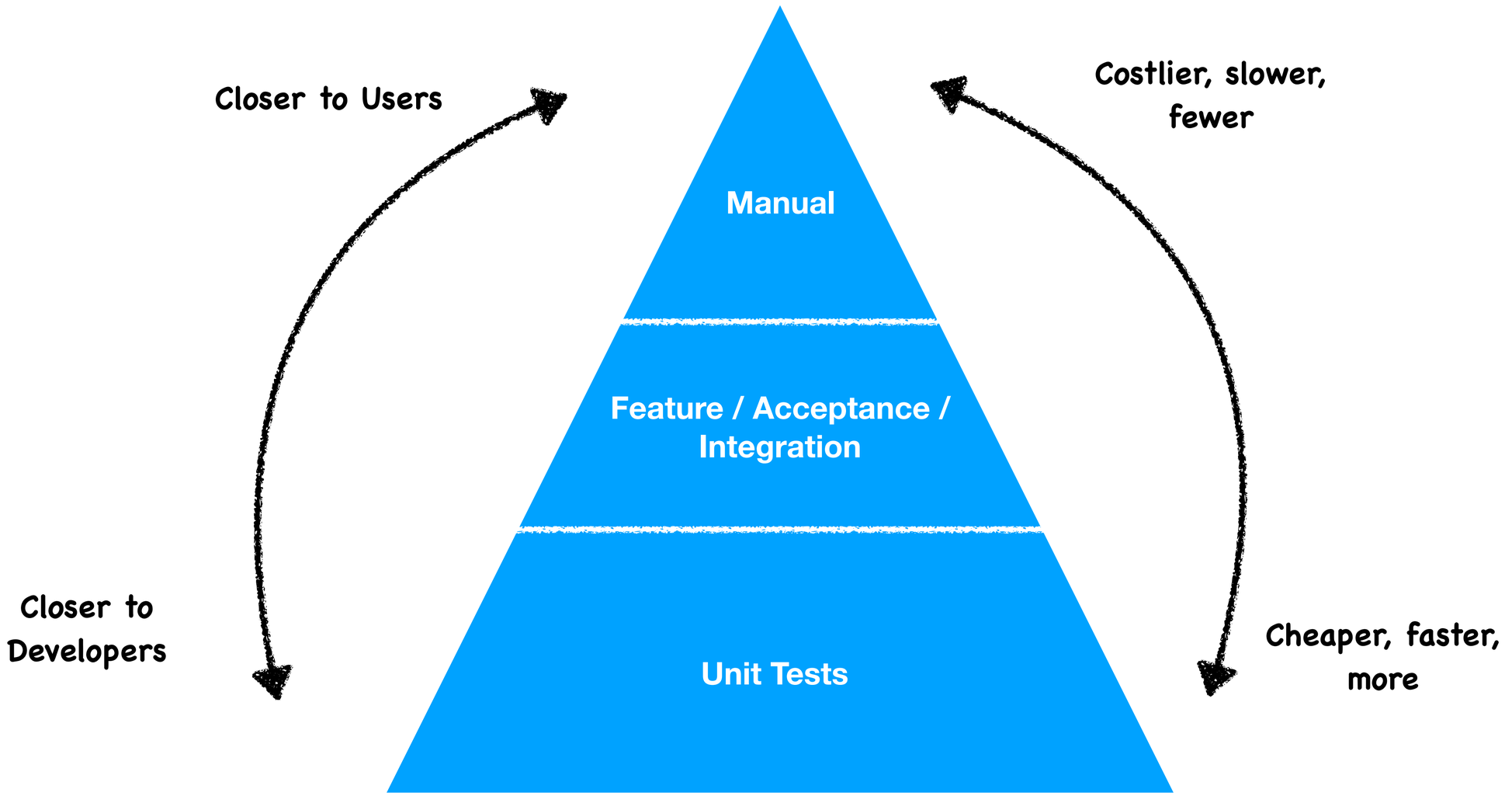
Engineering Day #1

I'm NBT from QE

- data engineer
- data scientist
- performance analyst
- application ops
- end user computing
- infrastructure ops
- business analyst
- test engineering
- test manager
- data architect
- dev.ops
- infrastructure eng.
- network architect
- problem manager
- security architect
- software developer
- technical architect
- content designer
- interaction designer
- technical writer
- test manager

