CS169 Week 9 Discussion

Credit to Hezheng Yin, CS 169 Sp16

Announcements

- HW 7 due **tonight @ 11:59pm**
- Midterm is next week **11/5 from 7-9pm in Moffitt 101/Kroeber 160**

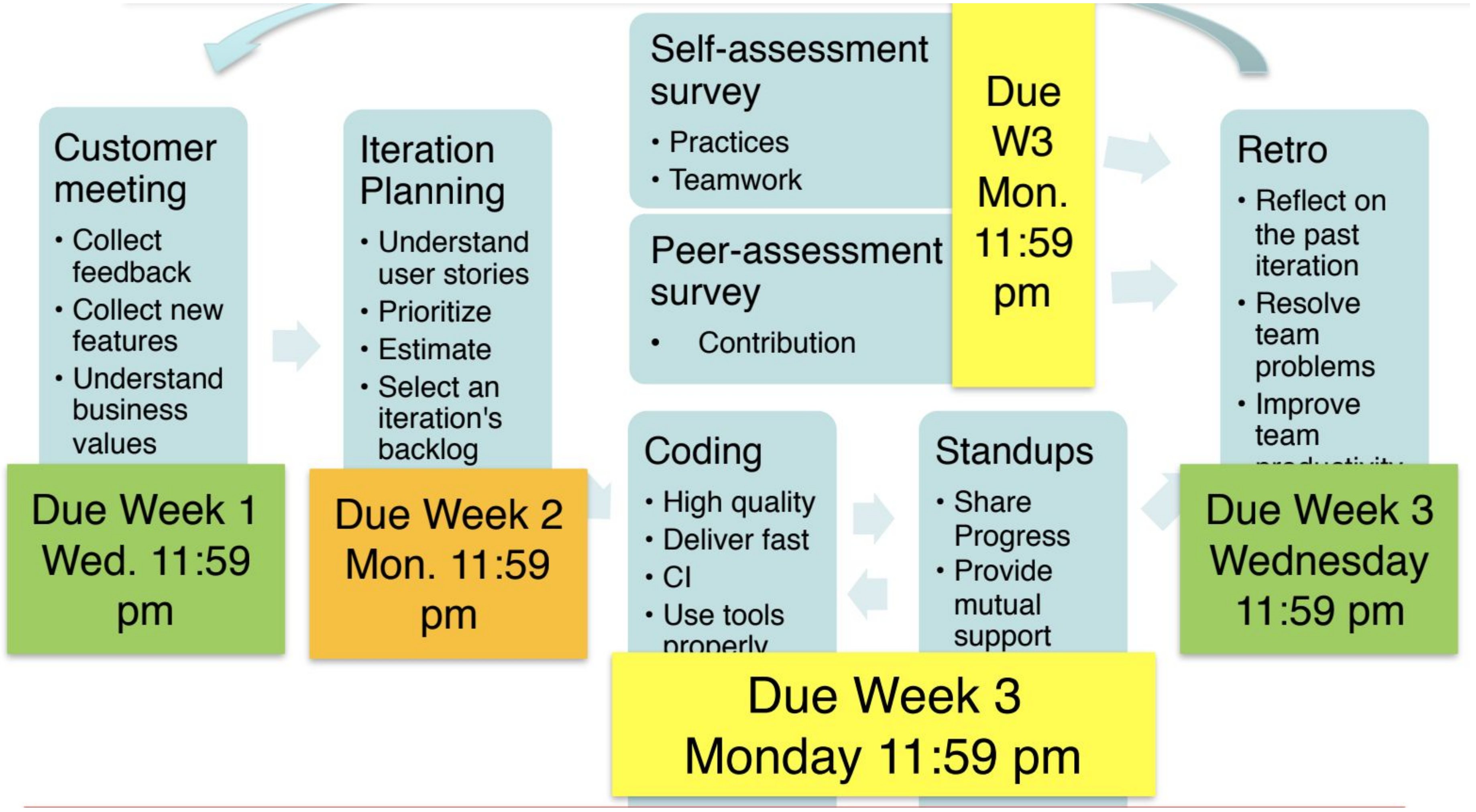
Project-Related Announcements - Check Piazza (pinned post @326)

- IPM & Customer Meeting checkpoints due **tomorrow @ 11:59pm**
- You should be having **>2 standups per week** (preferably in Slack)

Project Expectations

- **Check Piazza (pinned post @326)**
- Use the scaffolded **tools** for standup, IPM, and retro
  - Makes it easy for us and you!
  - If not, take detailed notes/record each meeting and send to your GSI
- Keep up with **bCourses checkpoints**
  - one for each meeting, self assessment, and peer review
- Any problems, reach out in your team's **Slack channel**

