T1A3- Terminal Application

The Journey of developing an Application

By Anthony B. Chung

Fuel Tracker

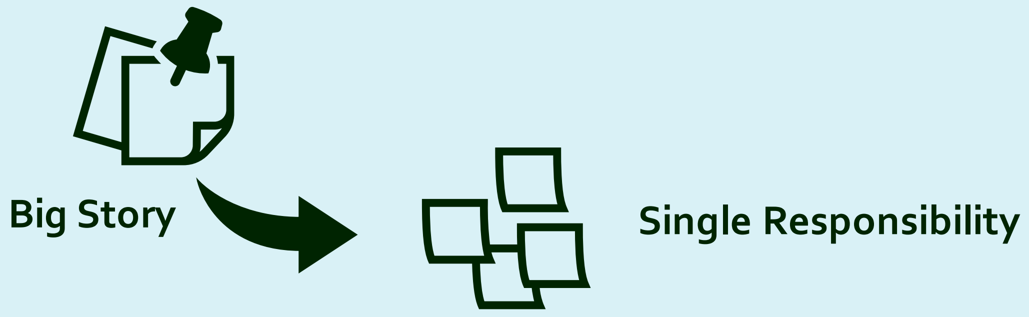
- Record Fuel Purchase

- Analyse Fuel usage
  - Cost per week or months.
  - Distance per week or months.
  - Litres per week or months.
  - Cost per distance.
  - Litre per distance.
  - Distance per litre

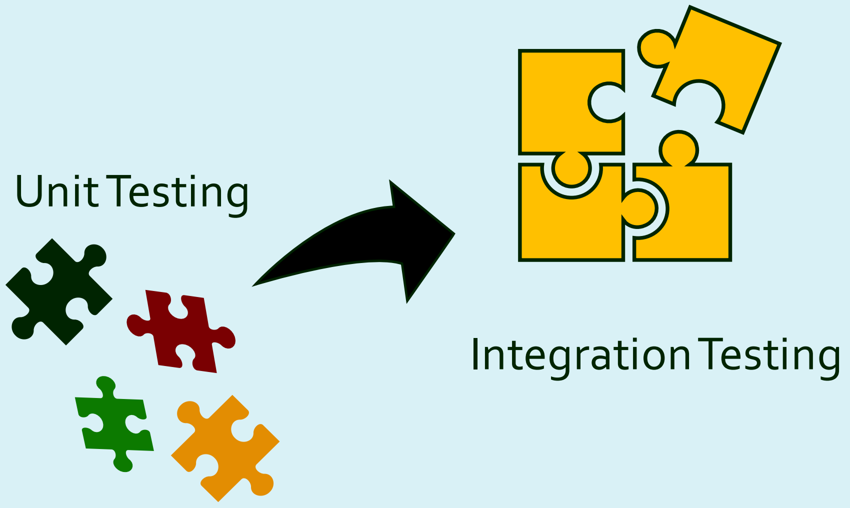
- Predict Cost per journey
  - Enter distance

Approach to the Project

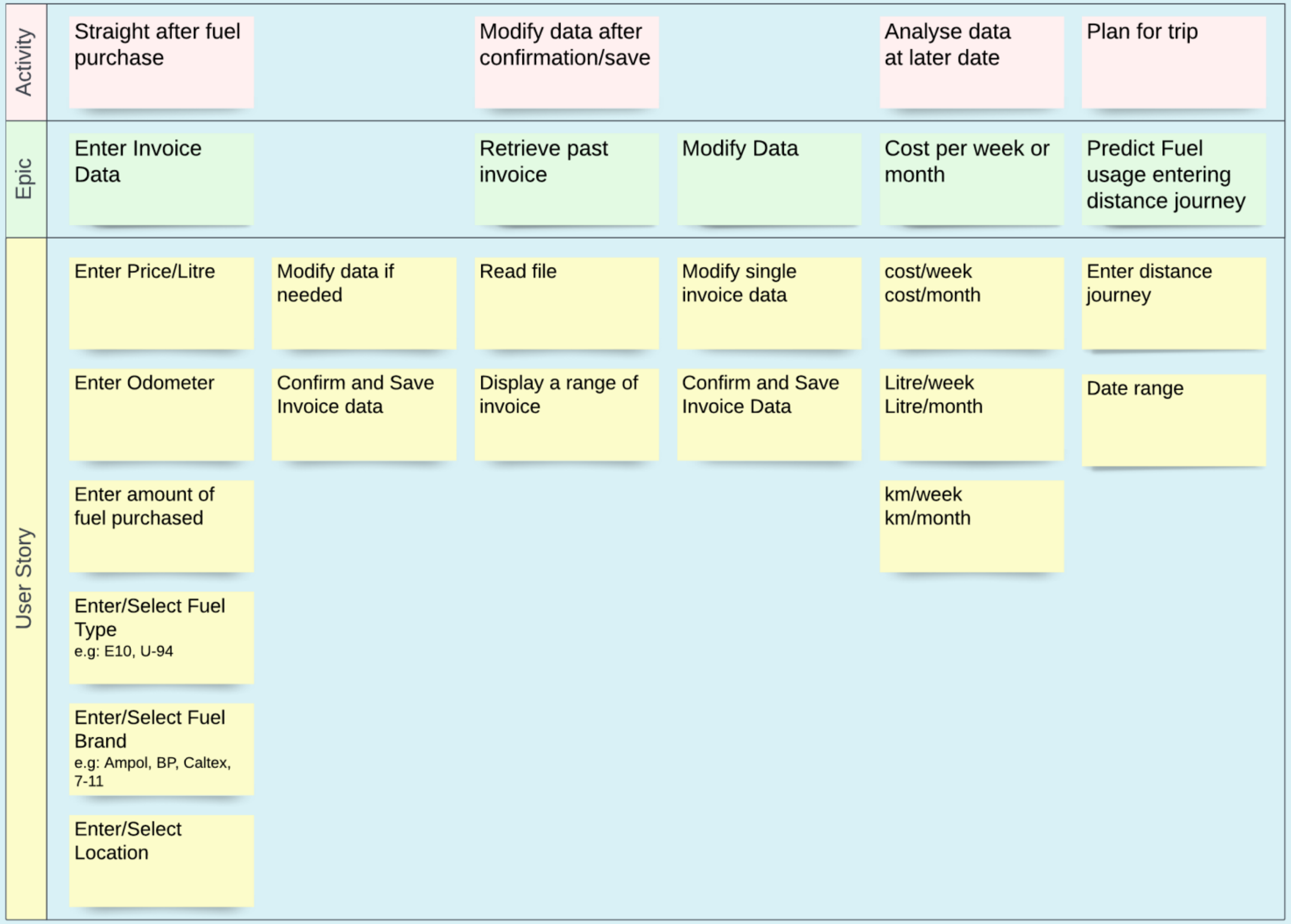
- Break down the problem
  - User Story Mapping



