Delivering Excellence in Software Engineering

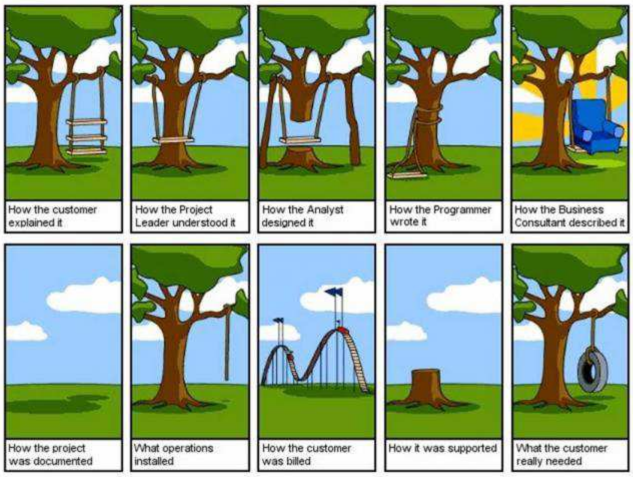
Software Testing

Introduction to Cucumber

Agenda

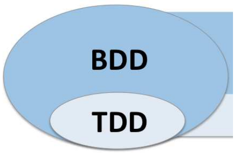
- **About BDD**
- **About Cucumber**
- **Gherkin**
- **Practices**

About BDD


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

About BDD

- **What is BDD?**

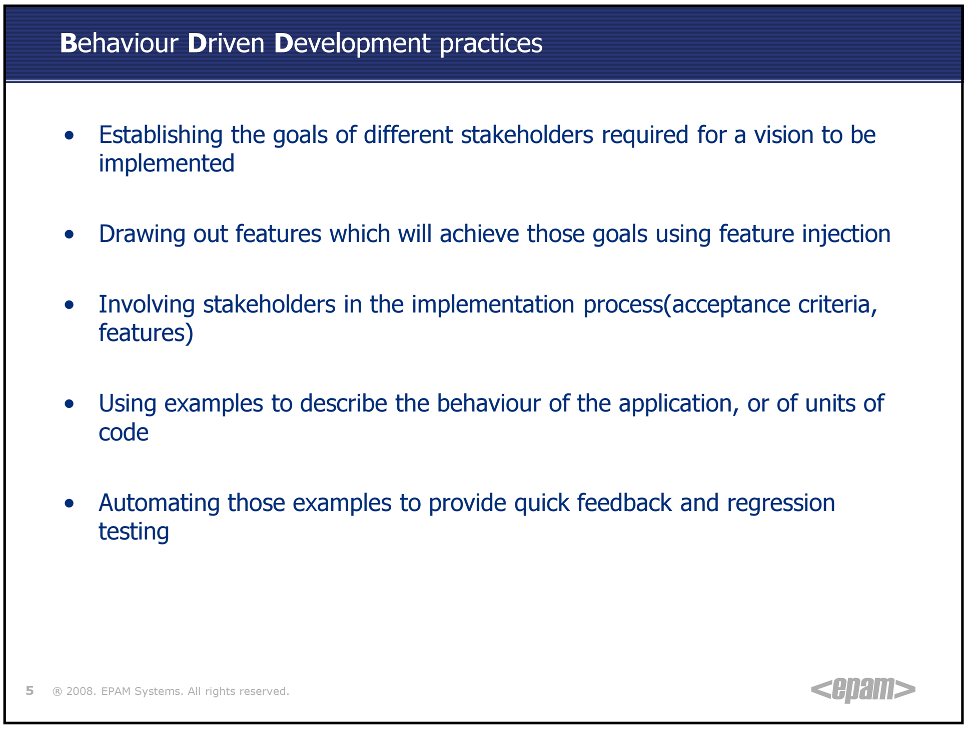


- BDD is an evolution in the thinking behind Test Driven Development and Acceptance Test Driven Planning.

- Unit test: *„Build the thing right."*
- Acceptance test: *„Build the right thing."*

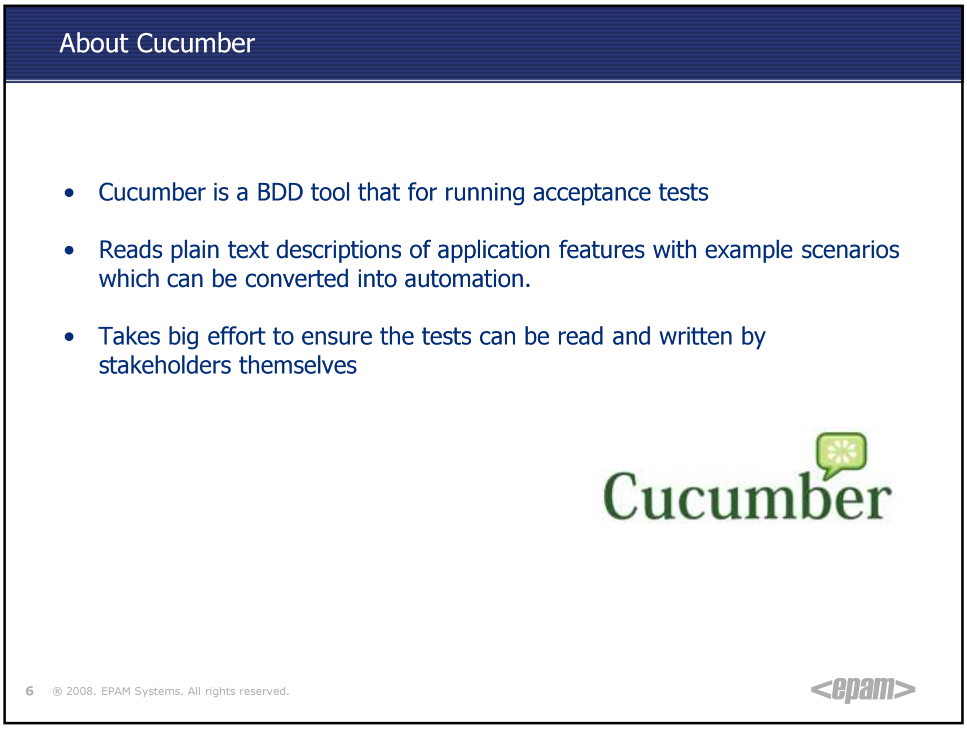
- BDD aims to help focus development on the delivery of prioritised, verifiable business value by providing a common vocabulary Ubiquitous Language) that spans the divide between Business and Technology.

