Cucumber (software)

Cucumber is a software tool that supports behavior-driven development (BDD).[7][8][9][10] Central to the Cucumber BDD approach is its ordinary language parser called Gherkin. It allows expected software behaviors to be specified in a logical language that customers can understand. As such, Cucumber allows the execution of feature documentation written in business-facing text.[11][7][8] It is often used for testing other software.[12] It runs automated acceptance tests written in a behavior-driven development (BDD) style.[13]

Cucumber was originally written in the Ruby programming language.[7][14][8] and was originally used exclusively for Ruby testing as a complement to the RSpec BDD framework. Cucumber now supports a variety of different programming languages through various implementations, including Java[15][8] and JavaScript.[16][17] The open source port of Cucumber in .Net is called SpecFlow.[18][19][20] For example, Cuke4php (https://github.com/olbrich/cuke4php) and Cuke4Lua (https://github.com/dgfitch/Cuke4Lua) are software bridges that enable testing of PHP and Lua projects, respectively. Other implementations may simply leverage the Gherkin parser while implementing the rest of the testing framework in the target language.

Contents

Cucumber

Developer(s)	Aslak Hellesøy,[1] Joseph Wilk,[2] Matt Wynne,[3] Gregory Hnatiuk,[4] Mike Sassak[5]
Stable release	3.1.2[6] / 13 July 2018
Repository	github.com/cucumber/cucumber-ruby (https://github.com/cucumber/cucumber-ruby)
Written in	Ruby
Operating system	Cross-platform
Type	Behavior driven development framework / Test tool
License	MIT License
Website	cucumber.io (https://cucumber.io)

Gherkin language

Gherkin is the language that Cucumber uses to define test cases. It is designed to be non-technical and human readable, and collectively describes use cases relating to a software system.[7][8][21][22] The purpose behind Gherkin's syntax is to promote behavior-driven development practices across an entire development team, including business analysts and managers. It seeks to enforce firm, unambiguous requirements starting in the initial phases of requirements definition by business management and in other stages of the development lifecycle.

In addition to providing a script for automated testing, Gherkin's natural language syntax is designed to provide simple documentation of the code under test.[22] Gherkin currently supports keywords in dozens of languages.[22][23][7][8]

Language Operations[22]

```
# List available languages
cucumber --i18n help

# List a language's keywords
cucumber --i18n $LANG
```

Syntax

Syntax is centered around a line-oriented design, similar to that of Python. The structure of a file is defined using whitespace and other control characters.[22] # is used as the line-comment character, and can be placed anywhere in a file.[22] Instructions are any non-empty and non-comment line. They consist of a recognized Gherkin keyword followed by a string.[24]

All Gherkin files have the `.feature` file extension. They contain a single Feature definition for the system under test and are an executable test script.[24]

Features, scenarios, and steps

Cucumber tests are divided into individual Features. These Features are subdivided into Scenarios, which are sequences of Steps.

Features

A feature is a Use Case that describes a specific function of the software being tested. There are three parts to a Feature[24]

- The `Feature:` keyword
- The Feature name (on the same line as the keyword)
- An optional description on the following lines

Example Feature definition

```
Feature: Withdraw Money from ATM

    A user with an account at a bank would like to withdraw money from an ATM.
```

```
    Provided he has a valid account and debit or credit card, he should be allowed to make the
transaction. The ATM will tend the requested amount of money, return his card, and subtract
amount of the withdrawal from the user's account.

    Scenario: Scenario 1
        Given preconditions
        When actions
        Then results

    Scenario: Scenario 2
        ...
```

Scenarios

Each Feature is made of a collection of scenarios. A single scenario is a flow of events through the Feature being described and maps 1:1 with an executable test case for the system.[24] Keeping with the example ATM withdrawal feature, a scenario might describe how a user requests money and what happens to their account.

```
Scenario: Eric wants to withdraw money from his bank account at an ATM
    Given Eric has a valid Credit or Debit card
    And his account balance is $100
    When he inserts his card
    And withdraws $45
    Then the ATM should return $45
    And his account balance is $55
```

