

# The Coder's Dojo

Emily Bache

[emily@bacheconsulting.com](mailto:emily@bacheconsulting.com)

<http://emilybache.blogspot.com>



# Session outline

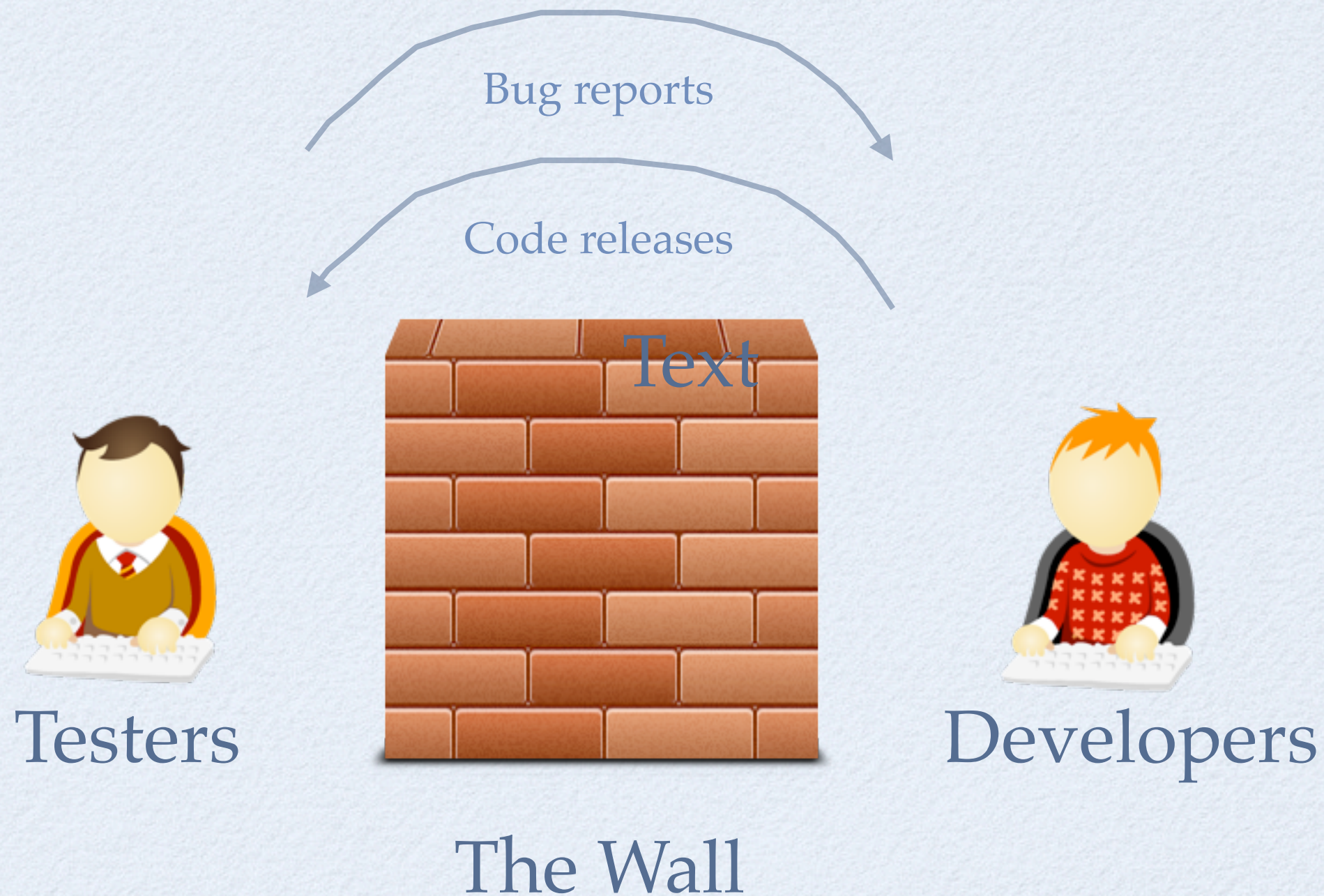
- Why should developers test? (slides)
- What is TDD? (slides)
- TDD vs Test Last (slides, discussion)
- How can I learn TDD? (slides)
- Randori (coding)
- Experiences leading Coding Dojos (slides, discussion)



# Why should developers Test?



# Traditional Testing Strategy:





# Developers are blind to bugs

- Developers don't want to break their beautiful creation
- Subconsciously use their knowledge of the implementation to avoid weak spots



circa 1940, Paris, Blindfold typing competition  
(Jhayne, flickr)

material copyright Bache consulting, emily@bacheconsulting.com



# A smart mouse ...





# JUnit



“Keep the bar green to keep the code clean”



# Engineering practices

(Exploratory) Testing

Automated Checks

Continuous Integration

Automated Build

Version Control



# Testing vs Checking

- Requires a sapient human
- turns up new information
- asks new questions
- is exploration and learning
- Can be automated
- confirms our beliefs to be true
- asks the same question as yesterday
- is making sure the program doesn't fail

See also Michael Bolton's article

<http://www.developsense.com/blog/2009/08/testing-vs-checking/>



# Example - Fitnessse

Hudson

search

Hudson » fitnessse » Test Report

ENABLE AUT

[Back to Dashboard](#)

[Status](#)

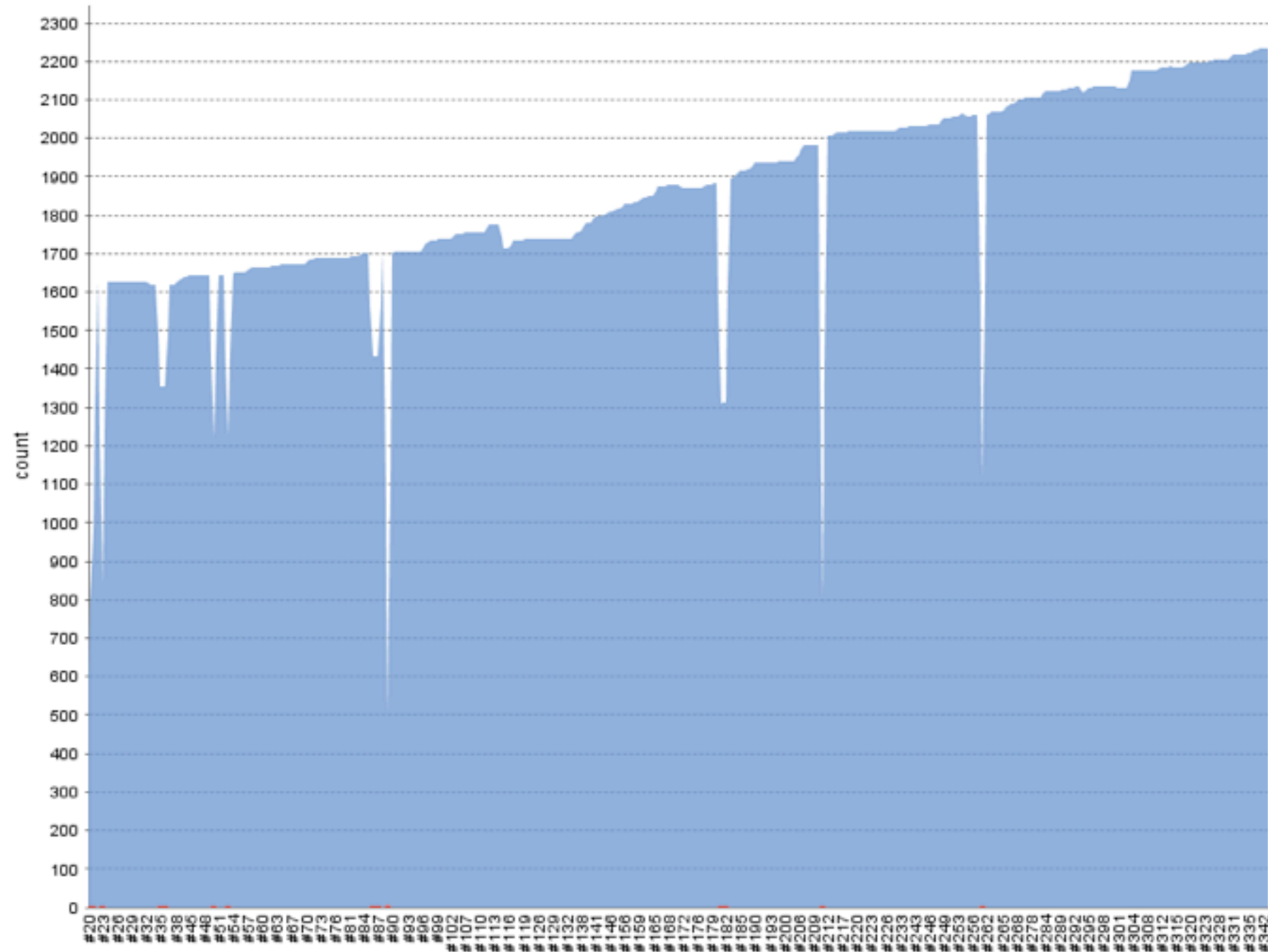
[Changes](#)

[Workspace](#)

[Git Polling Log](#)

**Build History** [\(trend\)](#)

- #343 [Feb 28, 2010 11:20:24 AM](#)
- #342 [Feb 27, 2010 9:45:25 AM](#)
- #338 [Feb 26, 2010 1:36:31 PM](#)
- #336 [Feb 24, 2010 3:34:25 PM](#)
- #335 [Feb 24, 2010 6:00:55 AM](#)
- #334 [Feb 17, 2010 6:23:36 AM](#)
- #333 [Feb 16, 2010 5:26:38 PM](#)
- #331 [Feb 9, 2010 5:02:22 PM](#)
- #330 [Feb 5, 2010 8:24:10 AM](#)
- #329 [Feb 3, 2010 7:02:33 PM](#)
- #328 [Feb 3, 2010 12:10:37 PM](#)
- #325 [Feb 2, 2010 8:02:11 PM](#)
- #324 [Feb 1, 2010 1:02:58 PM](#)
- #323 [Feb 1, 2010 10:02:02 AM](#)
- #322 [Feb 1, 2010 9:02:46 AM](#)
- #321 [Jan 30, 2010 7:37:25 AM](#)
- #320 [Jan 28, 2010 4:02:08 PM](#)
- #319 [Jan 27, 2010 1:02:12 AM](#)
- #318 [Jan 25, 2010 10:28:06 PM](#)
- #315 [Jan 17, 2010 5:51:22 PM](#)