  „A language structured around the domain model and used by all team members to connect all the activities of the team with the software."



Behaviour Driven Development practices

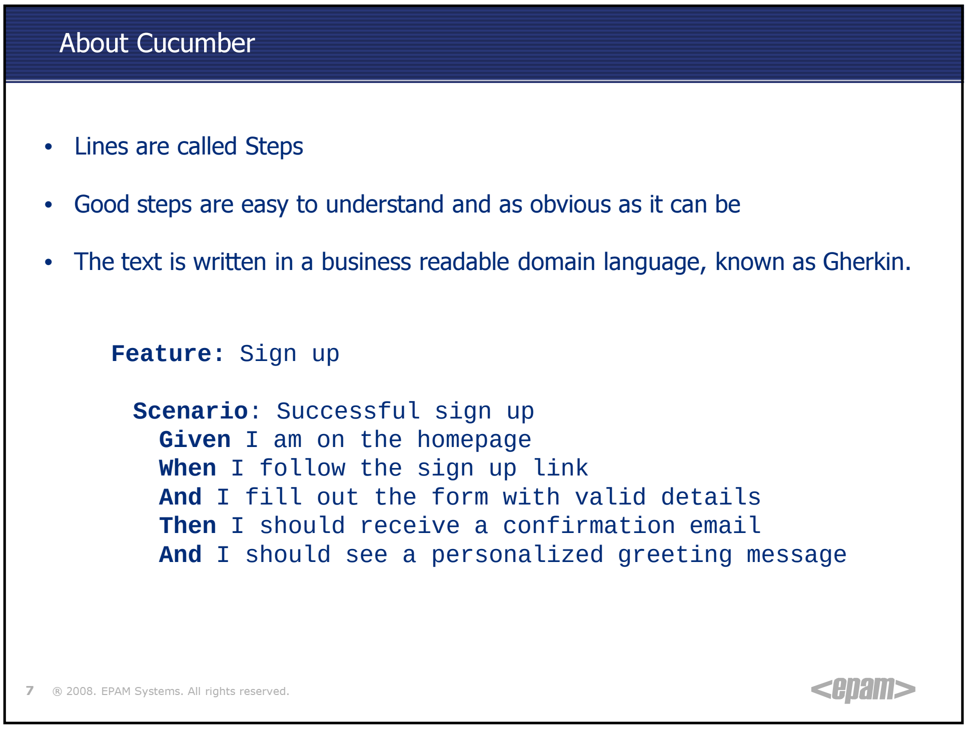
- Establishing the goals of different stakeholders required for a vision to be implemented
- Drawing out features which will achieve those goals using feature injection
- Involving stakeholders in the implementation process(acceptance criteria, features)
- Using examples to describe the behaviour of the application, or of units of code
- Automating those examples to provide quick feedback and regression testing



About Cucumber

- Cucumber is a BDD tool that for running acceptance tests
- Reads plain text descriptions of application features with example scenarios which can be converted into automation.
- Takes big effort to ensure the tests can be read and written by stakeholders themselves

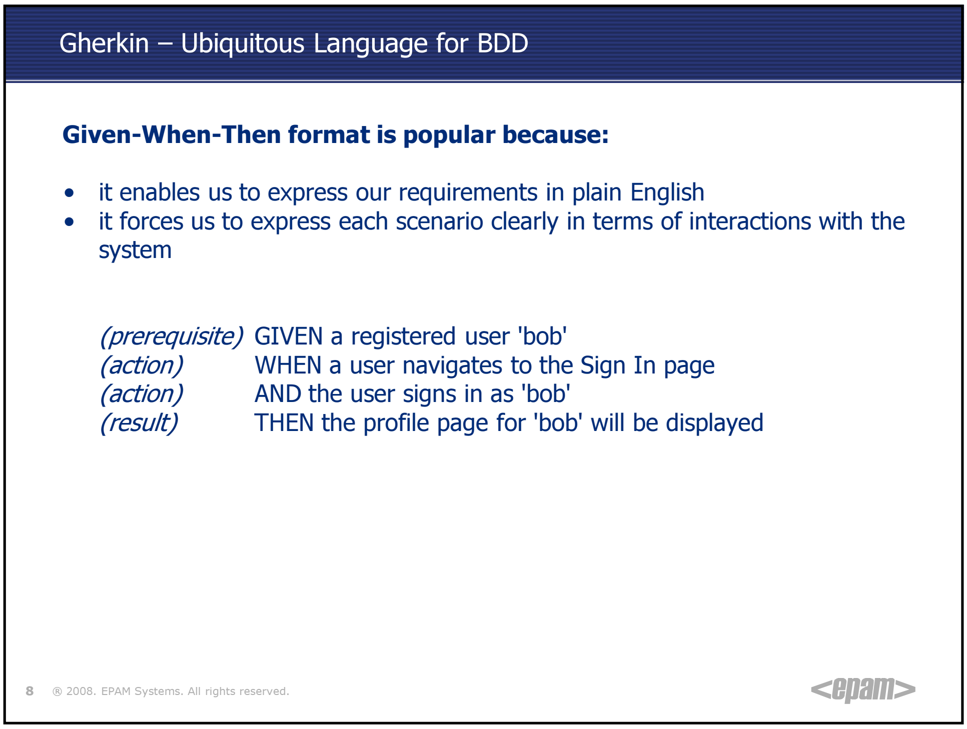
Cucumber



About Cucumber

- Lines are called Steps
- Good steps are easy to understand and as obvious as it can be
- The text is written in a business readable domain language, known as Gherkin.

```
Feature: Sign up

  Scenario: Successful sign up
    Given I am on the homepage
    When I follow the sign up link
    And I fill out the form with valid details
    Then I should receive a confirmation email
    And I should see a personalized greeting message
```



