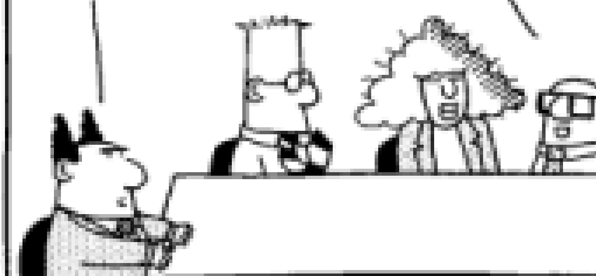
CS169 Week 5 Section

Liang (Leon) Gong

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Administrivia & Agenda

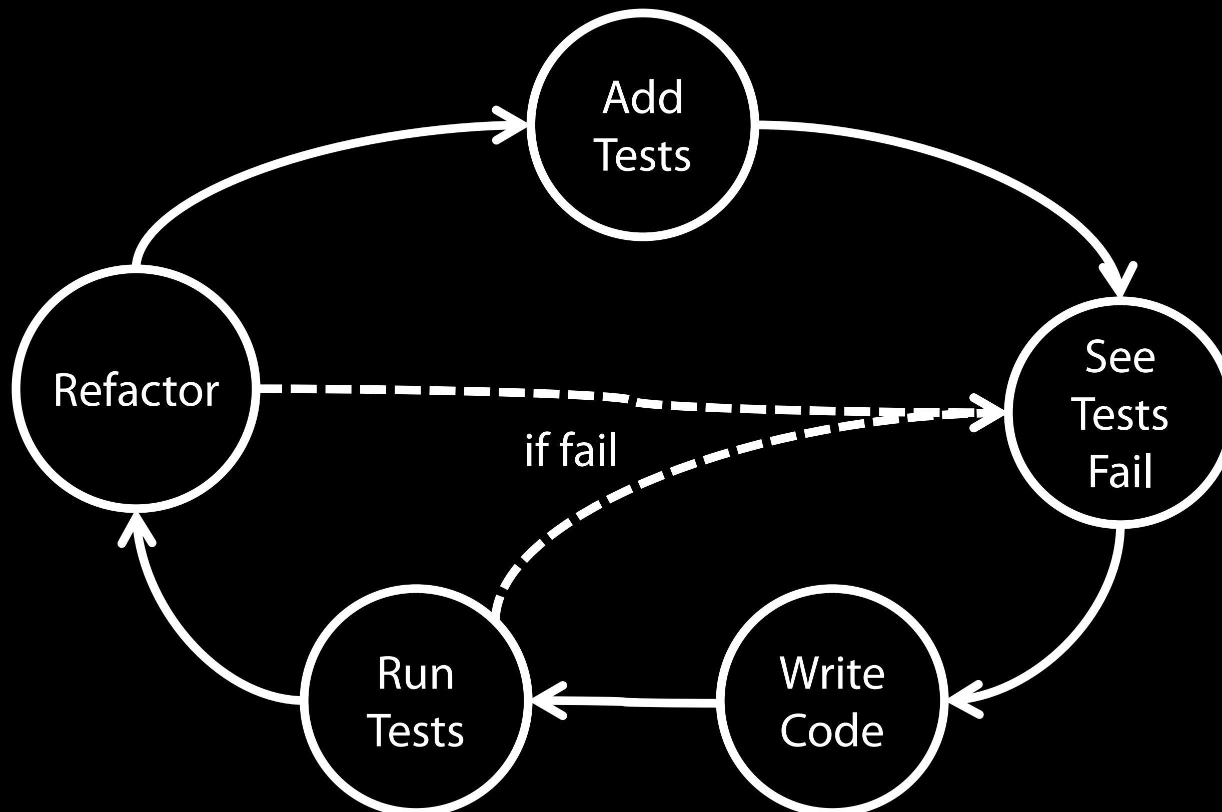
Today's Agenda: Rspec, TDD



Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

TDD: Test Driven Development

- Requirements: test cases
- Development: write code to pass the tests



Adopted from Charlies Xue's Slides

TDD Test Case Lifecycle

For each test, we have the Unit Under Test (UUT)