Typical Schedule (slightly different for Iter1)

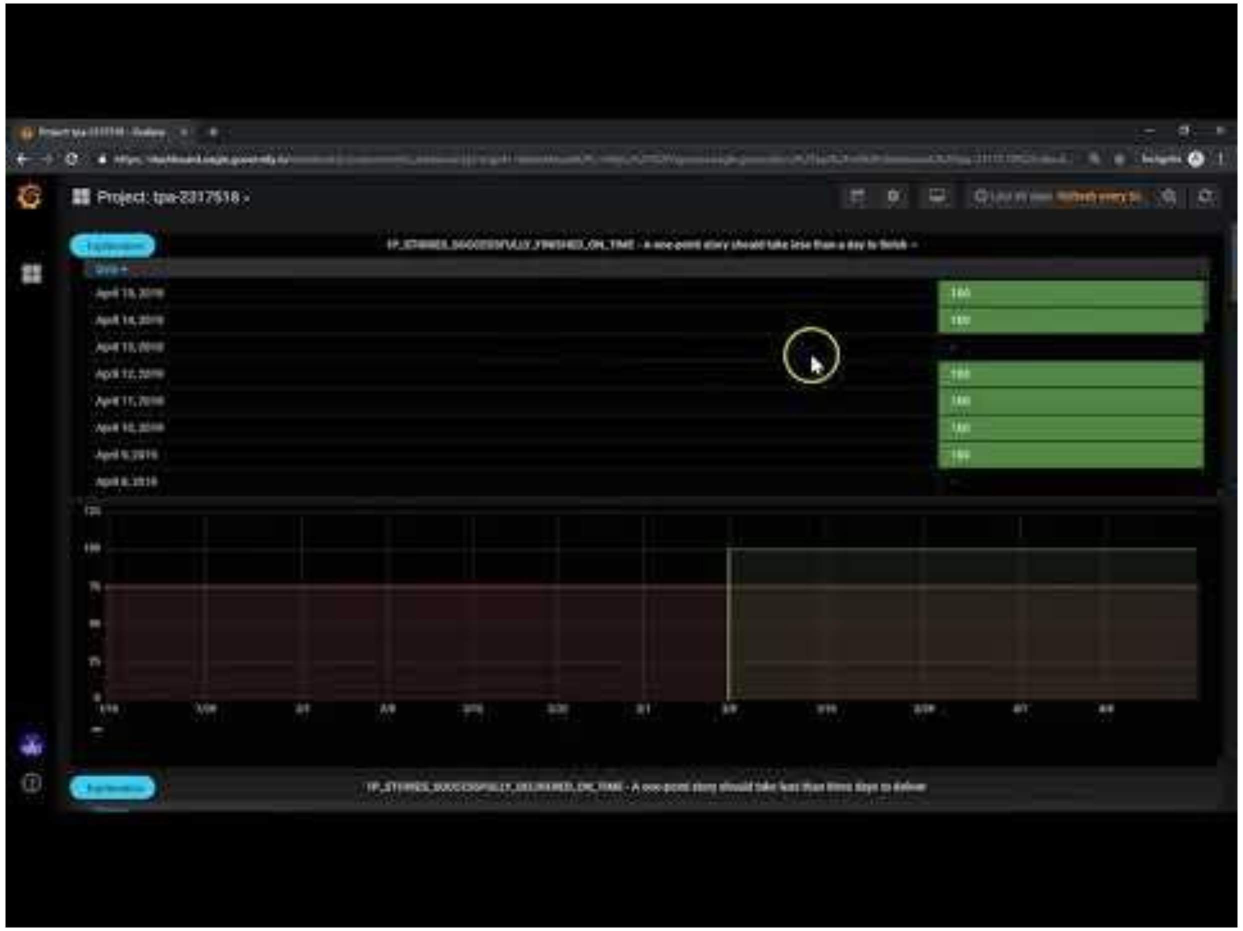


EAGLE: A Reflection Tool

- EAGLE is a tool that analyzes your data from Pivotal Tracker, to ensure you're following recommended Agile practices

- Watch for your team's EAGLE link - to be released this week!

