

Midterm 1 Review Session

Logistics

- Midterm tonight (10/8)
 - 7 - 9pm, GBP 100
- HW 4 peer reviews due 10/11 11:59pm
- HW 5 due 10/13 11:59pm

Agenda

1. Testing (BDD/Cucumber)
2. Sinatra
3. Rails (MVC/ActiveRecord)
4. Ruby
5. REST/URIs
6. Software Development Cycles
7. Q&A

Testing

Testing

Program testing can be used to show the presence of bugs, but never to show their absence!

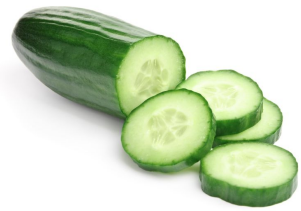
- Edsger Dijkstra

BDD

Behavior-driven design asks questions about **behavior of the app** (not implementation) before and during development to reduce miscommunication between stakeholders.

One way to test is with **Cucumber**.

Cucumber example



Step 1: Describe your feature's behavior in plain English.

Feature: Addition

As a math student

So that I can avoid mistakes

I want to be given a sum of two numbers.

Scenario: Add 2 numbers

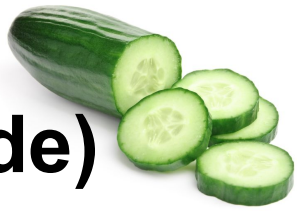
Given I have entered 2 into the calculator

Given I have entered 7 into the calculator

When I press add

Then the result should be 12

Step 2: Step definition (regex + real code)



```
Given /I have entered (.*) into the calculator/ do |n|  
  calculator = Calculator.new  
  calculator.push(n.to_i)  
end
```


Debugging

R: Read the error message

A: Ask an informed question

P: Post online (StackOverflow, Piazza)

(or)

S: Search the web (Google, StackOverflow)

Ways to Debug

Instrumentation: print things.

Stop the show: raise the object in question as an exception, view the exception page generated by Rails.

Print to log: use `logger.debug(msg)` to print to log

Use a debugger: set breakpoints and examine the state of your app at any time

Sinatra

Sinatra



Ruby domain-specific-language for building web applications.

Example

```
# app.rb  
require 'sinatra'  
  
get '/' do  
  'Hello world!'  
end
```



Example

```
# app.rb  
require 'sinatra'  
  
get '/' do  
  'Hello world!'  
end
```

```
$ ruby app.rb # starts on localhost:4567
```



Example



```
# app.rb  
require 'sinatra'
```

```
get '/' do  
  'Hello world!'  
end
```

```
$ ruby app.rb # starts on localhost:4567
```

```
$ curl localhost:4567 # returns:  
Hello there!
```

POST



```
# app.rb
require 'sinatra'

post '/data' do
  params.to_s
end
```

```
$ curl -d "hello=there" -X POST localhost:4567/data # returns ?
```