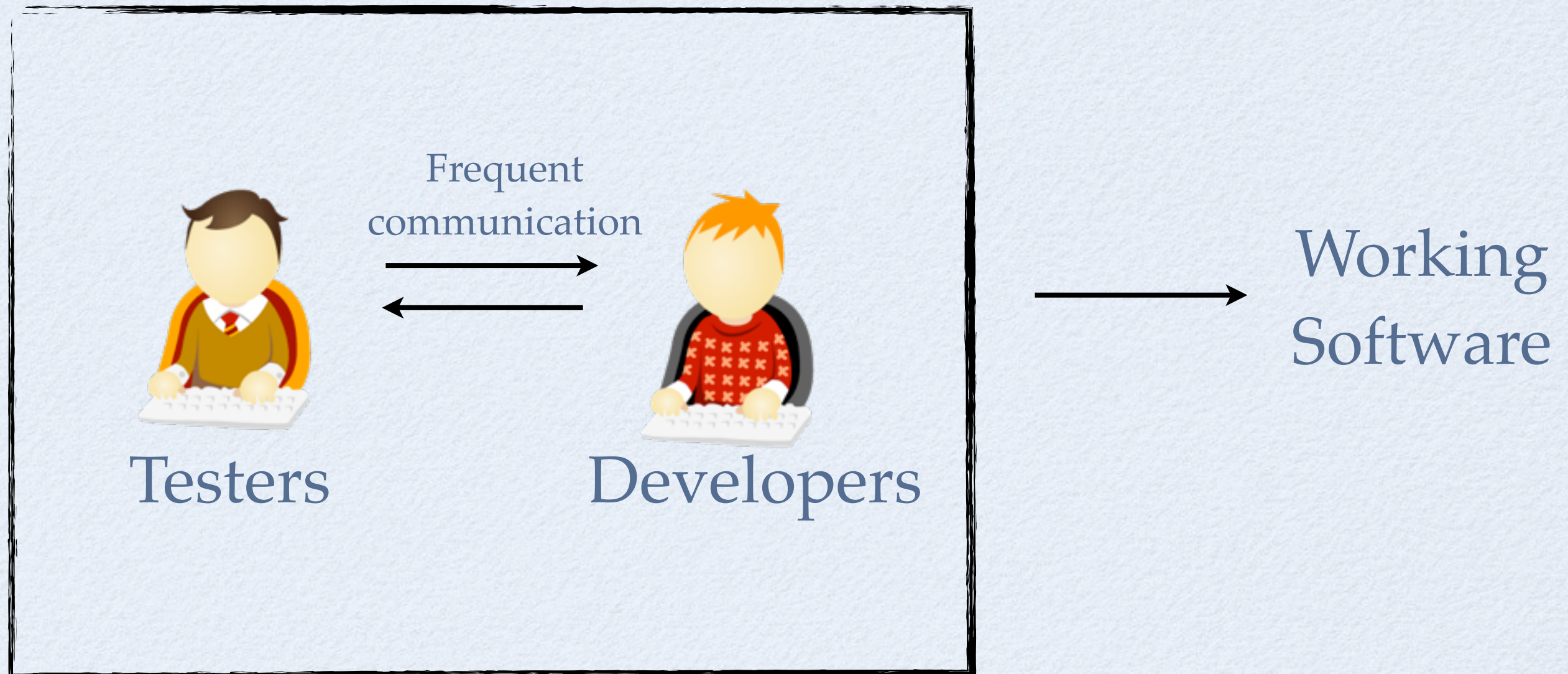


X Find: hudson Next Previous Highlight all Match case



# Testers and Devs on same team

## Timebox



The team delivers working, tested software  
*every iteration*



# Automated Checks

- Catch regression errors -
  - Safety net for refactoring
- Documentation / examples of usage
- Force internal interfaces / modules / units

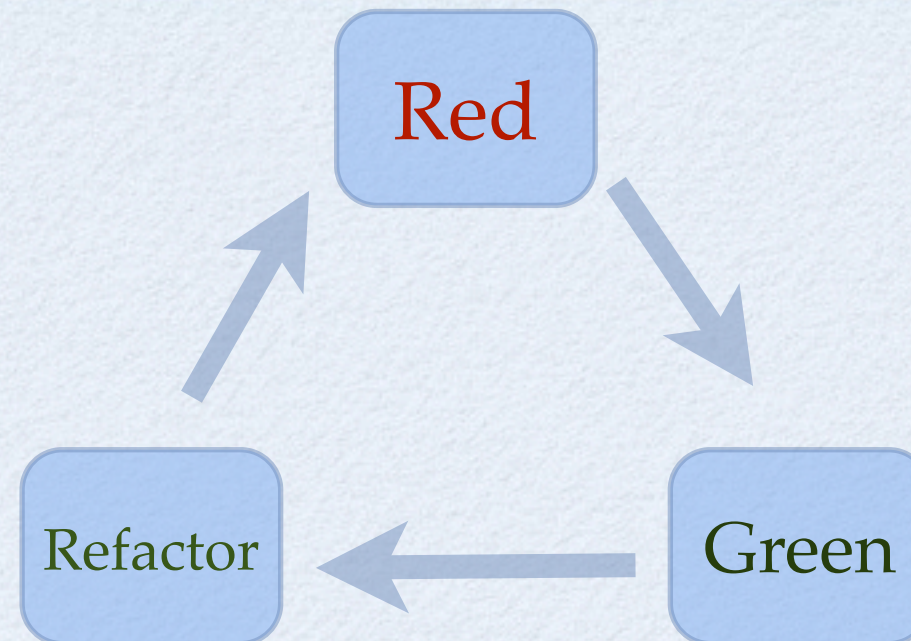




# What is TDD?



# Bob Martin's Laws of TDD



- Don't write production code unless it is to make a failing unit test pass.
- Don't write any more of a unit test than is sufficient to fail.
- Don't write any more production code than is sufficient to pass the one failing unit test.

<http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd>



# TDD moves

## Starting

Analyse Problem  
Test List  
Guiding Test

## Red

Declare & Name  
Arrange-Act-Assert  
Satisfy compiler

## Refactor

Remove Fake  
Remove Code Smell  
(No new functionality)  
Note new test cases

## Green

Implement solution  
Fake it  
Start over



# TDD moves (starting)

**Starting**  
Analyse Problem  
Test List  
Guiding Test

- Basic analysis of problem
- Review existing code in area
- Identify where the new functionality will be called from
- Guiding Test (take from story acceptance criteria)
- Make a list of test cases

**Move On When:**  
You have an achievable goal & have chosen first test



# TDD moves (get to red)

## Red

Declare & Name  
Arrange-Act-Assert  
Satisfy compiler

- Declare and name a test
- Arrange - Act - Assert (start with Assert)
- Just enough Production code to satisfy compiler

## Move On When:

Test code describes missing function, executes, and is **Red**



# TDD moves (get to Green)

## Green

Implement solution

Fake it

Start over

- Change the production code - implement just enough to make the test pass
- Fake it (if you're unsure)
- When all else fails, remove the failing test, get back to green, and write an easier test.

Move On When:

Tests all execute and are **Green**



# TDD moves (refactor)

## Refactor

Remove Fake

Remove Code Smell

(No new functionality)

Note new test cases

- Change production code: remove fakes, code smells
- Change test code: improve readability
- Don't add functionality not required by tests.
- Note down new test cases on list.

**Move On When:**

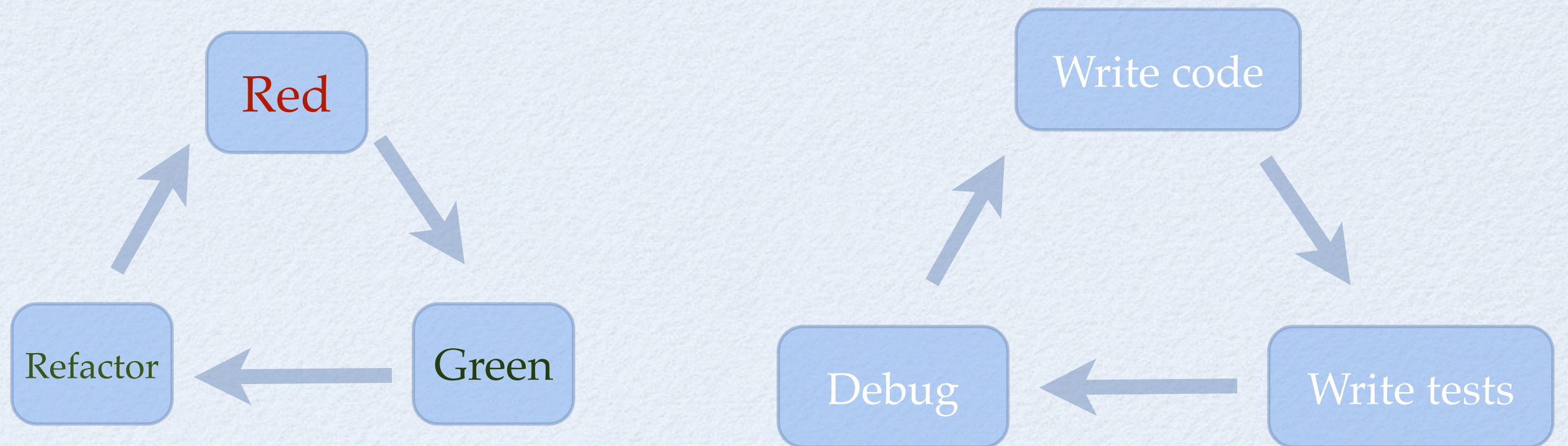
Code is Clean, tests all execute and are **Green**



# Why TDD over Test Last?



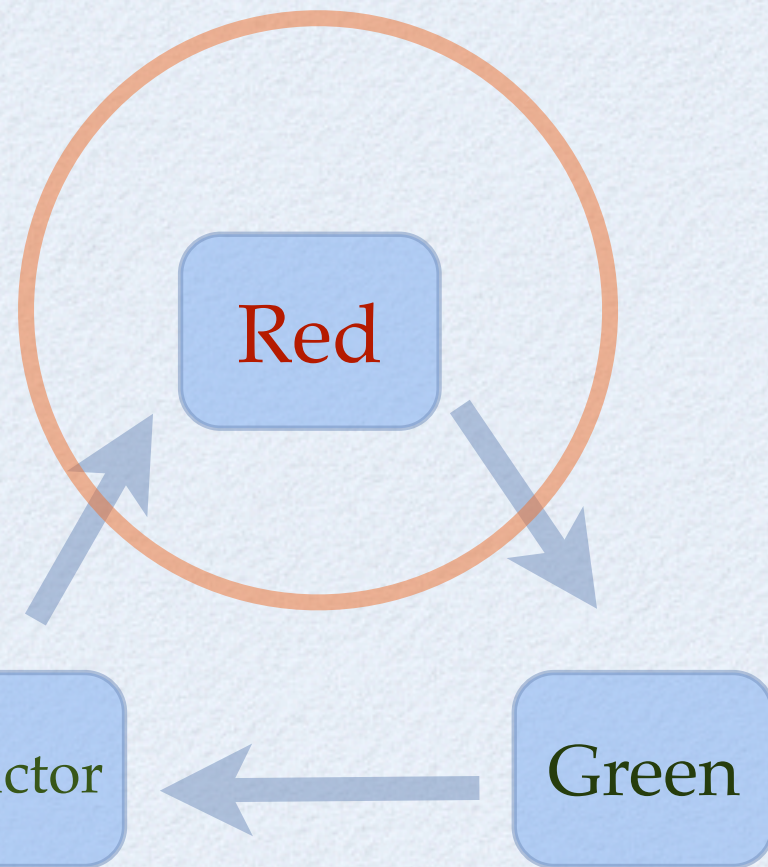
# TDD vs Test Last





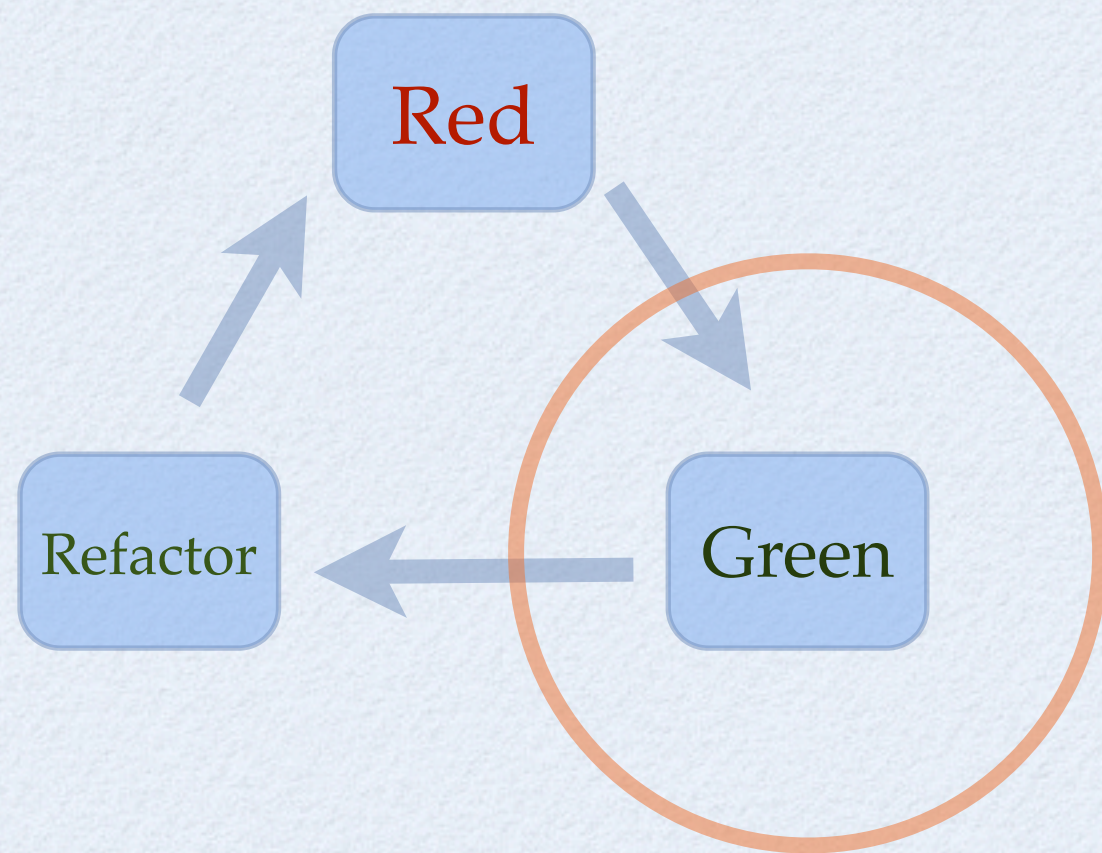
# TDD & feedback

- Writing a failing test tells you:
  - what problem you are solving
  - if your SUT requires many collaborators
  - if SUT has an inconvenient API
- *before you commit to an implementation*