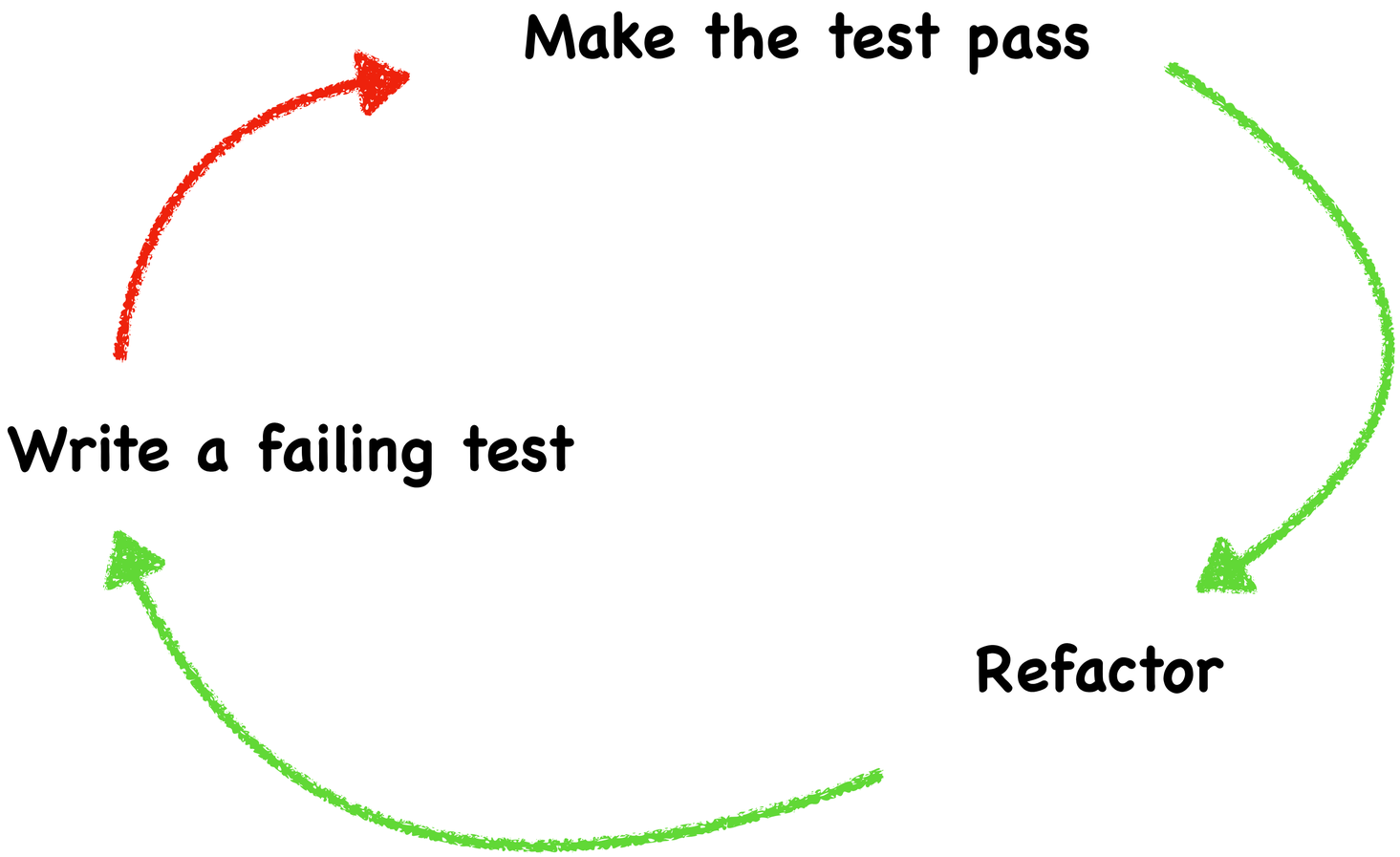
Automation Test Pyramid



Alphabet Soup

- TDD: Test Driven Development
- BDD: Behaviour Driven Development
- ATDD: Acceptance Test Driven Development

Red-Green-Refactor



Feature Testing with Cucumber

DSLs, DSLs everywhere

Cucumber

RSpec

Capybara

All powered by Ruby

Feature Testing with Cucumber

- Cucumber gives us a single source of truth about how the system should behave
- Specifications, test and documentation is the same artifact -- a living document
- Written from the point-of-view of a user (which may be another machine)

Setup Step #1: Getting Ruby

Windows

Download the Ruby installer from

https://rubyinstaller.org/

Follow the instructions and make sure you install the DevKit when prompted

Linux

Check if ruby is already installed: `ruby -v`

If not:

```
sudo apt update
sudo apt install \
  ruby-full
```

Mac

You have a grown-up's machine.

