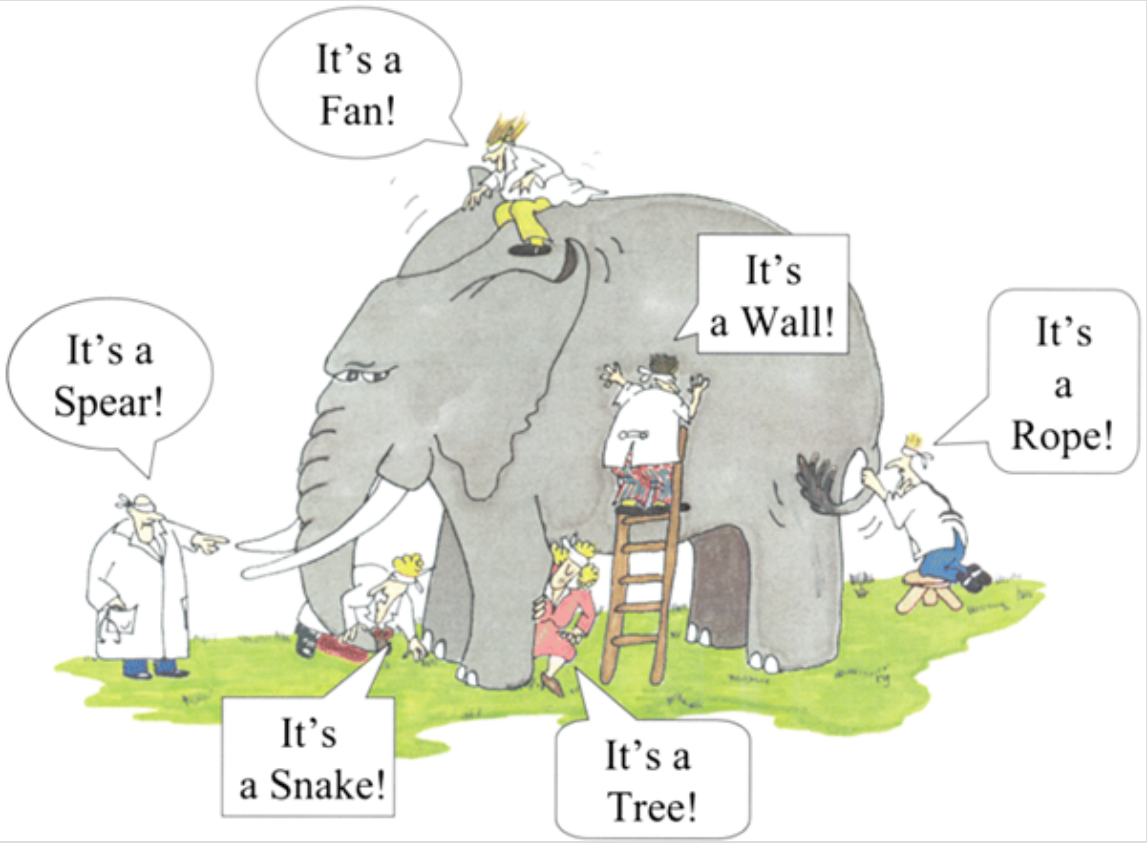
BDD is a Software Development Process that reduces the gap between Business and Technical Teams by:

encouraging collaboration across roles of Business and Agile Teams to build shared understanding of the problem to be solved.

working in rapid, small iterations to increase feedback and the flow of value.

producing system documentation that is automatically checked against the System's behavior.

using a common language (called *Gherkin*) and specialized tools to support the delivery of software.



BENEFITS OF USING BDD

Drives Stronger Collaboration:

BDD increases collaboration between the Business and IT Teams.

Focuses on Business Value:

BDD puts great emphasis on the Business Value and Needs.