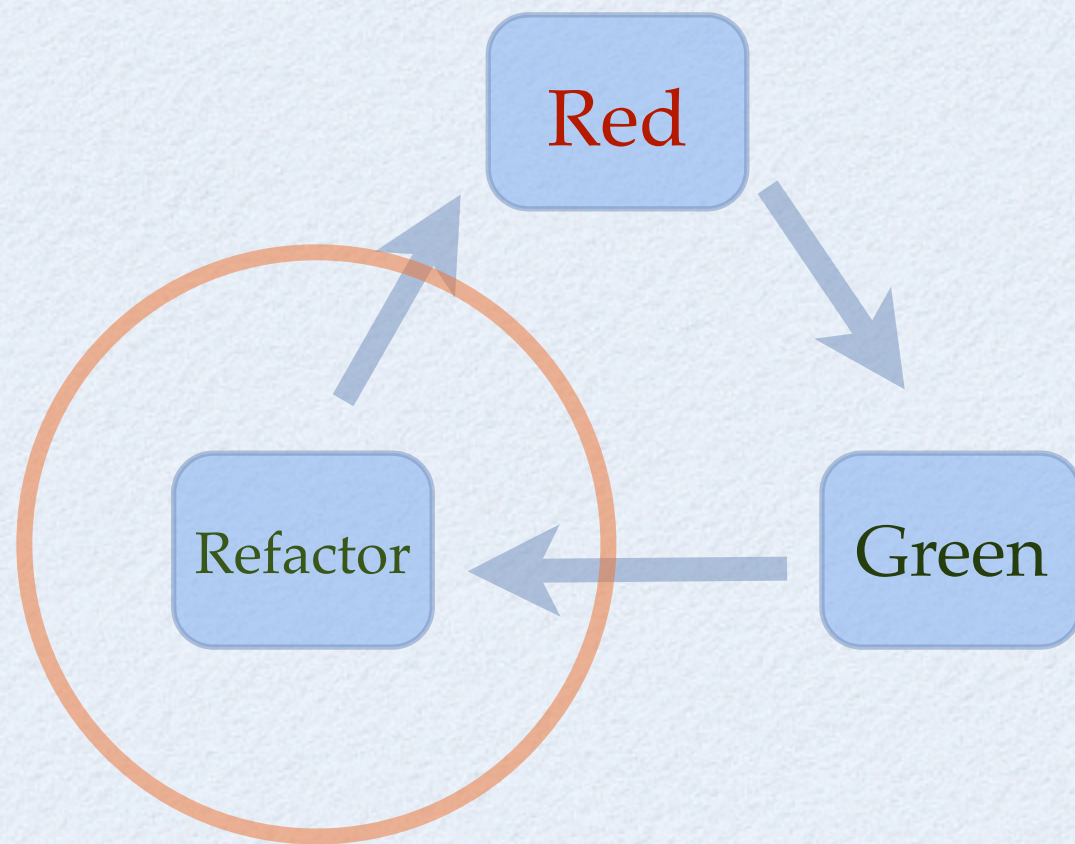
# TDD & feedback



- Getting to Green:
- makes you write only just enough functionality
- Avoid over design & code bloat



# TDD & feedback



- Refactoring happens frequently
- Many opportunities to improve your design



# TDD - no silver bullet

“TDD constrains the problem space we are looking at, as well as provides feedback on how well we are addressing this problem.

Of course TDD isn't the only thing imposing constraints on us, nor our only source of feedback.”

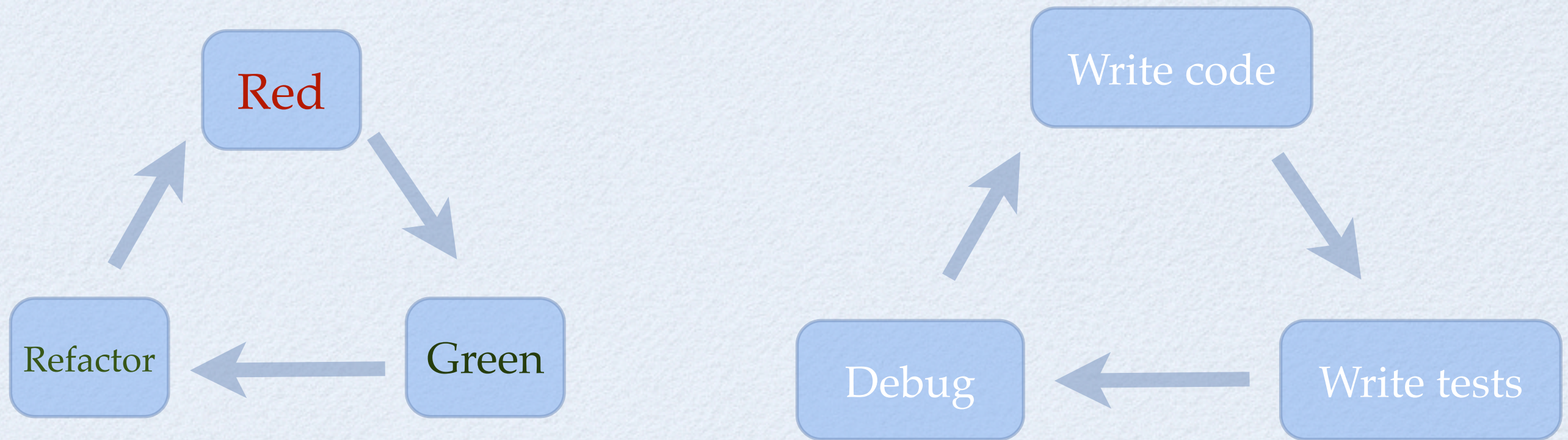
- *David Tchepak*

From his article

<http://www.davesquared.net/2011/03/why-learning-tdd-is-hard-and-what-to-do.html>



# Checking & Testing



- TDD -> more like Testing
- Test-Last -> more like Checking

*TDD is a lot more fun to do*



# Discussion



# Should developers do TDD?

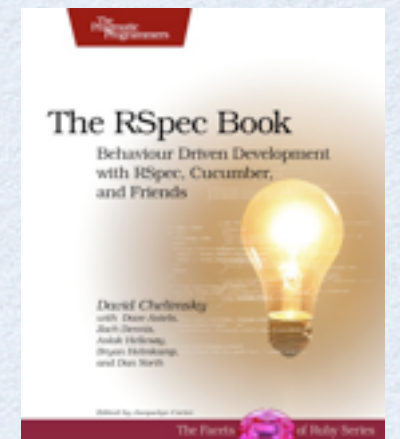
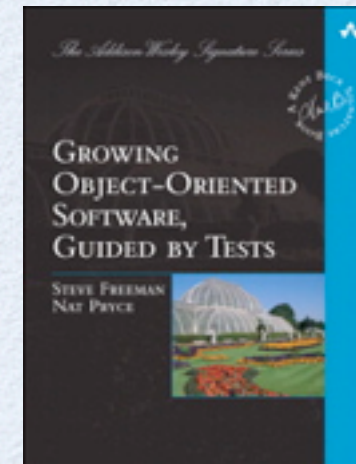
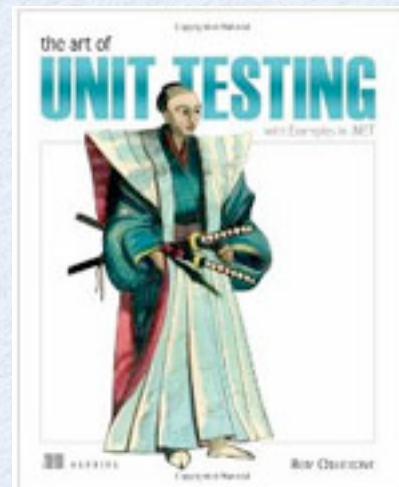
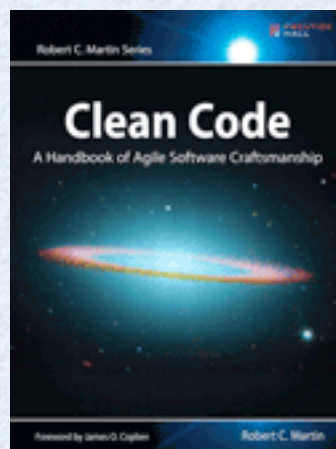
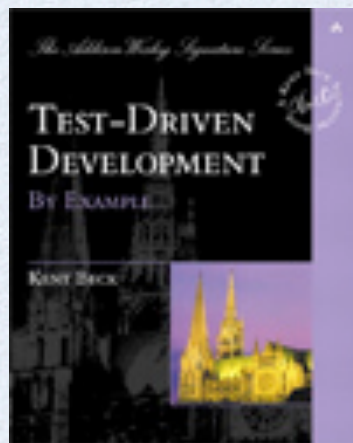
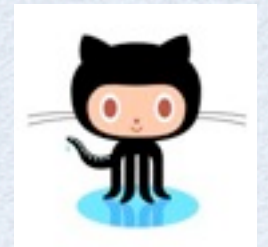
- Do you test your own code? Do you need separate testers?
- What can developers do to influence the long term project velocity? (give examples)
- Do you think it matters whether you write the tests first or last?