- Test Driven Development
  - RSpec-3
  - Unit Testing
  - Integration Testing



Initial User Story Map

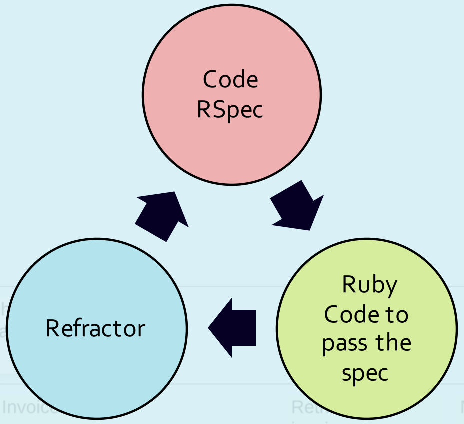


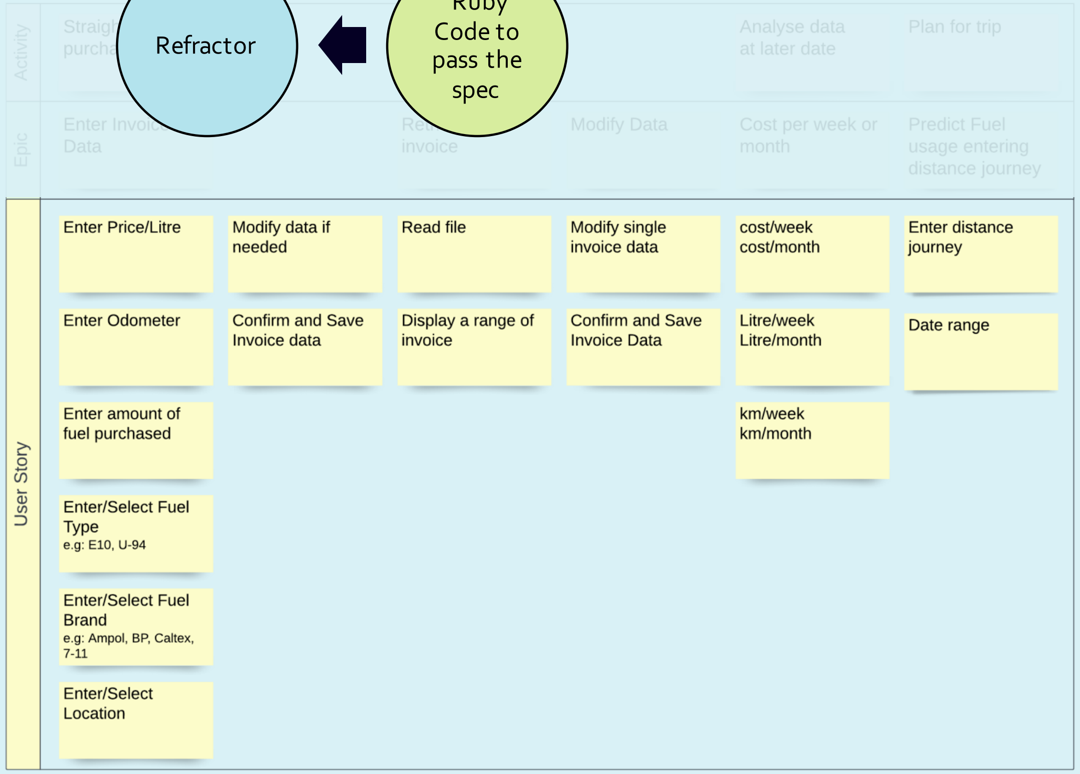
Activity	Straight after fuel purchase		Modify data after confirmation/save		Analyse data at later date	Plan for trip
Epic	Enter Invoice Data		Retrieve past invoice	Modify Data	Cost per week or month	Predict Fuel usage entering distance journey
User Story	Enter Price/Litre	Modify data if needed	Read file	Modify single invoice data	cost/week cost/month	Enter distance journey
	Enter Odometer	Confirm and Save Invoice data	Display a range of invoice	Confirm and Save Invoice Data	Litre/week Litre/month	Date range
	Enter amount of fuel purchased				km/week km/month	
	Enter/Select Fuel Type e.g: E10, U-94					
	Enter/Select Fuel Brand e.g: Ampol, BP, Caltex, 7-11					
	Enter/Select Location					

- User Story Map is **not finalised**
- It **changes** throughout the project

- From a user perspective.
  - Activities
    - General way the user will use the application
  - Epics
    - What the user will do during each activities.
  - User Story/Tasks
    - Break down the story into individual task.
    - Each task is independent to each other.

Predicting Project Timeline





Activity	Straigh purcha				Analyse data at later date	Plan for trip
Epic	Enter Invoic Data		Re invoice	Modify Data	Cost per week or month	Predict Fuel usage entering distance journey
User Story	Enter Price/Litre	Modify data if needed	Read file	Modify single invoice data	cost/week cost/month	Enter distance journey
	Enter Odometer	Confirm and Save Invoice data	Display a range of invoice	Confirm and Save Invoice Data	Litre/week Litre/month	Date range
	Enter amount of fuel purchased				km/week km/month	
	Enter/Select Fuel Type e.g: E10, U-94					
	Enter/Select Fuel Brand e.g: Ampol, BP, Caltex, 7-11					
	Enter/Select Location					

Test Driven Development

- Robert C. Martin - Advanced TDD: The Transformation Priority Premise (https://vimeo.com/97516288)
- He did an example in 5 to 10 mins.

