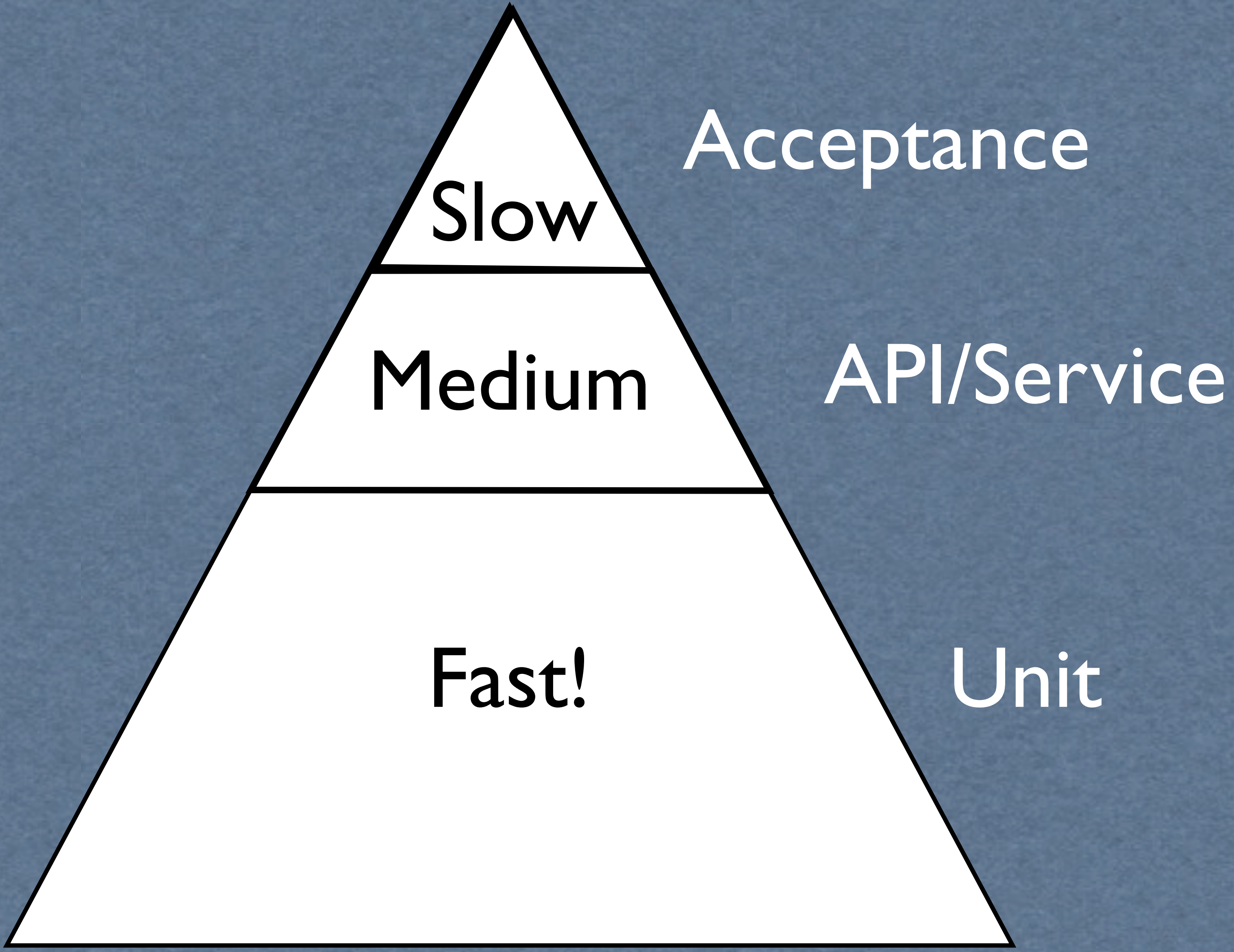
Proper Testing:

How what you test and how you test it can make better software and faster tests.

