Developer Testing-2

Chantri Polprasert

Outline

- Minitest
- Statistical Analysis Tools
- Behaviour-driven development
- Conclusions

Minitest

What is Minitest

- A lightweight, built-in testing framework included with Ruby.
- Comes automatically when you install Rails (no extra gem needed).
- Provides tools for unit tests, integration tests, system tests, and performance tests.
- Fast, simple, and works out of the box — great for both small and large apps.

Key Features

- Part of Ruby's standard library (no setup required).
- Supports TDD (Test-Driven Development) and BDD (Behavior-Driven Development) styles.
- Provides assertions (assert_equal, assert_nil, etc.) to verify behavior.
- Integrates seamlessly with Rails generators (rails generate model, rails generate scaffold).
- Automatically organizes tests in the /test directory.
- Produces test reports showing passes, failures, and errors.

Minitest

Types of Tests in Rails (via Minitest)

- Model tests → Validate logic, associations, and validations.
- Controller tests → Verify HTTP responses and redirects.
- System tests → Simulate user interaction (via Capybara).
- Integration tests → Test how components work together.

test_helper.rb

- A setup file that runs before every Minitest test in your Rails app.
- It configures the testing environment, loads Rails, and includes shared settings or libraries (like SimpleCov).
- All tests automatically include it with: require “test_helper”

```
ENV["RAILS_ENV"] ||= "test"
require_relative "../config/environment"
require "rails/test_help"

module ActiveSupport
  class TestCase
    # Run tests in parallel with specifie
    parallelize(workers: :number_of_proce

    # Setup all fixtures in test/fixtures
    fixtures :all

    # Add more helper methods to be used
  end
end
require "simplecov"
SimpleCov.start "rails" #Start tracking
```

Outline

- Minitest
- Statistical Analysis Tools
- Behaviour-driven development
- Conclusions

Statistical Analysis Tools: Static analysis

- It will be easier to maintain a code base if it is fully covered by a regression test suite.
- However, what if developers get tests to pass using bad programming practices?
- What if the code passes the tests but is so complex that it is hard for anyone to understand and maintain?
- Static analysis tools run a variety of algorithms on your source code to identify problems that may need fixing.
- Examples: method complexity, too much reference to fields of other objects, using deprecated style...

Purpose of Statistical & Analysis Tools in Software Engineering

- Measure code quality, test effectiveness, and software maintainability.
- Support data-driven decision-making in development, QA, and DevOps.
- Detect anomalies, defects, and performance issues early in the lifecycle.
- Provide quantitative feedback for continuous improvement (CI/CD, Agile, DevOps).
- Enable empirical studies on productivity, reliability, and team behavior.

Categories of Analysis in Practice

- **Code Quality Analysis:** evaluates complexity, duplication, smells, and maintainability.
- **Test & Coverage Analysis:** measures effectiveness of test suites and code coverage.
- **Performance & Profiling Analysis:** identifies runtime bottlenecks and resource usage.
- **Security & Dependency Analysis:** detects vulnerabilities and unsafe libraries.
- **Statistical & Empirical Analysis:** uses data (e.g., commits, bugs, metrics) to find trends and correlations.

Common Tools for Software Metrics & Code Analysis

Category	Tools	Description
Code Quality	**RubyCritic**, **Reek**, **Flay**, **Flog**, **Rails Best Practices**	Analyze code smells, duplication, and complexity in Ruby/Rails projects.
Test Coverage	**SimpleCov**, **RSpec Stats**, **Minitest Reporters**	Quantify coverage percentage and test suite reliability.
Security Analysis	**Brakeman**, **Bundler-Audit**, **Hakiri**, **Dependabot**	Scan for vulnerabilities and outdated dependencies.
Performance Profiling	**New Relic**, **Scout**, **Datadog**, **Rack Mini Profiler**	Measure request latency, memory leaks, and DB query performance.
Code Review Platforms	**SonarQube**, **CodeClimate**, **Codacy**	Centralized dashboards for maintainability, duplication, and technical debt.
Project Analytics	**GitHub Insights**, **GitLab CI Analytics**, **Jira Reports**	Track commits, issues, and velocity trends for team productivity.

Statistical Analysis Tools: RubyCritic

- There are excellent analysis tools for Ruby, many of which are included in the rubycritic gem:
- RubyCritic provides visual reports highlighting code smells, code structure, ease of testing, and test coverage in your Ruby application.
    - **Reek**: gem for detecting **code smell**
    - **Flay**: gem for Ruby code **duplication**
    - **Flog**: gem for code **complexity**
    - **Churn**: Tracks how often files change—high churn + high complexity = risky code.
- In Gemfile, under test environment, type 'gem "rubycritic", require: false', save and run bundle install
- Once done, run 'bundle exec rubycritic' to execute rubycritic and view the report:

```
group :test do
  # Use system testing [https://gui
  gem "capybara"
  gem "selenium-webdriver"
  gem "simplecov", require: false
  gem "rubycritic", require: false
end
```