- Setup (change the UUT to the desired state)
  - *before(:each), before(:all)*
- Execution (run method to the UUT)
  - *stub, double, allow* etc.
- Validation (assert that the new state of the UUT matches expected behavior)
  - *should, expect, should_receive* etc.
- Teardown (clean up the database)
  - *after(:each), after(:all)*

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

TDD Test Case Lifecycle

Let's say we want to make sure our admin updates their email correctly

- Setup (create an admin)
- Execution (run `admin.update_attribute(:email, email)`)
- Validation (assert that `admin.email == email`)
- Teardown (destroy the `admin` model)

Adopted from Charlies Xue's Slides

Good Unit Tests

- Run fast
  - short setups, run times, and break downs.
- Correct & Reliable
  - Be careful with flaky test cases
  - Test cases should not corrupt actual data
- Run in isolation
  - Does not depend on each other
  - you should be able to reorder them
- Readable
  - Serves as documentation
  - Use data that makes them easy to read and to understand.

Adopted from Charlies Xue's Slides

TDD Test Case Lifecycle

For each test, we have the Unit Under Test (UUT)

- Setup (change the UUT to the desired state)
  - `before(:each), before(:all)`
- Execution (run method to the UUT)
  - `stub, double, allow` etc.
- Validation (assert that the new state of the UUT matches expected behavior)
  - `should, expect, should_receive` etc.
- Teardown (clean up the database)
  - `after(:each), after(:all)`

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

TDD Test Case Lifecycle

For each test, we have the Unit Under Test (UUT)

- Setup (change the UUT to the desired state)
  - *before(:each), before(:all)*
- Execution (run method to the UUT)
  - *stub, double, allow* etc.
- Validation (assert that the new state of the UUT matches expected behavior)
  - ***should, expect, should_receive*** etc.
- Teardown (clean up the database)
  - *after(:each), after(:all)*

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

The spec file

```
require "spec_helper"
require "movie"

describe "A Movie" do
    it "has correct title" do
        movie = Movie.new("Star Trek")
        movie.title.should == "Star Trek"
    end
end
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

The spec file

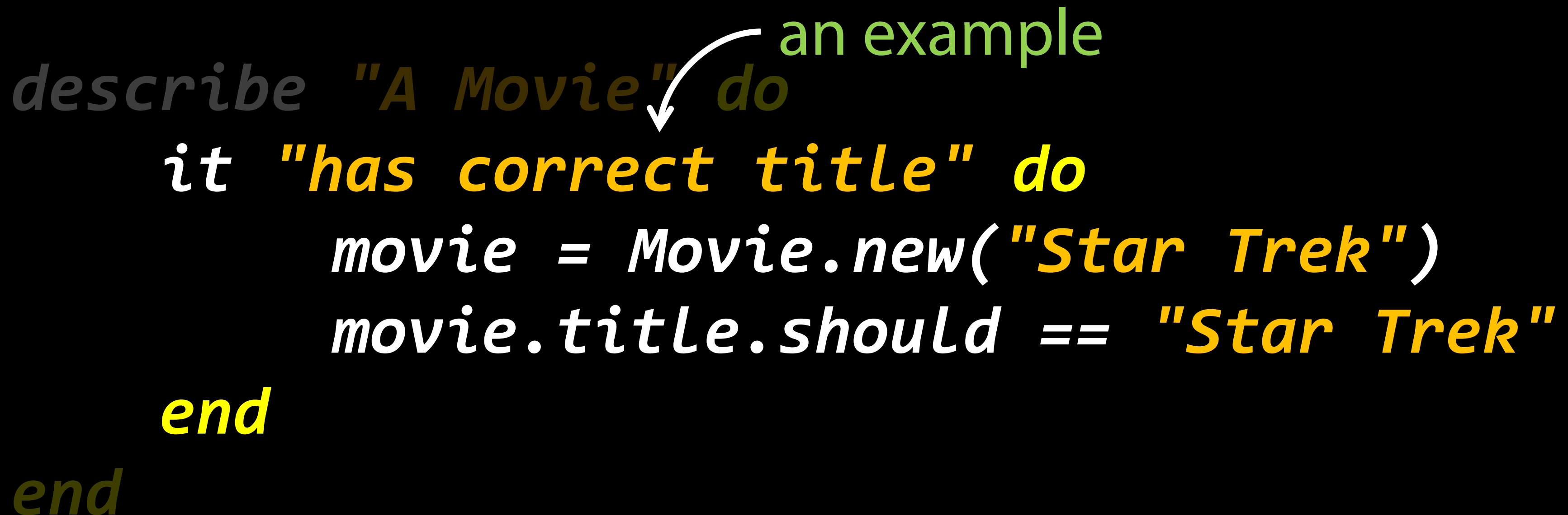
```
require "spec_helper"
require "movie"