  (will announce on Piazza)

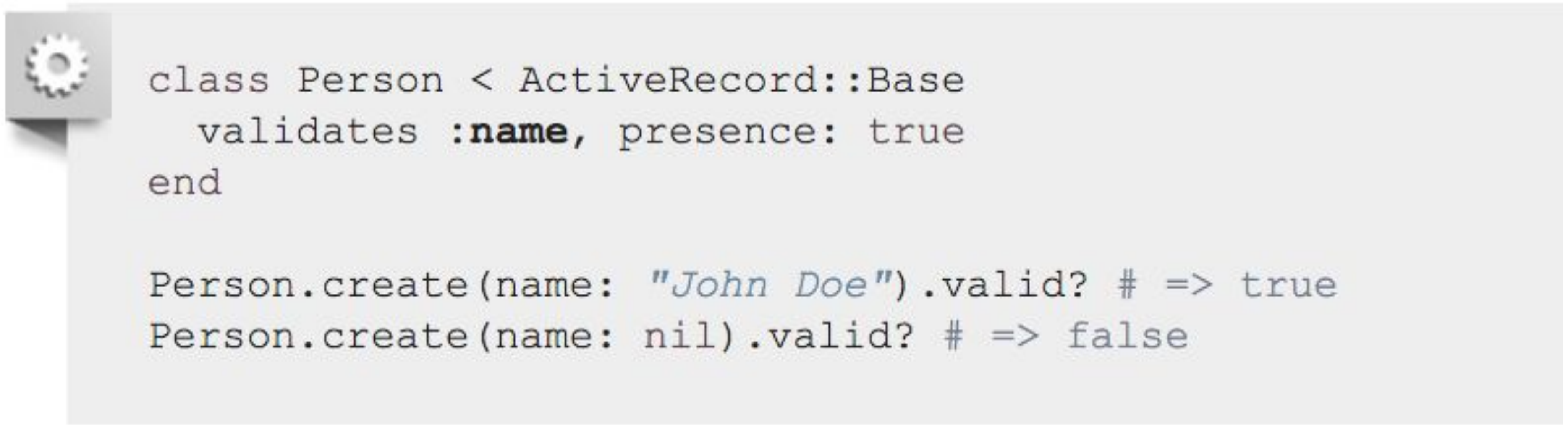


IP_STORIES_SUCCESSFULLY_FINISHED_ON_TIME - A one-point story should take less than a day to finish

IP_STORIES_SUCCESSFULLY_DELIVERED_ON_TIME - A one-point story should take less than three days to deliver

What is Rails Validation?

Rails Validation Hello World



```
class Person < ActiveRecord::Base
  validates :name, presence: true
end

Person.create(name: "John Doe").valid? # => true
Person.create(name: nil).valid? # => false
```

What is Rails Validation?

- **Goal:** DRY out checks that are done for every object

- Run before an object is stored to the DB
- You can write custom functions for more complex checks!

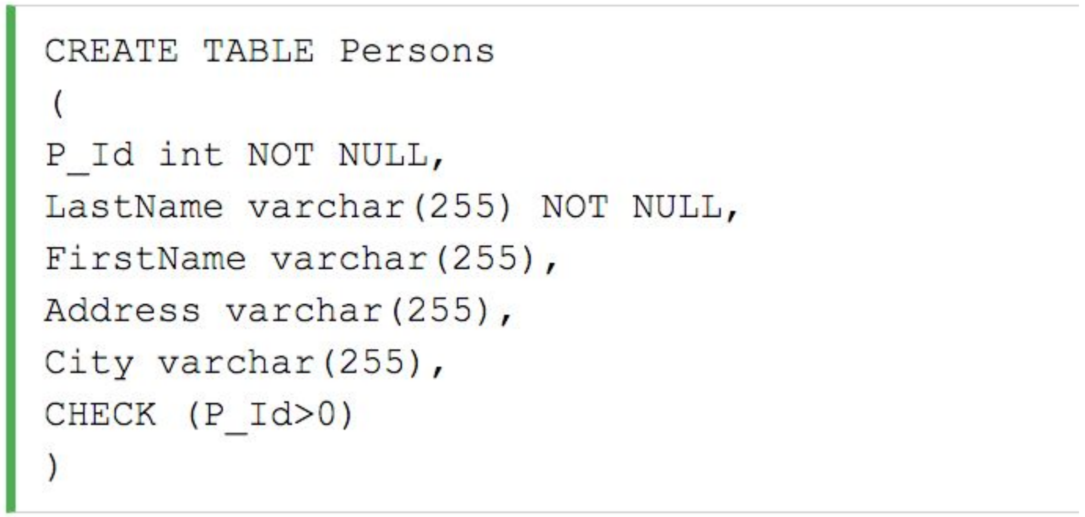
```
class User < ActiveRecord::Base
        validates :username, :presence => true
        validate :over_18

        def over_18
            if age < 18
                errors.add(:age, “This user is
                    too young.”)
            end
        end
end
```

Ex. Validation where User.username != nil

```
> user = User.new
   => #<User id: nil, username: nil, admin: nil, created_at: nil, updated_at: nil>
> user.valid?
   => false
> user.errors
   => #<ActiveModel::Errors:0x007f92c8df2d18 @base=#<User id: nil, username: nil,
   admin: nil, created_at: nil, updated_at: nil>, @messages={:username=>["can't be
   blank"]}>
> user.save
    (0.1ms)  begin transaction
    (0.1ms)  rollback transaction
> user.save!
    (0.1ms)  begin transaction
    (0.0ms)  rollback transaction
   ActiveRecord::RecordInvalid: Validation failed: Username can't be blank
```