# How can I learn TDD?



# read books, blogs, code...



- other people's experience and wisdom
- read code, discuss code, analyse code - that you didn't write yourself



# Pair Programming



*Attribution: flickr user Lisamarie Babik*



# The Coder's Dojo

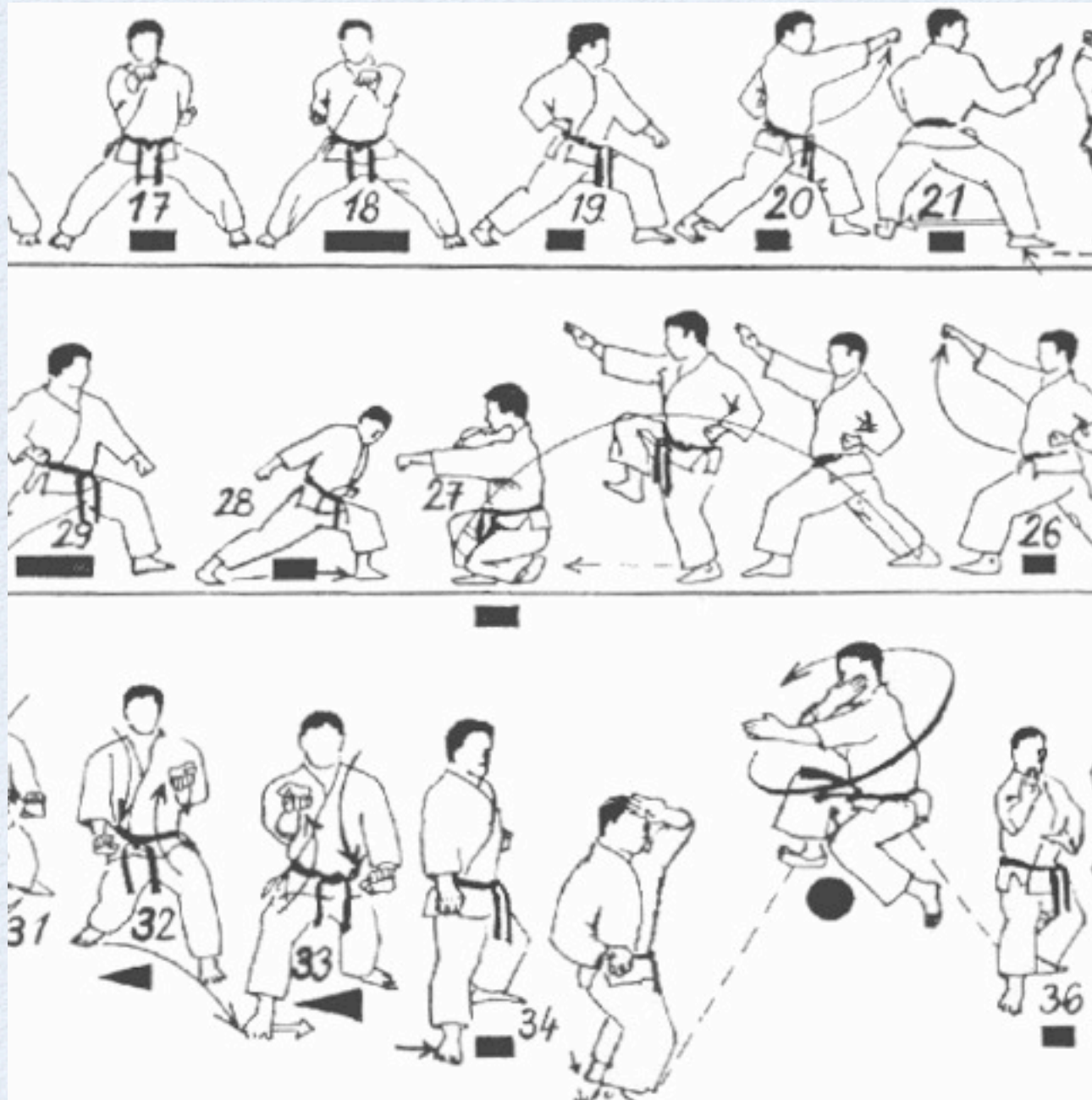
- The dojo is the place where you go every week to practice and learn karate



see <http://codingdojo.org>



# Code Kata: TDD moves



‘Pragmatic’ Dave Thomas



# KataRomanNumerals

Write a program to convert normal numbers to roman numerals:

1	-->	I
10	-->	X
7	-->	VII



# KataMinesweeper

Write a program that outputs the same field including the hint numbers, like this

* . . .	→	* 1 0 0
. . . .		2 2 1 0
. * . .		1 * 1 0
. . . .		1 1 1 0



# What happens at a dojo meeting?



*Image courtesy Danilo Sato*

- 3 – 10 programmers
- meet as equals
- do some coding together
- practice TDD moves



# Prepared Kata

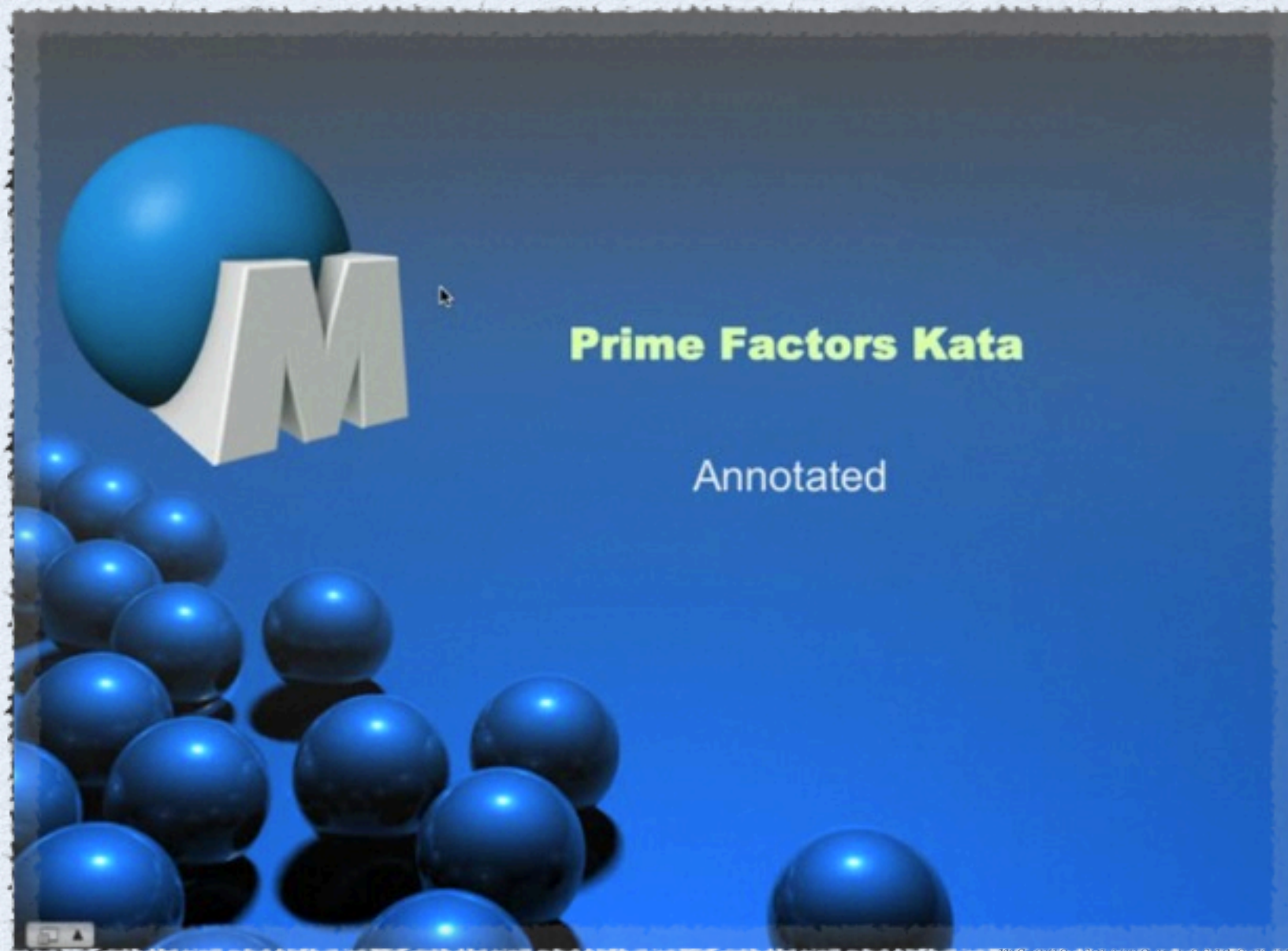
- One person presents their best solution
- They have practiced it many times.
- Group contributes feedback and questions





# Screencasts

- Watching someone else code
- music or commentary



material copyright Bache consulting, [emily@bacheconsulting.com](mailto:emily@bacheconsulting.com)



# Randori

- Everyone contributes
- Split into pairs, or work all together on a projector
- Ping pong TDD or time limits





# Dojo Rules

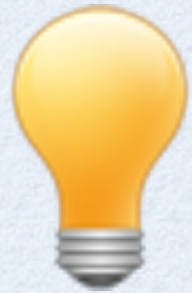
- The person with the keyboard decides what to type
- Ask for help if stuck, offer help if asked
- Be strict about TDD
- Offer design criticism on a green bar





# Test-Infected

- You're writing some code using TDD, awkward and slow and unfamiliar.
- You refactor it.
- Some tests fail unexpectedly!
- You realize the tests saved you
- *That feeling of safety*
- You want more of that!





# Importance of safe

- Step outside normal hierarchy
- Mistakes expected
- Moderation / leadership important



# Dojo Discussions

The code you write in the dojo leads to *concrete* discussions

```
PartnerData newPartner = new PartnerData("Disney");  
AddressData address = new AddressData("24 Stars Ave",  
                                         "91210", "Hollywood", "USA"));  
newPartner.setAddress(address);  
UserData contact = new UserData("Emily", "emily@test.com",  
                                   "031 123456");  
newPartner.setContactPerson(contact);
```



# Dojo Discussions

```
PartnerData newPartner = new PartnerData("Disney")  
    .withAddress(  
        new AddressData("24 Stars Ave",  
            "91210", "Hollywood", "USA"))  
    .withContactPerson(  
        new UserData("Emily", "emily@test.com",  
            "031 123456"));
```

Is this better? Cleaner?



# The Coders Dojo

Emily Bache

[emily@bacheconsulting.com](mailto:emily@bacheconsulting.com)

<http://emilybache.blogspot.com>



# Experiences

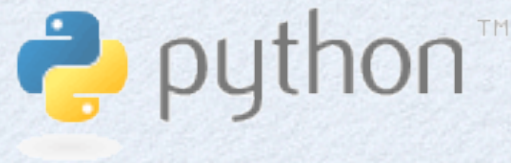


# Different kinds of groups

- language user groups
- with colleagues in the same team
- open / free taught course
- agile coach leading course at a client



# Language user groups



• js



- Python, Ruby, JavaScript, Scala groups
- Small numbers of skilled enthusiasts
- Dojo meetings are fun!



# With colleagues



- Stop work a little early, dojo in evening
- Boss sponsors with time & pizza
- Fun, community building



# Open / free taught course



JDojo@Gbg

- 5 sessions announced in advance as a course
- 4pm - 6pm (may count as work)
- Host company uses it for marketing & recruitment



# Agile coach leads dojo at client

- Raise competency in whole dept / team
- Discuss: what is clean code for this team?
- “This would never work on our code”