POST



```
# app.rb
require 'sinatra'

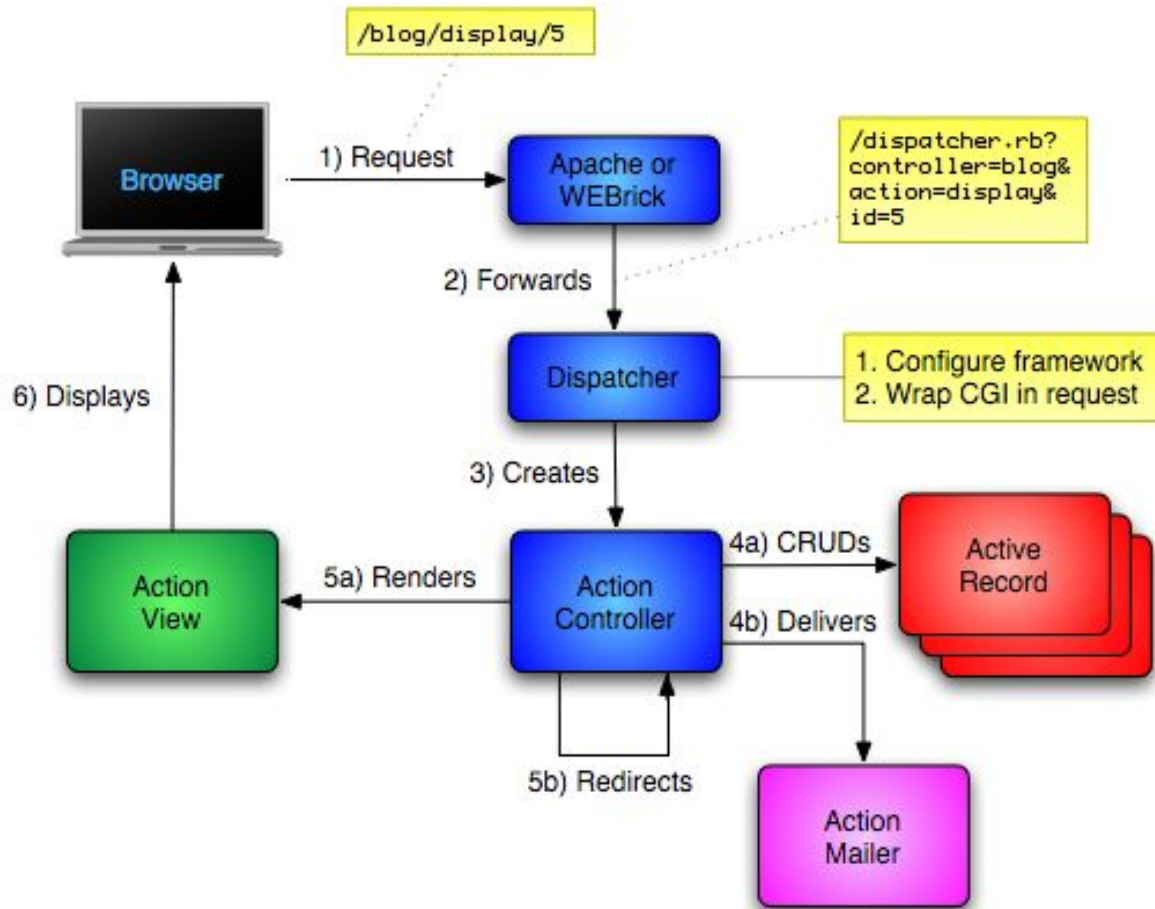
post '/data' do
  params.to_s
end
```

```
$ curl -d "hello=there" -X POST localhost:4567/data # returns ?
{"hello"=>"there"}
```

Rails

MVC, ActiveRecord

MVC



MVC

Model: methods to get/manipulate data (ActiveRecord).

`Movie.where(..), Movie.find(..)`

Controller: get data from Model, make available to View.

```
def show
```

```
  @movie = Movie.find(params[:id])
```

```
end
```

View: display data, allow user interaction (`*.erb`).

ex: show details of a movie (description, rating)

Rails MVC

Model: app/models/hangperson.rb

Controller: app/controllers/game_controller.rb

View:

- app/views/game/**new**.html.erb
- app/views/game/**show**.html.erb
- app/views/game/**win**.html.erb
- app/views/game/**lose**.html.erb

ActiveRecord

Automated mapping between classes and tables, attributes and columns

Basic operations on object: CRUDI
(Create, Read, Update, Delete, Index)

Acts a bridge between memory and database

AR Example

```
class Article < ActiveRecord::Base {  
  :id    => :integer,  
  :title => :string,  
  :content => :text  
}
```

AR Example

```
a = Article.new
```

```
a.title = "Week 5"
```

```
a.save
```

id	title	content
1	First record	Hello world
2	Week 3 section	Active record etc.
3	Week 4 section	Rails etc.

Ruby

Everything is an Object

1 + 2


Everything is an Object

1 + 2  1.send(:+, 2)

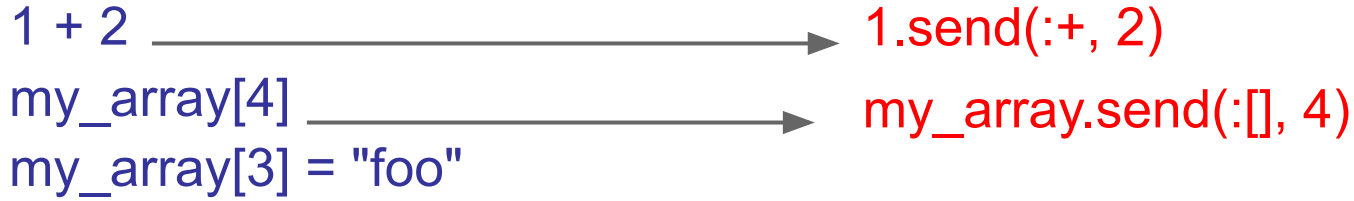
Everything is an Object

1 + 2  1.send(:+, 2)
my_array[4]


