Workflow

the conventional practices of a person or team with regards to modifying code and incorporating those modifications

Your Own Git Repo

branch, add feature (and passing tests), merge into master

Sharing a Repo

branch, add feature (and passing tests), PR to master, receive comments on PR, make updates, get merged (hopefully)

branching

for every feature

committing

every. time. something. new. works
(or gets better)

... and sometimes for backups

pull requests

a request to incorporate changes from another branch
(or another fork)

code reviews

- another pair of eyes on changes.
- team member comments, suggests alternatives.
- some companies require changes to "pass a code review"
- before they're put in master

merging

incorporating a branch into another branch (most often, into the master branch)

merge conflict

when git can't decide what code to keep

happens when master has changed since the last time you branched

fork repo

one that was copied from an already existing repository

how might this workflow be different?

Pair Programming

Two programmers, one text editor.

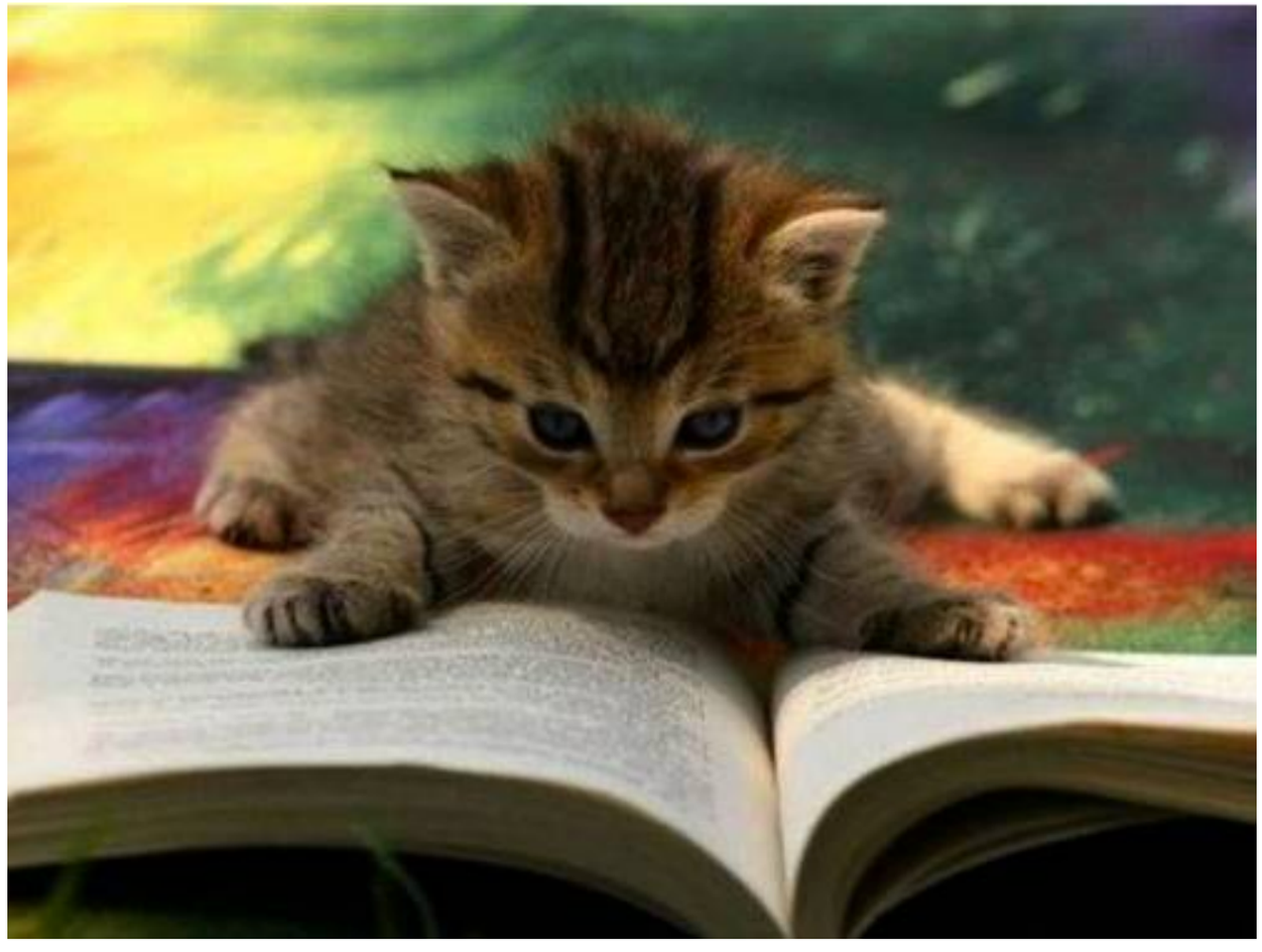
Why?