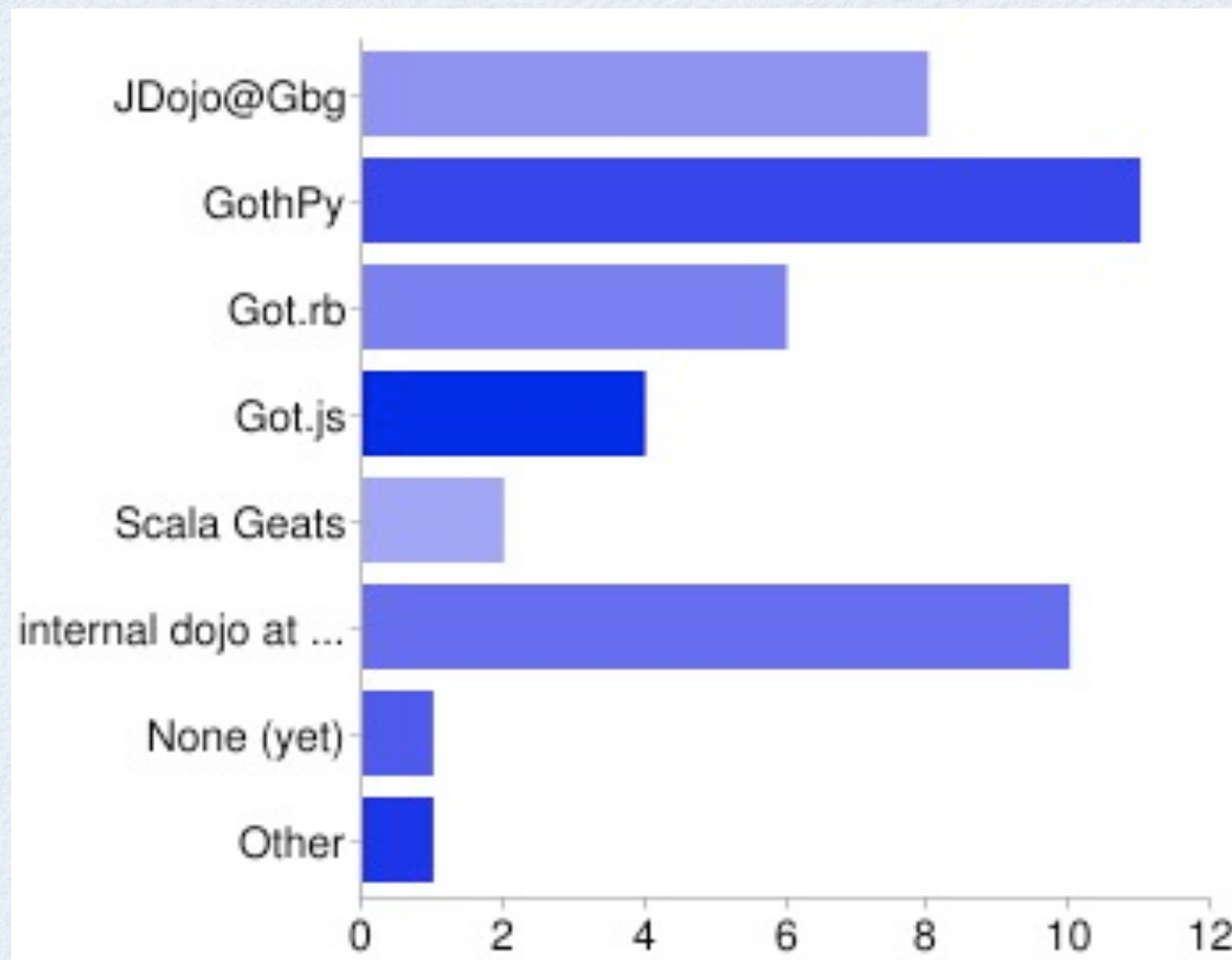




# What past participants have learnt



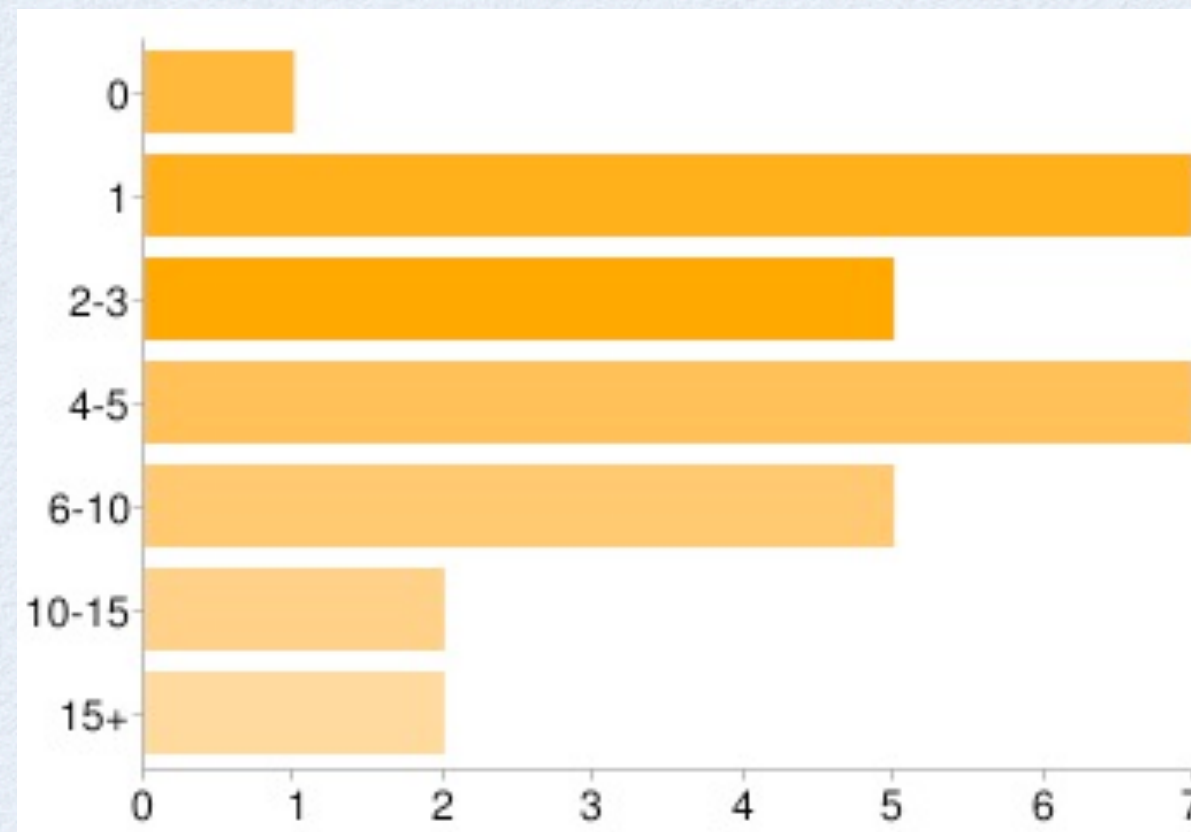
# Survey of various dojos



29 responses, most have been to more than one kind



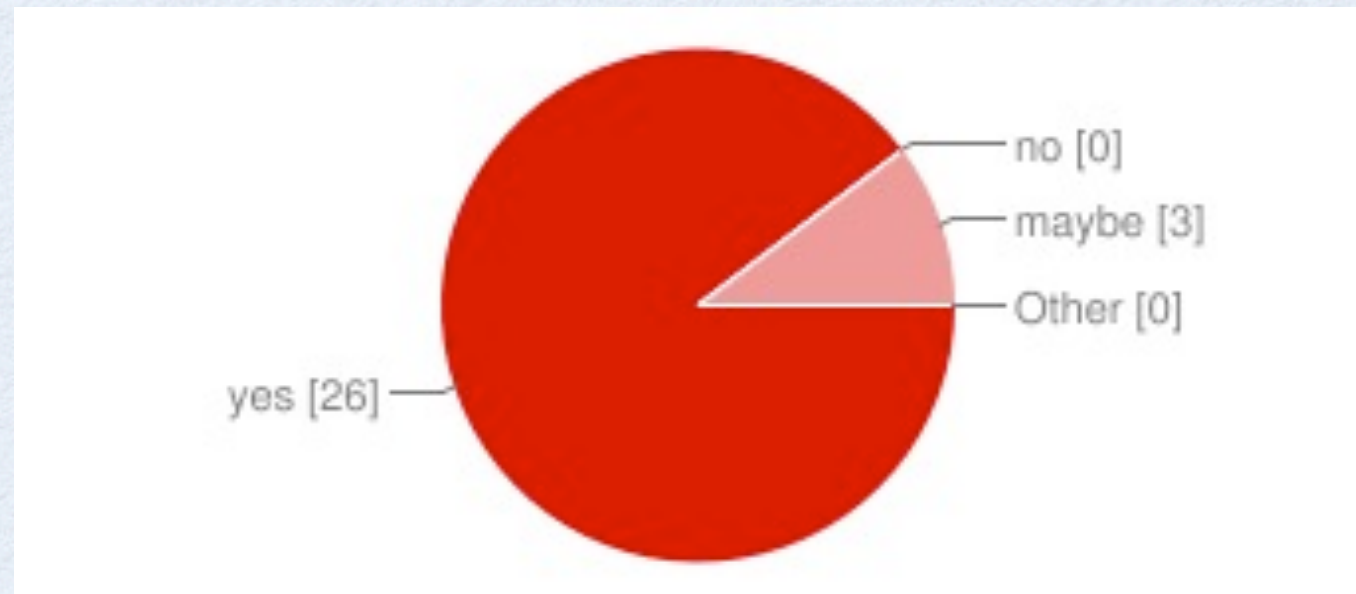
# Based on 4-5 meetings...



Roughly how many dojo meetings have you been to?



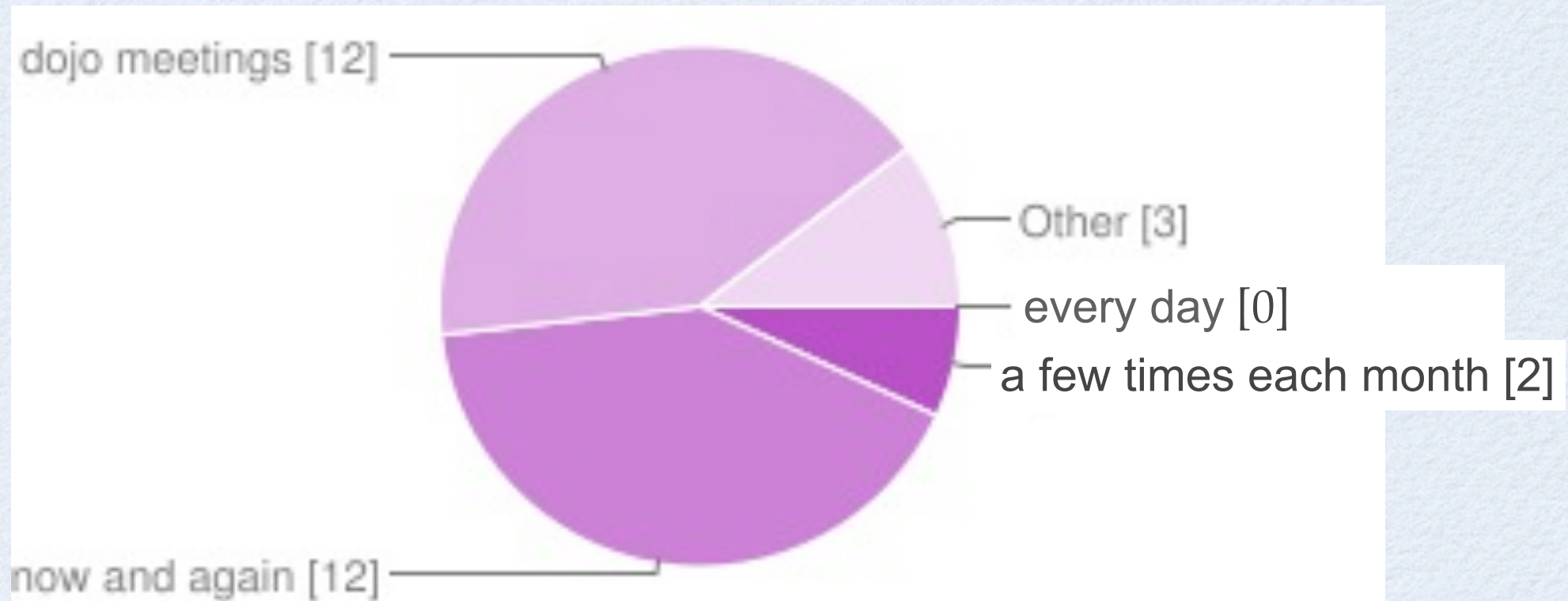
# Well received



Would you encourage others  
who want to improve their programming  
skills to come to dojo meetings?



# Practicing at home?



Have you ever practiced a Kata by yourself at home?