describe "A Movie" do
    it "has correct title" do
        movie = Movie.new("Star Trek")
        movie.title.should == "Star Trek"
    end
end
```

a group of examples

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

The spec file



Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

The spec file

```
require "spec_helper"
require "movie"

describe "A Movie" do
    it "has correct title" do
        movie = Movie.new("Star Trek")
        movie.title.should == "Star Trek"
    end
end
```

- new alternative syntax
- both are readable

```
expect(movie.title).to eq "Star Trek"
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

The spec file

```ruby
require "spec_helper"
require "movie"

describe "A Movie" do
    it "has correct title" do
        movie = Movie.new("Star Trek")
        movie.title.should == "Star Trek"
    end
end
```

Readable

```ruby
movie.title.should == "Star Trek"
```

- new alternative syntax
- both are readable

```ruby
expect(movie.title).to eq "Star Trek"
```

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

The Syntactic Sugar

```
movie.title.should == "Star Trek"
→ movie.title.should.==("Star Trek")
```

- ***val.should*** returns a built-in Matcher
- its methods (e.g., ==, >, < etc.) contain assertion

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

The Syntactic Sugar

```
movie.title.should == "Star Trek"
→ movie.title.should.==("Star Trek")
```

- *val.should* returns a built-in Matcher
- its methods (e.g., ==, >, < etc.) contain assertion

```
expect(movie.title).to eq("Star Trek")
→ expect(movie.title).to(eq("Star Trek"))
```

- *expect(val)* returns an ***ExpectationTarget***
- *eq(val)* returns a ***Matcher***

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

The Syntactic Sugar

```
movie.title.should == "Star Trek"
→ movie.title.should.==("Star Trek")

movie.title.should eq "Star Trek"
→ movie.title.should(eq("Star Trek"))

expect(movie.title).to eq("Star Trek")
→ expect(movie.title).to(eq("Star Trek"))

expect(movie.title).to be == "Star Trek"
→ expect(movie.title).to(be.==("Star Trek"))
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

The Syntactic Sugar

```
movie.title.should == "Star Trek"
→ movie.title.should.==("Star Trek")

movie.title.should eq "Star Trek"
→ movie.title.should(eq("Star Trek"))

expect(movie.title).to eq("Star Trek")
→ expect(movie.title).to(eq("Star Trek"))

expect(movie.title).to be == "Star Trek"
→ expect(movie.title).to(be.==("Star Trek"))
```

TDD Test Case Lifecycle

For each test, we have the Unit Under Test (UUT)

- Setup (change the UUT to the desired state)
  - *before(:each), before(:all)*
- Execution (run method to the UUT)
  - *stub, double, allow* etc.
- Validation (assert that the new state of the UUT matches expected behavior)
  - *should, expect, should_receive* etc.
- Teardown (clean up the database)
  - *after(:each), after(:all)*

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

TDD Test Case Lifecycle

For each test, we have the Unit Under Test (UUT)

- Setup (change the UUT to the desired state)
  - *before(:each), before(:all)*
- Execution (run method to the UUT)
  - *stub, double, allow* etc.
- Validation (assert that the new state of the UUT matches expected behavior)
  - *should, expect, should_receive* etc.
- Teardown (clean up the database)
  - *after(:each), after(:all)*

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

before, after

- Execute arbitrary code before and after each example
- Control the environment of examples
  - ***before(:each)***: run before each example
  - ***before(:all)***: run once before all examples in a group
  - **Similarly,** ***after(:each)*** and ***after(:all)***

- Example:
  - use ***before(:each)*** to prepare data, and ***after(:each)*** to clean the data

before, after