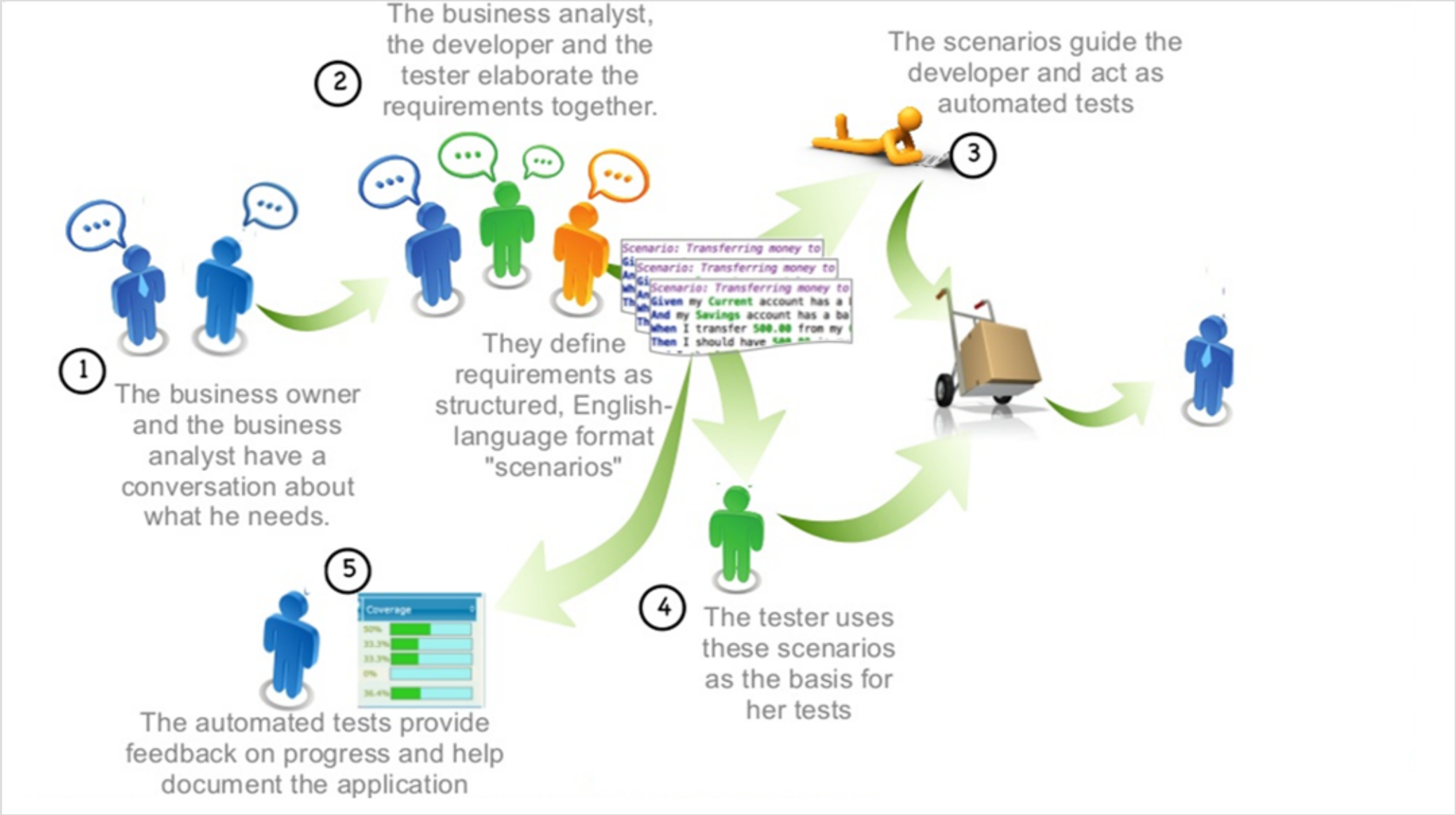
Uses a Common language:

In order to reduce gaps in interpretation, BDD uses a common language (*Gherkin*) to specify Requirements.

The Three Amigos

- Product Owner:
  - defines *what* problem must be solved.
- Developers:
  - suggest *how* the solution to the problem will be implemented.
- Testers:
  - *verify* that the software solution is built correctly.