Everything is an Object


1 + 2  1.send(:+, 2)
my_array[4]  my_array.send(:[], 4)


Everything is an Object



Everything is an Object

1 + 2  1.send(:+, 2)


my_array[4]  my_array.send(:[], 4)


my_array[3] = "foo"  my_array.send(:[]=, 3, "foo")


Everything is an Object


1 + 2 → 1.send(:+, 2)
my_array[4] → my_array.send(:[], 4)
my_array[3] = "foo" → my_array.send(:[]=, 3, "foo")
if (x == 3)

Everything is an Object


1 + 2  1.send(:+, 2)


my_array[4]  my_array.send(:[], 4)


my_array[3] = "foo"  my_array.send(:[]=, 3, "foo")


if (x == 3)  if (x.send(:==, 3)) ...


Everything is an Object

1 + 2  1.send(:+, 2)

my_array[4]  my_array.send(:[], 4)

my_array[3] = "foo"  my_array.send(:[]=, 3, "foo")

if (x == 3)  if (x.send(:==, 3)) ...

my_func(z) 

Everything is an Object

1 + 2 → 1.send(:+, 2)

my_array[4] → my_array.send(:[], 4)

my_array[3] = "foo" → my_array.send(:[]=, 3, "foo")

if (x == 3) → if (x.send(:==, 3)) ...

my_func(z) → self.send(:my_func, z)

Getter/Setter

```
def balance
  @balance
end
```

```
def balance=(new_amount)
  @balance = new_amount
end
```

Getter/Setter

```
def balance  
  @balance  
end
```



```
attr_accessor :balance
```

```
def balance=(new_amount)  
  @balance = new_amount  
end
```

REST & URIs

REST

REpresentational **S**tate **T**ransfer (2000)

Focuses on performing operations on application resources

Everything is a resource, with different possible representations (JSON/HTML)

Multiple HTTP verbs, most common is GET

Others include POST, PATCH, UPDATE, DELETE

URIs

Uniform Resource Identifier

Useful for interacting with online resources (e.g., images, stylesheets, form submission targets)

Examples:

https://en.wikipedia.org/wiki/Uniform_Resource_Identifier#Examples

Example

<https://www.etsy.com:443/search?q=test#copy>

Example

- https
- www.etsy.com
- 443
- /search
- ?q=test
- #copy

Example

- https - **scheme**
- www.etsy.com - **hostname**
- 443 - **port**
- /search - **path**
- ?q=test - **query string**
- #copy - **fragment**

Software Development Cycles

Waterfall

1. Requirements analysis & specification
2. Architectural design
3. Implementation & integration
4. Verification
5. Operation & maintenance

What was the problem with Waterfall?

Spiral

Built prototypes in each iteration

Plans and documents evolve with changes to product

What doesn't this work for?

Agile

1. **Individuals and interactions** over processes & tools
2. **Working software** over comprehensive documentation
3. **Customer collaboration** over contract negotiation
4. **Responding to change** over following a plan.

Differences

1. Is specification required?
2. Are customers unavailable?
3. Is the system to be built large?
4. Is the system to be built complex (e.g., real time)?
5. Will it have a long product lifetime?
6. Are you using poor software tools?
7. Is the project team geographically distributed?
8. Is team part of a documentation-oriented culture?
9. Does the team have poor programming skills?
10. Is the system to be built subject to regulation?

Testing in P&D vs Agile

How would you expect testing in Agile to be different from testing in P&D models?

Testing in P&D vs Agile

How would you expect testing in Agile to be different from testing in P&D models?

Agile involves constantly testing for every iteration, even before code is written.

P&D starts testing after implementation

Testing in P&D vs Agile

What else is different?

Testing in P&D vs Agile

What else is different?

In P&D expensive formal methods (human or computer) can be used to prove that code follows specification.