In some cases, one might want to test multiple scenarios at once to perform Equivalence partitioning and Boundary-value analysis. A `Scenario Outline` provides a technique to specify multiple examples to test against a template scenario by using placeholders.[24] For example,

```
Scenario Outline: A user withdraws money from an ATM
    Given <Name> has a valid Credit or Debit card
    And their account balance is <OriginalBalance>
    When they insert their card
    And withdraw <WithdrawalAmount>
    Then the ATM should return <WithdrawalAmount>
    And their account balance is <NewBalance>

    Examples:
      | Name   | OriginalBalance | WithdrawalAmount | NewBalance |
      | Eric   | 100             | 45               | 55         |
      | Gaurav | 100             | 40               | 60         |
      | Ed     | 1000            | 200              | 800        |
```

At runtime the scenario is run against each row in the table. Column values are substituted for each of the named placeholders in the scenario.

Steps

The crux of a Scenario is defined by a sequence of Steps outlining the preconditions and flow of events that will take place. The first word of a step is a keyword, typically one of[24]

- `Given` - Describes the preconditions and initial state before the start of a test and allows for any pre-test setup that may occur
- `When` - Describes actions taken by a user during a test
- `Then` - Describes the outcome resulting from actions taken in the When clause

Occasionally, the combination of Given-When-Then uses other keywords to define conjunctions

- `And` - Logical and
- `But` - Logically the same as `And`, but used in the negative form[25]

```
Scenario: A user attempts to withdraw more money than they have in their account
    Given John has a valid Credit or Debit card
    And his account balance is $20
    When he inserts his card
    And withdraws $40
    Then the ATM displays an error
    And returns his card
    But his balance remains $20
```

Tags

Gherkin's Feature structure forces organisation. However, in cases where this default organisation is inconvenient or insufficient, Gherkin provides Tags. Tags are `@-prefixed` strings and can be placed before[24]

- `Feature`
- `Scenario`
- `Scenario Outline`
- `Examples`

An element can have multiple tags and inherits from parent elements.[22][24]

Cucumber

Step definitions

Steps in Gherkin's `.feature` files can be considered a method invocation.[26][22] Before Cucumber can execute a step it must be told, via a step definition, how that step should be performed.

Definitions are written in Ruby and conventionally filed under `features/step_definitions/*_steps.rb`.[22] Definitions start with the same keywords as their invocation (including Gherkin's full language support).[22] Each definition takes two arguments[22]

- Either a regular expression or string with $variables
- A block containing ruby code to execute

Example using regular expressions

```
Given /(.*) has a valid Credit or Debit card/ do |name|
  # Ruby code
end
```

Example using strings and $variables. Note that at runtime the string is converted into a regular expression, and any $variable is converted to match `(.*)`.[22]

```
Given "$name has a valid Credit or Debit card" do |name|
  # Ruby code
end
```

Hooks

Hooks are Cucumber's way of allowing for setup to be performed prior to tests being run and teardown to be run afterwards. They are defined as executable Ruby blocks, similar to JUnit methods marked with `@Before, @After` annotations. Conventionally they are placed under `support/`, and are applied globally.[22] Three basic types of hooks exist[22]

- `Before` - Runs before a scenario
- `After` - Runs after a scenario
- `Around` - Assumes control and runs around a scenario

Additional hooks include[22]

- `BeforeStep`
- `AfterStep`
- `AfterConfiguration` - Runs after Cucumber configuration and is passed an instance of the configuration

`Before, After, and Around` hooks optionally take a list of tags filtering scenarios that they apply to. A list of tags in the same string is treated as `OR`, while individual arguments are treated as `AND`; tags can be optionally negated by being preceded with `~`.[22]

Example of a tagged before hook

```
Before('@ATM') do |scenario|
  # Ruby code
end
```

Hooks are often used to maintain database state, typically by cleaning up prior to running a scenario. It is also possible to start and roll back a transaction using `Before` and `After` hooks, and many Cucumber extensions provide an `@txn` tag for such a purpose.[24]

Integrations and implementations

Non Ruby implementations of Cucumber exist for popular languages including Java, JavaScript, and Python.[24] Support also exists for integration testing frameworks. A complete list of implementations can be found on Cucumber. Cucumber has integrated testing tools working well with many Continuous Integration configurations. There are cucumber plugins for popular CI tools like Jenkins and TeamCity and also for IDEs like Eclipse and RubyMine.