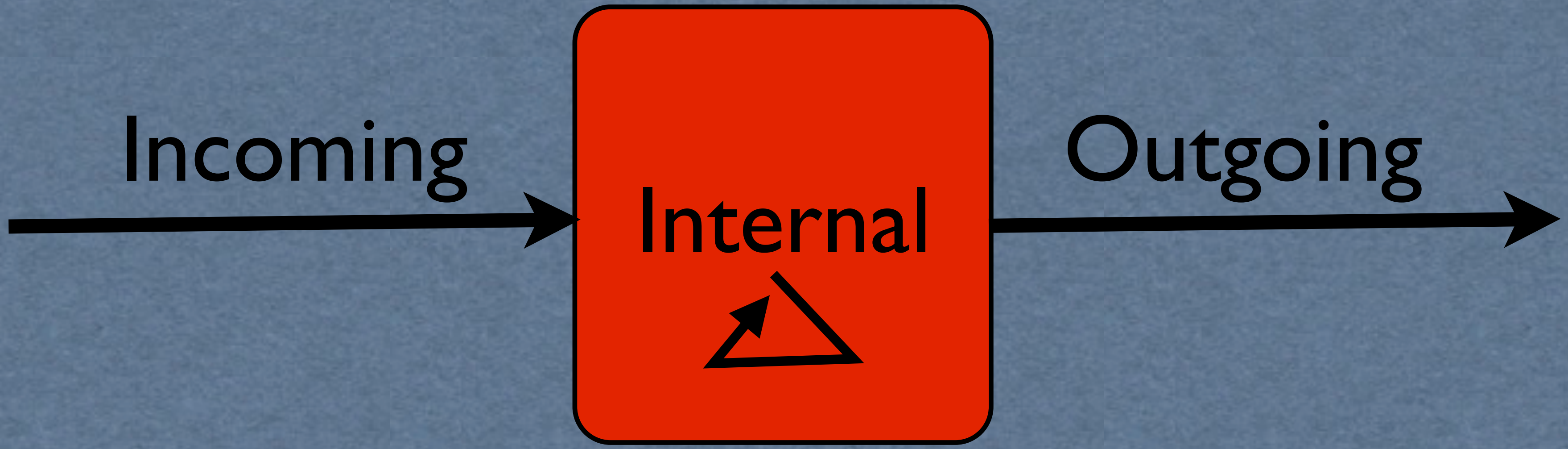
Behavior Driven Development (BDD)

Test what your software does,
NOT how it does it.

Types of Tests



Object Oriented Programming is not so much about objects, but rather the messages they send to each other.



There are two kinds of messages:

Informational or Query

Modification or Command

	Query	Command
Incoming	🟩	🟩
Outgoing	🟥	🟩
Internal	🟥	🟥

Some Guidelines

- Don’t use factories
- Don’t persist your models*
- Your models should contain
  - Relationships
  - Validations
  - Simple Query Methods
- Use a real object for the object under test
- Use mocks for all collaborators
- Inject your collaborators when possible
- Wrap external services/APIs

```
describe Car do
  subject(:car) { Car.new(color, fuel_tank) }
  let(:fuel_tank) { double('fuel_tank', fuel: fuel, burn: nil) }
  let(:color) { nil }
  let(:fuel) { nil }

  context 'when the car is red' do
    let(:color) { :red }
    specify { expect(car).to be_popular }
  end

  context 'with a tank with 5 gallons of fuel' do
    let(:fuel) { 5 }
    specify { expect(car.range).to eq(100) }
  end

  context 'when driving 20 miles' do
    let(:fuel) { 1 }
    it 'increases the odometer by 20' do
      expect {car.drive(20)}.to change(car, :odometer).by(20)
    end
    it 'burns 1 gallon of fuel' do
      fuel_tank.should_receive(:burn).with(1)
      car.drive(20)
    end
  end
end
```

```ruby
class Car
  attr_reader :color, :fuel_tank, :odometer

  def initialize(color, fuel_tank=FuelTank.new)
    @color = color
    @fuel_tank = fuel_tank
    @odometer = 0
  end

  def popular?
    color == :red
  end

  def range
    fuel_tank.fuel * 20
  end

  def drive(miles)
    @odometer += miles
    fuel_tank.burn(miles / 20)
  end
end
```

```
class Meeting < ActiveRecord::Base

  def can_give_kudo?(user)
    topics.none?{ |topic| topic.given_kudo?(user) }
  end

end
```

```ruby
describe Meeting do

  subject(:meeting) { Meeting.prototype(on_date) }
  let(:user) { create(:user) }
  let!(:presenter_one) { create(:user, :name => "Russ Smith") }
  let!(:presenter_two) { create(:user, :name => "Gabe Evans") }
  let!(:presenter_three) { create(:user, :name => "Judd Lillestrand") }
  let(:on_date) { Date.today }
  let(:topic1) { create(:topic) }
  let(:topic2) { create(:topic) }
  let(:topic3) { create(:topic) }

  before do
    meeting.time_slots.each_with_index do |ts, idx|
      ts.presenter = User.all[idx]
      ts.topic = send("topic#{idx+1}")
    end
    meeting.save
    meeting.time_slots.each_with_index do |ts, idx|
      send("topic#{idx+1}").update_attribute(:meeting_id, meeting.id)
    end
    meeting.reload
  end

  describe '#can_give_kudo?' do
    context 'when the user has given a kudo' do
      before { topic1.give_kudo_as(user) ; meeting.reload }

      specify { expect(meeting.can_give_kudo?(user)).to be_false }
    end
  end
end
```

```ruby
describe Meeting do

  subject(:meeting) { Meeting.new }
  let(:user) { double('user') }
  let(:topic_with_kudo) { double('topic', given_kudo?: true) }
  let(:topic_without_kudo) { double('topic', given_kudo?: false) }

  before { meeting.stub(:topics).and_return(topics) }

  describe '#can_give_kudo?' do
    context 'when the user has given a kudo' do
      let(:topics) { [topic_with_kudo] }

      specify { expect(meeting.can_give_kudo?(user)).to be_false }
    end

    context 'when the user has not given a kudo' do
      let(:topics) { [topic_without_kudo] }

      specify { expect(meeting.can_give_kudo?(user)).to be_true }
    end
  end
end
```

QUESTIONS?