Gherkin – Ubiquitous Language for BDD

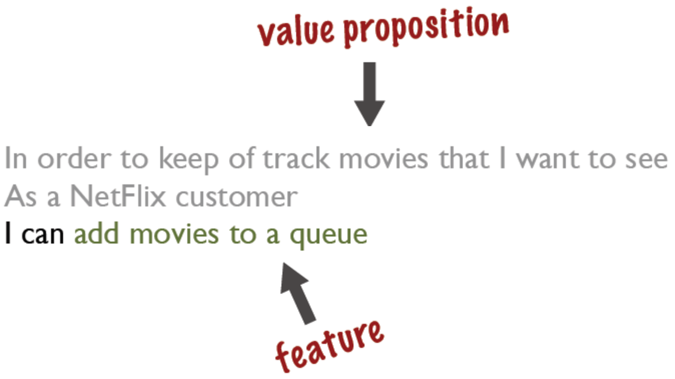
Given-When-Then format is popular because:

- it enables us to express our requirements in plain English
- it forces us to express each scenario clearly in terms of interactions with the system

(prerequisite)	GIVEN a registered user 'bob'
(action)	WHEN a user navigates to the Sign In page
(action)	AND the user signs in as 'bob'
(result)	THEN the profile page for 'bob' will be displayed

Gherkin - Feature

Feature: Adding movies to the queue


value proposition

In order to keep of track movies that I want to see
As a NetFlix customer
I can add movies to a queue

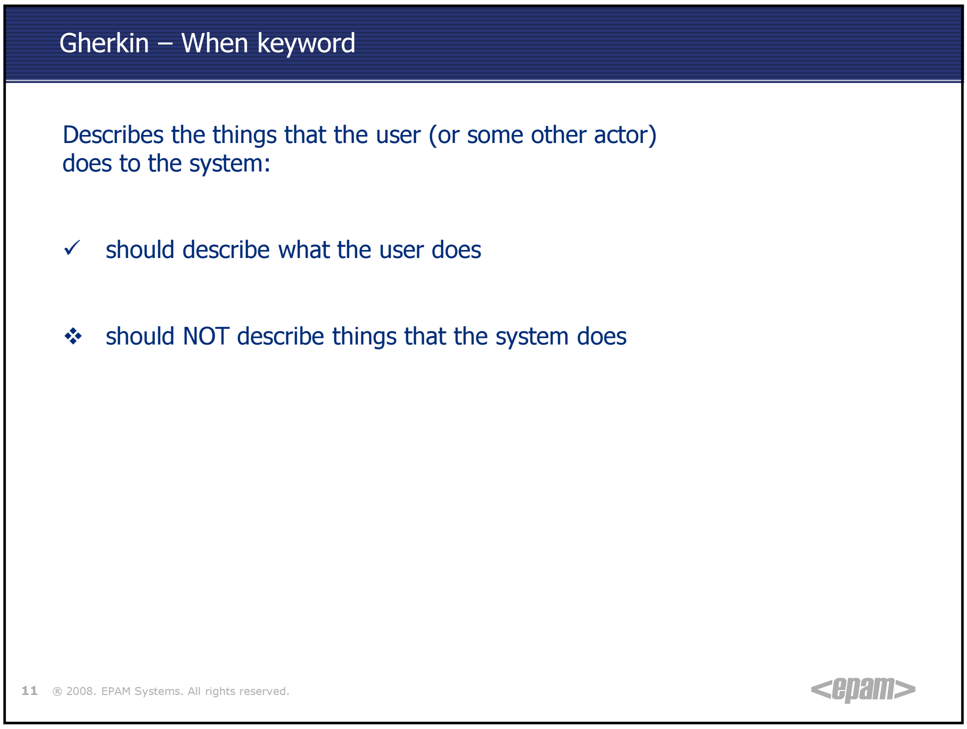
feature

Gherkin – Given keyword

Sets up the initial state for the scenario we are testing:

- ✓ may interact with the system
- ✓ should be expressed as a pre-existing condition

- ❖ should NOT perform interactions relevant to the scenario
- ❖ should NOT be expressed like an action



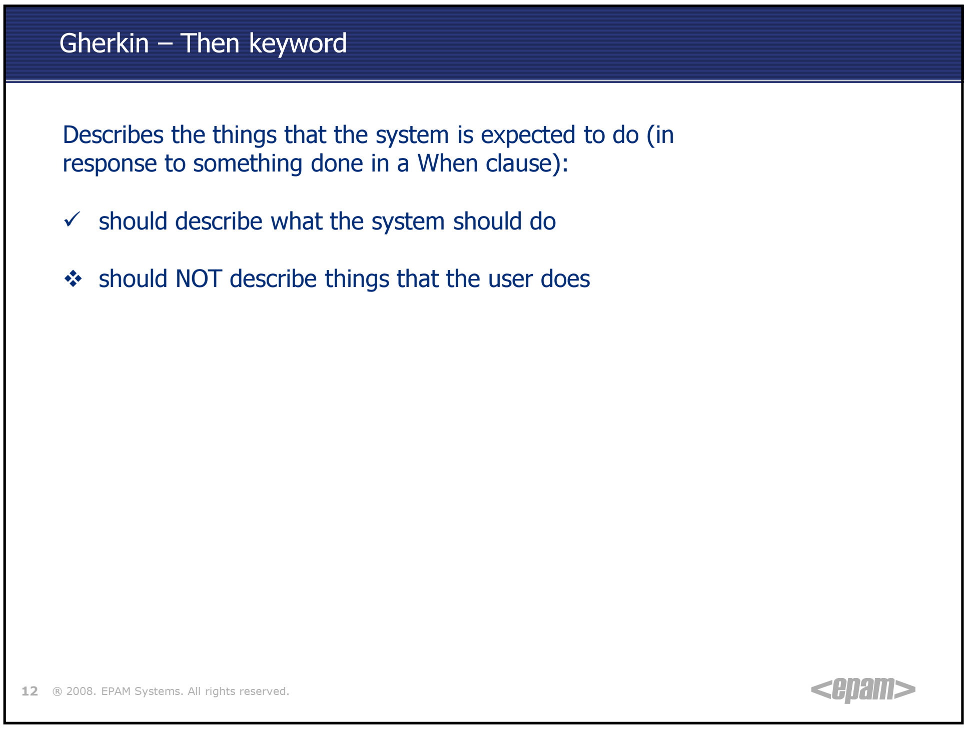
Gherkin – When keyword

Describes the things that the user (or some other actor) does to the system:

✓ should describe what the user does

❖ should NOT describe things that the system does

11



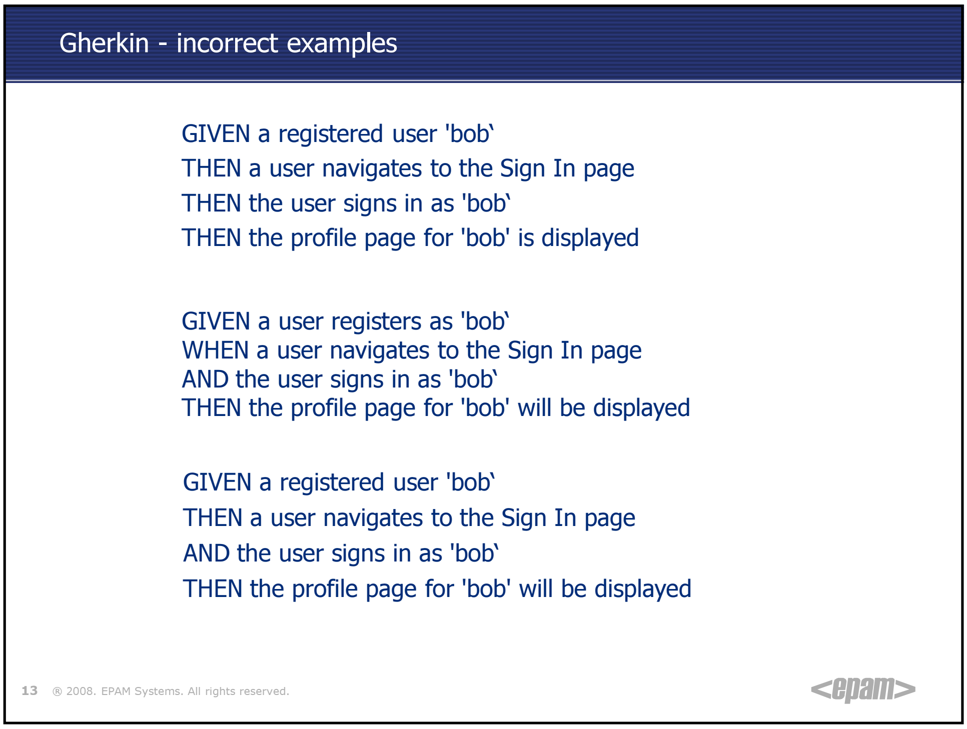
Gherkin – Then keyword

Describes the things that the system is expected to do (in response to something done in a When clause):

✓ should describe what the system should do

❖ should NOT describe things that the user does

12

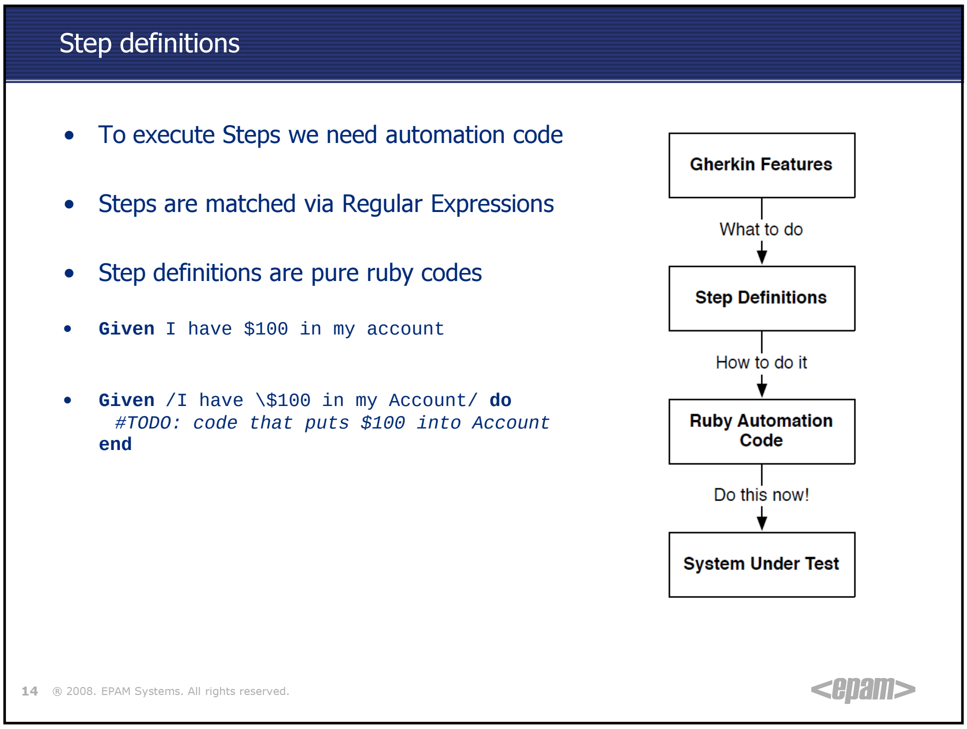


Gherkin - incorrect examples

GIVEN a registered user 'bob'
THEN a user navigates to the Sign In page
THEN the user signs in as 'bob'
THEN the profile page for 'bob' is displayed

GIVEN a user registers as 'bob'
WHEN a user navigates to the Sign In page
AND the user signs in as 'bob'
THEN the profile page for 'bob' will be displayed