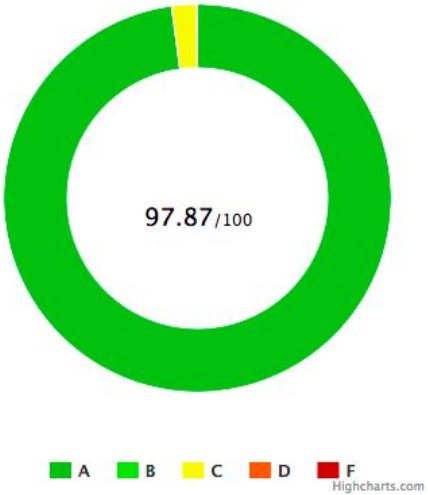
RUBYCRITIC

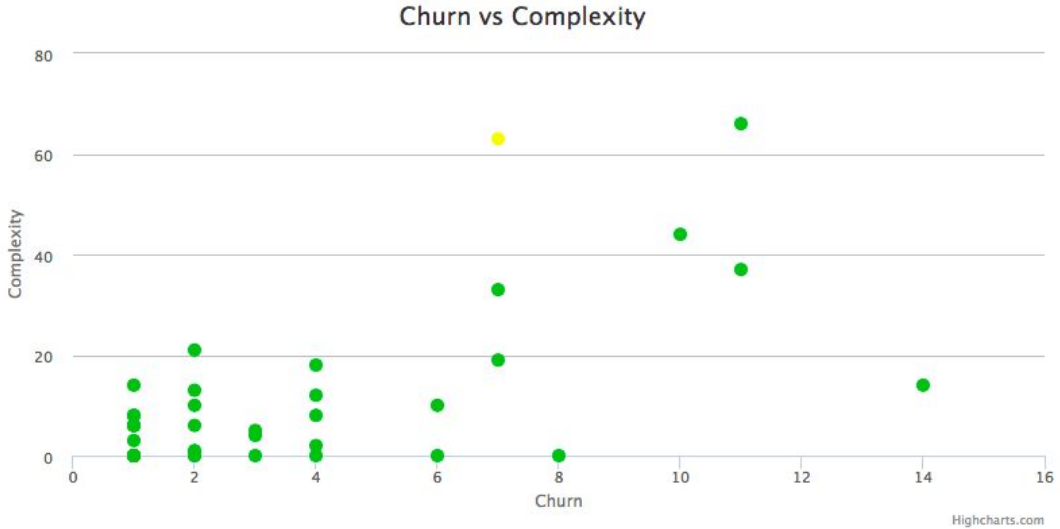
Overview

Code

Smells

Overview



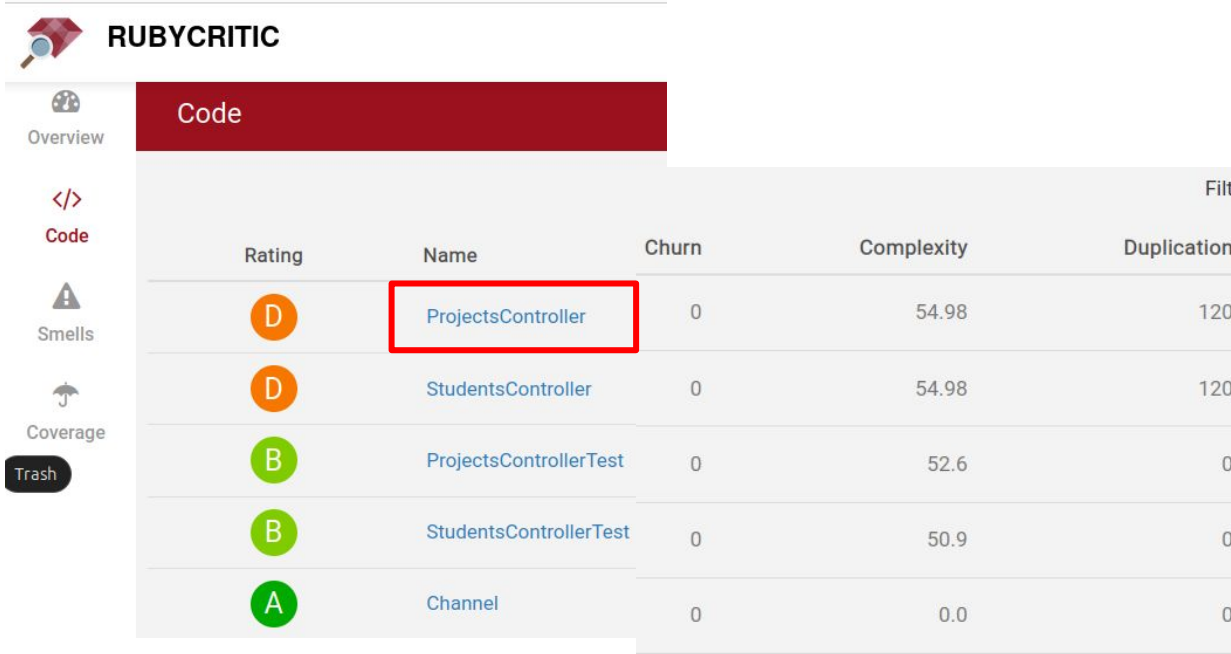


Summary

A	46 files	145 churns	54 smells
B	0 files	0 churns	0 smells
C	1 files	7 churns	2 smells
D	0 files	0 churns	0 smells
F	0 files	0 churns	0 smells

Code metric

Show a score for each class, including indicators for churn, complexity, duplication, and smells.



