Unit Testing and Test Driven Development

(With some slides created by
Steven Bourke, PhD Student, UCD)

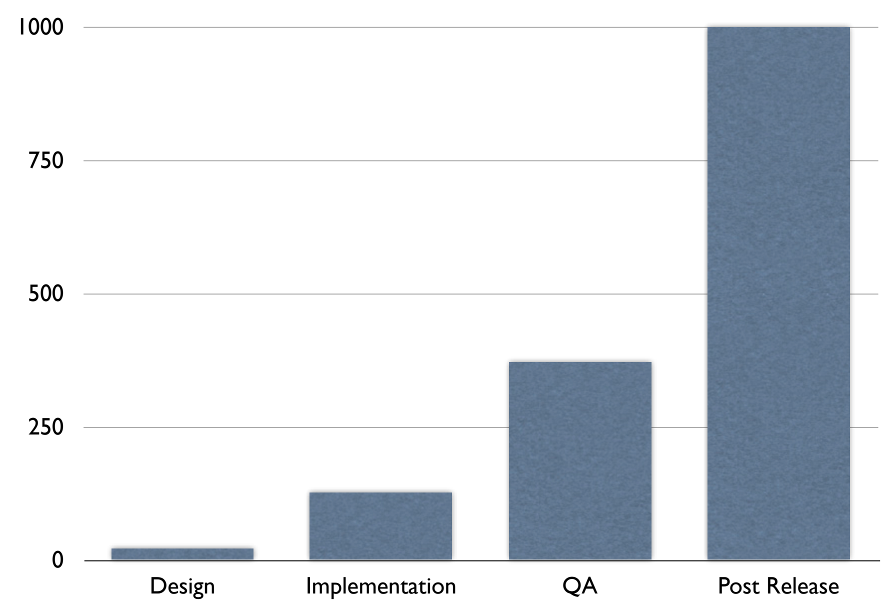
What we’ll cover today...

- Testing
- Unit Tests in Ruby
- Test-Driven Development
- Case Study

Why do you test your code?

To find bugs?	Good idea! But there may not be any bugs to find.
To show there are no bugs?	Can you ever be sure of this in a realistic system?
To improve confidence that it works?	✓
So I know if it breaks in the future?	✓

Relative cost of finding bugs



A simple Ruby class

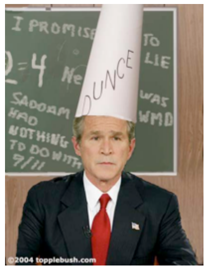
```
class LightHouse
  def initialize
    @light = :OFF
  end

  def press_button
    if @light == :OFF
      @light = :ON
    else
      @light = :OFF
    end
  end

  def on_or_off?
    @light
  end
end
```

... and a test script

```
lh = LightHouse.new
if lh.on_or_off? != :OFF
  puts "error in lighhouse initilisation"
end
lh.press_button
if lh.on_or_off? != :ON
  puts "lighthouse failed to switch on"
end
lh.press_button
if lh.on_or_off? != :OFF
  puts "lighthouse failed to switch off"
end
lh.press_button
if lh.on_or_off? != :ON
  puts "lighthouse failed to switch on"
end
puts "tests finished"
```

What happens to the test script?

"I run it to make sure my code is ok. The I delete it. Sure it's done its job, it's just clutter after that."



"I comment it out but still hang on to it just in case I want to run it again, like if I'm updating the class again and want to be sure there are no new bugs"

"I write my tests as **unit tests** and keep them. I can run them anytime to ensure I haven't accidentally broken my code."

Kent Beck, co-creator of the Java testing framework **JUnit**.

Anatomy of a Unit Test in Ruby

```
require 'test/unit'
require 'foobar.rb'

class TC_FooBar < Test::Unit::TestCase

  def setup
    ...
  end

  def test_one
    ...
    assert ...
  end

  def test_two
    ...
    assert_equal ...
  end
end
```

to use the unit testing framework

`TC_FooBar` is test case for the `FooBar` class.

`setup` is executed before **each** test method

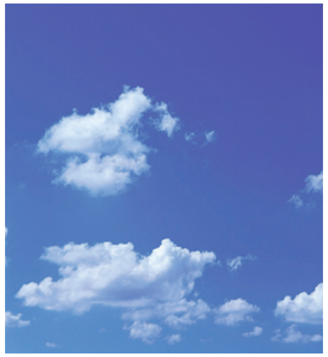
Each test method must be named `testxxx`.

Test methods will always contain **assertions**.

Assertions

Assertions are the basis of unit tests.

Assert things that you expect to be true.


The sky is blue... TRUE

The clown is happy... FALSE

Sample Unit Test

```
require 'test/unit'
require 'lighthouse.rb'

class TC_Lighthouse < Test::Unit::TestCase
  def setup
    @my_light = LightHouse.new
  end

  def test_initialize
    assert(my_light.on_or_off? == :OFF,
          'new lightbulb is not off')
  end

  def test_sequence
    9.times{my_light.press_button}
    assert(my_light.on_or_off? == :ON,
          'lightbulb off after 9 presses')
  end
end
```

Unit Tests and Ruby

Unit Testing is all the rage. Unit Testing frameworks exist for all popular languages.

The most well-known unit testing framework is **JUnit**, for Java.

Unit testing is much more important for a dynamically-typed language like Ruby, than it is for statically-typed languages like Java or C++.

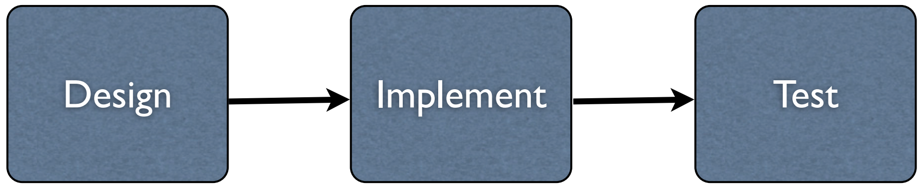
Why?

Test Driven Development (TDD)

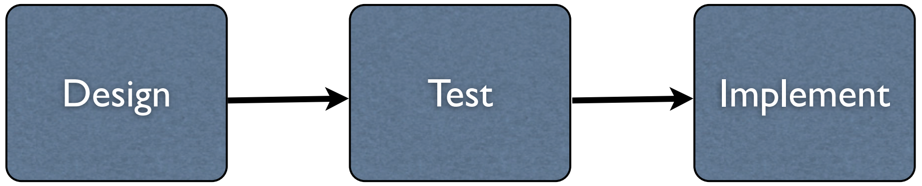
The notion of developing automated tests has lead to the idea of making testing more central to the development process.

We'll look at this idea briefly...

“Traditional” Approach to Software Development



Test-Driven Development



Huh? How can you test before you implement?

Write Unit Tests that fail

Unit Tests should fail at first...

This provokes you to write code to make them pass.

Write code to make Unit Tests pass

Implement code that can pass your unit tests

Now tidy up your code

Tidy up your code, make sure there is no duplicated code or ugly bits. (This is a whole area in itself, termed **refactoring**.)

TDD Example

See Point example.

Key Points

Testing is a vital part of software development.

Unit testing involves writing test cases for individual classes. A **unit testing framework** can manage these test cases and run them on command.

Test-Driven Development (TDD) is where test cases are written first, as a way of driving the development of software. This is a key part of the **Agile** approach to software development.

Write test cases for any classes you develop in your assignments for this module.