Test-driven development (TDD)
1. step definitions for new story, may require new code to be written
2. TDD says: write unit & function tests

Spec tests individual modules that contribute to those behaviors

Unit tests should be FIRST
Fast: run tests quickly
Independent: no tests depend on others, so can run any subset in any order
Repeatable: run N times, get same result
Self-checking: test can automatically detect if passed
Timely: written about the same time as code under test

RSpec, a Domain-Specific Language for testing
Run the tests in one file:spec filename

x = Math.sqrt(9)
expect(x).to eq 3
expect(sqrt(9)).to be_within(.5).of(3)
expect(x.odd?).to be_true
=>expect(x).to be_odd
expect(m).to be_a_kind_of Movie
expect {m.save!}.to
raise_error(ActiveRecord::RecordInvalid)
expect {m.save!}.to change{Move.count}.by(1)

Matchers to test Fails apps' behaviors
expect(response).to
render_Template("movies/index")

calling TMDb
When I fill in "Search Terns" with "Inception"
And I press "Search TMDb"
Then I expect to be on

The Code You Wish You Had
What should the controller method do that receives the search form
1. call a method that will search TMDb for specified movie
2. if match found: select (new) "Search Results" view to display match

TDD for the Controller action:
Add a route to config/routes.rb
post '/movies/search_tmdb'
Create an empty view
touch app/views/movies/search_tmdb.html...

Seams
A place where you can change app's behavior without changing source code
Useful for testing: isolate behavior of some code from that of other code it depends on

should_receive uses Ruby's open classes to create a seam

expect(5).to be <=> result is wrong!! return three things
Checking for rendering
New seam concept:stub
Movie.stub(:find_in_tmdb)
expect(obj).to_receive(a).with(b).and_return()
class-or-instance.stub(method).and_return(value)

expect(obj).to match-condition
Rails-specific matcher:
response()
render_template()

Seam: generic temporally modify the code to test
it's a place where you can change the behavior of your application without changing the source code.
3.
Mock: a class that implements an interface and allows the ability to dynamically set the values to return/exceptions to throw from particular methods and provides the ability to check if particular methods have been called/not called.
Stub: Like a mock class, except that it doesn't provide the ability to verify that methods have been called/not called.

Test techniques we know:
3. expect(obj).to_receive(a).with(b).and_return().stub (method).and_return(value)
4. expect(obj).to match-condition

Rails-specific matchers:
response()
render_template

What kinds of tests:
4. Unit (one method/class) runs fast High coverage Fine resolution many mocks;
5. Functional or module (a few methods classes)
6. Integration/system Few mocks; tests interfaces Run slows
7. Mutation testing: if introduce deliberate error in code, dos some test break
8. Fuzz testing: 10,000 monkeys throw random input
9. Black-box vs. white-box/glass-box

Drying out MVC
Don't repeat yourself
cross-cutting concerns: logically centralized but may appear multiple places in implementation

Aspect-Oriented Programming
Advice is a specific piece of code that implements a cross-cutting concern
Point cuts are the places you want to inject advice at runtime

Validations
AOP: where are the point cuts
Controller filters: can change the flow of the execution
-by calling redirect_to or render

adjust to the rspec by using stub

Associations & Foreign Keys
Simple model
Cartesian Product
*foreign key in one table refers to the primary key of another table

ActiveRecord Associations
- allows manipulating DB-managed associations more Rubyistically
- After setting things up correctly, you don't have to worry about keys and joins

class Movie < ActiveRecord::Base
  has_many :reviews
end
class Review < ActiveRecord::Base
  belongs_to :movie
end

reviews table gets a foreign key field that has primary key f Movie a review is for
Dereference movie.reviews == database join (lazily) to find reviews
Dereference review.movie == lookup

```
class AddReviews < ActiveRecord::Migration
  def self.up
    create_table :reviews do |t|
      t.integer 'potatoes'
      t.references 'movie'
      t.references 'moviegoer'
    end
  end
end
```

@movie.reviews #Enumerable of reviews
@review.movie # what movie is reviewed
- You can add new reviews for a movie:

```
@movie = Movie.where("title='Fargo'")
@movie.reviews.build(:potatoes => 5)
@movie.reviews.create(:potatoes => 5)
  # how are these different from just new() & create()?
@movie.reviews.where(:potatoes => 5)
  # returns an Enumerable, just like Movie.where()
```

build a object lazily

Many-to-many associations
many-to-many
moviegoer<-reviews->movies
moviegoer:has_many :reviews
movie: has_many :reviews
review : belongs_to ...
belongs_to ...