Code metric

The line of code where an issue is found will be highlighted

- Assumes too much
- No comment
- Duplicate code

RUBYCRITIC

Overview
Code
Smells
Coverage

app/controllers / projects_controller.rb

D

70 lines of codes
9 methods
6.1 complexity/method
0 churn
54.98 complexity
120 duplications

```
1.  class ProjectsController < ApplicationController
      ⚠ InstanceVariableAssumption
      ProjectsController assumes too much for instance variable '@project'
      ⚠ IrresponsibleModule
      ProjectsController has no descriptive comment
2.    before_action :set_project, only: %i[ show edit update destroy ]
3.
4.    # GET /projects or /projects.json
5.    def index
6.      @projects = Project.all
7.    end
8.
9.    # GET /projects/1 or /projects/1.json
10.   def show
11.   end
12.
13.   # GET /projects/new
14.   def new
15.     @project = Project.new
16.   end
```

```
test "should destroy project" do
  assert_difference("Project.count", -1) do
    @project.students.each do |s|
  ⚠ UncommunicativeVariableName
  ProjectsControllerTest has the variable name 's'
      s.delete
    end
```

```
# POST /projects or /projects.json
def create
  ⚠ DuplicateCode
  Similar code found in 2 nodes    Locations: 0 1
  ⚠ TooManyStatements
  ProjectsController#create has approx 10 statements
    @project = Project.new(project_params)
```

Integration Examples

- Continuous Integration (CI):
  - Add RubyCritic to your CI (GitHub Actions, GitLab CI, Jenkins) to fail builds when maintainability drops.
- Code Review:
  - Share HTML reports with your team for refactoring discussions.
- Automation:
  - Combine with SimpleCov to track both quality and coverage trends over time.

Limitations

- Static analysis only (doesn’t analyze runtime behavior).
- May flag false positives for certain metaprogramming-heavy code.
- Report size can be large for big projects.

SimpleCov

What is SimpleCov

- A code coverage analysis tool for Ruby and Rails applications.
- Measures how much of your code is executed when running tests.
- Helps ensure your tests actually cover the important parts of your app.

Basic setup:
In test/test_helper.rb

```
require "simplecov"
SimpleCov.start "rails" #Start tracking coverage
```

Then, run rails test

Report: open coverage/index.html

Key Features

- Tracks which files, classes, and lines are covered by tests.
- Generates HTML reports (green for covered, red for missed).
- Supports RSpec, Minitest, Cucumber, and other Ruby test frameworks.
- Provides percentage metrics for total and per-file coverage.
- Integrates easily into CI pipelines e.g. GitLab

SimpleCov: coverage/index.html

All Files (75.82% covered at 0.91 hits/line)

13 files in total.
91 relevant lines, **69** lines covered and **22** lines missed. (75.82%)

File	% covered
app/channels/application_cable/channel.rb	0.00 %
app/channels/application_cable/connection.rb	0.00 %
app/jobs/application_job.rb	0.00 %
app/mailers/application_mailer.rb	0.00 %
app/controllers/projects_controller.rb	87.88 %
app/controllers/students_controller.rb	87.88 %
app/controllers/application_controller.rb	100.00 %
app/helpers/application_helper.rb	100.00 %
app/helpers/projects_helper.rb	100.00 %
app/helpers/students_helper.rb	100.00 %
app/models/application_record.rb	100.00 %
app/models/project.rb	100.00 %
app/models/student.rb	100.00 %

Showing 1 to 13 of 13 entries

app/controllers/projects_controller.rb
87.88% lines covered
33 relevant lines. 29 lines covered and 4 lines missed.

```
1.  #Project controller for project entities.
2.  class ProjectsController < ApplicationController
3.    before_action :set_project, only: %i[ show edit update destroy ]
4.
5.    # GET /projects or /projects.json
6.    def index
7.      @projects = Project.all
8.    end
9.
10.   # GET /projects/1 or /projects/1.json
11.   def show
12.   end
13.
14.   # GET /projects/new
15.   def new
16.     @project = Project.new
17.   end
18.
19.   # GET /projects/1/edit
20.   def edit
21.   end
22.
23.   # POST /projects or /projects.json
24.   def create
25.     @project = Project.new(project_params)
26.
27.     respond_to do |format|
28.       if @project.save
29.         format.html { redirect_to @project, notice: "Project was successfully created." }
30.         format.json { render :show, status: :created, location: @project }
31.       else
32.         format.html { render :new, status: :unprocessable_entity }
33.         format.json { render json: @project.errors, status: :unprocessable_entity }
34.       end
35.     end
36.   end
37.
```

Outline

- Minitest
- Statistical Analysis Tools
- Behaviour-driven development
- Conclusions

Behaviour-driven development: Introduction

- Best practices in TDD (test-driven development) are evolving.
- A recent trend is behavior-driven development (BDD).
- BDD aims to take TDD closer to the business stakeholders.
- An agile software development methodology that emphasizes collaboration among developers, testers, and non-technical stakeholders to ensure that the software meets business needs.
- Focus on defining requirements in terms of desired behaviors and user stories, fostering clearer communication and alignment across teams.

