Testing Frameworks (MiniTest)

Lecture 32
MiniTest and RSpec

- Many popular testing libraries for Ruby
- MiniTest (replaces older Test::Unit)
  - Comes built-in
  - Looks like JUnit (mapped to Ruby syntax)
  - Well-named!
- RSpec
  - Installed as a library (i.e. a "gem")
  - Looks different from JUnit (and even Ruby!)
  - Most unfortunate name!
- RSpec view is that test cases *define* expected behavior—they *are* the spec!
  - What is wrong with that view?
Writing MiniTest Tests

- Require runner and UUT
  ```ruby
  require 'minitest/autorun'
  require 'card'
  ```
- Test fixture = subclass of `MiniTest::Test`
  ```ruby
  class TestCard < MiniTest::Test
  ```
- Test case = method in the fixture
  - Method name must begin with `test_`
    ```ruby
    def test_identifies_set ... end
    ```
  - Contains assertion(s) exercising a single piece of code / behavior / functionality
  - Should be `small` *(i.e. test one thing)*
  - Should be `independent` *(i.e. of other tests)*
- Test Suite = collection of fixtures
Example: test_card.rb

```ruby
require 'minitest/autorun'
require 'card' # assume card.rb on load path

class TestCard < MiniTest::Test

  def test_has_number
    assert_respond_to Card.new, :number
  end

  def test_remembers_number
    @card = Card.new 1, "oval", "open", "red"
    assert_equal 1, @card.number
  end
end
```
Execution Model

TestCard

instance of

@card

has_number()
remembers()

instance of

has_number()
remembers()
Execution Model: Implications

- Separate instances of test class created
  - One instance / test case

- Test cases don't have side effects
  - Passing/failing one test does not affect others

- Can not rely on order of tests
  - Randomized order of execution
  - Controllable with --seed command-line option
  - Also controllable by invoking `i_suck_and_my_tests_are_order_dependent!`

- Fixture: common set-up to all test cases
  - Field(s) for instance(s) of class being tested
  - Factor initialization code into its own method
  - This method must be called `setup`
Good Practice: setup

- Initialize a fixture with a setup method (rather than initialize method)

- Reasons:
  - If the code being tested throws an exception during the setup, the output is much more meaningful
  - Symmetry with teardown method for cleaning up after a test case
require 'minitest/autorun'
require 'card' #assume card.rb is on load path

class TestCard < Minitest::Test

  def setup
    @card = Card.new 1, "oval", "open", "red"
  end

  def test_has_number
    assert_respond_to @card, :number
  end

  def test_remembers_number
    assert_equal 1, @card.number
  end

end
Execution Model

TestCard

instance of

@card
setup()
has_number()
remembers()

instance of

@card
setup()
has_number()
remembers()
MiniTest Assertion Methods

- Most have two versions: `assert` & `refute`
  - Example: `assert_nil, refute_nil`
  - No need for negation (use `refute` instead)
- Most take an optional message
  ```ruby
  assert_empty Library.new, 
  "A new library contains no books"
  ```
  - Message appears when assertion fails
- Specials:
  - `pass/flunk` – always passes/fails
  - `skip` – skips the rest of the test case
- Performance benchmarking also available
Common Assertions

- Boolean condition: `assert/refute`
  ```ruby
  assert @books.all { |b| b.available? }
  ```

- Is nil: `(assert/refute) _nil`
  ```ruby
  refute_nil @library.manager
  ```

- Is empty: `(assert/refute) _empty`
  ```ruby
  assert_empty Library.new
  ```
Asserting Equality

- Assert two objects are `==` equal
  ```ruby
  assert_equal expected, actual
  ```
  - Compares *object* values (*i.e.* `==` in Ruby)
  - Failure produces useful output
    ```ruby
    TestCard#test_total_number_of_cards
    Expected: 81
    Actual: 27
    ```
  - Compare with basic `assert exp == actual`
    ```ruby
    TestCard#test_shuffle_is_permutation
    Failed assertion, no message given
    ```

- Assert two objects are aliased
  ```ruby
  assert_same @table.north, @players.first
  ```
  - Compares *reference* values (*i.e.* `.equal?()`)
More Assertions

- String matches a regular expression
  ```ruby
  assert_match /CSE.*/, @course.name
  ```

- Collection includes a particular item
  ```ruby
  assert_includes @library, @book
  ```

- Object is of a particular type
  ```ruby
  assert_instance_of String, @book.title
  ```

- Object has a method
  ```ruby
  assert_respond_to @student, :alarm
  ```

- Block raises an exception
  ```ruby
  assert_raises ZeroDivisionError do
    @library.average_book_cost
  end
  ```
Good Practice: Comparing Floats

- Never compare floating point numbers directly for equality
  ```ruby
  assert_equal 1.456, calculated,
  "Low-density experiment"
  ```

- Numeric instabilities make exact equality problematic for floats

- Better approach: Equality with tolerance (delta or epsilon)
  ```ruby
  assert_in_delta Math::PI, (22.0 / 7.0),
  0.01, "Archimedes algorithm"
  ```

- Delta for error, epsilon for relative error
Good Practice: Organization

- Keep tests in the same *project* as the code
  - They are part of the build, the repo, *etc.*
  - Helps to keep tests current
- Separate tests and implementation
  - \set\lib – contains card.rb (implementation)
  - \set\tests – contains test_card.rb (tests)
- Name test classes consistently
  - TestCard tests Card
- Test fixture is a Ruby program
  - [setapp] $ ruby tests\test_card.rb
  - Test needs to be able to find UUT (require)
  - Add location of UUT to load path
    - [setapp] $ ruby -I lib tests\test_card.rb
Alternative Syntax