- 60 mins per story (yellow card)
- 17 individual stories

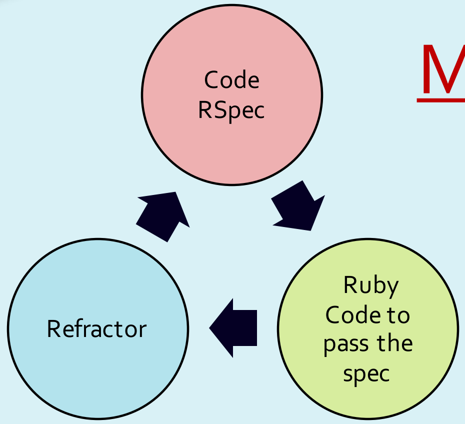
Total Time = 60 x 17

=17 hours

First Task

Confirm and Save Invoice data

- Timed first task
- To get an accurate timeline



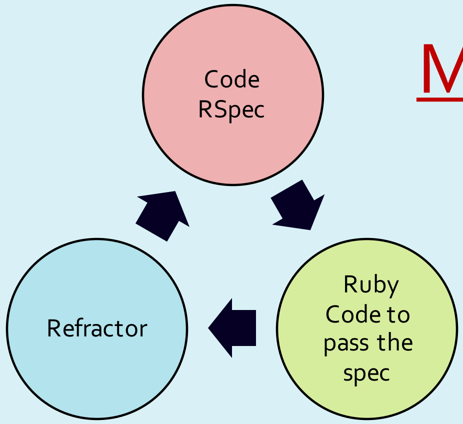
MUST USE TDD !!!!!

- RSpec coding was the Road Block
  - Coding for the App
  - Coding for RSpec

- Total time :3 hours
- 3 hours x 17
  = 51 hours

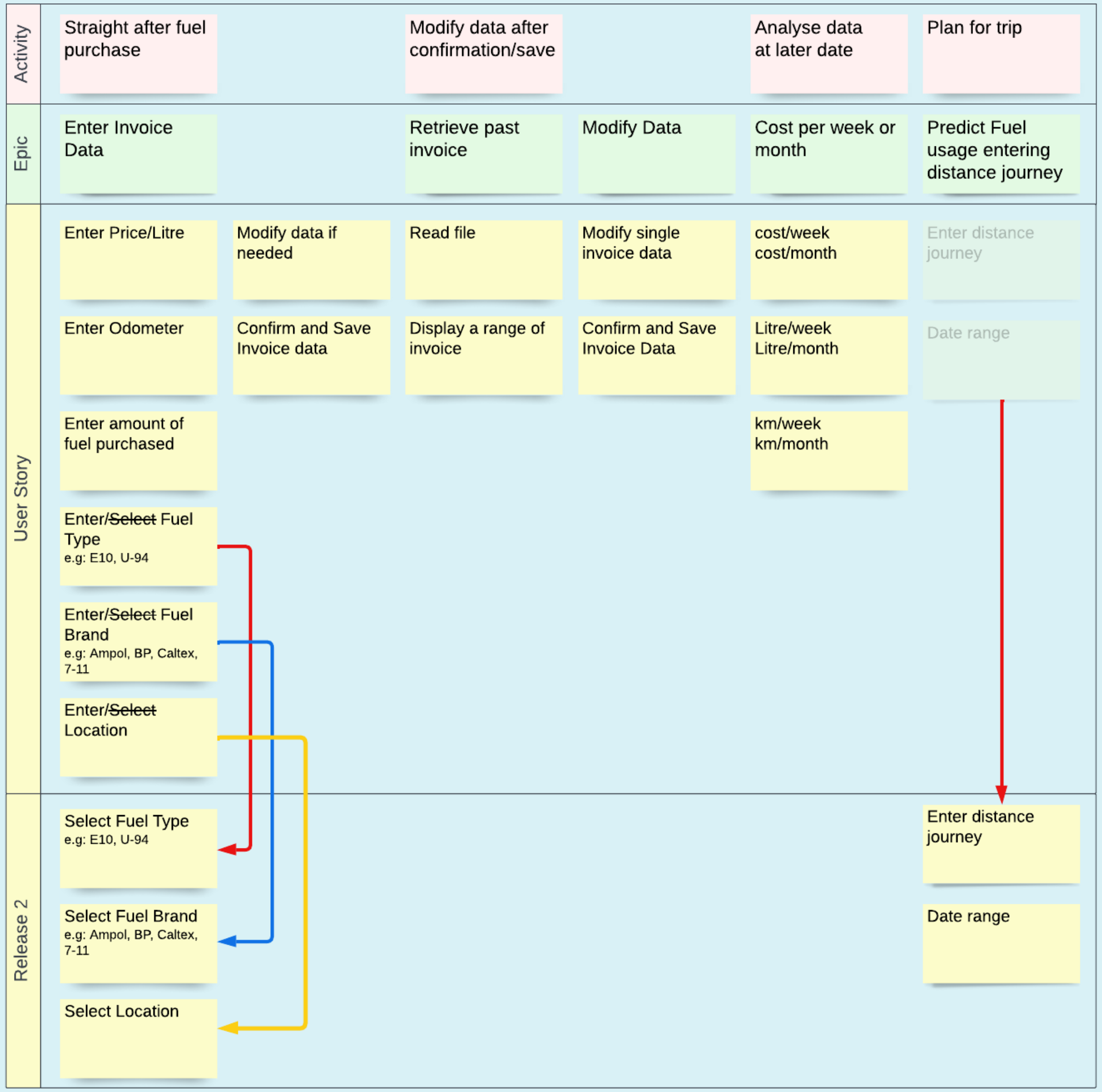
At the CROSS-ROAD

- Skip RSpec-3
- Skip the TDD
- Write up all the code then do RSpec-3



MUST USE TDD !!!!!

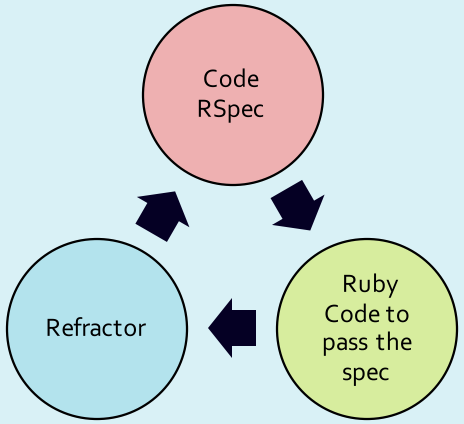
User Story Map Version 2 **Minimum Value Product**