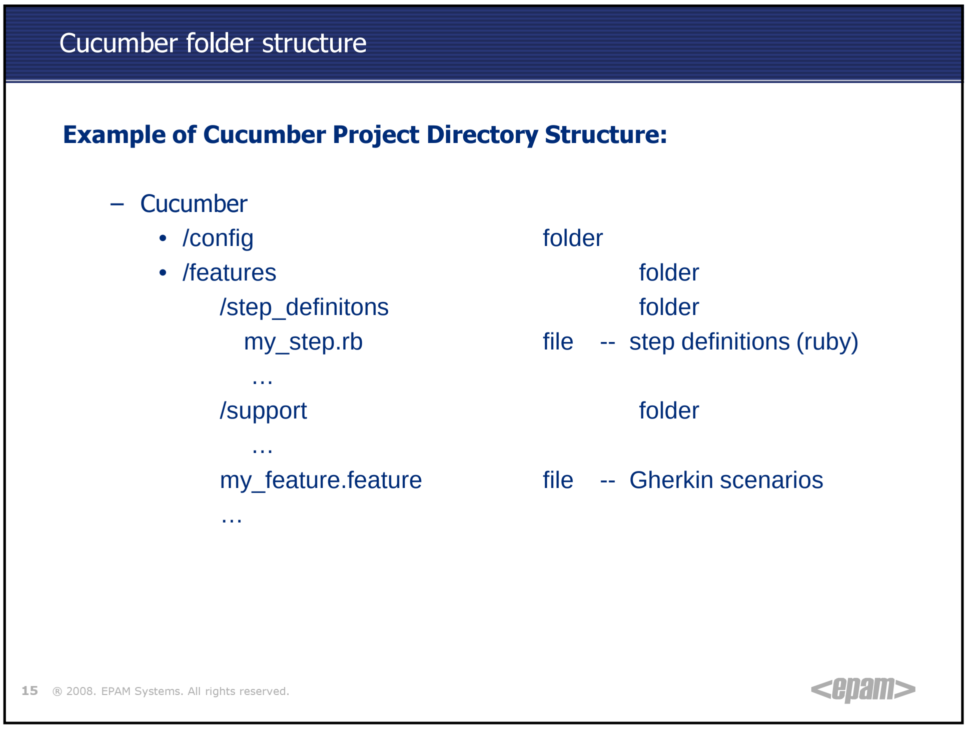
GIVEN a registered user 'bob'
THEN a user navigates to the Sign In page
AND the user signs in as 'bob'
THEN the profile page for 'bob' will be displayed



Step definitions

- To execute Steps we need automation code
- Steps are matched via Regular Expressions
- Step definitions are pure ruby codes
- `Given I have $100 in my account`
-
```
Given /I have \$100 in my Account/ do
  #TODO: code that puts $100 into Account
end
```

Gherkin Features
What to do
Step Definitions
How to do it
Ruby Automation Code
Do this now!
System Under Test



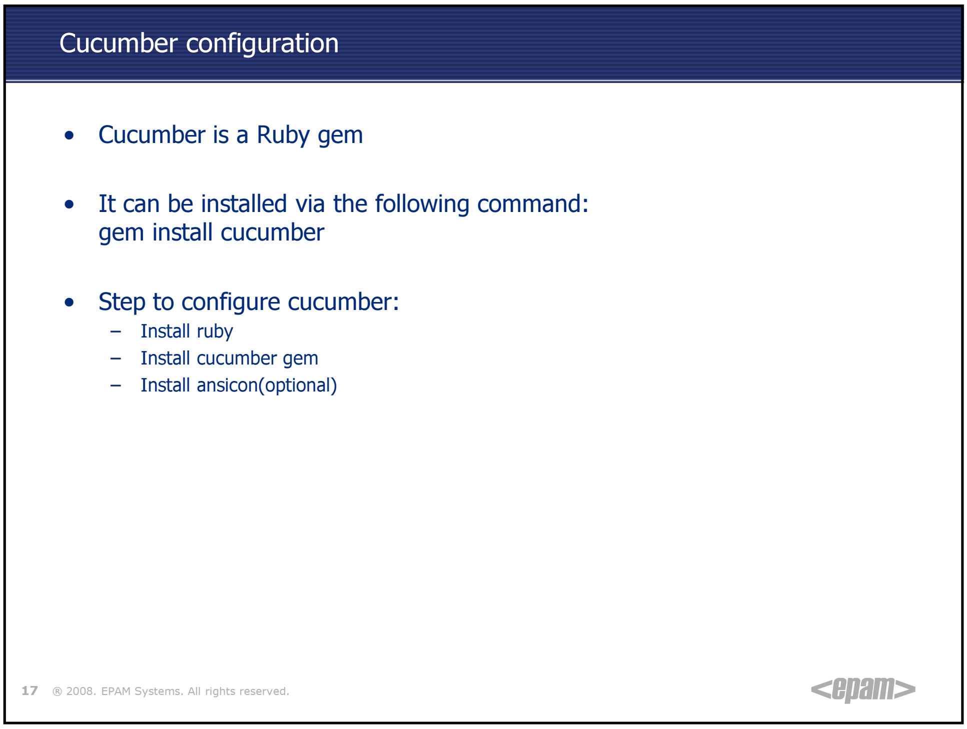
Cucumber folder structure

Example of Cucumber Project Directory Structure:

```
– Cucumber
    • /config                       folder
    • /features                          folder
        /step_definitons                 folder
           my_step.rb               file   --  step definitions (ruby)
            …
        /support                         folder
            …
        my_feature.feature          file   --  Gherkin scenarios
        …
```