# We asked about:

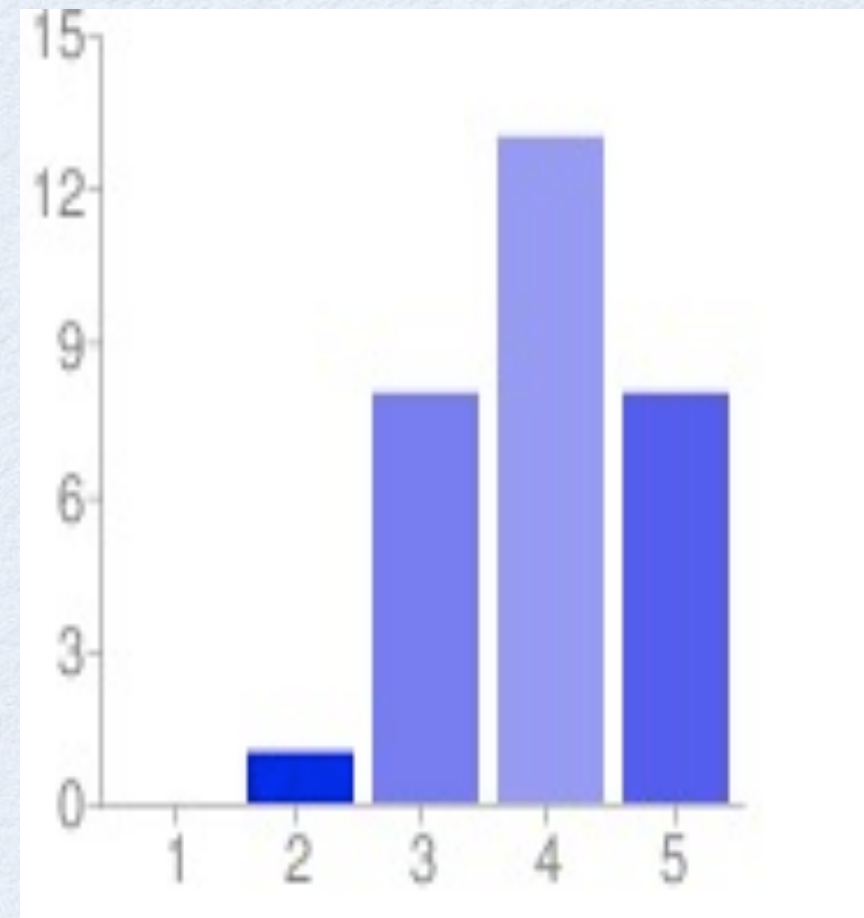
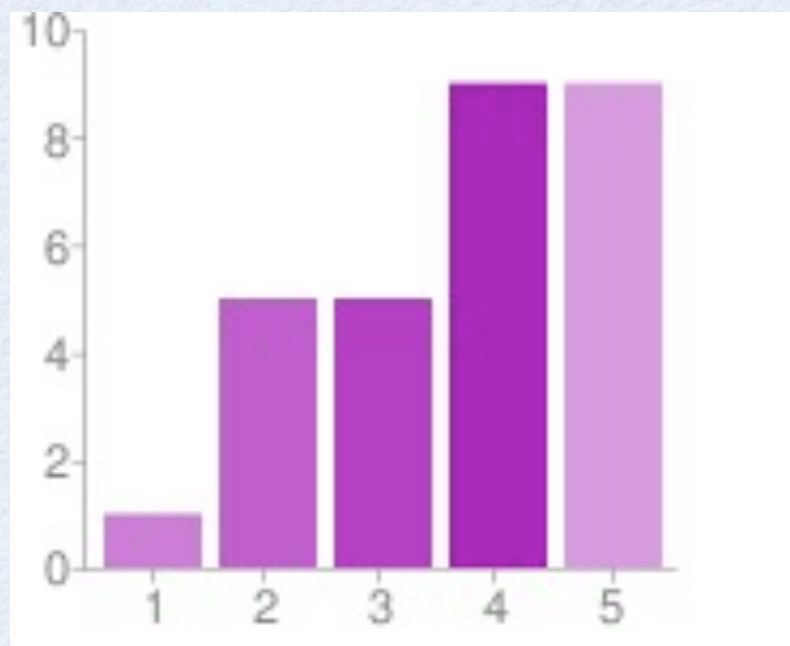
- TDD
- Mocking
- Refactoring
- IDE & frameworks
- Language skills

Image attribution: Flickr user crashmaster



# Language Skill

*Scale from 1-5  
beginner - expert*

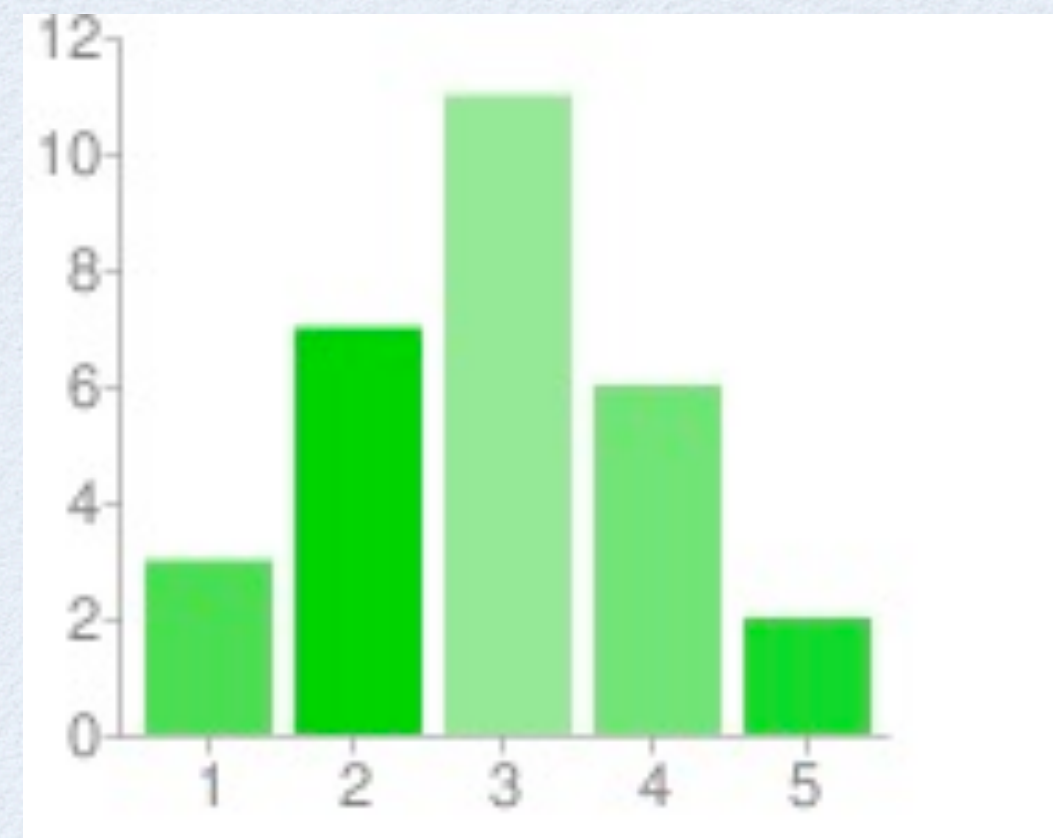
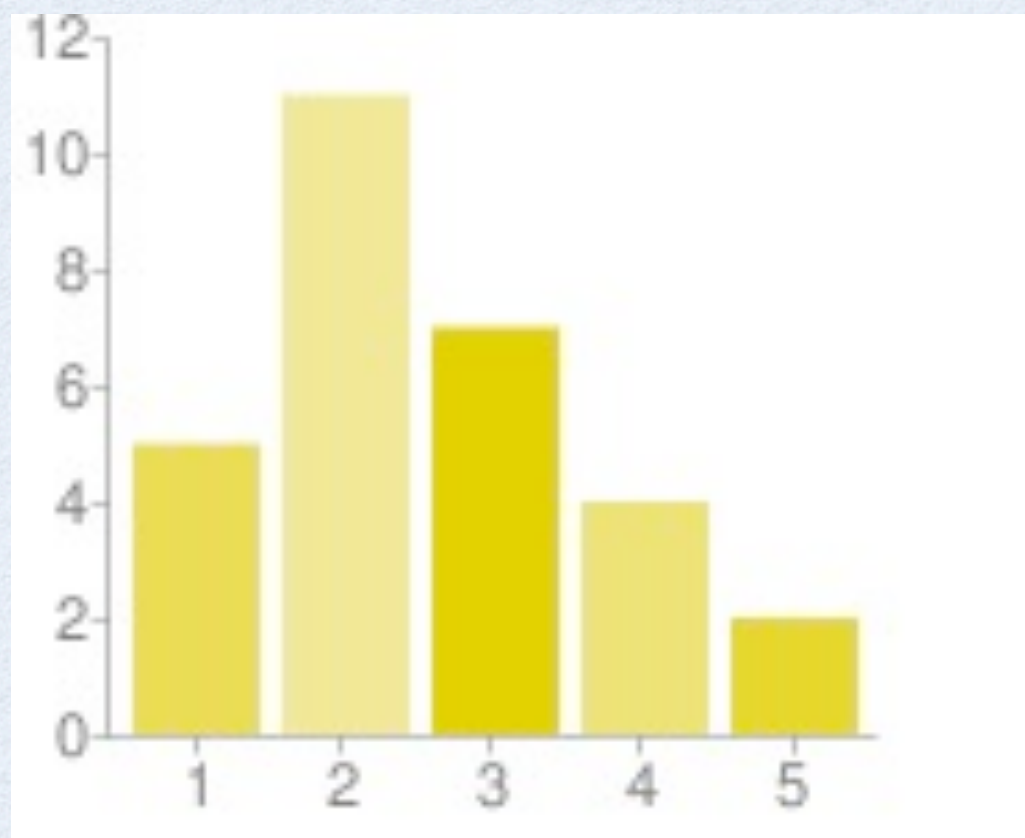


How comfortable were / are you  
with the programming language?