Testing in P&D vs Agile

What else is different?

In P&D developers write unit tests, but dedicated QA developers write module, integration, system, and acceptance tests.

Q&A

Unused

Rational Unified Process

4 Phases (*can iterate*)

1. Inception: business case
2. Elaboration: use cases, architecture, prototype
3. Construction: implement + test
4. Transition: move to production environment; get customer acceptance

Good in that it combines business case

Poetry Mode

A way to reduce clutter by removing curly braces and omitting parentheses around unambiguous method calls (especially hashes).

Poetry Mode: Example

(redirect_to(login_page)) and return() unless logged_in?

redirect_to login_page and return unless logged_in?

Cucumber vs. Rspec



Cucumber: (integration testing) for higher level tests describing functionality as viewed from the user's perspective.

RSpec: (unit testing) for lower-level tests describing details for how your classes, methods, models, controller, etc. actually work.

TDD/Rspec

TDD is for verification (trying to get working code)

Rspec is a DSL used for testing

Extensive use of seams and

metaprogramming for testing (more later)

Expectations check whether object state matches what you expect

An example:

```
describe "ItemController" do
  describe "POST #add_to_cart" do
    context "when logged in" do
      it "should render the cart page" do
        @item = Item.create!(:name => "Swiffer", :price => 35.00)
        post :add_to_cart, :item_id => @item.id
        response.should redirect_to(cart_path(current_user))
      end
      ...
    end
    context "when not logged in" do
      it "should redirect to the login page" do
        @item = Item.create!(:name => "Swiffer", :price => 35.00)
        post :add_to_cart, :item_id => @item.id
        response.should redirect_to(login_path)
      end
      ...
    end
  end
end
```

An example:

```
describe "ItemController" do
  describe "POST #add_to_cart" do
    context "when logged in" do
      it "should render the cart page" do
        @item = Item.create!(:name => "Swiffer", :price => 35.00)
        post :add_to_cart, :item_id => @item.id
        response.should redirect_to(cart_path(current_user))
      end
      ...
    end
    context "when not logged in" do
      it "should redirect to the login page" do
        @item = Item.create!(:name => "Swiffer", :price => 35.00)
        post :add_to_cart, :item_id => @item.id
        response.should redirect_to(login_path)
      end
      ...
    end
  end
end
```

An example:

```
describe "ItemController" do
  describe "POST #add_to_cart" do
    before(:each) do
      @item = Item.create!(:name => "Swiffer", :price => 35.00)
    end
    context "when logged in" do
      it "should render the cart page" do
        post :add_to_cart, :item_id => @item.id
        response.should redirect_to(cart_path(current_user))
      end
    end
    context "when not logged in" do
      it "should redirect to the login page" do
        post :add_to_cart, :item_id => @item.id
        response.should redirect_to(login_path)
      end
    end
    ...
  end
end
end
```

Seams

Isolate behavior of code that interacts with other pieces of code (i.e. client/server) should_receive, stub, and mocks

Seams enable just enough functionality for some *specific* behavior under test

Why would you want to use these?

Fixtures/Factories/Misc

Fixtures: static data that is loaded into the tests

Factories: Helper methods that make it easy to create objects dynamically for individual tests optionally with default field values (i.e.

```
FactoryGirl.create!(:user))
```

When would you use a fixture? A factory?

Questions about Rspec?

What makes a good test?

Automatic : Invoking of tests as well as checking results for PASS/FAIL should be automatic

Thorough: Coverage; Although bugs tend to cluster around certain regions in the code, ensure that you test all key paths and scenarios.. Use tools if you must to know untested regions

Repeatable: Tests should produce the same results each time.. every time. Tests should not rely on uncontrollable params.

Independent: Very important.

Tests should **test only one thing** at a time. Multiple assertions are okay as long as they are all testing one feature/behavior. When a test fails, it should pinpoint the location of the problem.

Tests **should not rely on each other** - Isolated. No assumptions about order of test execution. Ensure 'clean slate' before each test by using setup/teardown appropriately

Scrum

Two pizza team

ScrumMaster: remove obstacles; keep team focused; enforce code style

Product Owner: liaison between the customer and the team

Daily scrum meeting; informal meetings

2-4 week sprints