RESTful Routes for Associations
in config/routes.rb:
resources :movies
becomes
resources :movies do
  resources :reviews
end

```
def create
  @movie = Movie.find(params[:movie_id])
  @review = @movie.reviews.build(params[:review
  ...
end

def new
  @movie = Movie.find(params[:movie_id])
  @review = @movie.reviews.new
end
```

Views

```
%h1 Edit

= form_tag movie_review_path(@movie,@review) do |f|

 ...Will f create form fields for a Movie or a Review?

  = f.submit "Update Info"

= link_to 'All reviews for movie', movie_reviews_path(@movie)
```

Convention over configuration (also known as coding by **convention**) is a software design paradig which seeks to decrease the number of decisions th developers need to make, gaining simplicity, and nc necessarily losing flexibility.

DRYing Out Queries with Reusable Scopes
Scopes

```
class Movie < ActiveRecord::Base
  has_many :reviews

  scope :for_kids, :conditions =>
     ['rating IN (:ratings)', :ratings => %w(G PG)

  scope :with_good_reviews, lambda { |cutoff|
    joins(:reviews).
      group(:movie_id).
      having("AVG(reviews.potatoes) > #{cutoff}")
    }
end

Movie.for_kids
Movie.for_kids.with_good_reviews(4)
Movie.with_good_reviews(4).for_kids
```

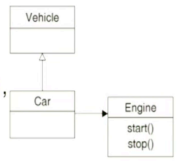
Associations wrap-up
Referential Integrity
Various possibilities depending on app
-delete those reviews?
has_many :reviews, :dependent => :destroy
orphaned?
has_many :reviews, :dependent => :nullify

Associations wrap-up
DataMapper

CRC Cards and UML

Unified Modeling Language:notation for describing various artifacts in OOP systems

- Car is a subclass of Vehicle
- Engine is a *component* of Car
- Engine class includes start(), stop() methods



CRC Cards

Ticket	
Responsibilities	*Collaborators*
Knows its price	
Knows which showing it's for	Showing
Computes ticket availability	
Knows its owner	Patron

Effective Branching part1: Branch Per Feature
keep branch short

Design Reviews(meeting where authors present design), Code Reviews(held after design implemented P), Plan-And-Document
Effective Branching part2: Branches & Deploying

SAMOSAS: (Start and stop meeting promptly/ Agend a created in advance; no agenda, no meeting/ Minute s recorded so everyone can recall results/ One speak er at a time; no interrupting talker/ Send material in a dvance, since reading is faster/ Action items at end of meeting, so know what each should do as a result of the meeting/ Set the date and time of the next meetin g)

Bugs, Teams, Legacy intro
Fixing Bugs: The Five R's
No Bug Fix Without a Test
5. Report
6. Reproduce the bug
7. Regression test
8. Repair
9. Release the fix (commit and deploy)

Even in non-agile organizations
Report
Pivotal Tracker
Github "issues" feature
Bugzilla

Reclassify as "not a bug"
Repair == test fails in presence of bug passes in absence of bug

Exploration: determine where you need to make changes
Refactoring: is the code around change points tested? testable?

Approach the legacy code
Get the code running in development
Learn the user stories
Inspect database schema
model interaction diagram automatically
gem install railroady

Establishing Ground Truth with **Characterization Test**

Code Smells
SOFA captures symptoms that often indicate
is it **short**
does it do **one thing**
does it have **few arguments (otherwise hard to cover, hard to testing)**
is it a consistent feel of **abstraction**?