```
describe "launch the rocket" do
  before(:each) do
    @rocket = Rocket.new
  end

  it "launch the rocket" do
    expect(@rocket.launch).to be_true
  end

  it "..." do
    # uses @rocket
  end
end
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

context

- ***context*** is an alias method of ***describe***
- ***describe***: wrap a set of tests against one functionality
- ***context***: wrap a set of tests against one functionality under the same state

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

context

```ruby
describe "launch the rocket" do
  before(:each) do
    @rocket = Rocket.new
  end

  context "all ready" do
    before(:each) do
      @rocket.ready = true
    end

    it "launch the rocket" do
      expect(@rocket.launch).to be true
    end
  end

  context "not ready" do ... end
end
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

TDD Test Case Lifecycle

For each test, we have the Unit Under Test (UUT)

- Setup (change the UUT to the desired state)
  - *before(:each), before(:all)*
- Execution (run method to the UUT)
  - *stub, double, allow* etc.
- Validation (assert that the new state of the UUT matches expected behavior)
  - *should, expect, should_receive* etc.
- Teardown (clean up the database)
  - *after(:each), after(:all)*

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

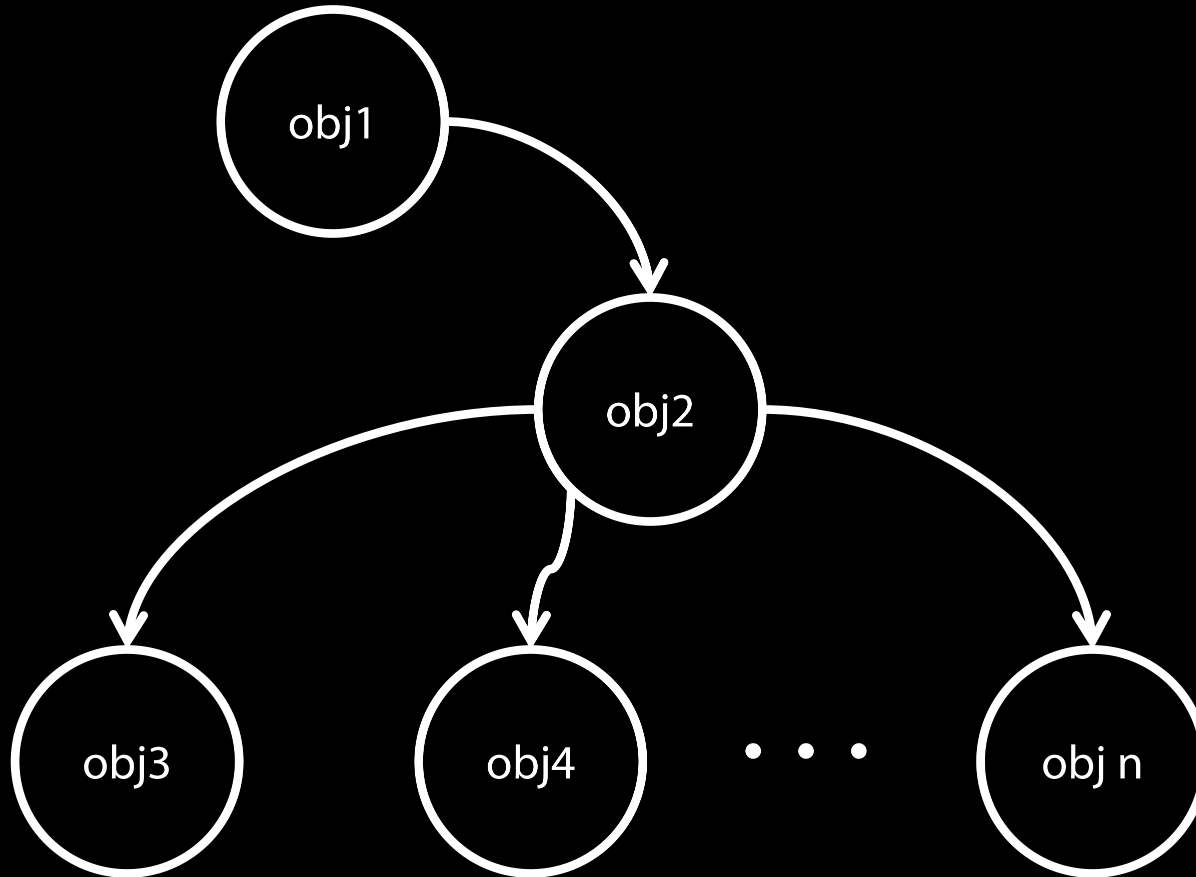
TDD Test Case Lifecycle

For each test, we have the Unit Under Test (UUT)

- Setup (change the UUT to the desired state)
  - `before(:each), before(:all)`
- Execution (run method to the UUT)
  - `stub, double, allow` etc.
- Validation (assert that the new state of the UUT matches expected behavior)
  - `should, expect, should_receive` etc.
- Teardown (clean up the database)
  - `after(:each), after(:all)`

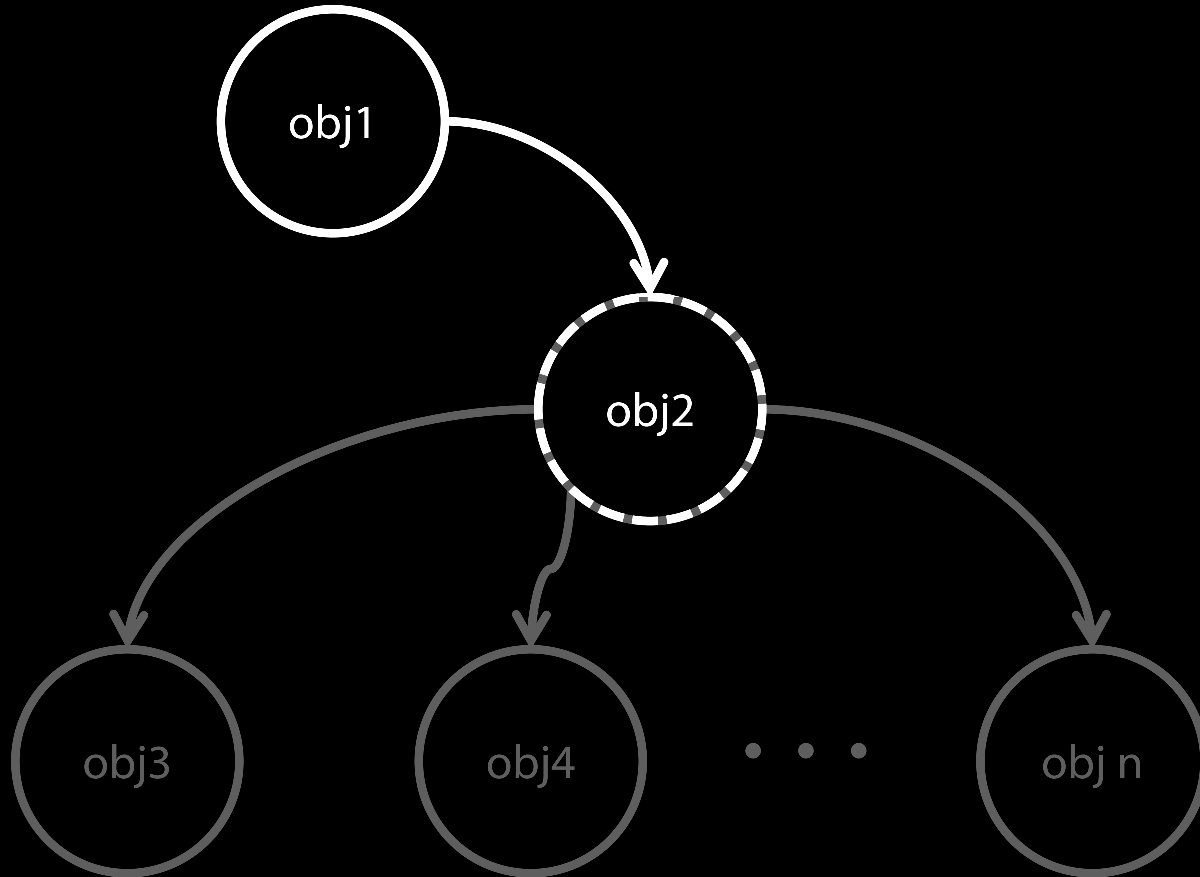
Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Mock and Stub



The state of obj3, obj4, …, obj n
may depends on network, specific events, or database

Mock and Stub



The state of obj3, obj4, …, obj n
may depends on network, specific events, or database

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Stub

Replace implementation (no assertions)