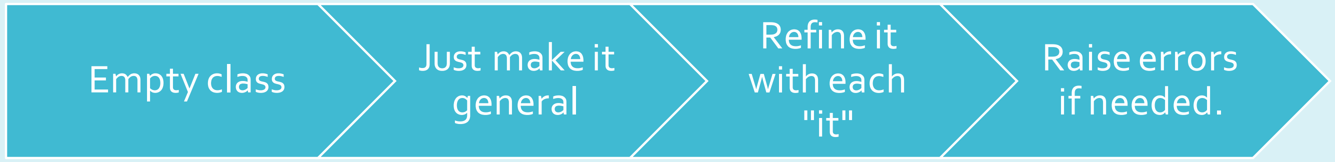


Activity	Straight after fuel purchase		Modify data after confirmation/save		Analyse data at later date	Plan for trip
Epic	Enter Invoice Data		Retrieve past invoice	Modify Data	Cost per week or month	Predict Fuel usage entering distance journey
User Story	Enter Price/Litre	Modify data if needed	Read file	Modify single invoice data	cost/week cost/month	Enter distance journey
	Enter Odometer	Confirm and Save Invoice data	Display a range of invoice	Confirm and Save Invoice Data	Litre/week Litre/month	Date range
	Enter amount of fuel purchased				km/week km/month	
	Enter/~~Select~~ Fuel Type e.g: E10, U-94					
	Enter/~~Select~~ Fuel Brand e.g: Ampol, BP, Caltex, 7-11					
	Enter/~~Select~~ Location					
Release 2	Select Fuel Type e.g: E10, U-94					Enter distance journey
	Select Fuel Brand e.g: Ampol, BP, Caltex, 7-11					Date range
	Select Location					

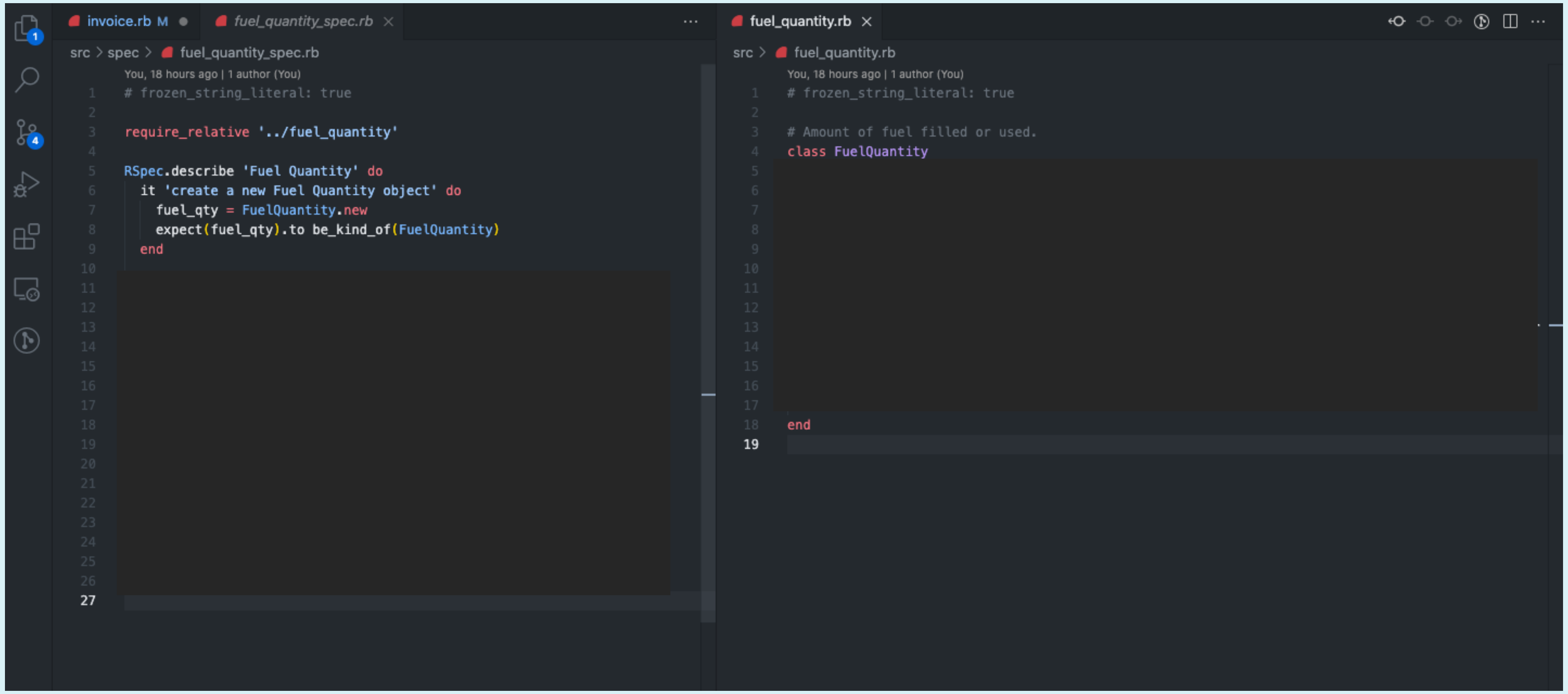
- Enter Data Invoice
  - Moving the select feature to version 2
  - Reduce the file handling (3 extra files not need to be built for version 1)

- Predict Fuel usage for journey
  - Moved it to version 2.