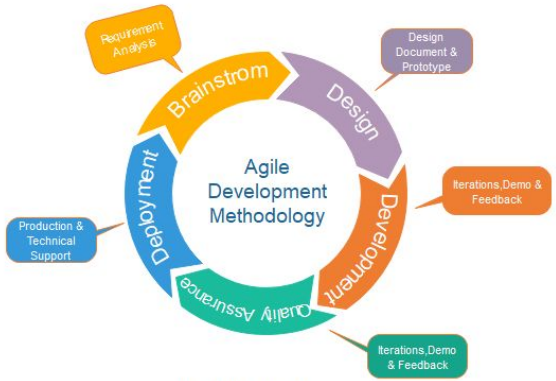
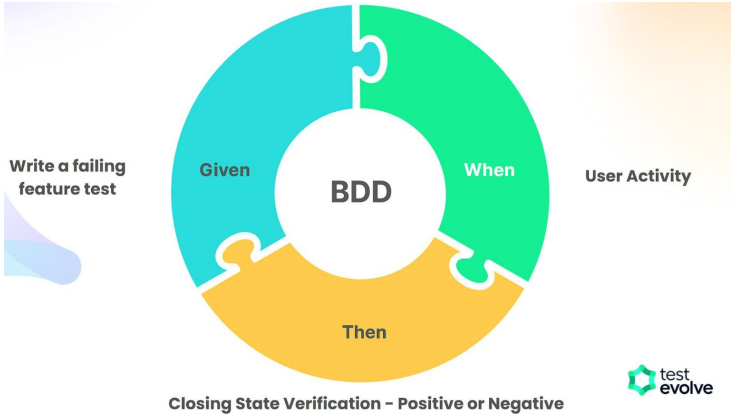


Fig. Agile Model

BDD in Agile

- Tests are written in plain English, and not in a programming language.
  - **Context/Given**: The starting state of your use case
  - **Event/When**: The action performed by the user in your use case
  - **Outcome/Then**: The expected results for your use case
- Example: In a music streaming app, you want the app to return the exact song that a user searches for. A BDD test would look like this:
  - **Context**: Given you're on a search page
  - **Event**: Type the correct name of the song you're searching
  - **Outcome**: Return the song that you searched for along with options to play online or download



https://www.testevolve.com/blog/behavior-driven-development-bdd

BDD Advantages

- **Improved Communication**: By using plain language to describe scenarios, BDD fosters better understanding among all team members, reducing misinterpretations and ensuring everyone is aligned on project goals25.
- **Early Defect Detection**: Since BDD focuses on defining behaviors upfront, it helps identify ambiguities and missing requirements early in the development process, thereby minimizing costly rework.
- **User-Centric Development**: BDD emphasizes the user's perspective by focusing on how users interact with the software. This ensures that development efforts are closely aligned with user needs and priorities.

