Cucumber is an open source tool to support Behavior Driven Development with plain text specifications and unobtrusive automation in Ruby. Alternative implementations of Cucumber exist for Java, .NET, and several other platforms.

**Installation**

Cucumber is installed with Ruby's package manager, RubyGems. Assuming you already have a current version of Ruby (1.8.7 or 1.9.3 as of this writing), to install Cucumber simply open a command window and run

gem install cucumber

This will install Cucumber along with its dependencies.

Note: If you intend to use Cucumber under Ruby on Rails 3.x, you'll want to install Cucumber using Bundler instead.

**Potential Drawbacks**

**Extra overhead** - Compared to writing tests in a general-purpose language like Ruby, Cucumber introduces extra overhead. If you're not going to have the BDD-style interactions, if you're not going to have non-programmers read scenarios, and if you're disciplined enough to use domain language in your tests, you may prefer to write functional tests in test/unit or rspec.

**Regular expressions**- I consider regular expressions to be a strong point of Cucumber, but many people, even developers, are uncomfortable with them. If you use Cucumber, you'll find it hard to stay away from regular expressions, so this may be a reason to choose a different tool. To overcome this, I wrote an article and cheat sheet highlighting the most useful subset of regular expressions for Cucumber users.

So, what do I like about Cucumber?

**Separation of examples and automation** - Cucumber's unobtrusive automation means product people are more likely to engage with features and scenarios than if code were mixed in. Whether or not product people actually write scenarios, they should be able to read, verify, and adjust them.

**Some structure but not too much** - The Given-When-Then syntax of Cucumber scenarios imposes some structure on scenarios while leaving plenty of room for teams to grow their own language to describe their system. Similarly, the use of regular expressions for mapping between steps and step definitions leaves you room for flexibility in language but naturally limits how much flexibility you employ, lest the regular expressions become unreadable.

**Pressure to grow a ubiquitous language** - One of the apparent weaknesses of Cucumber as a test automation tool is that step definitions are all global. This quickly reveals, via ambiguous step matches, whether you use the same words to mean more than one thing in your domain. This pressure, plus the precision of specification by concrete examples, helps a team grow a precise language to express their domain.

**Just enough syntax** - The Gherkin language walks a fine line between natural language and a programming language. The Background and Scenario Outline features support simple refactoring to remove excess duplication. But more complex structures such as procedure calls and includes are left out because they would hurt readability for non-programmers.

Installation

Cucumber for Ruby is a ruby gem, and can be installed from the command line:

gem install cucumber

If you are using [Bundler](http://gembundler.com/), just add it to your Gemfile:

group :test do

gem 'cucumber'

end

Then initialize a features directory:

cucumber --init

Running

To see the full list of options:

cucumber --help

Otherwise, to run all features:

cucumber

# Ruby on Rails

[Cucumber-Rails](https://github.com/cucumber/cucumber-rails) is a generator that generates and modifies files in a Rails project so it can be used with Cucumber.

## Installation

Before you can use the generator, add the gem to your project's Gemfile as follows:

group :test, :development do

gem 'cucumber-rails', :require => false

# database\_cleaner is not required, but highly recommended

gem 'database\_cleaner'

end

Install the gem:

bundle install

Run the generator:

rails generate cucumber:install

## Running

With Rake:

rake cucumber

Without Rake:

[bundle exec] cucumber

Describe behavior in plain text.

Write step definition in ruby.

# BLOG

## Let's talk about testing

# Web testing with Cucumber

### Test driven development

The word “driven” here refers to the way of writing the tests first, then the code to make the tests pass. Test driven development is more of a developer oriented approach, a more low level way of doing things.

### Behaviour driven development

In BDD on the other hand, the tests are described in a natural language, which makes the tests more accessible to people outside of development or testing teams. It can describe the functionality as specifications do.

### Feature

Here is an example of a simple feature description from a Cucumber feature file:

Feature: Website Navigation

In order to navigate website pages

I need to be able to click on menu item

The feature is an explanation of what we want to accomplish. It doesn’t really do much by itself but works are a specification for the system and the test.

### Scenario

Now this feature needs to be tested with several scenarios. For example, user can be on a different page in the beginning of the test or the user can click on a different link. One scenario is shown below. It describes an operation, clicking on a menu item on our website:

Scenario: Going to page Services

Given that I am on spriteCloud Home

When I click on link Services

Then the page title should be "spriteCloud - Services"

### Step definitions

This is the part that we need to explain to Cucumber so that it knows what to do. In this case, we are using Watir with the Watir-webdriver gem. We also use RSpec gem to be able to use the should operation.

|  |
| --- |
| **require** 'spec'  **require** "watir-webdriver" |

Then we define some constants, one for the site address and another one for referring to the browser. Note the syntax for starting an instance of Firefox with webdriver.