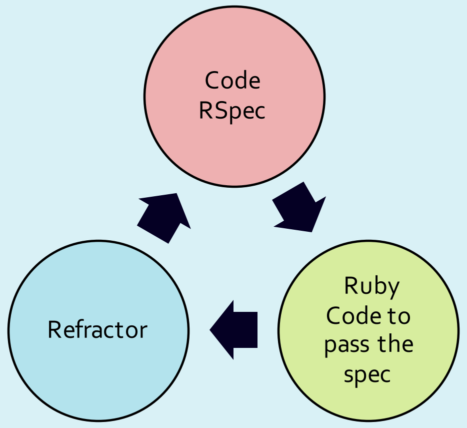


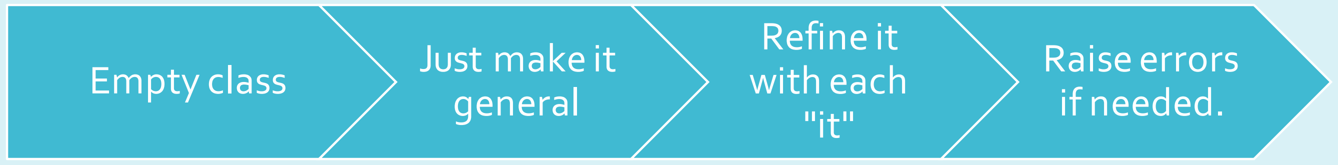
1. Build an empty class



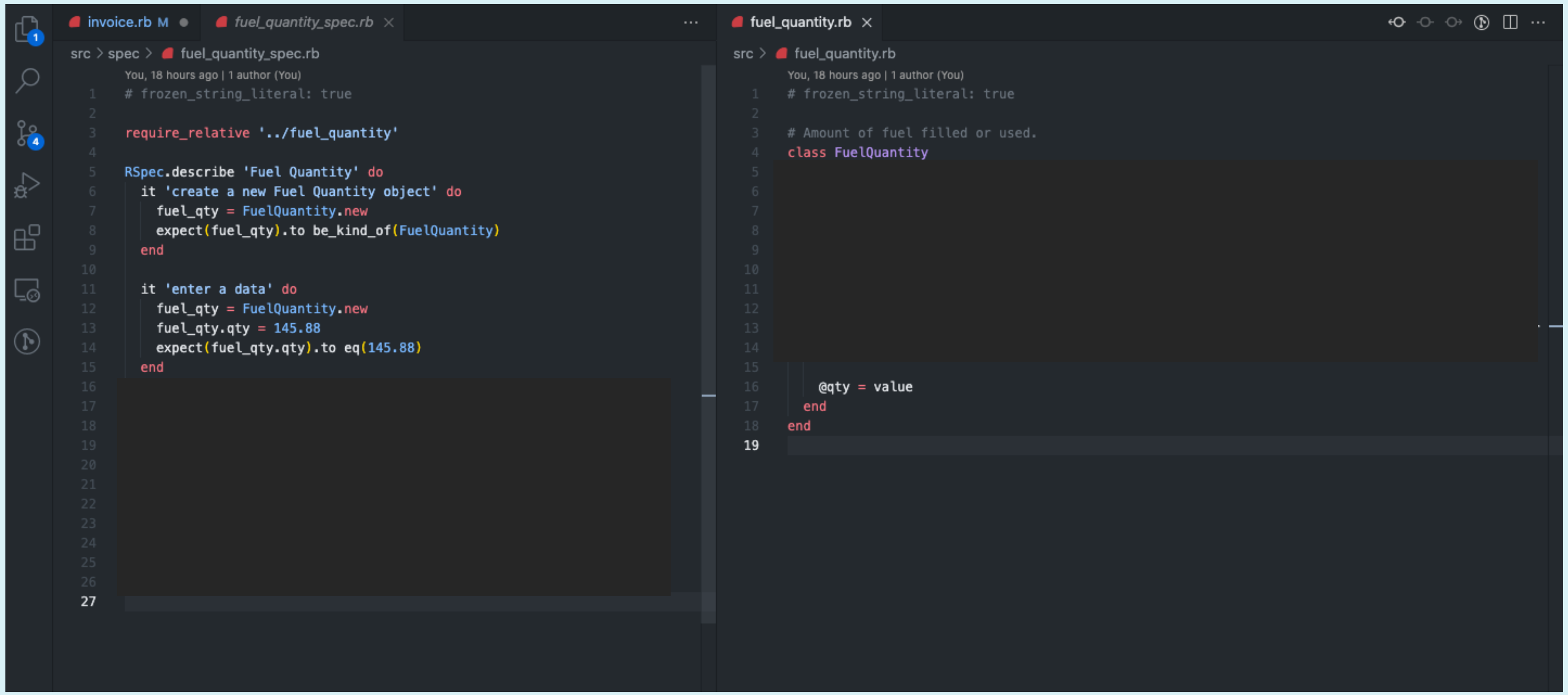
```ruby
# frozen_string_literal: true

require_relative '../fuel_quantity'

RSpec.describe 'Fuel Quantity' do
  it 'create a new Fuel Quantity object' do
    fuel_qty = FuelQuantity.new
    expect(fuel_qty).to be_kind_of(FuelQuantity)
  end
```

```ruby
# frozen_string_literal: true

# Amount of fuel filled or used.
class FuelQuantity














end
```





1. Build an empty class
2. Accept any object


invoice.rb M
fuel_quantity_spec.rb
src > spec > fuel_quantity_spec.rb
You, 18 hours ago | 1 author (You)

```ruby
# frozen_string_literal: true

require_relative '../fuel_quantity'

RSpec.describe 'Fuel Quantity' do
  it 'create a new Fuel Quantity object' do
    fuel_qty = FuelQuantity.new
    expect(fuel_qty).to be_kind_of(FuelQuantity)
  end

  it 'enter a data' do
    fuel_qty = FuelQuantity.new
    fuel_qty.qty = 145.88
    expect(fuel_qty.qty).to eq(145.88)
  end
```

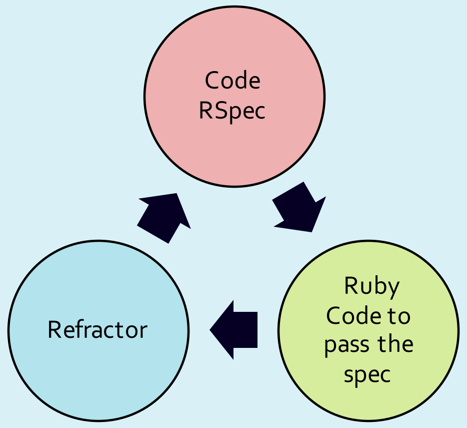
[redacted]

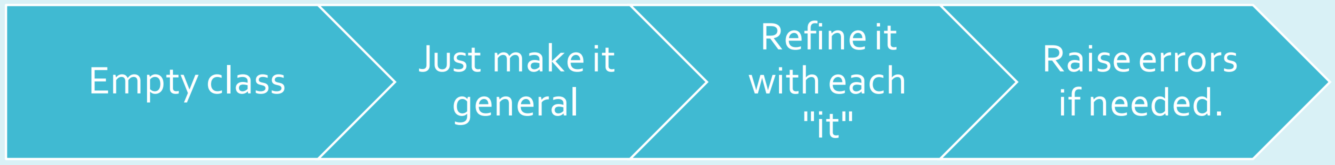
fuel_quantity.rb
src > fuel_quantity.rb
You, 18 hours ago | 1 author (You)

```ruby
# frozen_string_literal: true

# Amount of fuel filled or used.
class FuelQuantity
```