Fewer Bugs

15% fewer bugs according to this study.

Learning

Share knowledge.
Explain your reasoning and test your assumptions.



Happiness

Makes coding more social. Helps you get unstuck.

Roles

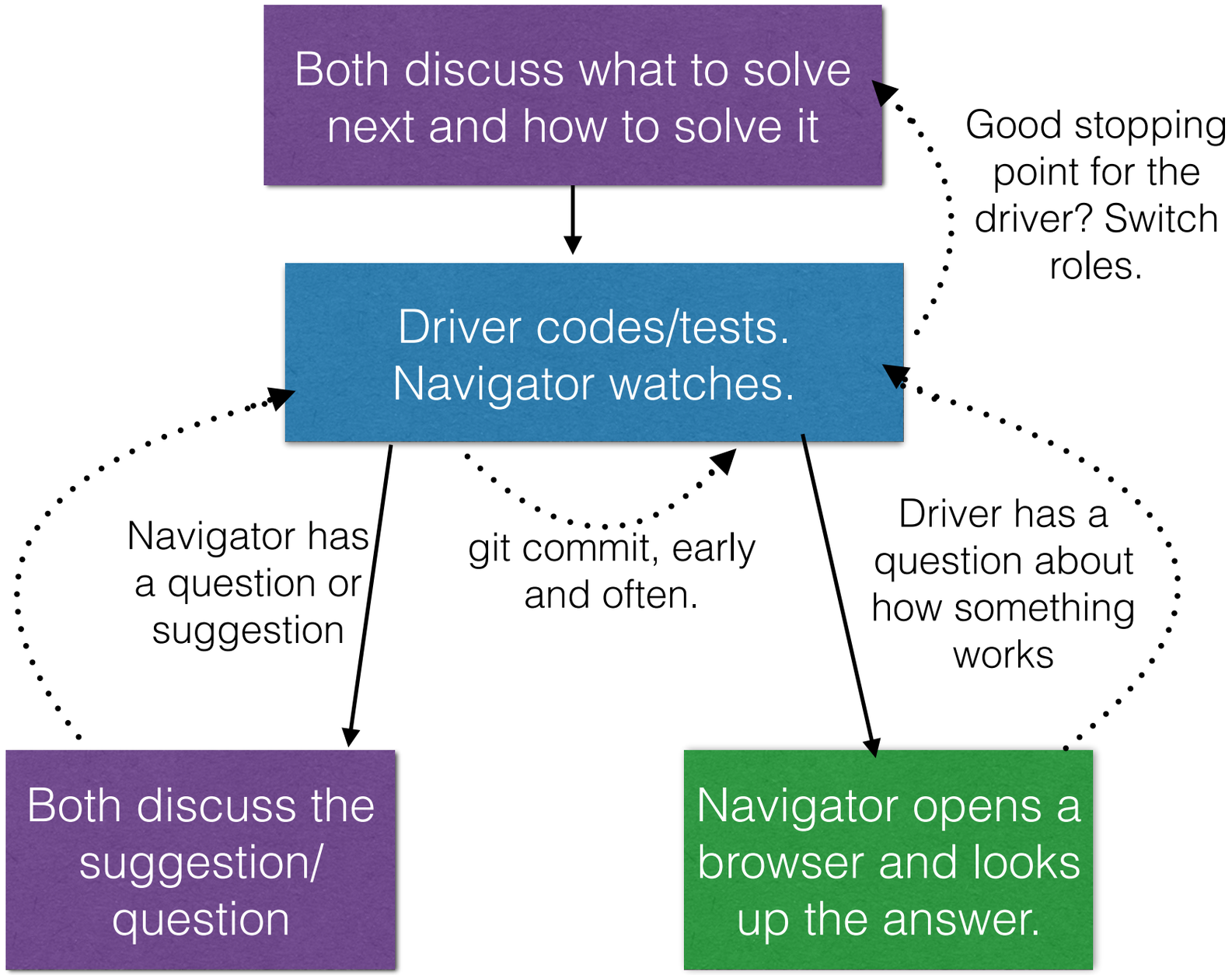
Driver

Write the code. Test the code.