Typical BDD Process Flow



GHERKIN KEYWORDS

Primary Keywords:

- **Feature**
- **Scenario**
- **Given, When, Then, And, But** for steps (or ***)**
- **Background**
- **Scenario Outline Examples**

Secondary Keywords:

- **|** (Data Tables)
- **@** (Tags)
- **#** (Comments)

A SIMPLE SCENARIO EXAMPLE

(User Story): As a Bank customer
I want to withdraw cash from an ATM
so that I have money to buy dinner

Scenarios:

1. ATM card is valid
    1. Account Balance has sufficient funds+ATM Machine has sufficient funds
    2. Account Balance has insufficient funds +ATM Machine has sufficient funds
    3. Account Balance has sufficient funds+ATM Machine has insufficient funds
2. ATM card is in valid

```
#Given: Some precondition step
#When: Some key actions
#Then: To observe outcomes or validation
#And,But: To enumerate more Given,When,Then steps
#Scenario Outline: List of steps for data-driven as an Examples and <placeholder>
```

Feature: Withdraw Cash from an ATM.

Scenario: Account has sufficient funds

Given the account balance is '$100'
And the ATM card is valid
And the machine contains more than '$100'
When the Account Holder requests '$80'
Then the ATM should dispense '$80'
And the account balance should be '20'
And the ATM card should be returned

Scenario: Account has insufficient funds

Given the account balance is '$40'
And the ATM card is valid
And the machine contains more than '$100'
When the Account Holder requests '$80'
Then the ATM should not dispense any cash
And the ATM should display a message "Insufficient Funds"
And account balance should still be '40'
And the ATM card should be returned

A SIMPLE SCENARIO OUTLINE

(User Story): As a Bank customer
I want to withdraw cash from an ATM using Saving or Current A\C
so that I have money to buy dinner

Scenario Outline: Account has sufficient funds
Given the account balance is '$100' in <"Account">
And the ATM card is valid
And the machine contains more than '$100'
When the Account Holder requests '$80'
Then the ATM should dispense '$80'
And the account balance should be '20'
And the ATM card should be returned

Examples:
|Account |
|Saving A\C |
|Current A\C|

IMPERATIVE STYLE:

There are two ways we write our gherkin Statement

- Imperative Style
- Declarative Style

Imperative Style: Imperative tests communicate details. They are closely tied to the mechanics of the current UI, they often require more work to maintain. Any time the implementation changes, the tests need to be updated too.

```
Example-01
Given I visit "/login"
When I enter "Bob" in the "user name" field
And I enter "tester" in the "password" field
And I press the "login" button
Then I should see the "welcome" page
```

```
Example-2
Scenario: Free subscribers see only the free articles
Given users with a free subscription can access "FreeArticle1" but not "PaidArticle1"
When I type "freeFrieda@example.com" in the email field
And I type "validPassword123" in the password field
And I press the "Submit" button
Then I see "FreeArticle1" on the home page
 And I do not see "PaidArticle1" on the home page

Example-3
Scenario: Subscriber with a paid subscription can access "FreeArticle1" and "PaidArticle1"
 Given I am on the login page
 When I type "paidPattya@example.com" in the email field
 And I type "validPassword123" in the password field
 And I press the "Submit" button
 Then I see "FreeArticle1" and "PaidArticle1" on the home page
```

DECLARATIVE STYLE:

Declarative Style: Declarative style describes the behaviour of the application, rather than the implementation details. Declarative scenarios read better as “living documentation". A declarative style helps you focus on the value that the customer is getting, rather than the keystrokes they will use. One way to make scenarios easier to maintain and less brittle is to use a declarative style.

```
Scenario: Login Scenario
 When "Bob" logs in

Scenario: Free subscribers see only the free articles
 Given Frieda has a free subscription
 When Frieda logs in with her valid credentials
 Then she sees a Free article on the home page

Scenario: Subscriber with a paid subscription can access both free and paid articles
 Given Paid Patty has a basic-level paid subscription
 When Paid Patty logs in with her valid credentials
 Then she sees a Free article and a Paid article on the home page
```

AVOID CONJUNCTIVE STEPS

One action per step makes your steps more modular and increases reusability

Given I am on the home page
When I login as an admin
Then I should be on my dashboard and I should see "You have successfully logged in."

Should be refactored, break the last step into 2 steps:
Given I am on the home page
When I login as an admin
Then I should be on my dashboard
And I should see "You have successfully logged in."