```
receiver = double("receiver")
receiver.stub(:message) { "val" }
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Stub

Replace implementation (no assertions)

- Create a dummy *receiver* object

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Stub

Replace implementation (no assertions)

- Create a dummy *receiver* object

```
receiver = double("receiver")
receiver.stub(:message) { "val" }
```

- dummy method (*receiver.message*) that returns *"val"*

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Stub

Replace implementation (no assertions)

```
receiver = double("receiver")
receiver.stub(:message) { "val" }

receiver.stub(:message).and_return("val")
allow(receiver).to(receive(:message))
                 .and_return ("val")
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Mock

Replace implementation and make assertions

```
expect(obj).to(receive(:meth))
             .with(param_obj)
             .and_return(val)
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Mock

Replace implementation and make assertions

- Create a stub for ***obj.meth()***
- Assert to be called

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Mock

Replace implementation and make assertions

- Create a stub for ***obj.meth()***
- Assert to be called

```
expect(obj).to(receive(:meth))
           .with(param_obj)
           .and_return(val)
```

- assert the parameter to be ***param_obj***

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Mock

Replace implementation and make assertions

- Create a stub for ***obj.meth()***
- Assert to be called

```
expect(obj).to(receive(:meth))
           .with(param_obj)
           .and_return(val)
```

- assert the parameter to be ***param_obj***

- define the return value
- No assertion

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Mock (*should* vs *expect*)

```
obj.should_receive(:method_name)
   .with(param_val)
   .and_return(val)

------------------------------

expect(obj).to receive(:method_name)
               .with(param_obj)
               .and_return(val)
```

Mock (*should* vs *expect*)

Remember to return a value

```
obj.should_receive(:method_name)
    .and_return(val)
    .and_call_original

--------------------------------

expect(obj).to receive(:method_name)
                .and_return(val)
                .and_call_original
```

Looks like just making an assertion,
but it also replace the implementation.

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Mock (*should* vs *expect*)

Remember to return a value

```
obj.should_receive(:method_name)
    .and_return(val)
    .and_call_original

---------------------------------

expect(obj).to receive(:method_name)
                .and_return(val)
                .and_call_original
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Mock (*should* vs *expect*)

Remember to return a value

```
obj.should_receive(:method_name)
    .and_return(val)
    .and_call_original

--------------------------------
expect(obj).to receive(:method_name)
                .and_return(val)
                .and_call_original
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

BDD & TDD

Both write tests first, then write code to pass the tests.

Behavior Driven Development

- BDD testing from the perspective of a customer.
- `Then I should see 15$ on the screen`
- Black box (test at feature level)

Test Driven Development

- TDD testing from the perspective of a developer:
- `assertEquals(price, 15)`
- White box (test at class/method level)

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Pair Programming

http://github.com/JacksonGL/CS169-URDb

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Pair Programming

http://github.com/JacksonGL/CS169-URDb

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.