# Comfortable with IDE

*Scale from 1-5  
beginner - expert*

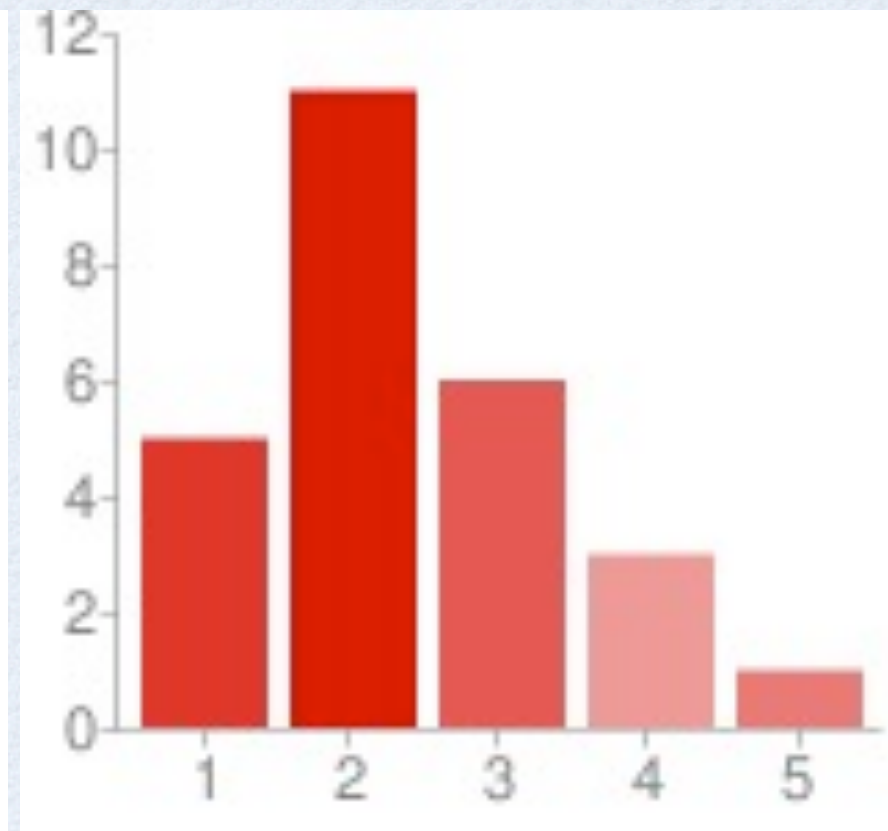
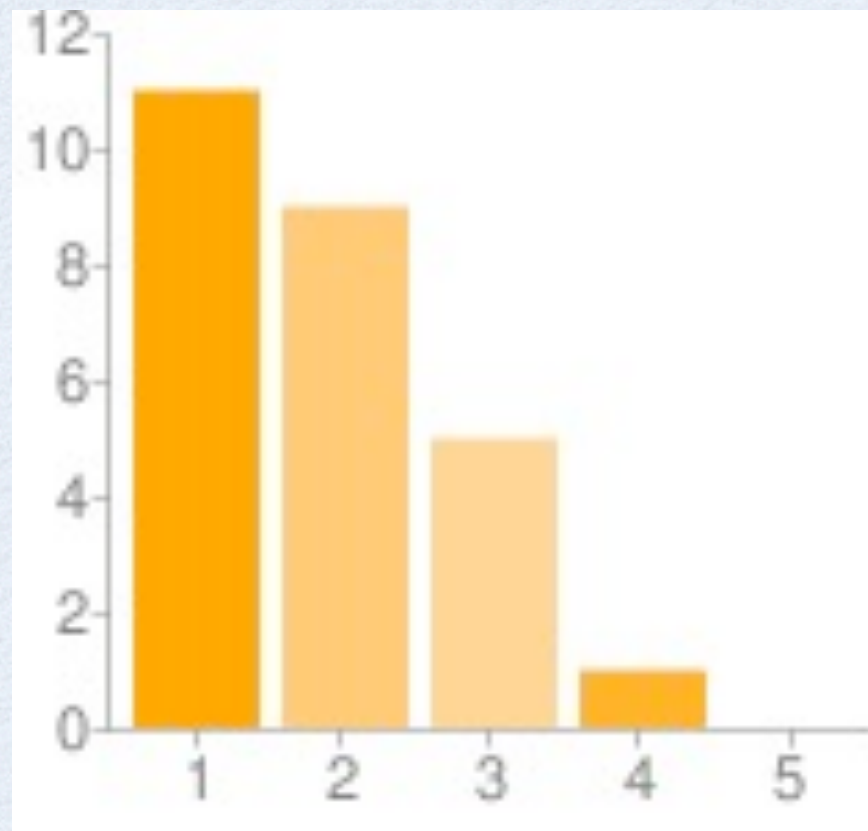


How comfortable were / are you with the IDE-editor?



# Mock objects

*Scale from 1-5  
beginner - expert*

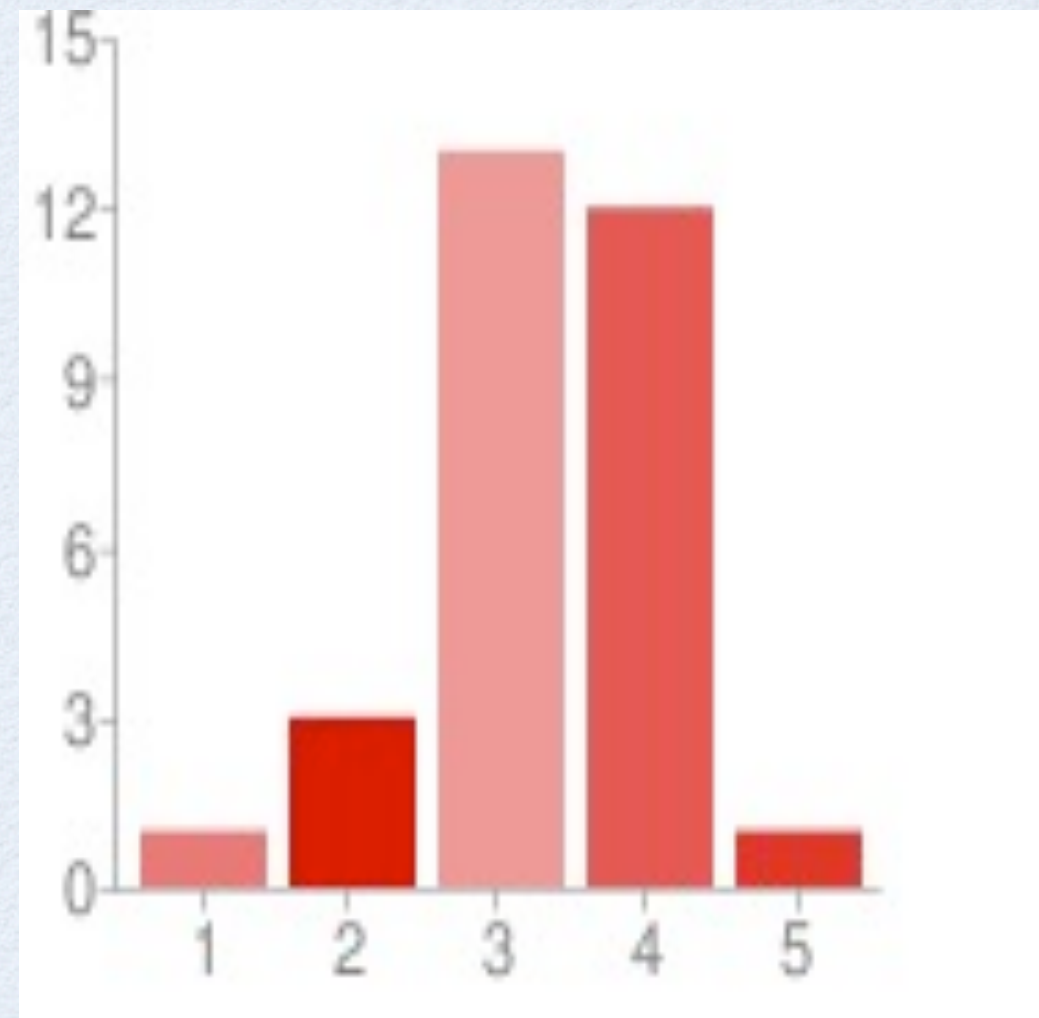
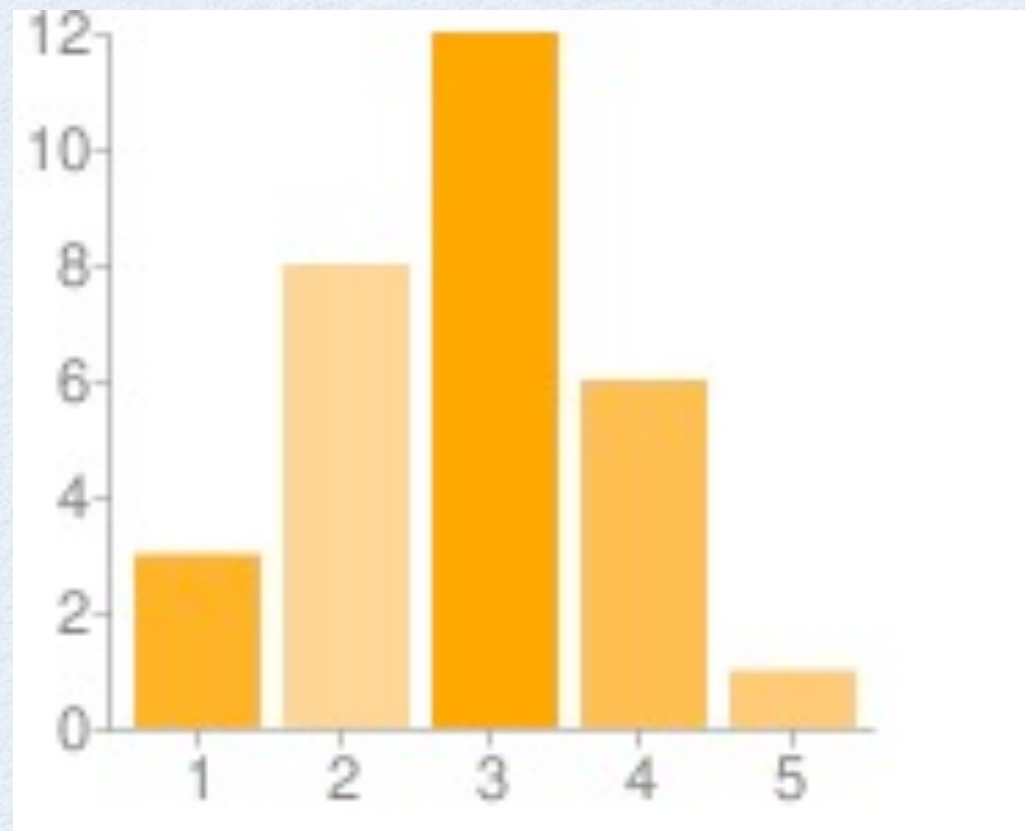


How good were / are you at using Mock objects?



# Refactoring

*Scale from 1-5  
beginner - expert*

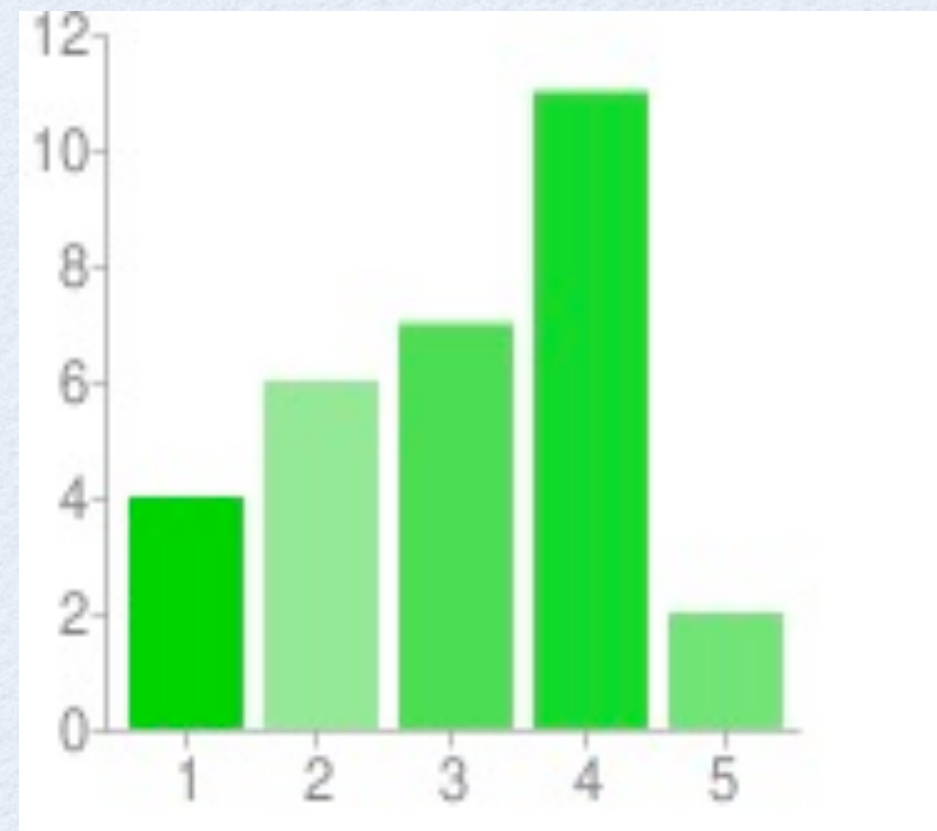
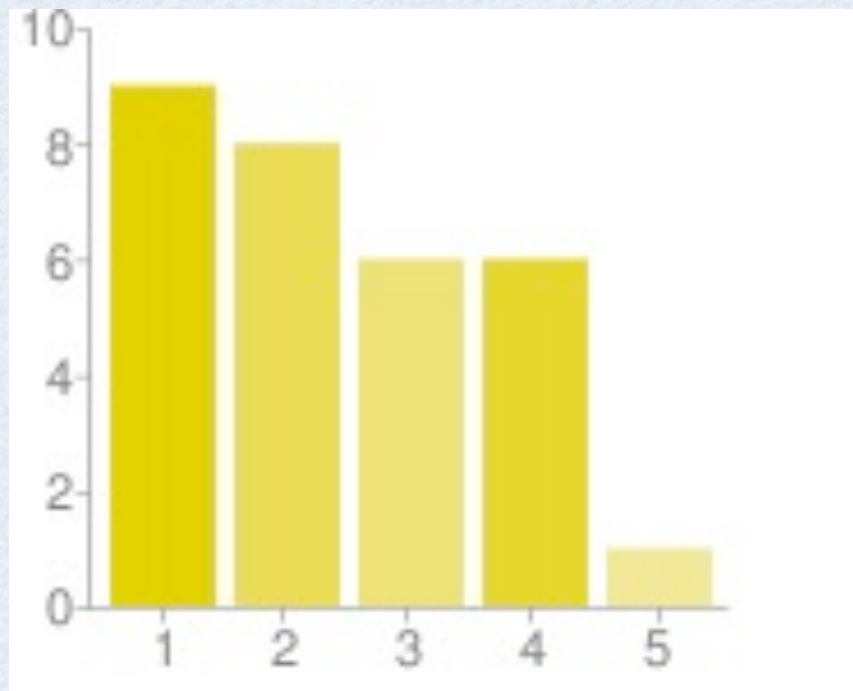


How good at refactoring were / are you?