Below is an example of a step definition written for Java with Cucumber-JVM.[27]

```
@Given("(.*) has a valid Credit or Debit card")
public void has_card(String name) {
    // Java code
}
```

Formatter plugins

Cucumber uses Formatter Plugins to provide output. Several common formats are provided by default, including[24]

- JSON
- HTML
- JUnit

Available formats are not standardized across different Cucumber implementations, so offerings may differ.[24] Cucumber also supports rich output formats like images and videos.

Browser automation

Cucumber does not provide built in browser automation. However, it does work well with existing programs such as Selenium and WATiR-WebDriver.[28] It does support running tests with transactions through leveraging other programs such as ActiveRecord.[29]

Cucumber command-line

Cucumber comes with a built-in command line interface that covers a comprehensive list of instructions. Like most command line tools, cucumber provides the `--help` option that provides a summary of arguments the command accepts.[30]

```
$ cucumber --help
        -r, --require LIBRARY|DIR        Require files before executing the features.
        --i18n LANG                      List keywords for in a particular language.
                                         Run with "--i18n help" to see all languages.
        -f, --format FORMAT              How to format features (Default: pretty).
        -o, --out [FILE|DIR]             Write output to a file/directory instead of
        ...
```

Cucumber command line can be used to quickly run defined tests. It also supports running a subset of scenarios by filtering tags.

```
$ cucumber --tags @tag-name
```

The above command helps in executing only those scenarios that have the specified `@tag-name`.[30] Arguments can be provided as a logical `OR` or `AND` operation of tags. Apart from tags, scenarios can be filtered on scenario names.[30]

```
$ cucumber --name logout
```

The above command will run only those scenarios that contain the word 'logout'.

It is also useful to be able to know what went wrong when a test fails. Cucumber makes it easy to catch bugs in the code with the `--backtrace` option.[30]

Cucumber can also be configured to ignore certain scenarios that have not been completed by marking them with the Work In Progress tag `@wip`. When Cucumber is passed the `--wip` argument, Cucumber ignores scenarios with the `@wip` tag.

References

1. "Aslak Hellesøy" (http://aslakhellesoy.com/). Aslakhellesoy.com. Retrieved 2012-01-24.
2. "Joseph Wilk | on AI, The Web, Usability, Testing & Software process" (http://blog.josephwilk.net/). Blog.josephwilk.net. Retrieved 2012-01-24.
3. "Tea-Driven Development" (http://blog.mattwynne.net/). Blog.mattwynne.net. Retrieved 2012-01-24.
4. "ghnatiuk's Profile" (https://github.com/ghnatiuk). GitHub. Retrieved 2012-01-24.
5. "msassak's Profile" (https://github.com/msassak). GitHub. Retrieved 2012-01-24.
6. "Releases - cucumber/cucumber-ruby" (https://github.com/cucumber/cucumber-ruby/releases). Retrieved 9 August 2018 – via GitHub.
7. "The Pragmatic Bookshelf | The Cucumber Book" (https://web.archive.org/web/20120121002246/http://pragprog.com/book/hwcuc/the-cucumber-book). Pragprog.com. Archived from the original (http://pragprog.com/book/hwcuc/the-cucumber-book) on 2012-01-21. Retrieved 2012-01-24.
8. Rose, Seb; Wynne, Matt; Hellesøy, Aslak (15 February 2015). *The Pragmatic Bookshelf | The Cucumber For Java Book* (https://pragprog.com/book/srjcuc/the-cucumber-for-java-book). Pragprog.com. Retrieved 2019-04-28.
9. "What is Cucumber?" (https://cucumber.io/docs/guides/overview/). *cucumber*. Retrieved 2019-06-08.
10. Aslak Hellesøy. "The world's most misunderstood collaboration tool" (https://cucumber.io/blog/the-worlds-most-misunderstood-collaboration-tool/). *cucumber*.
11. Fox, Armando; Patterson, David (2016). *Engineering Software as a Service*. Strawberry Canyon. pp. 218–255. ISBN 978-0-9848812-4-6.
12. "Automated testing with Selenium and Cucumber" (https://www.ibm.com/developerworks/library/a-automating-ria/). *www.ibm.com*. 2013-08-06. Retrieved 2017-02-09.
13. Soeken, Mathias; Wille, Robert; Drechsler, Rolf (2012-05-29). Furia, Carlo A.; Nanz, Sebastian (eds.). *Objects, Models, Components, Patterns*. Lecture Notes in Computer Science. Springer Berlin Heidelberg. pp. 269–287. doi:10.1007/978-3-642-30561-0_19 (https://doi.org/10.1007%2F978-3-642-30561-0_19). ISBN 9783642305603.
14. "The Pragmatic Bookshelf | The RSpec Book" (https://web.archive.org/web/20120121003856/http://pragprog.com/book/achbd/the-rspec-book). Pragprog.com. 2010-12-02. Archived from the original (http://pragprog.com/book/achbd/the-rspec-book) on 2012-01-21. Retrieved 2012-01-24.
15. "Cucumber-jvm" (https://github.com/cucumber/cucumber-jvm). cucumber. Retrieved 2018-03-08.
16. "Cucumber-js" (https://github.com/cucumber/cucumber-js). cucumber. Retrieved 2018-03-08.
17. Naidele Manjunath; Olivier de Meulder (2019-02-01). "No Code? No Problem — Writing Tests in Plain English" (https://open.nytimes.com/no-code-no-problem-writing-tests-in-plain-english-537827eaaa6e). Times Open. Retrieved 2019-04-29.
18. "Binding Business Requirements to .NET Code" (https://specflow.org/). *SpecFlow*. Retrieved 2019-04-29.
19. "SpecFlow" (https://github.com/techtalk/SpecFlow). *GitHub*. Retrieved 2019-04-29.
20. Richard Lawrence; Paul Rayner (2018). *Behavior-Driven Development with Cucumber*. Addison Wesley.