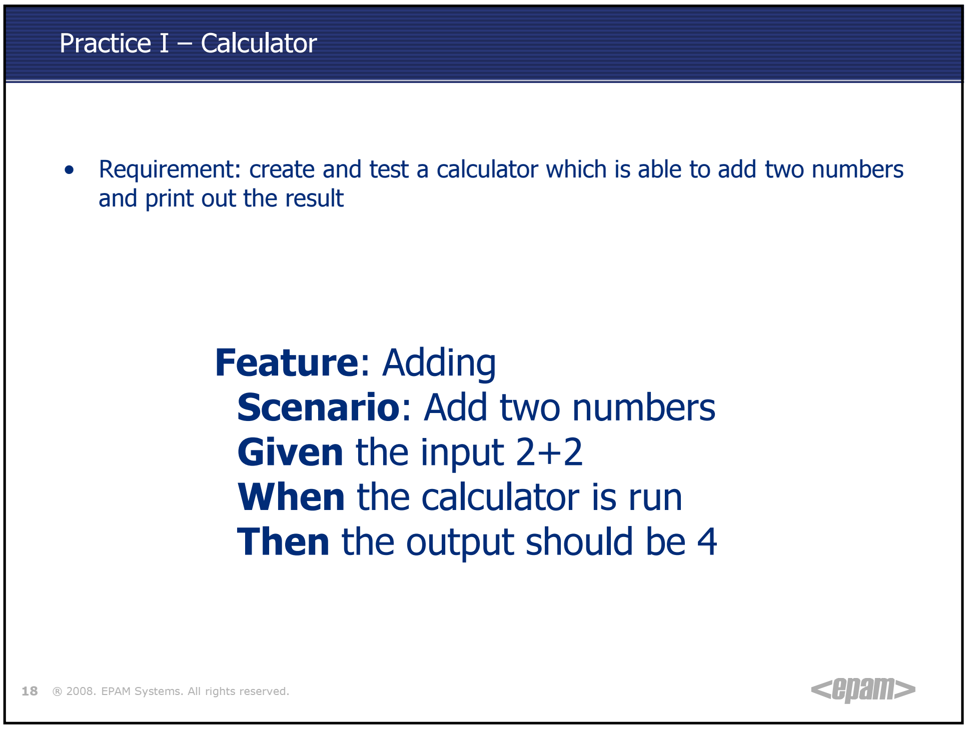
Delivering Excellence in Software Engineering

Practices



Cucumber configuration

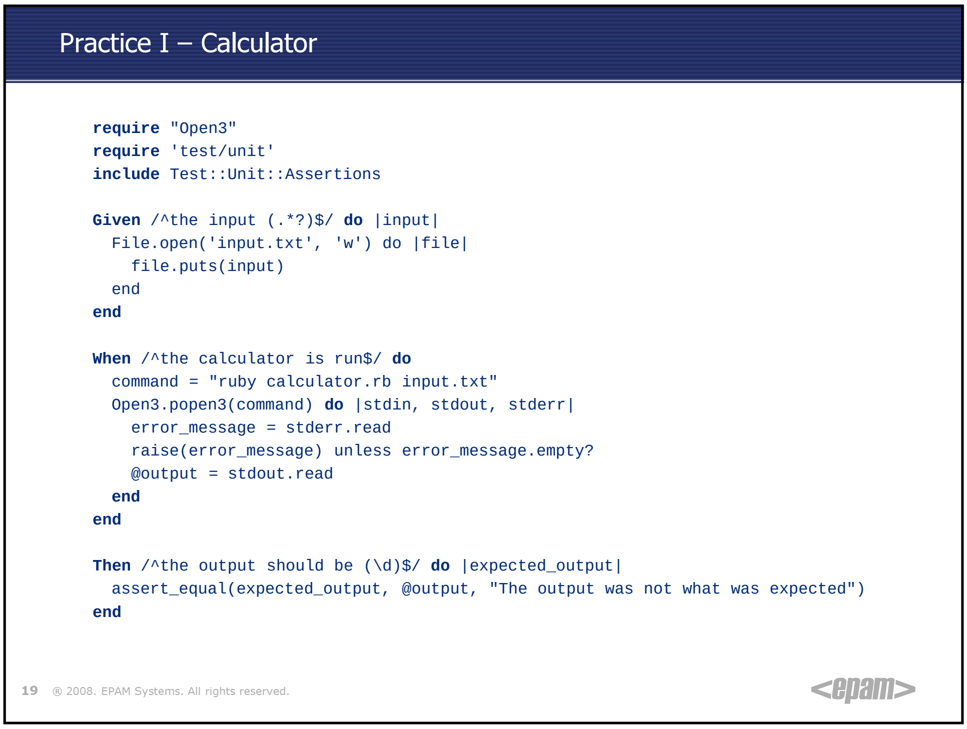
- Cucumber is a Ruby gem
- It can be installed via the following command:
  gem install cucumber
- Step to configure cucumber:
  - Install ruby
  - Install cucumber gem
  - Install ansicon(optional)



Practice I – Calculator

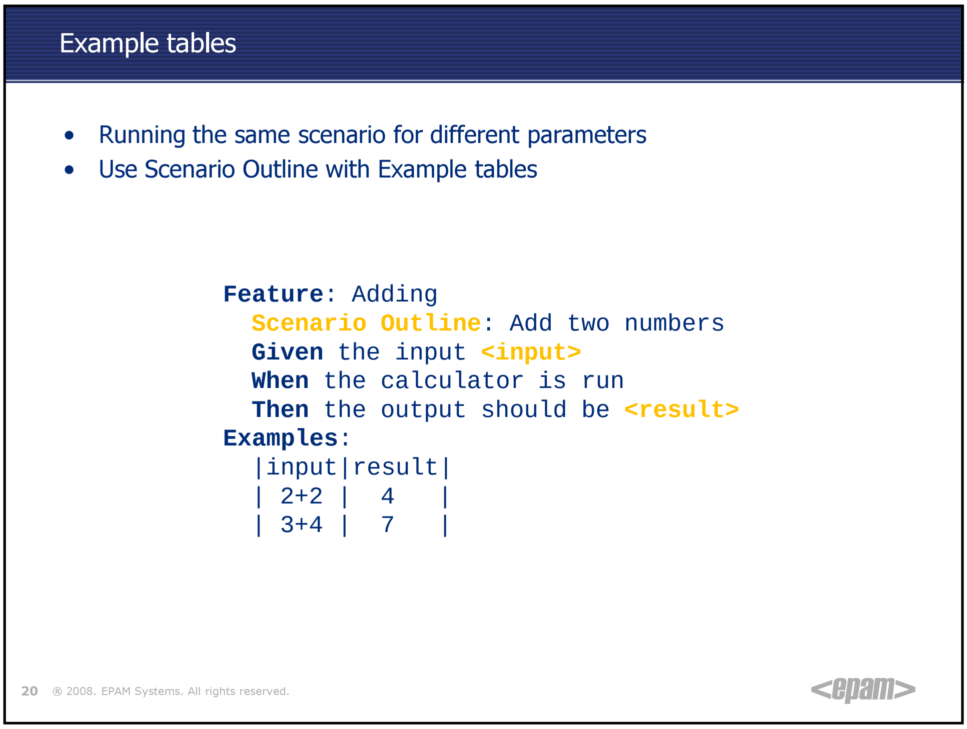
- Requirement: create and test a calculator which is able to add two numbers and print out the result

