Cucumber

Ram Sharma

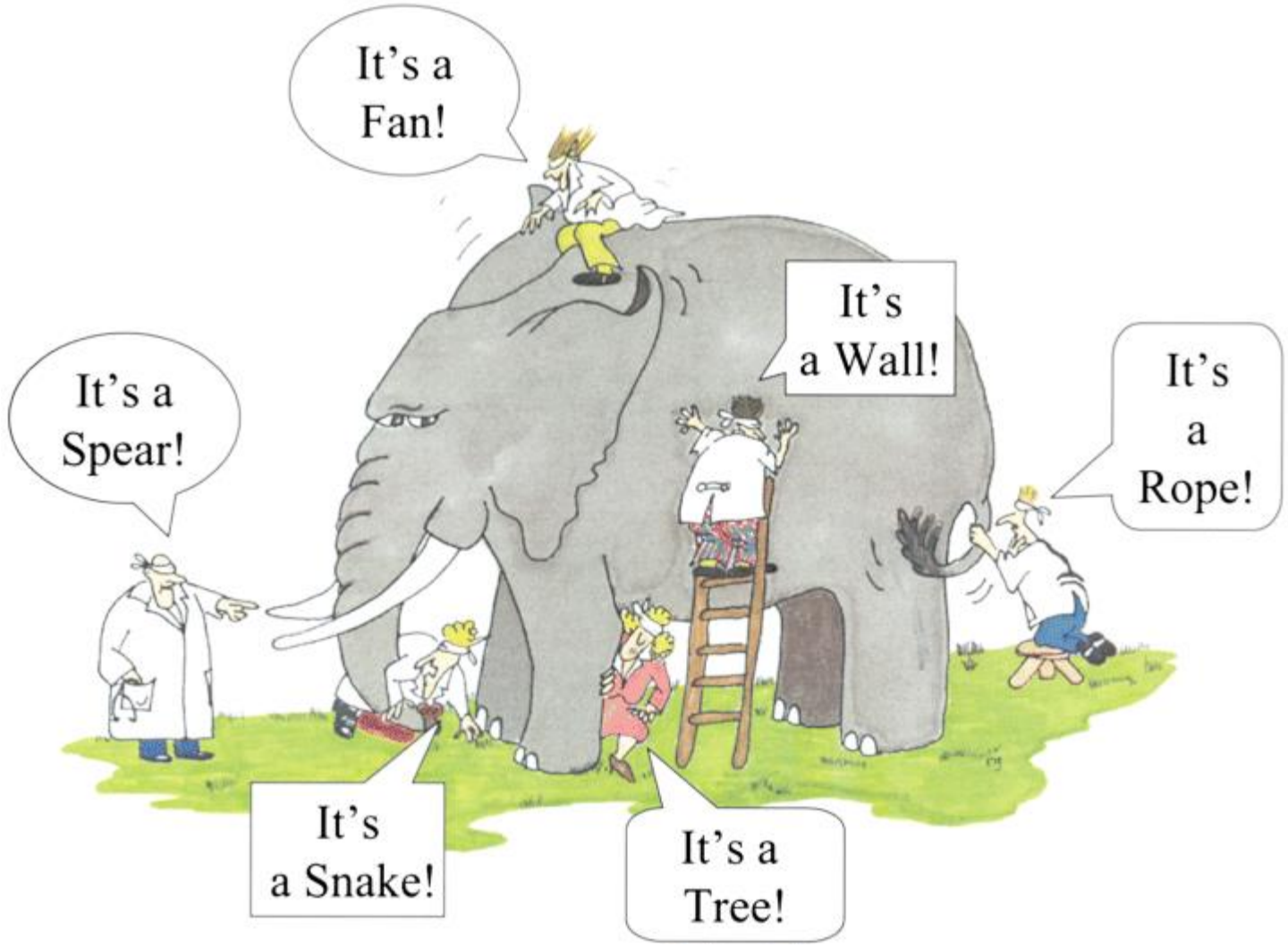
What is BDD?

- Behaviour-driven development (BDD) is a software development methodology in which an application is specified and designed by describing how its behaviour should appear to an outside observer.
- BDD is largely facilitated through the use of a simple domain-specific language (DSL) using **natural language constructs**.
- BDD is more about having conversations with stakeholders in such a way that it will uncover and clarify requirements.
- BDD suggests to test behaviours, so instead of thinking of how the code is implemented, we spend a moment thinking of what the scenario is.

How is it different from TDD?

- BDD focuses on the **behavioural aspect** of the system rather than the implementation aspect of the system that TDD focuses on.
- BDD gives a clearer understanding as to what the system should do from the **perspective of the developer and the customer**. TDD only gives the developer an understanding of what the system should do.
- BDD is when we write *behaviour & specification* then drives our software development
- BDD uses a more verbose style so that it can be read almost like a sentence. The ability to read your tests like a sentence is a cognitive shift in how you will think about your tests

Perception



What is Cucumber

- Cucumber follows BDD(Behavior driver development).
- Cucumber uses Gherkin language.
- It is mostly used in Agile Team.
- It requires three discipline(amigo) to discuss about requirement:
  - Tester
  - Developer
  - Product Owner
- Cucumber can generates reports in various formats : HTML, JSON

Keywords in Cucumber

- Feature
- Background
- Scenario
- Given, When, Then, And, But (Steps)
- Scenario Outline
- Examples

How Cucumber Works???

- It require three parties:
  - Feature file
  - Step Definition file
  - Runner class
- Runner will initiate the execution
- Cucumber can be started with JUnit, TestNG and others like Serenity

CucumberOptions

- features = path of feature files to be executed
- plugin = Types of reports to be generated
- glue = path of package of Step definitions
- dryRun = To run the scenario without executing
- tags = To run the specified tags

Parameterization

- When I enter 10 skills
- And I enter 8.9 as gpa
- Then I should get "high" as skill level

- @When("^I enter **(\\d+)** skills$")
- public void i_enter_skills(int arg1) {}

- @When("^I enter **(\\d+)\\.(\\d+)** as gpa$")
- public void i_enter_as_gpa(int arg1, int arg2) {}

- @Then("^I should get **\"([^\"]*)\"** as skill level$")
- public void i_should_get_as_skill_level(String arg1) {}

Datatable and Examples

- For one step if user wants to send multiple data, we can use **DataTable**.
- First row is also considered in Data table.
- We can convert Data table to List<List<String>>.
  - List<List<String>> myTableData = arg1.raw();
- DataTable can be used along with parameter(s) for a step.
- If user wants to run a scenario with multiple set of data, **Scenario Outline and Examples** can be used.
- First row will indicate the names used in the steps with enclosed with angular bracket.
- During the execution, data from rows will be replaced with respective name in the steps.

Cucumber Tags and Hooks

- Feature/Scenario can be tagged by using ***@tagName***.
- They can be tagged with more than one tags.
- To run a scenario with a tag, need to specify in tags of CucumberOptions.
  - Ex: tags={"@tag1", "~@tag3"}
  - Run scenario/feature with tag1 and not with tag3

Hooks

- Like TestNG and Junit, Cucumber also have hooks as below:
  - @Before
  - @After
- They can take Scenario as input parameter.
- They can also be tagged by tag as parameter.
- They can be ordered using @After(order = value).