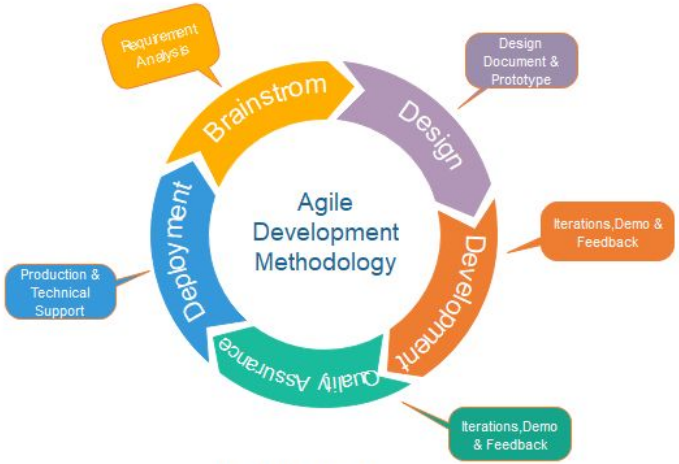


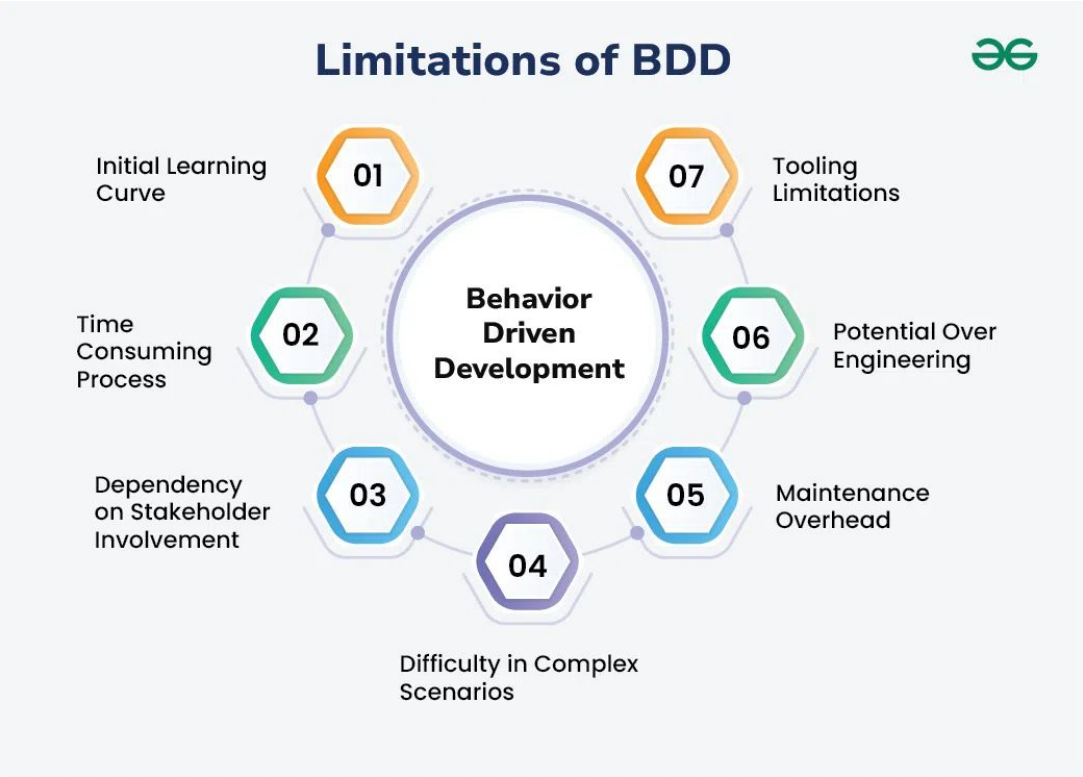
Fig. Agile Model

BDD vs TDD

Aspect	BDD	TDD
Focus	Desired behaviors and outcomes	Code correctness
Stakeholder Involvement	High (includes business stakeholders)	Primarily developers
Language	Natural language (e.g., Gherkin)	Programming language
Documentation	Improved documentation*	Separate test cases

BDD scenarios serve as executable documentation for the system's behavior. This documentation remains up-to-date as tests are regularly executed, providing a living specification of the software.

BDD Limitations



BDD Tools

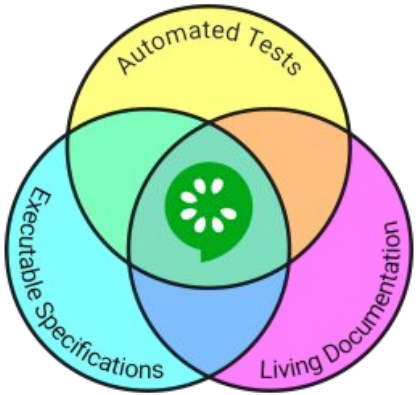
- Cucumber: Support various programming languages (Ruby, Java, Javascript…). Write executable specifications in plaintext using **Gherkin** syntax
- Behave: Python BDD
- RSpec: BDD framework for Ruby
- Not covered in Minitest but can be installed in Rails

```
# Comment
@tag
Feature: Eating too many cucumbers may not be good for you

  Eating too much of anything may not be good for you.

  Scenario: Eating a few is no problem
    Given Alice is hungry
    When she eats 3 cucumbers
    Then she will be full
```

BDD Cucumber

```
Scenario: Breaker guesses a word
  Given the Maker has chosen a word
  When the Breaker makes a guess
  Then the Maker is asked to score
```

- A tool used in BDD that facilitates the creation and execution of automated acceptance tests.
- Allows teams to write specifications in a human-readable format, bridging the gap between technical and non-technical stakeholders.
- Read executable specifications written in plain text and validates that the software does what those specifications say.
- The specifications consists of multiple examples, or scenarios.
- Each scenario is a list of steps for Cucumber to work through. Cucumber verifies that the software conforms with the specification and generates a report indicating ✅ success or ❌ failure for each scenario.
- Gherkin is a set of grammar rules that makes plain text structured enough for Cucumber to understand.



BDD Cucumber

Example Cucumber acceptance test for the Course Information System:

```
Feature: Browsing

  Scenario: Get course info

    Anyone should be able to get information about a course.

    Given I want information about a course
    When I visit the main page
    Then I should see a link for the course I want information about
    When I visit the course page
    Then I should see the information for the course
```

BDD Cucumber

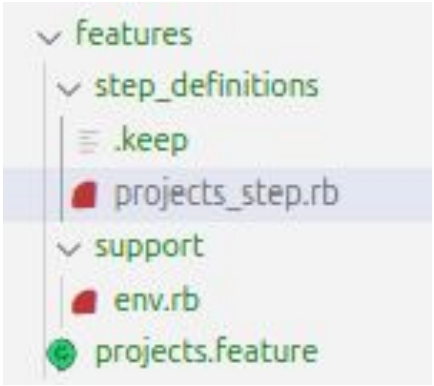
```
Feature: Browsing

  Scenario: Get course info                          # features/browsing.feature:3

    Anyone should be able to get information about a course.
    Given I want information about a course          # features/browsing.feature:7
      Undefined step "I want information about a course" (Cucumber::Undefined)
      features/browsing.feature:7:in 'Given I want information about a course'
    When I visit the main page                       # features/browsing.feature:8
      Undefined step: "I visit the main page" (Cucumber::Undefined)
      features/browsing.feature:8:in 'When I visit the main page'
```