Alternative to Rails Validation

- Database Constraint
  - Pros: share database with others
  - Cons: database-dependent
- Client-side validation
  - Pros: enhance user experience
  - Cons: unreliable if used alone
- Controller-level validation
  - Pros: Can’t think of pros....
  - Cons: not MVC, we want skinny controllers



```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CHECK (P_Id>0)
)
```

Rails Filters

- Similar to validations, declare `before_action` & `after_action`
- Common filter use case – `require_login`

```
class ApplicationController < ActionController::Base
  before_action :require_login

  private

  def require_login
    unless logged_in?
      flash[:error] = "You must be logged in to access this
section"
      redirect_to new_login_url # halts request cycle
    end
  end
end
```

Rails Association

- Why do we need association?
- How does association work?
  - Database level: foreign key column in table
  - Programming Model level: ruby metaprogramming
- What do developers (us) need to do?
  - Database level: write migration to add foreign key column
  - Programming Model level: call rails helper methods, e.g. has_many

Rails Association

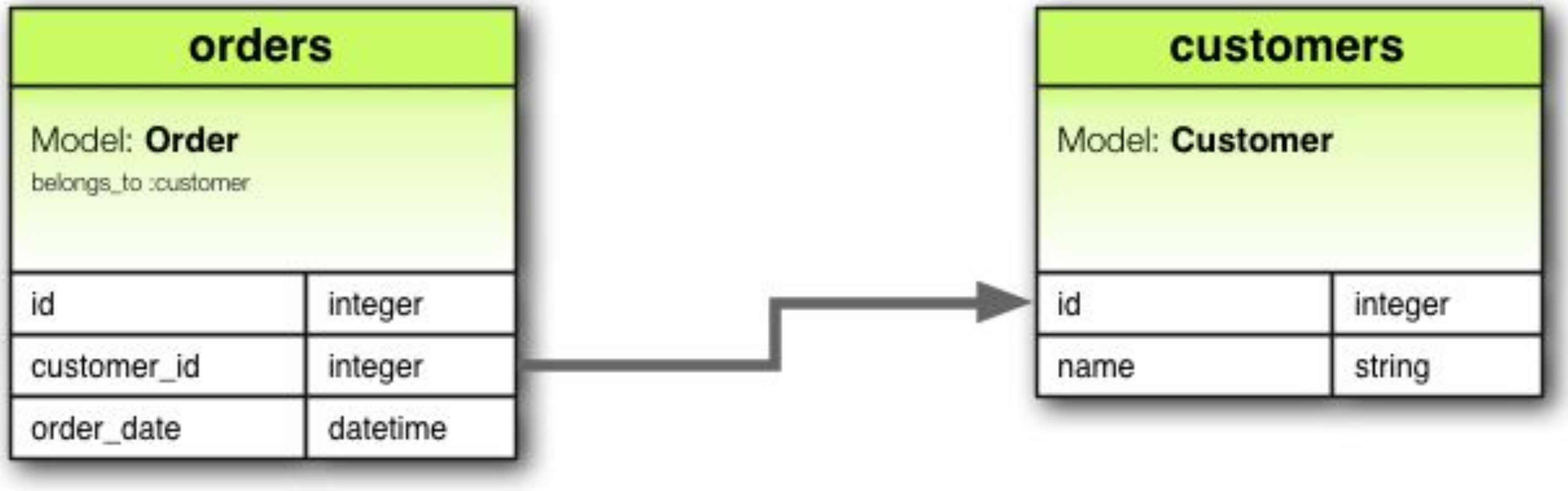
Conceptually:

- Object A `belongs_to` Object B, Object B `has_many` or `has_one` Object A's
    - Relationship can be one-to-many, or one-to-one
- Ex. an Order belongs to a Customer, a Customer has many Orders
- Ex. a smartphone belongs to a Person, a Person has one smartphone

In Practice:

- The foreign key goes on the model with "belongs_to"

Example for One-to-Many Association



```
class Order < ActiveRecord::Base
  belongs_to :customer
end
```

```
class Customer < ActiveRecord::Base
  has_many :orders
end
```

Example for One-to-Many Association

```
class CreateCustomers < ActiveRecord::Migration
  def change
    create_table :customers do |t|
      t.string :name
      t.timestamps null: false
    end

    create_table :orders do |t|
      t.belongs_to :customer, index:true
      t.datetime :order_date
      t.timestamps null: false
    end
  end
end
```