Ingeniería de Software

Testing

Juan Pablo Sandoval

¿Que es testing?

Software ***testing*** *es un proceso usado para evaluar la* ***correctitud****,* ***completitud*** *y la* ***calidad*** *de un programa de computador. Esto puede incluir un conjunto de actividades realizadas con el fin de encontrar errores en programas de manera que puedan ser corregidos antes de lanzar el producto a los usuarios finales.*

¿Por qué el Software Testing es importante?

Software Bugs pueden ser
costosos y incluso peligrosos

Pero, ¿qué causa los defectos?

Sabemos que las personas cometemos errores, somos falibles (contrario a in-falibles). Los errores o equivocaciones pueden venir de diferente lugar. Por ejemplo:

- *programadores: errores en el diseño, construcción, etc.*
- *usuarios finales: utilizan el software de forma “no esperada”.*

Clasificación Inicial

- ***Error**: acción humana que produce un resultado incorrecto.*
- ***Defecto**: presencia de una imperfección que puede ocasionar fallas.*
- ***Falla**: comportamiento observable incorrecto con respecto a los requisitos.*

***¿están relacionados?** Un error introduce un defecto en el software que se manifiesta a través de una falla en las pruebas.*

Pero que podría malir sal (lo primero que sale mal)

Varios factores pueden llegar a defectos o fallas:

- *errores en la especificación, diseño, implementación*
- *errores en el uso del sistem*
- *condiciones del medio ambiente*
- *daño intencional*
- *entre otros*

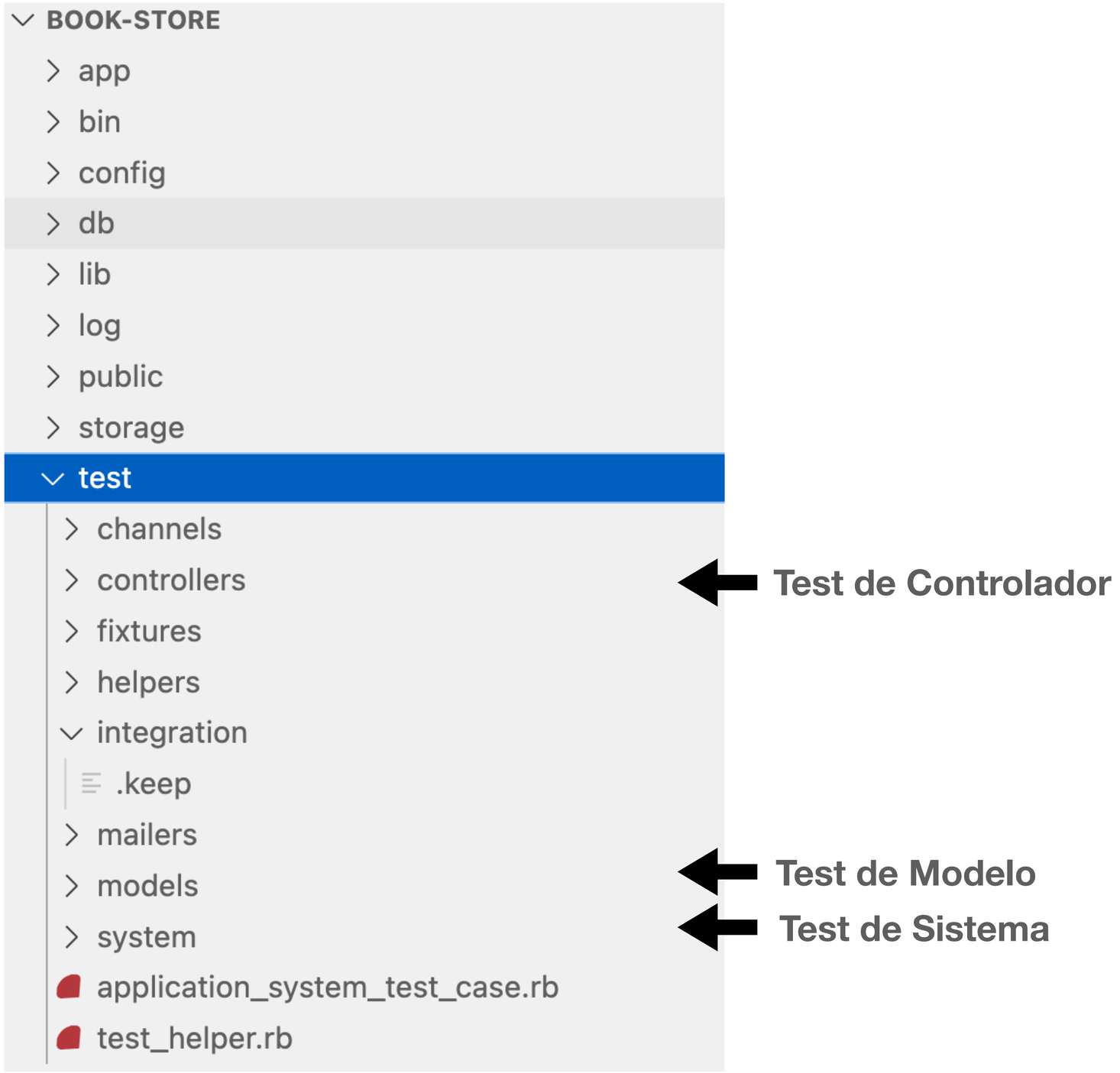
¿Por qué Testing es importante?