Navigator

Read the code as the driver writes it,
make suggestions, ask questions.

Workflow


Both discuss what to solve next and how to solve it
Driver codes/tests. Navigator watches.
Good stopping point for the driver? Switch roles.
Navigator has a question or suggestion
git commit, early and often.
Driver has a question about how something works
Both discuss the suggestion/ question
Navigator opens a browser and looks up the answer.

Save Your Changes

To make a git commit of all your changes so far, in the terminal:

```
$ git add .
$ git commit -m "this awesome thing we just did"
$ git push origin whateverBranchYoureOn
```

Switch Roles

The new driver needs to pull down the latest changes

```
$ git pull origin whateverBranchYoureOn
```

Constructive Criticism

Be kind. Be generous. Be patient.

Views

in general: the presentation layer
in rails: templates for creating http responses

ERB

the template language that comes with rails

stands for "embedded ruby"

put ruby directly in the template.
rails runs that ruby to create the final document

```
<% %>
```

Executes the ruby code within the brackets.

```
<%= %>
```

Prints something into erb file.

```
<% -%>
```

Avoids line break after expression.

```
<%# %>
```

Comments out code within brackets; not sent to client (as opposed to HTML comments).

instance variables

if defined in the controller action are available in the template. use of absent instance variables evaluate to nil

```
@user = User.all
```

partials

erb files that represent parts of a page and can be included in other templates during rendering

name with a leading underscore, like "shared/_ad_banner.html.erb"

```
<%= render "shared/ad_banner" %>

<h1>Products</h1>

<p>Here are a few of our fine products:</p>
...

<%= render "shared/footer" %>
```

```
<%= render :partial => "image_tag", :locals => {:image => i} %>
```

routes

directions for how to map urls to controller actions
(and vice versa)

http methods

flags in the metadata of a request.
there are conventions, but the server has discretion.

GET, PUT, POST, DELETE

Post vs Put

Post: ask the server to accept data/resource and process it at the specified uri

Put: ask the server to place a resource at the specified uri

resources

method that setups up CRUD endpoints for a model

(Create, Read, Update, Destroy)

```
resources :topics
```

match

method to create a route matching a given url structure

```
match 'photos', to: 'photos#show', via: [:get, :post]
```

```
match 'photos', to: 'photos#show', via: :all
```

get

a shortcut for match when the request type is a get

```
get '/patients/:id', to: 'patients#show'
```

path variables

provided by rails to create urls

```
<%= link_to 'Patient Record', patient_path(@patient) %>
```

params

a hash available in the controller action that includes segment keys and url parameters

segment key

part of the url structure passed to **match**. prepended with a colon. represents a value that will serve as a key in params in the controller action

```
get '/patients/:id', to: 'patients#show'
```

url parameter

named data passed in with an http request via the url

also available in **params**

```
http://example.com/over/there?name=ferret
```

Demo Requirements

- create jokes, quotes
- /jokes -- all the jokes in the system
- /jokes/:index -- index-th joke in the database
- /jokes/random -- random joke
- /inspirations -- all the inspirational quotes
- /inspirations/random -- random inspiring quote
- /inspirations/random/:index-th

Homework

/both/random -- use partials to put a random joke and an inspiration in the same view. rewrite the views we made in class to use those partials

/jokes/:num -- a page with :num random (unique) jokes. if :num is larger than the number of jokes in the system, render a different template with an error message that says you don't have enough jokes.

/inspirations/:num -- a page with :num random (unique) inspirations. ditto about the error message above (use the same template, but say you don't have enough quotes)

Read

http://richonrails.com/articles/partials-in-ruby-on-rails

http://guides.rubyonrails.org/action_view_overview.html
(through section 3.2.2)

https://guides.github.com/introduction/flow/index.html

http://guides.rubyonrails.org/routing.html
(through section 3.5)