BDD Cucumber

```
Then I should see a link for the course I want information about
                                                    # features/browsing.feature:9
  Undefined step: "I should see a link for the course I want information about'
  features/browsing.feature:9:in 'Then I should see a link for the course I wa
When I visit the course page                        # features/browsing.feature:10
  Undefined step: "I visit the course page" (Cucumber::Undefined)
  features/browsing.feature:10:in 'When I visit the course page'
Then I should see the information for the course
                                                    # features/browsing.feature:11
  Undefined step: "I should see the information for the course" (Cucumber::Unde
  features/browsing.feature:11:in 'Then I should see the information for the c

1 scenario (1 undefined)
5 steps (5 undefined)
0m0.005s
```

BDD Cucumber Installation

```
group :test do
  # Use system testing [https://guides.rubyonrails.org/te
  gem "capybara"
  gem "selenium-webdriver"
  gem "simplecov", require: false
  gem "rubycritic", require: false
  gem "cucumber-rails", require: false
  #database cleaner, not required but highly recommended
  gem "database_cleaner"
end
```

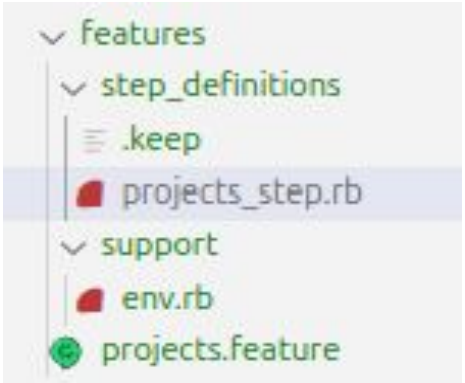


- Then, install by running: ‘bundle install’
- Install cucumber for rails:: ‘rails generate cucumber:install’
- Run ‘bundle exec cucumber’

```
Share your Cucumber Report with your team at https://reports.cucumber.io

Command line option:     --publish
Environment variable:    CUCUMBER_PUBLISH_ENABLED=true
cucumber.yml:            default: --publish

More information at https://cucumber.io/docs/cucumber/environment-variables/

To disable this message, specify CUCUMBER_PUBLISH_QUIET=true or use the
--publish-quiet option. You can also add this to your cucumber.yml:
default: --publish-quiet
```

Features in Cucumber

- **A plain-text description** of a behavior or functionality of your application.
- Represents a **user-facing** capability — what the system should do, not how it does it.
- Each feature is stored as a .**feature** file inside the features/ directory.
- Serves as **living documentation** — readable by developers, testers, and business stakeholders.
- Defines **scenarios** that describe expected behavior in natural language.
- Provides the link between business requirements and executable tests.



Basic Structure of a Feature File (*.feature)

```
Feature: User login
  In order to access my account
  As a registered user
  I want to log into the system

  Scenario: Successful login
    Given I am on the login page
    When I fill in "Email" with "user@example.com"
    And I fill in "Password" with "password"
    Then I should see "Welcome back!"
```

Keyword	Meaning
`Feature:`	Describes the functionality or module under test
`Scenario:`	Describes one example or use case of that feature
`Given`	Sets up the initial context
`When`	Describes an action or event
`Then`	Describes the expected outcome

Features → step_definitions

- Each step (Given, When, Then) connects to Ruby code called a step definition.
- Step definitions live in features/step_definitions/*.rb:

```ruby
Given("I am on the login page") do
  visit login_path
end

When("I fill in {string} with {string}") do |field, value|
  fill_in field, with: value
end

Then("I should see {string}") do |text|
  expect(page).to have_content(text)
end
```

FactoryBot

```
Given("there is a student named {string}") do |name|
  @student = create(:student, name: name)
end
```

- A fixture replacement library for Ruby and Rails.
- Used in RSpec, Minitest, and Cucumber to generate model instances.
- Easy and readable way to create data directly from Ruby step definitions:
- To set up, add factory_bot_rails in Gemfile in :development, :test group
- Install gem with ‘bundle install’
- Create features/support/‘factories.rb and add configuration

```
Given There is a project
And The project has students
```

```
group :development, :test do
  gem 'factory_bot_rails'
end
```

FactoryBot Key Features

- Defines factories (templates) for your models.
- Generates objects using methods like:
  - build(:user) → builds in memory (not saved)
  - create(:user) → builds and saves to DB
  - build_stubbed(:user) → simulates saved object (faster, not persisted)
  - attributes_for(:user) → returns a hash of attributes
- Supports associations between models (e.g., Project → Students).
- Allows sequences and traits for variation and reuse.
- Integrates automatically with Rails and ActiveRecord via factory_bot_rails.

FactoryBot Example

features/support/factories.rb

```ruby
FactoryBot.define do
  factory :user do
    name { "Alice" }
    email { "alice@example.com" }
    password { "password123" }
  end
end
```

features/step_definitions/project_steps.rb

```ruby
user = create(:user)
expect(user.name).to eq("Alice")
```

FactoryBot Association

Associate John (Student) to a factory :project

```
factory :project do
  title { "AI Research" }
end

factory :student do
  name { "John" }
  project   # automatically
end
```

Factory bot cheat sheet (https://devhints.io/factory_bot)

Factory Bot cheatsheet

Introduction

Factory Bot is a helper for writing factories for Ruby tests. It was previously known as Factory Girl. For older versions, use `FactoryGirl` instead of `FactoryBot`.