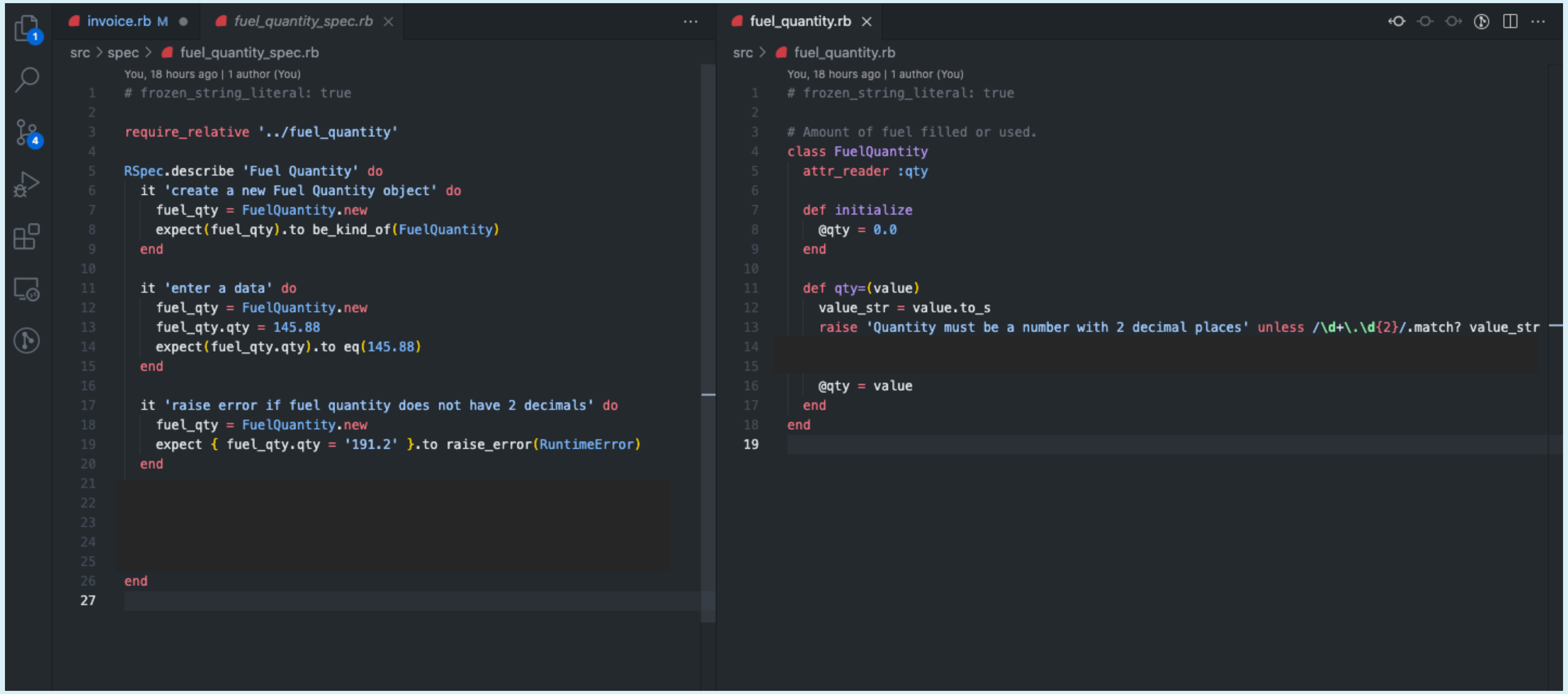
[redacted]

```ruby
    @qty = value
  end
end
```



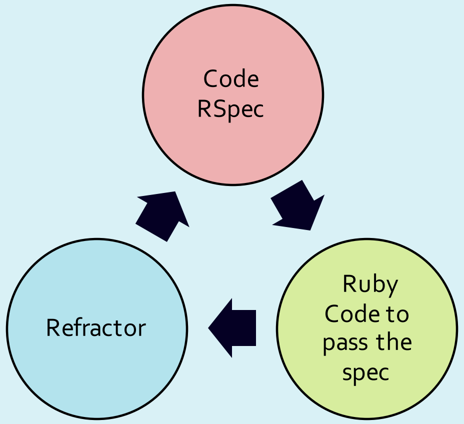


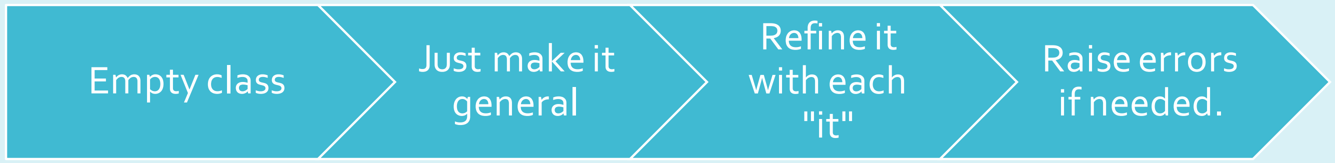
1. Build an empty class
2. Accept any object
3. Must be 2 decimal points


invoice.rb M
fuel_quantity_spec.rb
src > spec > fuel_quantity_spec.rb
You, 18 hours ago | 1 author (You)