Ruby is already installed.

Setup Step #2: Getting Bundler

- Ruby's libraries are called Gems
- Bundler is a dependency manager for Gems
  - It will install Gems for us, but first we need to install it manually
  - It is a Gem itself
- (Windows) `gem install bundler`
- (Linux / MacOS) `sudo gem install bundler`

Setup Step #3: Initialising Gems

- In a terminal...
- Make and change directory to `<your project>/functional-tests`
- Run `bundle init` —this will create a file called Gemfile
- Edit the Gemfile in your favorite editor and add the line `gem 'cucumber'`
- In a terminal, run `bundle install`
- About what just happened

Setup Step #3: Initialising Cucumber

- In a terminal & in directory `<your project>/functional-tests`
- Run `cucumber --init`
- About what just happened

```
# We’re going to write our first feature
# This is Gherkin - Cucumber's specification language

# Create a file <your project>/functional-tests/features/hello_world.feature
# with the following content

# BTW, This is a comment
Feature: web server responds with a HTTP response code

  The web server in front of my service must behave just like any other web server and any
  response must include a HTTP response code in the header.

  Scenario: When the endpoint does not exist
    When I call a non-existent endpoint
    Then I get a Not Found error
```

```
Feature: web server responds with a HTTP response code
  The web server in front of my service must behave just like any other web server and any
  response must include a HTTP response code in the header.

  Scenario: When the endpoint does not exist # features/hello_world.feature:11
    When I call a non-existent endpoint      # features/hello_world.feature:12
    Then I get a Not Found error             # features/hello_world.feature:13

1 scenario (1 undefined)
2 steps (2 undefined)
0m0.003s

You can implement step definitions for undefined steps with these snippets:

When("I call a non-existent endpoint") do
  pending # Write code here that turns the phrase above into concrete actions
end

Then("I get a Not Found error") do
  pending # Write code here that turns the phrase above into concrete actions
end
```

```
# We’re going to add placeholder steps
# This uses Cucumber DSL and plain-old-ruby

# Create a file <your project>/functional-tests/features/step_definitions/http_steps.rb
# with the following content

When("I call a non-existent endpoint") do
  pending # Write code here that turns the phrase above into concrete actions
end

Then("I get a Not Found error") do
  pending # Write code here that turns the phrase above into concrete actions
end
```

```
Feature: web server responds with a HTTP response code
  The web server in front of my service must behave just like any other web server and any
  response must include a HTTP response code in the header.

  Scenario: When the endpoint does not exist # features/hello_world.feature:11
    When I call a non-existent endpoint      # features/step_definitions/http_steps.rb:1
      TODO (Cucumber::Pending)
      ./features/step_definitions/http_steps.rb:2:in `"I call a non-existent endpoint"'
      features/hello_world.feature:12:in `When I call a non-existent endpoint'
    Then I get a Not Found error             # features/step_definitions/http_steps.rb:5

1 scenario (1 pending)
2 steps (1 skipped, 1 pending)
0m0.002s
```

Adding a library (Gem)

- We like using a HTTP client called HTTParty
- We also want to add RSpec
- We need to add the library to our Gemfile, then import it into our code
- In a terminal, run `bundle install`

Gemfile

```
# frozen_string_literal: true

source "https://rubygems.org"

git_source(:github) {|repo_name| "https://
github.com/#{repo_name}" }

# gem "rails"
gem 'cucumber'
gem 'rspec'
gem 'httparty'
```