# Write Unit Tests Often?

*Scale from 1-5  
never - daily*

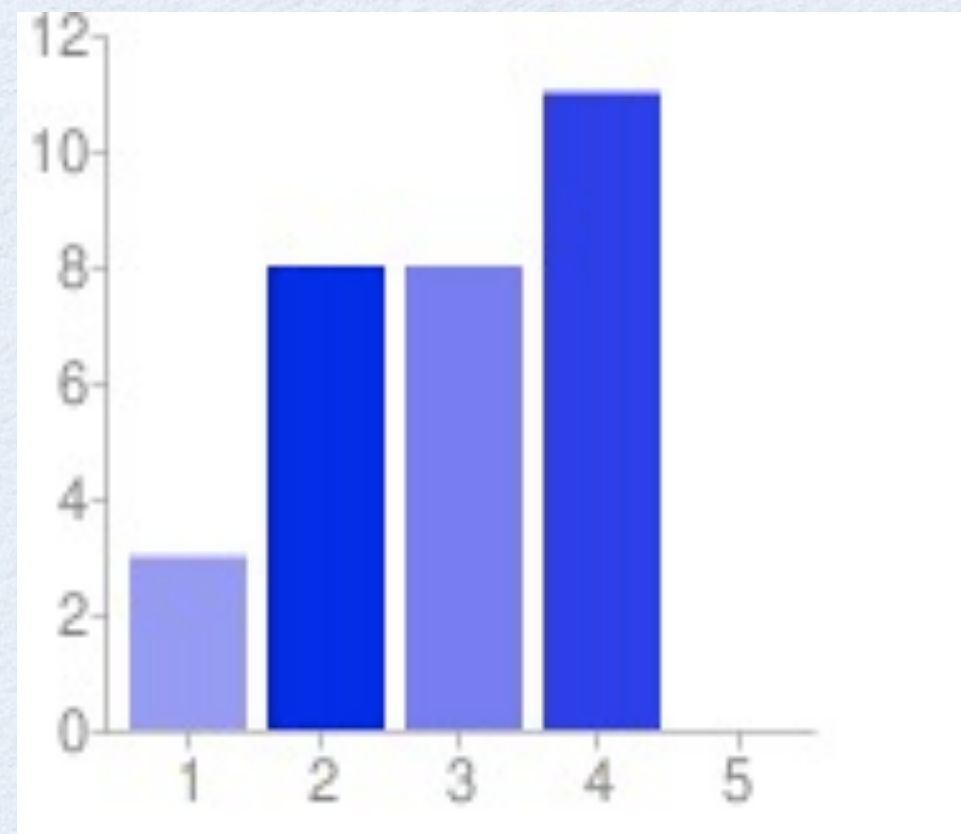
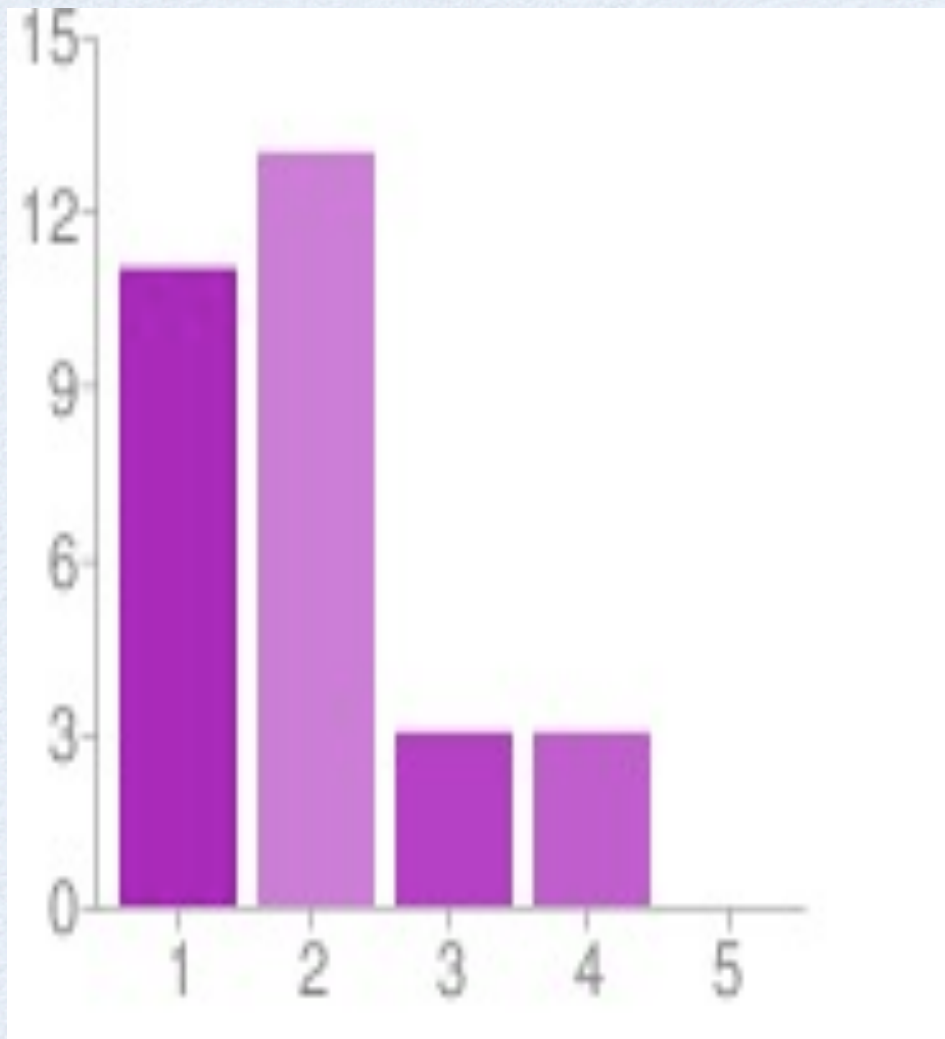


How often did / do you write unit tests?



# Test-Driven Development

*Scale from 1-5  
beginner - expert*



How good at Test-Driven Development were / are you?



# How to get started



# You could join a dojo!

- Lots of them exist already
- It's fun, rewarding and you get to meet new people
- Or, hire a coach to run one at your company!



Image attribution: Flickr user lumaxart



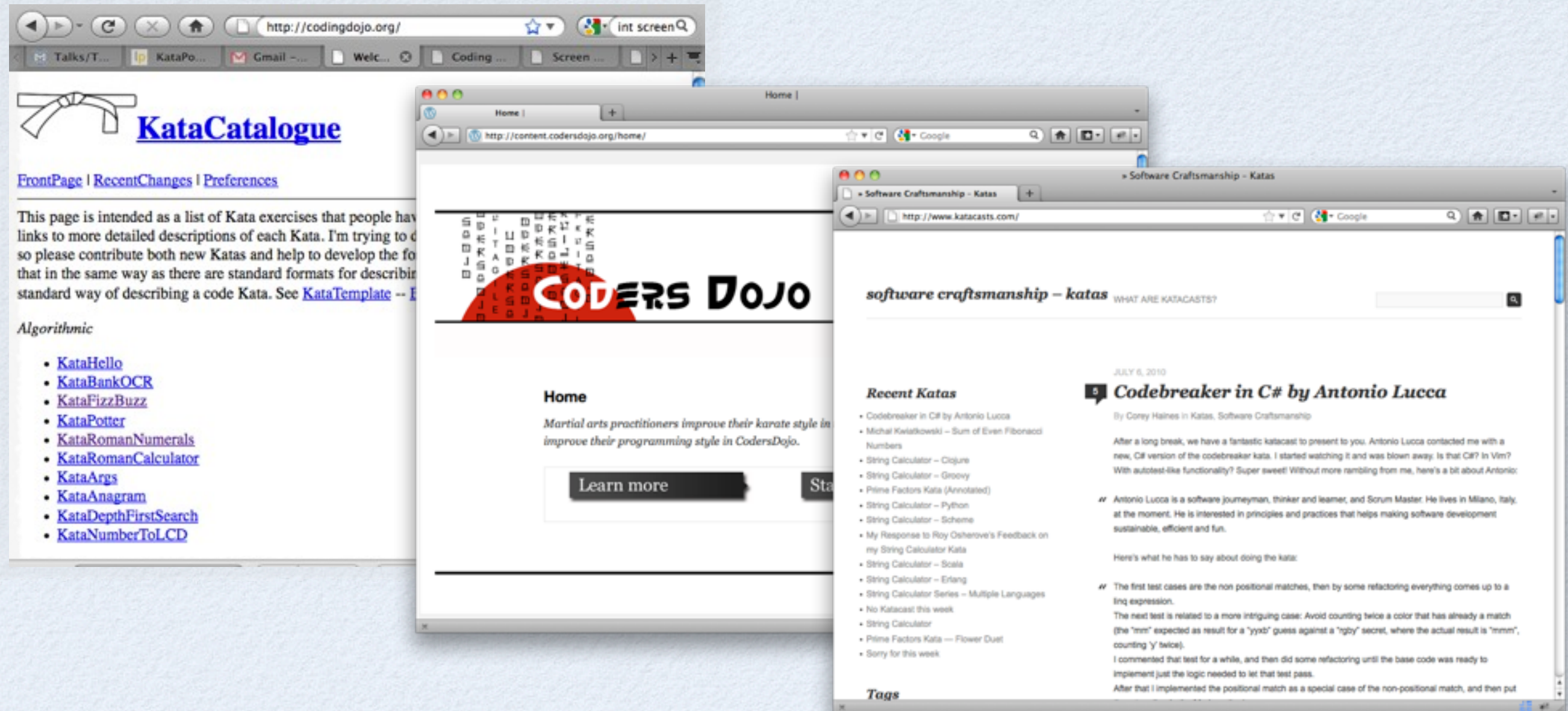
# You could start a dojo!



- tools: meetup.com, google groups
- persuade a friend to help organize
- Get someone to pay for pizza!



# Online Resources



- <http://codingdojo.org>
- <http://codersdojo.org>
- <http://katacasts.com>

- <http://codingkata.org>
- <http://cyber-dojo.com>
- <http://katacatalog.com>

material copyright Bache consulting, emily@bacheconsulting.com



# Still trying stuff out...

- give out kata in advance?
- what to do with code afterwards?
- groups for beginners and different groups for experts?
- if/when to bring in legacy code
- what form to use when? (prepared, randori, pairs etc)



# The Coders Dojo

Emily Bache

[emily@bacheconsulting.com](mailto:emily@bacheconsulting.com)

<http://emilybache.blogspot.com>