- *Podemos cometer errores!*
  - *Detectar los defectos de forma temprana facilita el desarrollo.*
  - *Reparar los defectos es mas barato si son encontrados mas antes.*
- *Queremos que el programa siga funcionando, si algo puede fallar, fallará!*
- *Queremos que el equipo de trabajo revise que todo sigue funcionando igual de forma rápida y confiable.*

Testing en Rails

Es posible realizar testing en diferentes niveles

- ***Unit Testing.*** *En Rails, trata de verificar si los clases del modelo que creamos en rails funciona correctamente en insolación (por si solas separadas del resto).*
- ***Integration Testing.*** *En rails, trata de verificar si los controladores creados en el proyecto están funcionando de forma correcta.*
- ***System Testing.*** *En rails, evalúa la funcionalidad del sistema como un todo, prácticamente se abre la página web y se la evalúa interactuando con ella. Se puede hacer manual y automática (i.e. con un bot).*


BOOK-STORE
app
bin
config
db
lib
log
public
storage
test
channels
controllers
fixtures
helpers
integration
.keep
mailers
models
system
application_system_test_case.rb
test_helper.rb
Test de Controlador
Test de Modelo
Test de Sistema

Ejemplo: Unit Test - Modelo

```ruby
class Book < ApplicationRecord
    validates :title, presence: true
    validates :author, presence: true
    validates :year, presence: true
end
```

Modelo

```ruby
class BookTest < ActiveSupport::TestCase
  test "should not save without title" do
    @book = Book.new(title:"",author:"Juan P.", year:2023)
    result = @book.save
    assert_not result,"saved the title without title"
  end
end
```

Test

Unit Test - Estructura

```ruby
class BookTest < ActiveSupport::TestCase
  test "should not save without title" do
    # inicialización, donde crean los objetos y datos necesarios
    @book = Book.new(title:"",author:"Juan P.", year:2023)

    # estimulo, donde se realiza la acción a evaluar
    result = @book.save

    # verificación, donde se evalúa si el resultado es el esperado
    assert_not result,"saved the book without title"
  end
end
```

Ejecutando el test

```
>rails test test/models/book_test.rb
Running 1 tests in a single process (parallelization
threshold is 50)
Run options: --seed 53778
# Running:
.
Finished in 0.008640s, 115.7407 runs/s, 115.7407
assertions/s.
1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
```

Algunas consideraciones

- Rails tiene 3 bases de datos, uno para pruebas, otro para producción y otro para desarrollo. (Ver archivo config/database.yml).
- Rails inicializara una base de datos dedicada para ejecutar cada prueba, de esta forma cada test se ejecutara de forma separada de forma aislada.

¿Qué otros asserts existen?

- *assert(test, [msg])*
- *assert_not(test, [msg])*
- *assert_equal (expected, actual, [msg])*
- *assert_not_same (expected, actual, [msg])*
- *assert_nil (obj, [msg])*
- *assert_not_nil (obj, [msg])*
- *assert_empty(obj, [msg])*

… entre muchos otros.

¿Qué es un fixture?