Factory Bot documentation
(rubydoc.info)

Getting started
(github.com)

Source code
(github.com)

Defining factories

```
FactoryBot.define do
  factory :user do
    first_name { 'John' }
    last_name  { 'Doe' }
    birthdate  { 21.years.ago }
    admin { false }

    sequence(:username) { |n| "user#{n}" }
  end
end
```

See: Defining factories

Using

Build a model

```
FactoryBot.build(:user)
```

Other ways

```
build(:user)            # → model (not saved)
create(:user)           # → model (saved)
attributes_for(:user)   # → hash
build_stubbed(:user)    # stubbed out attributes
```

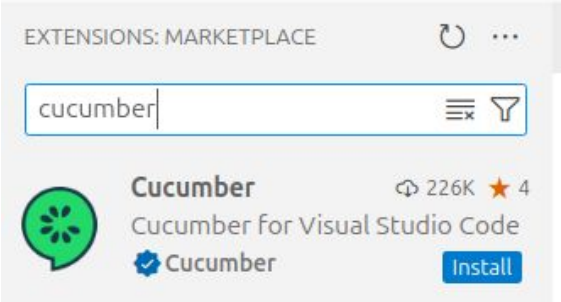
With options

```
build(:user, name: 'John')
```

Lists

BDD Cucumber: Create the first feature

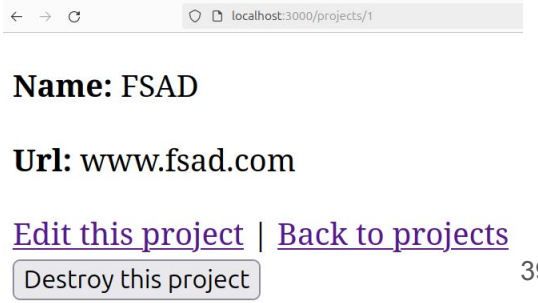
- In studentdb/features folder, create a new ‘projects.feature’ file
- Feature file describes behaviour and functionality of the software using Gherkin syntax



projects.feature

```
Feature: Projects

    Scenario: Get students info

    Anyone should be able to see the list of students working on a project

    Given There is a project
    And The project has students
    When I visit the projects page
    Then I should see the project in the list
    And I should see a show link for the project
    When I show the project details
    Then I should see a list of students on the project
```

- Run ‘bundle exec cucumber’



BDD Cucumber: Create the first feature

```
chantri@chantri-VMware-Virtual-Platform:~/studentdb2$ bundle exec cucumber
Using the default profile...
Feature: Projects

  Scenario: Get students info                                  # features/projects.feature:3
       Anyone should be able to see the list of students working on a project
    Given There is a project                                   # features/projects.feature:7
      Undefined step: "There is a project" (Cucumber::Core::Test::Result::Undefined)
      features/projects.feature:7:in `There is a project'
    And The project has students                               # features/projects.feature:8
      Undefined step: "The project has students" (Cucumber::Core::Test::Result::Undefined)
      features/projects.feature:8:in `The project has students'
    When I visit the projects page                             # features/projects.feature:9
      Undefined step: "I visit the projects page" (Cucumber::Core::Test::Result::Undefined)
      features/projects.feature:9:in `I visit the projects page'
```

BDD Cucumber: Undefined steps

Projects_steps.rb in /features/step_definitions

```ruby
Given('There is a project') do
  pending # Write code here that turns the phrase above into concrete actions
end

Given('The project has students') do
  pending # Write code here that turns the phrase above into concrete actions
end

When('I visit the projects page') do
  pending # Write code here that turns the phrase above into concrete actions
end

Then('I should see the project in the list') do
  pending # Write code here that turns the phrase above into concrete actions
end

Then('I should see a show link for the project') do
  pending # Write code here that turns the phrase above into concrete actions
end

When('I show the project details') do
  pending # Write code here that turns the phrase above into concrete actions
end

Then('I should see a list of students on the project') do
  pending # Write code here that turns the phrase above into concrete actions
end
```

BDD Cucumber: Step#1: There is a project

features/support/factories.rb

```ruby
FactoryBot.define do
    factory :project do
      name { 'FSAD' }
      url  { 'www.fsad.com' }
    end
  end
```

features/step_definitions/project_steps.rb

```ruby
Given('There is a project') do
    @project = FactoryBot.create(:project)
    #pending # Write code here that turns
end
```

```
Scenario: Get students info                                  # features/projects.feature:3
      Anyone should be able to see the list of students working on a project
  Given There is a project                                   # features/step_definitions/pr
  And The project has students                               # features/step_definitions/pr
    TODO (Cucumber::Pending)
    ./features/step_definitions/projects_step.rb:7:in `"The project has students"'
    features/projects.feature:8:in `The project has students'
  When I visit the projects page                             # features/step_definitions/pr
  Then I should see the project in the list                  # features/step_definitions/pr
  And I should see a show link for the project               # features/step_definitions/pr
  When I show the project details                            # features/step_definitions/pr
  Then I should see a list of students on the project        # features/step_definitions/pr


1 scenario (1 pending)
7 steps (5 skipped, 1 pending, 1 passed)
```