Feature: Adding
Scenario: Add two numbers
Given the input 2+2
When the calculator is run
Then the output should be 4



Practice I – Calculator

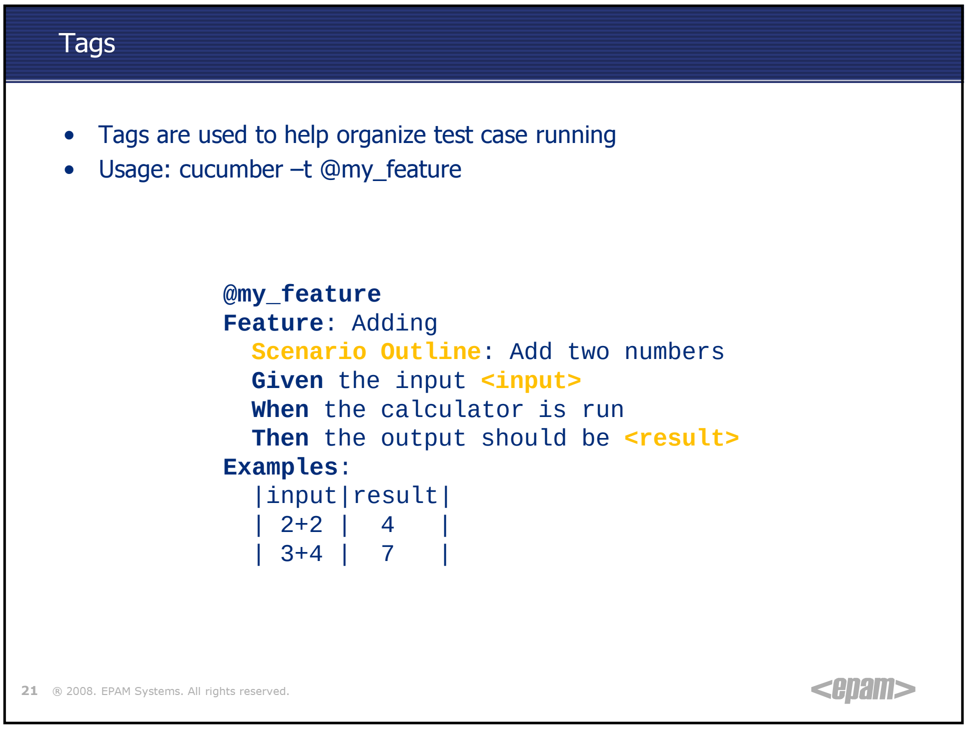
```
require "Open3"
require 'test/unit'
include Test::Unit::Assertions

Given /^the input (.*?)$/ do |input|
  File.open('input.txt', 'w') do |file|
    file.puts(input)
  end
end

When /^the calculator is run$/ do
  command = "ruby calculator.rb input.txt"
  Open3.popen3(command) do |stdin, stdout, stderr|
    error_message = stderr.read
    raise(error_message) unless error_message.empty?
    @output = stdout.read
  end
end

Then /^the output should be (\d)$/ do |expected_output|
  assert_equal(expected_output, @output, "The output was not what was expected")
end
```



Example tables

- Running the same scenario for different parameters
- Use Scenario Outline with Example tables

```
Feature: Adding
  Scenario Outline: Add two numbers
  Given the input <input>
  When the calculator is run
  Then the output should be <result>
Examples:
  |input|result|
  | 2+2 |  4   |
  | 3+4 |  7   |
```

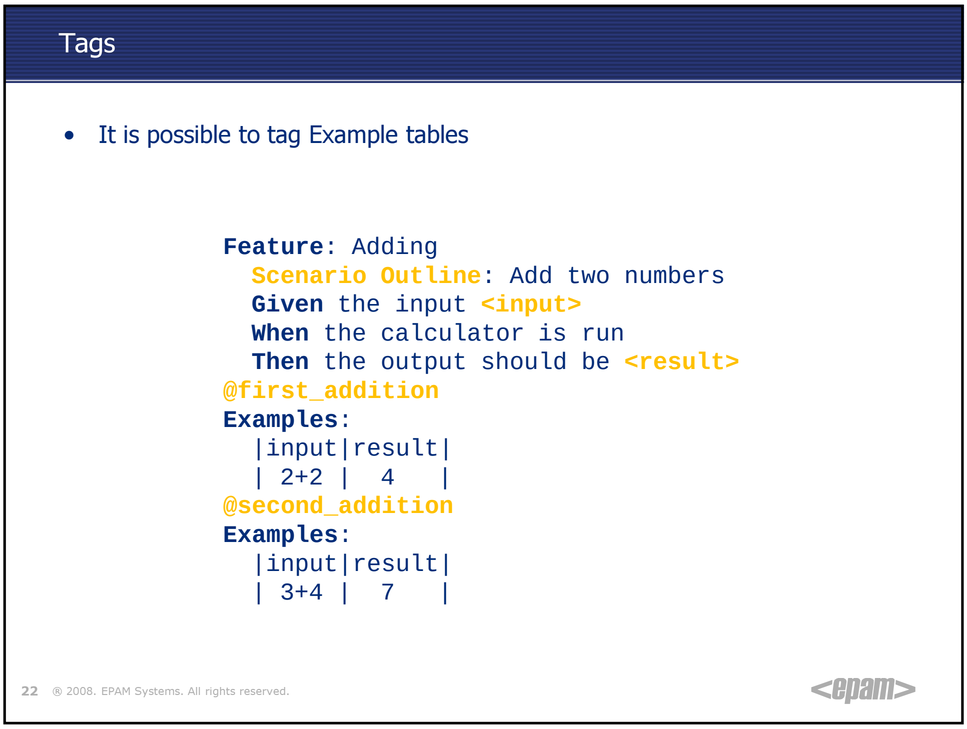


Tags

- Tags are used to help organize test case running
- Usage: cucumber –t @my_feature

```
@my_feature
Feature: Adding
  Scenario Outline: Add two numbers
  Given the input <input>
  When the calculator is run
  Then the output should be <result>
Examples:
  |input|result|
  | 2+2 |  4   |
  | 3+4 |  7   |
```

