

Techniques for Ruby testing

Test::Unit

- Cycle: set_up, test, tear_down

```
def setup
  @review_status = ReviewStatus.new
end

def test_should_supply_constraint_indicating_whether_review_is_awaiting_retailer_action
  assert_equal "status IN ('A')", @review_status.awaiting_retailer_action_constraint
end

def tear_down
  @review_status = nil
end
```

Naming Tests

- Conventions help you remember...
- We use 'should' to remind us that if a test fails we should ask the question "should the code still behave like this?"

```
def setup
  @review_status = ReviewStatus.new
end

def test_should_supply_constraint_indicating_whether_review_is_awaiting_retailer_action
  assert_equal "status IN ('A')", @review_status.awaiting_retailer_action_constraint
end

def tear_down
  @review_status = nil
end
```

Tests as automatic specification

```
class ReviewStatusTest < Test::Unit
  def setup
    @review_status = ReviewStatus.new
  end

  def test_should_supply_constraint
    assert_equal "status IN ('A')",
  end

  def tear_down
    @review_status = nil
  end
end
```

```
$ rake agiledox
ReviewStatusTest
  review_status should:
    - supply constraint
    indicating whether
    review is published
$
```

Tests as specification

review_status should:

- convert to different character for each status
- indicate form has been viewed
- indicate whether email requesting review completion has been sent
- indicate whether review form can be submitted
- indicate whether review form has been viewed
- indicate whether review has been rejected
- indicate whether review is awaiting retailer action
- indicate whether review is awaiting vetting
- indicate whether review is published
- publish review and store publication date
- refer review to retailer and store rejection date
- reject review and store rejection date
- submit review and store submission date
- supply constraint indicating whether email requesting review completion has been sent
- supply constraint indicating whether review form has been viewed
- supply constraint indicating whether review is awaiting retailer action
- supply constraint indicating whether review is awaiting vetting
- supply constraint indicating whether review is published

Blocks to establish context

```
def test_product_report
  run_as_admin(@selenium) {
    @selenium.open('/product_admin/')
    @selenium.assert_text_present('Reviewworld Administration')
  }
end

def test_access_denied_for_non_admin_user
  run_as_normal_user(@selenium) {
    @selenium.open('/product_admin/')
    @selenium.assert_text_present('You do not have sufficient privileges to access this page')
  }
end
```

Make your tests faster

- Avoid Db Access, File IO, Big object graphs
 - If you can...

Using Mocks over Fixtures

```
def setup
  @google = SchMock.new
end

def test_should_use_google_to_find_product_on_amazon_uk_site_and_extract_asin_from_url
  response = Struct.new(:resultElements).new([Struct.new(:URL).new("http://www.amazon.co.uk/exec
  @google.__expects(:search).with("Nikon Coolpix S3 site:www.amazon.co.uk").returns(response)
  asin = @google.asin(:manufacturer=>'Nikon', :model=>'Coolpix S3')
  @google.__verify
end
```

Stubbing built-in classes

```
TimeStub.with_time_now(hit_count.start) {  
  ProductReviewMarkHitCount.increment(:product => hit_count.product, :retailer => hit_count.retailer)  
}
```

```
def self.with_time_now(time, &block)  
  stub_now  
  begin  
    Time.test_now = time  
    yield  
  end  
  ensure  
    unstub_now  
  end  
end
```

```
def self.stub_now  
  Time.module_eval <<- "end_eval"  
  class <<self  
    attr_accessor :test_now  
    alias_method :__now__, :now  
    def now  
      test_now  
    end  
  end  
end_eval  
end  
  
def self.unstub_now  
  Time.module_eval <<- "end_eval"  
  class <<self  
    remove_method :test_now  
    remove_method :test_now=  
    remove_method :now  
    alias_method :now, :__now__  
  end  
end_eval  
end
```

More Stubbing & Mocking

```
module HReviewMockModule
  module ClassMethods
    def from_xml(xml)
      'HReview.from_xml'
    end
  end
  def to_review(ping)
    'HReview#to_review'
  end
end
```

```
def test_should_call_on_hreview_from_xml_method
  mocking HReview do
    review_collector = ReviewCollector.new(Ping.new, StringIO.new, logger)
    assert_equal 'HReview.from_xml', review_collector.hreview
  end
end
```

Tests should change the
way you code...

Conclusions

- Write lots of tests
- Decide on your own conventions
- Don't be afraid to code in a way that makes it easy to test
- Spend time to make those tests run quickly
- Run the tests more often
- Use Ruby idioms to make tests quicker