21. "cucumber/gherkin" (https://github.com/cucumber/cucumber/tree/master/gherkin). *GitHub*. Retrieved 2017-02-09.
22. "Gherkin Syntax" (https://cucumber.io/docs/gherkin/). *cucumber*. Retrieved 2019-07-09.
23. "Gherkin Supported Languages" (https://github.com/cucumber/cucumber/blob/master/gherkin/gherkin-languages.json). Gherkin. *cucumber/cucumber: Cucumber monorepo - building blocks for Cucumber in various languages*. Retrieved 2021-03-21 – via GitHub.
24. "Reference" (https://web.archive.org/web/20151025214933/https://cucumber.io/docs/reference). *cucumber*. Archived from the original (https://cucumber.io/docs/reference) on 2015-10-25. Retrieved 2016-01-17.
25. "Gherkin Language" (http://docs.behat.org/en/v2.5/guides/1.gherkin.html). *behat*. Retrieved 2016-01-17.
26. "Cucumber documentations" (https://github.com/cucumber/cucumber/wiki/A-Table-Of-Content). *GitHub*. 2019-01-23.
27. "Cucumber-JVM" (https://github.com/cucumber/cucumber-jvm). *GitHub*. Retrieved 10 February 2016.
28. "GitHub - watir/watir-webdriver: Watir-webdriver code has moved" (https://github.com/watir/watir-webdriver). 2018-06-09.
29. "GitHub - rails/rails: Ruby on Rails" (https://github.com/rails/rails/tree/master/activerecord). 2019-01-24.
30. Wynne, Matt; Hellesoy, Aslak. "The Cucumber Book" (https://www.safaribooksonline.com/library/view/the-cucumber-book/9781941222911/f_0090.html). *SafariBooksOnline*. Retrieved 22 January 2016.

External links

- Engineering Software as a Service: An Agile Approach Using Cloud Computing by Armando Fox and David Patterson (http://www.saasbook.info/about)
- Cucumber project (https://cucumber.io)
- Cucumber project documentation (https://cucumber.io/docs)
- At the Forge - Cucumber (http://www.linuxjournal.com/magazine/forge-cucumber), by Reuven M. Lerner in the Linux Journal
- Agile 2009 - Aslak Hellesoy - Cucumber test framework (http://agiletoolkit.libsyn.com/agile_2009_aslak_hellesoy_cucmber_test_framework), podcast by Bob Payne with Aslak Hellesøy
- Cucumber: The Latest in Ruby Testing (http://www.rubyinside.com/cucumber-the-latest-in-ruby-testing-1342.html), by Mike Gunderloy
- Specflow, Cucumber in .NET (http://www.specflow.org/)