BDD Cucumber: Step#2: The project has students

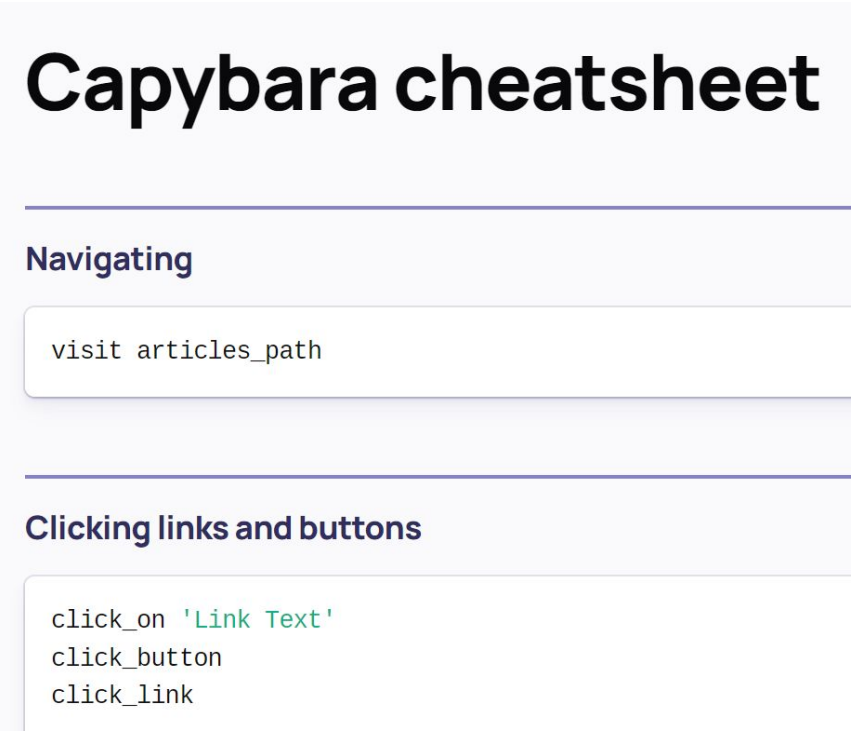
features/step_definitions/project_steps.rb

```
Given('The project has students') do
    s1 = FactoryBot.create(:student, project: @project)
    s2 = FactoryBot.create(:student, project: @project)
    #pending # Write code here that turns the phrase ab
end
```

features/support/factories.rb

```
#This will create students in the second step
factory :student do
    sequence(:name) { |n| "Student #{n}" }
    sequence(:studentid) { |n| (111+n).to_s}
end
```

BDD Cucumber: Step#3: ‘I visit the projects page’



Capybara cheatsheet

Navigating

```
visit articles_path
```

Clicking links and buttons

```
click_on 'Link Text'
click_button
click_link
```

Capybara

- A **web-automation and acceptance-testing library** for Ruby.
- Simulates how a **real user interacts** with your web application in a browser.
- Works with Rails
- Lets you write high-level tests that navigate pages, click links, fill forms, and verify visible content.
-

Why Capybara exist?

- Unit tests (like Minitest or RSpec model tests) only check logic.
- Capybara tests check **real user behavior** — visiting URLs, clicking, submitting forms.
- It ensures that the app behaves correctly **end-to-end** from the user's perspective.

Capybara methods

Method	Purpose
`visit('/projects')`	Opens a URL
`click_link('Show')`	Clicks a link
`fill_in('Name', with: 'AI Project')`	Fills a form field
`click_button('Create Project')`	Submits a form
`expect(page).to have_content('Project created')`	Checks visible text
`within('form') { ... }`	Scopes interactions to a section of the page

Relationship Between Capybara and Cucumber

Role	Description
Cucumber	Provides the **BDD framework** and natural-language “Given / When / Then” syntax.
Capybara	Executes the **browser actions** described in those steps.
Integration	When you run `bundle exec cucumber`, Cucumber parses `.feature` steps → calls Ruby step definitions → those definitions use Capybara to drive the app.

Example

projects.feature

```gherkin
When I visit the project page
Then I should see "AI Research"
```

Step definitions

```ruby
When("I visit the project page") do
  visit projects_path
end

Then("I should see {string}") do |text|
  expect(page).to have_content(text)
end
```

BDD Cucumber: Step#4: ‘I should see the project in the list’

Selectors

```
expect(page).to have_button('Save')
```

```
expect(page).to have_button('#submit')
```

```
expect(page).to have_button('//[@id="submit"]')
```

The `selector` arguments can be text, CSS selector, or XPath expression.

Conclusions

- BDD is a collaborative software development approach that focuses on the behavior of an application from the end-user's perspective.
- BDD scenarios are written in a simple format using "Given," "When," and "Then" to describe the context, action, and expected outcome.
- It encourages communication between developers, QA, and non-technical stakeholders, ensuring everyone has a shared understanding of requirements.
- Popular BDD frameworks include Cucumber, SpecFlow, and JBehave, which facilitate writing and automating tests.
- Benefits: BDD enhances test coverage, reduces misunderstandings in requirements, and ensures that the software aligns with user expectations.

References

- https://github.com/whitesmith/rubycritic?tab=readme-ov-file
- https://cucumber.io/
- https://github.com/thoughtbot/factory_bot_rails
- https://devhints.io/factory_bot
- https://github.com/thoughtbot/factory_bot/blob/main/GETTING_STARTED.md
- https://devhints.io/capybara
- https://github.com/whitesmith/rubycritic?tab=readme-ov-file