21

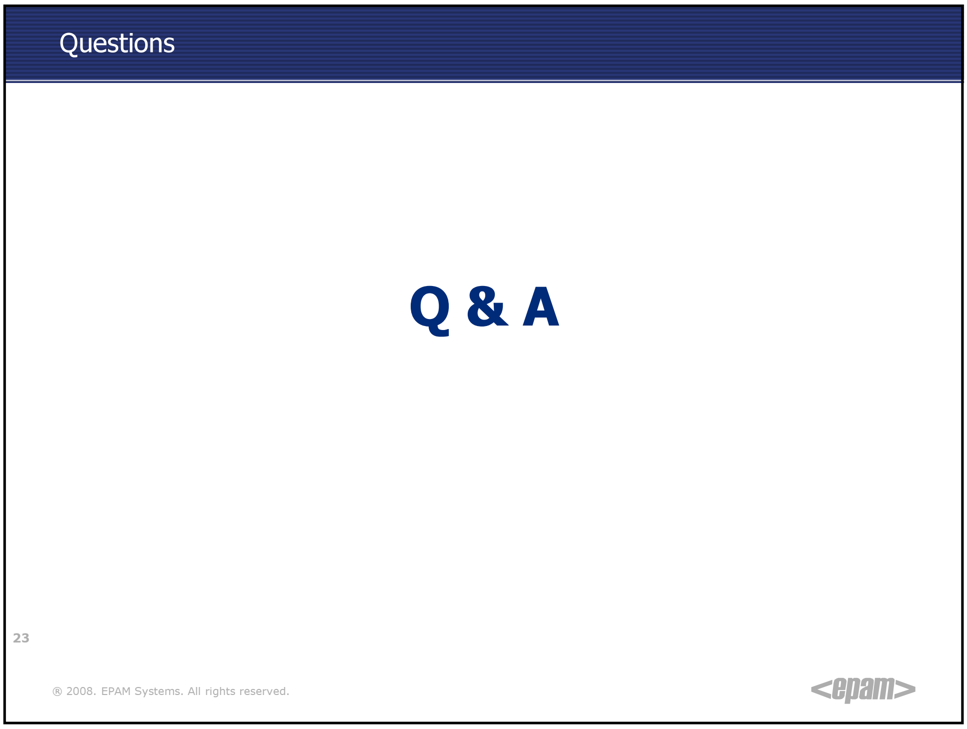


Tags

- It is possible to tag Example tables

```
Feature: Adding
  Scenario Outline: Add two numbers
  Given the input <input>
  When the calculator is run
  Then the output should be <result>
@first_addition
Examples:
  |input|result|
  | 2+2 |  4   |
@second_addition
Examples:
  |input|result|
  | 3+4 |  7   |
```

22



Questions

Q & A

Appendix

- **Helpful links:**

  Ruby 1.9.3 API

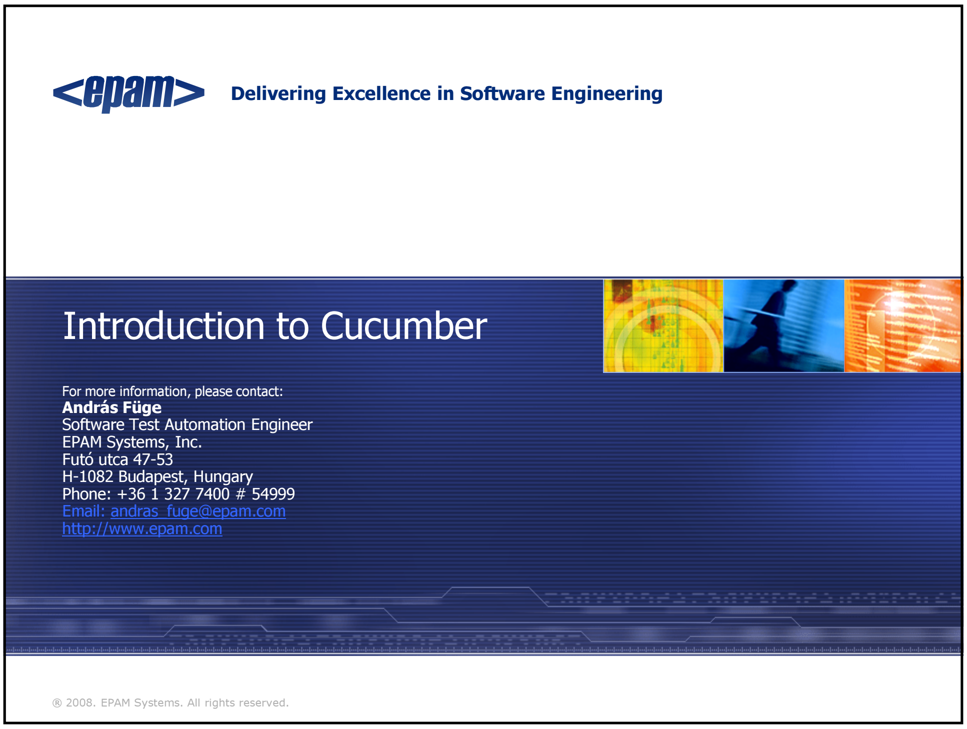
  Watir 1.6.5 API

  Watir WebDriver

  Regexp tutorial

  Cucumber

  Cuke Ninja



Delivering Excellence in Software Engineering

Introduction to Cucumber

For more information, please contact:
András Füge
Software Test Automation Engineer
EPAM Systems, Inc.
Futó utca 47-53
H-1082 Budapest, Hungary
Phone: +36 1 327 7400 # 54999
Email: andras_fuge@epam.com
http://www.epam.com