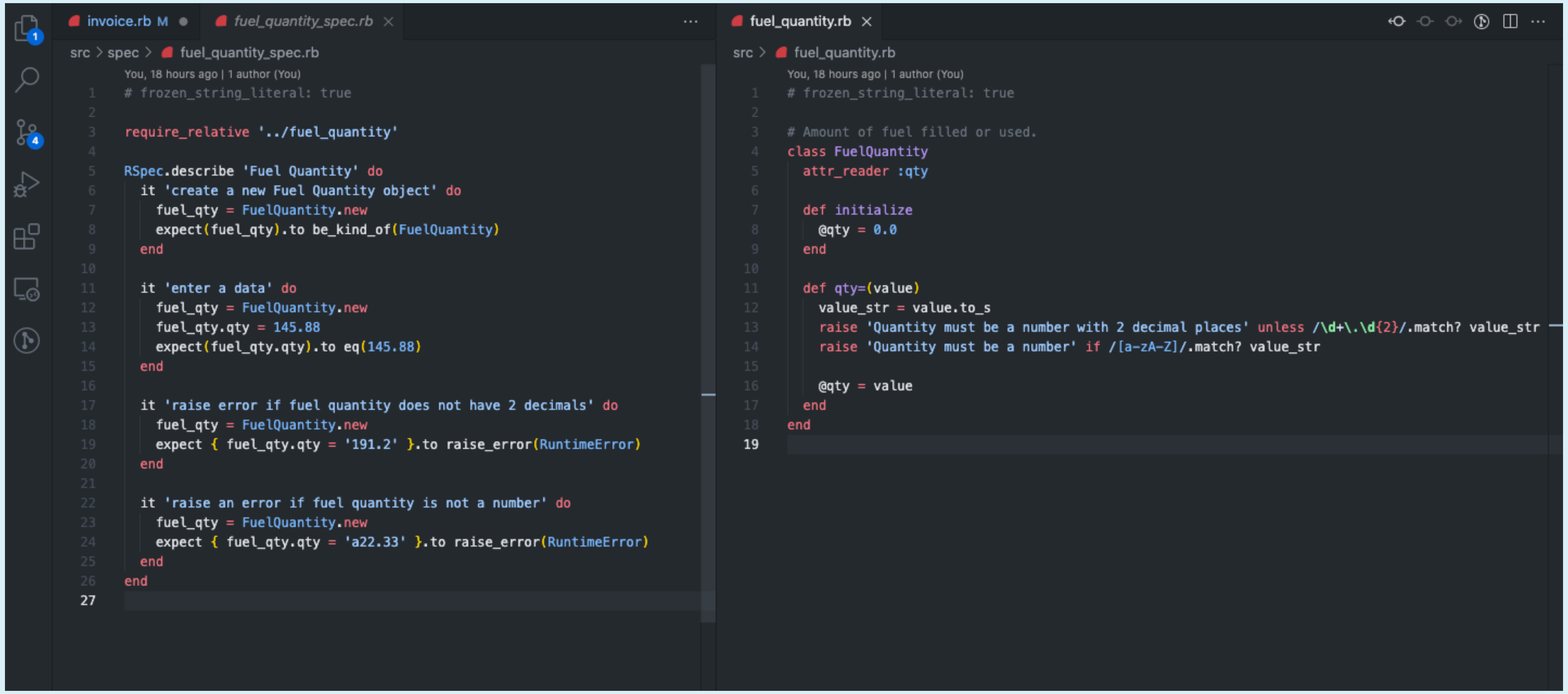
```ruby
# frozen_string_literal: true

require_relative '../fuel_quantity'

RSpec.describe 'Fuel Quantity' do
  it 'create a new Fuel Quantity object' do
    fuel_qty = FuelQuantity.new
    expect(fuel_qty).to be_kind_of(FuelQuantity)
  end

  it 'enter a data' do
    fuel_qty = FuelQuantity.new
    fuel_qty.qty = 145.88
    expect(fuel_qty.qty).to eq(145.88)
  end

  it 'raise error if fuel quantity does not have 2 decimals' do
    fuel_qty = FuelQuantity.new
    expect { fuel_qty.qty = '191.2' }.to raise_error(RuntimeError)
  end






end
```

```ruby
# frozen_string_literal: true

# Amount of fuel filled or used.
class FuelQuantity
  attr_reader :qty

  def initialize
    @qty = 0.0
  end

  def qty=(value)
    value_str = value.to_s
    raise 'Quantity must be a number with 2 decimal places' unless /\d+\.\d{2}/.match? value_str


    @qty = value
  end
end
```





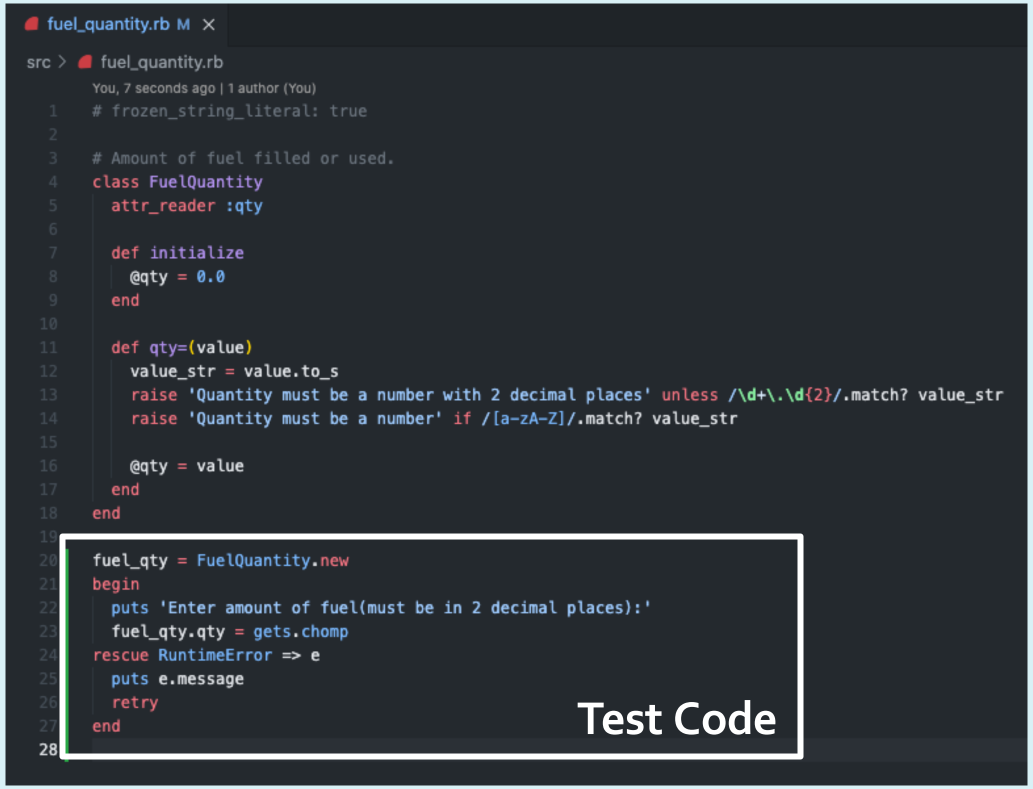
1. Build an empty class
2. Accept any object
3. Must be 2 decimal points
4. Must be a number with no letter


invoice.rb M
fuel_quantity_spec.rb
src > spec > fuel_quantity_spec.rb
You, 18 hours ago | 1 author (You)
fuel_quantity.rb
src > fuel_quantity.rb
You, 18 hours ago | 1 author (You)

```ruby
# frozen_string_literal: true

require_relative '../fuel_quantity'

RSpec.describe 'Fuel Quantity' do
  it 'create a new Fuel Quantity object' do
    fuel_qty = FuelQuantity.new
    expect(fuel_qty).to be_kind_of(FuelQuantity)
  end

  it 'enter a data' do
    fuel_qty = FuelQuantity.new
    fuel_qty.qty = 145.88
    expect(fuel_qty.qty).to eq(145.88)
  end

  it 'raise error if fuel quantity does not have 2 decimals' do
    fuel_qty = FuelQuantity.new
    expect { fuel_qty.qty = '191.2' }.to raise_error(RuntimeError)
  end

  it 'raise an error if fuel quantity is not a number' do
    fuel_qty = FuelQuantity.new
    expect { fuel_qty.qty = 'a22.33' }.to raise_error(RuntimeError)
  end
end
```

```ruby
# frozen_string_literal: true

# Amount of fuel filled or used.
class FuelQuantity
  attr_reader :qty

  def initialize
    @qty = 0.0
  end

  def qty=(value)
    value_str = value.to_s
    raise 'Quantity must be a number with 2 decimal places' unless /\d+\.\d{2}/.match? value_str
    raise 'Quantity must be a number' if /[a-zA-Z]/.match? value_str

    @qty = value
  end
end
```