- **Problem:** Cumbersome method names
  ```ruby
test_shuffle_changes_deck_configuration
  ```
- **Solution:** exploit Ruby language flexibility in API of testing library
  - Methods are available that change the syntax and structure of test cases
  - "Domain-specific language" for testing
- **Result:** MiniTest::Spec
  - Notation inspired by RSpec
Writing MiniTest::Spec Tests

- Require spec library (+ runner + UUT)
  ```ruby
  require 'minitest/spec'
  ```

- Test fixture (an “example group”) is a `describe` block
  ```ruby
  describe Card do ...
  ```
  - Can be nested, and identified by string
  - The block contains examples

- Test case (an “example”) is an `it` block
  ```ruby
  it 'identifies a set' ...
  ```
  - Contains *expectation*(s) on a single piece of code / behavior / functionality

- Expectations are methods on objects
  ```ruby
  @card.number.must_equal 1
  ```
require 'minitest/spec'
require 'minitest/autorunner'
require 'card'  # assume card.rb is on load path

describe Card, 'card for game of set' do

  it 'has a number' do
    Card.new.must.respond_to :number
  end

  it 'remembers its original number' do
    @card = Card.new 1, "oval", "open", "red"
    @card.number.must_equal 1
  end

end
Expectations vs. Assertions

- Positive and negative form
  - `must_be_empty`
  - `wont_be_empty`

- Have corresponding assertions
  - But different argument order
    - `assert_equal expected, actual`
    - `actual.must_equal expected`

- No string argument
  - Meaningful output comes from group and example name(s)
Obj.must_ + ...

- General expectation: Must be
  \texttt{x.must\_be :<, 10}

- Many other flavors of expectation...
  \texttt{x.must\_equal y}
  \texttt{x.must\_be\_same\_as y}
  \texttt{@library.manager.must\_be\_nil}
  \texttt{@shelf.must\_be\_empty}
  \texttt{@library.must\_include @book}
  \texttt{PI.must\_be\_within\_delta (22.0 / 7.0), .01}
  \texttt{@book.title.must\_be\_instance\_of String}
  \texttt{@course.name.must\_match /CSE.*/}
  \texttt{@student.must\_respond\_to :alarm proc { }
    \texttt{@library.average\_book\_cost}
  }.must\_raise ZeroDivisionError
Setup/Teardown

- Methods **before, after**
- Arguments :**each** or :**all**

```ruby
describe Student do
  before :each do
    @buck_id = BuckID.new "4328429"
    @s = Student.new buck_id
  end

  it 'should come to class' do ... end
end
```
Let: Lazy Initialization

describe Student do
  let(:student) { Student.new 1234 }

  describe 'sleep deprivation'
    it 'misses class' do
      student.awake?.must_equal false
    end
  end
end
RSpec: Set up and Use

- Install the rspec gem locally
  ```bash
  [~] $ gem install rspec
  ```
- Set up your program to use rspec
  ```bash
  [myapp] $ rspec --init
  ```
- Init creates several things in myapp/
  ```
  spec/  # put tests (foo_spec.rb) here
  spec/spec_helper.rb  # configures paths
  .rspec  # default command-line args
  ```
- Run tests
  ```bash
  [myapp] $ rspec spec/foo_spec.rb
  ```
require_relative '../student'

describe Student do  # example group
  it 'can drop a class' do  # example
    ...
  end

  context 'when attending lecture' do
    it 'stays awake during lecture' do
      ...
    end

    it 'stores info until exam' do
      ...
    end
  end
end
RSpec Expectations

- Verb is "should" (or "should_not")
  ```ruby
target.should condition #notice space
  ```

- Examples of condition
  - `==`, `equal`,
    ```ruby
    factor.should equal 34
    ```
  - `be_true`, `be_false`, `be_nil`, `be_empty`
    ```ruby
    list.empty?.should be_true
    ```
  - `have(n).items`, `have_at_most(n).items`
  - `include(item)`
    ```ruby
    list.should include(name)
    ```
  - `match(regex)`
  - `respond_to(method_name)`

- New form: `expect().to (or not_to)`
  ```ruby
  expect(a_result).to eq "OSU"
  ```
Stubs

- Top-down: testing a class that uses A, B, C
- Problem: We don't have A, B, C
  - Want quick approximations of A, B, C
  - Behave in certain way, returning canned answers
- Solution: Stub method
  - Takes a hash of method names & return values
  - Returns an object with those methods
    ```ruby
    stub_printer = stub :available? => true, :render => nil
    ```
- Another form adds (or changes) a method/return value of an existing object
  ```ruby
  long_str = 'something'
  long_str.stub! (:length).and_return(1000000)
  ```
Mocks

- Stubs passively allow the test to go through
-Mocks *monitor* how they are used (and will fail if they aren't used right)

```ruby
it 'should know how to print itself' do
  mock_printer = mock('Printer')
  mock_printer.should_receive(:available?).and_return(true)
  mock_printer.should_receive(:render).exactly(3).times
  @doc.print (mock_printer).should == 'Done'
end
```
Summary

- **MiniTest**
  - Test case: method named `test_`
  - Test fixture: class extending `Minitest::Test`

- **Execution model: multiple instances**
  - Independence of test cases

- **MiniTest::Spec**
  - Examples and expectations
  - String descriptions

- **RSpec**
  - Stubs and mocks