Fixture es una palabra “elegante” asociada a ejemplos de datos. En rails en particular nos ayuda a crear objetos del modelo fácilmente utilizando los mismos datos.

```yaml
jp1:
  author: Juan P. Sandoval
  title: Ingenieria de Software
  year: 2023

jp2:
  author: Juan P. Sandoval
  title: Testing
  year: 2022
```

app/test/fixtures/books.yml

Mismo Ejemplo pero usando fixtures

```ruby
class Book < ApplicationRecord
    validates :title, presence: true
    validates :author, presence: true
    validates :year, presence: true
end
```

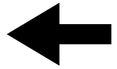
```ruby
class BookTest < ActiveSupport::TestCase
  test "should not save without title" do
    @book = books(:jp1)
    result = @book.save
    assert_not result,"saved the title without title"
  end
end
```

Ejemplo: Integration Test - Controller

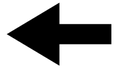
```
class BooksControllerTest <
ActionDispatch::IntegrationTest
  test "should get index" do
    get books_url
    assert_response :success
  end

  test "should get new" do
    get new_book_url
    assert_response :success
  end

end
```



Envia un request GET a la url "/books" y verifica que el response sea success.



Envia un request GET a la url "/books/new" y verifica que el response sea success.

Ejemplo: Integration Test - Controller

```
class BooksControllerTest < ActionDispatch::IntegrationTest
  test "should show book" do
    @book = Book.new(title:"Ing. Software",author:"Juan P.", year:2023)
    @book.save
    get book_url(@book)
    assert_response :success
  end

end
```

- *Crea un objeto libro en la base de datos*
- *Hace un GET request a "/book/id" , donde el id es el del libro recién creado.*
- *Verifica que el http response sea success.*

Ejemplo: Integration Test - Controller

```
test "should show book" do
    @book = Book.new(title:"Ing. Software",author:"Juan P.", year:2023)
    @book.save
    get "/books/#{@book.id}"
    assert_response :success
end
```

- *Uno puede armar la url manualmente si se confunde con los path helpers.*

Ejemplo 2: Integration Test - Controller

```
test "should create book" do
    assert_difference("Book.count") do
      post books_url, params:
           { book: { author: "Juan P.",
                     title: "Ing. Software",
                      year: 2022 } }
    end

    assert_redirected_to book_url(Book.last)
  end
```

Verifica que el contador incremento en uno

Verifica que se redirecciono a la url “books/id”, siendo el id el del ultimo libro

Ejemplo 3: Integration Test - Controller

```
test "should destroy book" do
    @book = Book.new(title:"Ing. Software",author:"Juan P.",
year:2023)
    @book.save
    assert_difference("Book.count", -1) do
      delete book_url(@book)
    end

    assert_redirected_to books_url
  end
```

Verifica que el contador decremento en uno

Verifica que se redirecciono a la url “books”, cuando se borra el controlador debería redireccionar a “/books”

Coverage

Instalar Simple COV

- Agregar Simple Cov a archivo Gemfile y ejecutar “bundle install”

```
gem 'simplecov', require: false, group: :test
```

- Agregar las siguientes instrucciones al archivo test/test_helper.rb (o spec_helper.rb)

```
require 'simplecov'
SimpleCov.start

# el contenido anterior al archivo debe ir aqui abajo
```

Resultado después de ejecutar los test

app/controllers/books_controller.rb

92.59% lines covered

27 relevant lines. 25 lines covered and 2 lines missed.

```
1.  class BooksController < ApplicationController
2.
3.    # GET /books
4.    def index
5.      @books = Book.all
6.
7.    end
8.
9.    # GET /books/1
10.   def show
11.     @book = Book.find(params[:id])
12.
```

Simple Con crea una carpeta coverage en tu proyecto con un conjuntos de html que muestra el resultado, para verlo abre el html en un browser.

Lineas de código no cubiertas

```
25.   # POST /books
26.   def create
27.     @book = Book.new(book_params)
28.     if @book.save
29.       logger.info "redirect"
30.       redirect_to book_url(@book), notice: "Book was successfully created."
31.     else
32.       render :new, status: :unprocessable_entity
33.     end
34.   end
```

Creamos un test para crear un libro de forma exitosa, pero no para ver que pasa si el libro no puede ser creado.

Set Up y Tear Down

Set up + Tear down

```
class BooksControllerTest < ActionDispatch::IntegrationTest
  #codigo ejecutado antes de cada test
  setup do

  end

  # codigo ejecutado despues de cada test
  teardown do
    # normalmente es buena idea resetear el cache
    Rails.cache.clear
  end

end
```

Mas información?

https://guides.rubyonrails.org/testing.html