Error Handling for all small task



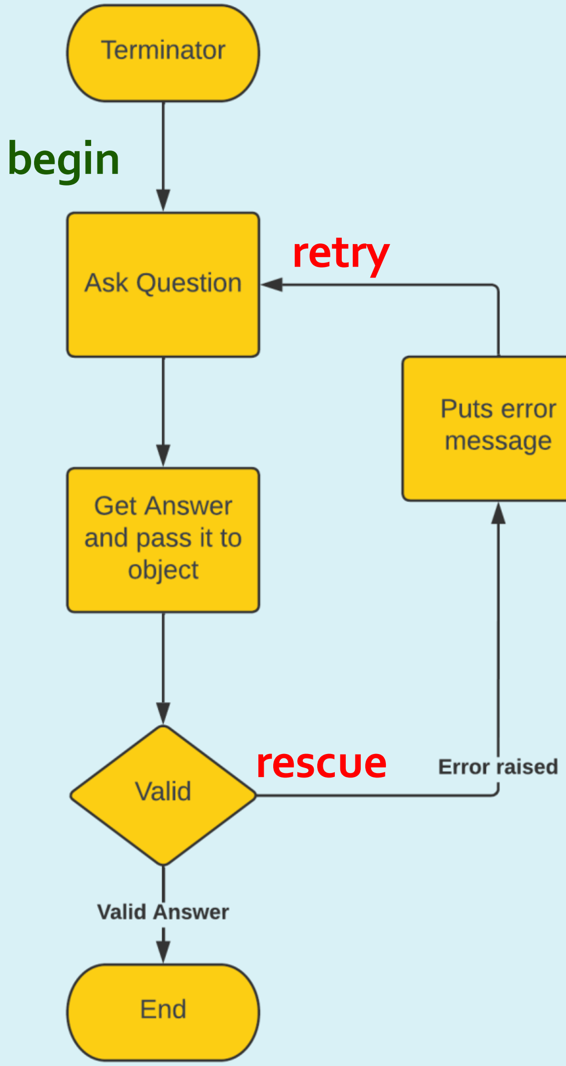
fuel_quantity.rb M ×

src > fuel_quantity.rb

You, 7 seconds ago | 1 author (You)

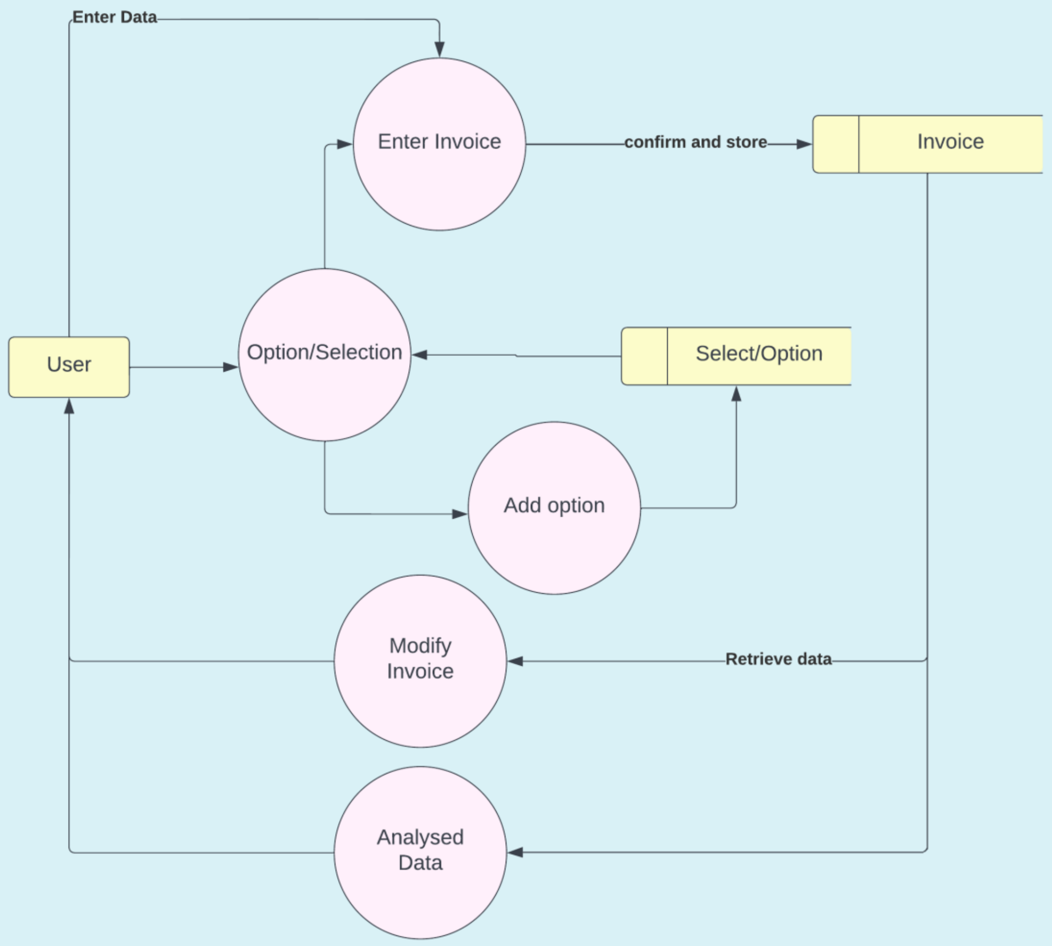
```ruby
# frozen_string_literal: true

# Amount of fuel filled or used.
class FuelQuantity
  attr_reader :qty

  def initialize
    @qty = 0.0
  end

  def qty=(value)
    value_str = value.to_s
    raise 'Quantity must be a number with 2 decimal places' unless /\d+\.\d{2}/.match? value_str
    raise 'Quantity must be a number' if /[a-zA-Z]/.match? value_str

    @qty = value
  end
end

fuel_qty = FuelQuantity.new
begin
  puts 'Enter amount of fuel(must be in 2 decimal places):'
  fuel_qty.qty = gets.chomp
rescue RuntimeError => e
  puts e.message
  retry
end
```

Test Code



Integration

- Confident with my classes due to TDD
- Connect them using the TDD process



Other requirements still needed

- Terminal Interface using TTY
- Installation Instruction
- User manual (might have demo data inside file)
- Command Line Script