AND

Scenario: *As an existing and enabled ATM user, I want to make a withdrawal to get money.*
Given *I authenticated with a card enabled*
And *The available balance in my account is positive*
When *I select the option to withdraw money*
And I *enter the amount of money that is less than the amount I have available and the ATM's available balance*
Then *I get the money*
And *The money I get is subtracted from the available balance of my account*
And *The system returns the card automatically*
And *The system displays the transaction completed message*

```
#Given: Some precondition step
#When: Some key actions
#Then: To observe outcomes or validation
#And,But: To enumerate more Given,When,Then steps
#Scenario Outline: List of steps for data-driven as an Examples and <placeholder>
```

AVOID REPETION

```
Feature: Test Background Feature
Description: The purpose of this feature is to test the Background keyword

Background: User is Logged In
 Given I navigate to the login page
 When I submit username and password
 Then I should be logged in

Scenario: Search a product and add the first product to the User basket
 Given User search for Lenovo Laptop
 When Add the first laptop that appears in the search result to the basket
 Then User basket should display with added item

Scenario: Navigate to a product and add the same to the User basket
 Given User navigate for Lenovo Laptop
 When Add the laptop to the basket
 Then User basket should display with added item
```

MANAGE DATA

Data Tables

With Cucumber data tables, you can pass parameters from feature files in tabular format. And you can then use this data in step definition methods in the form of **Lists** and **Maps**.

Business Value Data

```
@Register
  Scenario: Register with username & passowrd
    Given I am on travel portal
    When I register with details
      | UserName         | Sumit    |
      | Password         | IBM@123  |
      | Confirm Password | IBM@123  |
    Then I should be successfully register
```

DATA TABLES

Cucumber data table without header

```
Given I open my application
And I login with following credentials
     | admin | pass1234 |
```

```
List<String> list = dt.asList(String.class);
System.out.println("Username - " + list.get(0));
System.out.println("Password - " + list.get(1));
```

```
Map<String, String> test = table.asMap(String.class, String.class);
System.out.println(test.get("max"));
```

Cucumber data table with header

```
Given I open Facebook URL
And fill up the new account form with the following data
     | First Name | Last Name | Phone No    | Password | DOB Day | DOB Month | DOB Year |
Gender |
     | Test FN    | Test LN   | 0123123123 | Pass1234 |      01 | Jan       |     1990 |
Male   |
```

```
List<Map<String, String>> list = dt.asMaps(String.class, String.class);
```

```
Given I open Facebook URL and create new accounts with below data
     | First Name | Last Name | Phone No    | Password | DOB Day | DOB Month | DOB Year |
Gender |
     | Abc FN     | Abc LN    | 0123123123 | Pass1234 |      01 | Jan       |     1990 | Male
|
     | Def FN     | Def LN    | 0456456456 | Abcd1234 |      01 | Feb       |     1990 |
Female |
     | Xyz FN     | Xyz LN    | 0789789789 | Pass2018 |      01 | Mar       |     1990 |
Female |
```

```
List<List<String>> list = dt.asLists(String.class);
```

```
List<Map<String, String>> list = dt.asMaps(String.class, String.class);
```

HOOKS

Hooks & Scenario

```
@Before
public void beforeEveryScenario(Scenario scenario) {
String tag = scenario.getSourceTagNames().toString();
}


@After
public void afterEveryScenario(Scenario scenario) {
if (scenario.isFailed()) {
Screen print
}
}
```

TAGS

Manage Execution

@CucumberOptions. Some examples:

tags = {"@SmokeTest"} Execute all scenarios under the @SmokeTest tag

tags = {"@gui"} Execute all the scenarios under the @gui tag feature level tag

tags = {"@SmokeTest," "@RegressionTest"} Execute all scenarios that are under the @SmokeTest and @RegressionTest tags (AND condition).

tags = {"@RegressionTest", "~@SmokeTest"} executes all scenarios under the @RegressionTest tag, but ignores all scenarios under the @SmokeTest tag

tags = {"@gui," "~@SmokeTest," "~@RegressionTest"} ignores all the scenarios under the tag @SmokeTest and @RegressionTest but executes all those under the tag "@gui," if we follow the example it's like running all the scenarios of the feature that are not under any other tag.

CLI

mvn clean install -U

mvn test -Dcucumber.filter.tags="@Register“

LINKS

- https://github.com/gsumit1/CucumberBDD [Dev Branch]
- https://cucumber.io/docs/cucumber/
-