Plan-And-Document Perspective on Software Maintenance:
1/3 on development 2/3 on maintenance

Intro to Method-Level Refactoring
Quantitative: Metrics
Code-to-test ratio rake stats
C0 coverage SimpleCov
Assignment-Branch-Condition score
Cyclomatic

Refactoring: Idea
Start with code that has 1 or more problems/smells
Through a series of small steps, transform to code from which those smells are absent
Protect each step with tests
Minimize time during which tests are red

Report (Pivotal Tracker/Github Issues/ Bugzilla)/Repr oduce(With simplest possible test-minimize preconditions)or Reclassify (To "won't be fix ed" or "it's not a bug, it's a feature")/Regression test (Add after reproduction; check that old code still works with bug fix)/ Repair / Release fix

Lines of Test/ Lines of Code ~= 1.2 - 2.0
Coverage Levels:
-S0: every method called
-S1: every method from every call site
-C0: every statement (Ruby SimpleCov gem)
-C1: every branch in both directions
C1+decision coverage: every subexpression in condit ional
C2: every path (difficult, and disagreement on how va luable)

JavaScript: the Big Picture
A closure is a special kind of object that combines two things: a function, and the environment in which that function was created.
Lexical scoping

```
function init() {
  var name = "Mozilla"; // name is a local variable creat ed by init
  function displayName() { // displayName() is the inn er function, a closure
    alert (name); // displayName() uses variable decla red in the parent function
  }
  displayName();
}
init();
```

Subjective bias disclaimer:
JavaScript has unnecessary bad rep
Presence of JS should enhance app;
jQuert is the way to do client-side JS enhancements
Jasmine/TDD just as important for JS as for server
Jury still out on server-side JS

History:
Document Object Model lets JavaScript inspect modify document elements
2005: Google Maps, AJAX takes off

Use cases for JS today
Single-page apps
-Pivotal Tracker: still heavy server support
Standalone rich client apps (Google Docs)
Server-Side apps (Node.js)
Graceful degradation for SaaS-centric apps
JS, browsers, and scope
-**this** in global scope refers to global object
-Each HTML page gets its own JS global object
usually in application.html.haml </script src="">
Why use helper vs explicit tag? asset pipeline
Beautiful JavaScript?
Keep code separate from HTML
Red-Green-Refactor with Jasmine
CodeClimate now supports JavaScript
DOM&JavaScript

- DOM is a language-independent, hierarchical representation of HTML or XML document
- JavaScript API makes DOM data structures accessible from JS code
- JavaScript is just another language
- JavaScript is a single-threaded language.
- closures/scheme embedded

Summary:
1. be triggered by uyse
2. make http requests to server without triggering page reload

Q1: Client-side JavaScript code can interact with HTML page elements because this functionality:
is not part of the JavaScript

JavaScript basics

- no class
- prototypal inheritance
- Basic object type is a hash var movie = {title: 'U "releaseInfo" : {rating: 'G', year:2010}}
- Use JavaScript console window in modern browsers
- Variable Scope
  - var restricts scope of a variable
  - Functions are 1st class objects and closures
  - var MyApp = {foo1 : function() {…}}
- JS, browsers, and scope
- Functions are closures
- var make_times = function(mul) { return function(arg) {return arg * mul;}}
- Functions as properties
  - RP = {}
  - RP.setup = function() {};
- functions in JavaScripts? They can not execute concurrently with other function
- Every object has a prototypes
- Array.prototype.zip = function(other) {}

```
var Square = function(side) {
    this.side = side;
this.area = function() {return this.side*this.side}
    }
};
var p = Squre;
(new p(3)).area()
```

DOM Example

```
<p id = "notice">
</p>
e = document.getElementById('notice');
```

can always navigate the children

jQuery
A powerful framework for DOM manipulation
-adds many useful features over browsers' built in JSAPI
-homogenizes incompatible JSAPI's across browsers
Don't forget to load it
Defines a single polymorphic global function

$() or jQuery() with any CSS3 expression
document.getElementById
= $(window.document).find(selector)
Create element
var newDiv = $('<div class = 'main'));
Resulting elements have jQuery superpower on element

javaScript hack on eBay ={[]}+

Events and Callback
What:occurrences that affect the user interface
Events are usually associated with a DOM element
Bind element
2. Identify elements you will want to do stuff
3. Similarly, identify elements on which interaction will trigger stuff to happen
4. Write handler functions that cause the desired stuff to happen
5. In a setup function, bind the handlers

```
var hideTest = {  hidePara: function() { ... }
```