features/support/env.rb

```
require 'httparty'
require 'rspec/expectations'
```

```
# Edit <your project>/functional-tests/features/step_definitions/http_steps.rb
# to match this

When("I call a non-existent endpoint") do
  @response = HTTParty.get( 'http://localhost:3000/nobody_home' )
end

Then("I get a Not Found error") do
  pending # Write code here that turns the phrase above into concrete actions
end
```

```
Feature: web server responds with a HTTP response code
  The web server in front of my service must behave just like any other web server and any
  response must include a HTTP response code in the header.

  Scenario: When the endpoint does not exist # features/hello_world.feature:11
    When I call a non-existent endpoint      # features/step_definitions/http_steps.rb:1
    Then I get a Not Found error             # features/step_definitions/http_steps.rb:5
      TODO (Cucumber::Pending)
      ./features/step_definitions/http_steps.rb:6:in `"I get a Not Found error"'
      features/hello_world.feature:13:in `Then I get a Not Found error'

1 scenario (1 pending)
2 steps (1 pending, 1 passed)
0m0.082s
```

```
# Edit <your project>/functional-tests/features/step_definitions/http_steps.rb
# to match this

When("I call a non-existent endpoint") do
  @response = HTTParty.get( 'http://localhost:<port>/nobody_home' )
end

Then("I get a Not Found error") do
  expect( @response.code ).to eql 404
end
```

```
Feature: web server responds with a HTTP response code
  The web server in front of my service must behave just like any other web server and any
  response must include a HTTP response code in the header.

  Scenario: When the endpoint does not exist # features/hello_world.feature:11
    When I call a non-existent endpoint      # features/step_definitions/http_steps.rb:1
    Then I get a Not Found error             # features/step_definitions/http_steps.rb:5

1 scenario (1 passed)
2 steps (2 passed)
0m0.081s
```

```
# Adding another scenario <your project>/functional-tests/features/hello_world.feature
# with the following content

Feature: web server responds with a HTTP response code

  The web server in front of my service must behave just like any other web server and any
  response must include a HTTP response code in the header.

  Scenario: When the endpoint does not exist
    When I call a non-existent endpoint
    Then I get a Not Found error

  Scenario: When the endpoint does exist
    When I call an existent endpoint
    Then I do not get a Not Found error
```

```
# Edit <your project>/functional-tests/features/step_definitions/http_steps.rb
# to match this

When(/I call (?:a|an) (non-existent|existent) endpoint/) do |endpoint_type|
  if endpoint_type == 'non-existent'
    @response = HTTParty.get( 'http://localhost:3000/nobody_home' )
  else
    @response = HTTParty.get( 'http://localhost:3000/' )
  end
end

Then("I get a Not Found error") do
  expect( @response.code ).to eql 404
end

Then("I do not get a Not Found error") do
  expect( @response.code ).not_to eql 404
end
```

```
Feature: web server responds with a HTTP response code
  The web server in front of my service must behave just like any other web server and any
  response must include a HTTP response code in the header.

  Scenario: When the endpoint does not exist # features/hello_world.feature:11
    When I call a non-existent endpoint      # features/step_definitions/http_steps.rb:2
    Then I get a Not Found error             # features/step_definitions/http_steps.rb:10

  Scenario: When the endpoint does exist # features/hello_world.feature:15
    When I call an existent endpoint     # features/step_definitions/http_steps.rb:2
    Then I do not get a Not Found error  # features/step_definitions/http_steps.rb:14

2 scenarios (2 passed)
4 steps (4 passed)
0m0.186s
```

Pause for Breath

```
# Let's add a new feature test

# Create a file <your project>/functional-tests/features/albums.feature
# with the following content

Feature: My Favourite Albums

  People want to be able to get a list of all of their favourite albums so that they can
  compare their musical tastes with other people.

  Scenario: Listing all albums
    Given I ask for a list of all albums
     Then I will receive a machine-readable response
      And It will contain my favourite albums
```

```
# Edit <your project>/functional-tests/features/step_definitions/album_steps.rb
# to match this

When("I ask for a list of all albums") do
  @response = HTTParty.get( 'http://localhost:3000/album' )
end

Then("I will receive a machine-readable response") do
  expect{ @json = JSON.parse( @response.body ) }.not_to raise_error
end

Then("It will contain a list of my favourite albums") do
  expect( @json ).to be_a_kind_of( Array )
end
```

```
# Let's add a new feature test

# Create a file <your project>/functional-tests/features/song_locations.feature
# with the following content

Feature: Storing and retrieving the locations I like songs
  As a user, I want to be able to record the locations I like to listen
  to particular songs

  Scenario: Creating a location for a track
    When I record a location for a song
    Then The location I like to listen to that song is stored
```

```
# Edit <your project>/functional-tests/features/step_definitions/location_steps.rb
# to match this

When("I record a location for a song") do
  @payload = {
    'location' => '50.00, 25.00',
    'song_id'  =>
  }
  @response = HTTParty.post(
     'http://localhost:3000/songLocation',
     body: @payload.to_json,
    headers: { 'Content-Type' => 'application/json' } )
end

Then("The location I like to listen to that song is stored") do
  @validation_response = HTTParty.get( 'http://localhost:3000/songLocation' )
  @locations = JSON.parse( @validation_response.body )
  # we have to do this step to workaround an API issue
  @locations.each { |location| location.delete( 'song_location_id' ) }
  expect( @locations ).to include( @payload )
end
```