Little Languages

Lecture 19:

Code Review & Pair Programming

A little bit about myself...

- Currently a senior majoring in computer science and mathematics

- Interned at Google Mountain View (headquarters) summers of 2017 and 2018
  - Google Photos
  - Google Image Search

What is Code Review

- You write some code, and then ask your peer to review it.
- The reviewer(s) reads your code,
  - raises questions, and
  - provides suggestions on how to improve the code
- Line-by-line basis
- Voluntary
  - Anyone should be able to code review
  - You do not have to follow any suggestions

Why Code Review?

- Improve code quality
  - More eyes are more likely to catch more bugs
  - “The sooner you find bugs, the better”
  - Improve project reliability
- Promote responsibility of the author
  - Logical correctness
  - Readability

Why Code Review?

- Promote bi-directional mentoring and learning
- Promote openness in company culture
  - Nothing is a secret
  - Open to mistakes - everyone makes mistakes!
- Enforce uniform standard

Different Forms of Code Review

- Formal tool
- Over-the-shoulder
- Email threads
- Walkthroughs during meeting

Also may differ in terms of frequency, requirement, goals, etc.

Code Review at Google

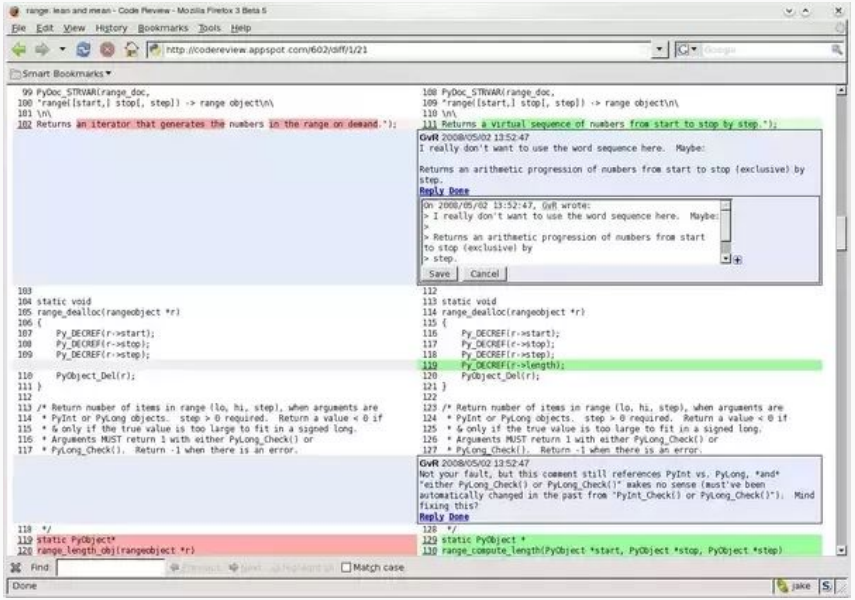
- Every line of code to be submitted needs to pass the code review first.

Code Review at Google

- To pass the code review, need 2 LGTMs (LGTM = looks good to me)
  - From an owner of the code section
  - From a developer with the language readability for the reviewed code
    - http://google.github.io/styleguide/

Code Review at Google

- Typically done through a web interface
  - The reviews that you sent out
  - The reviews that you need to do



Picture from: https://www.quora.com/What-is-Googles-internal-code-review-policy-process

Live Demo

Let’s do a real code review

https://github.com/QZHelen/CodeReview590

Recap: Code Review at Google

- Make edits to some code
- When it is ready to submit, send it out to code reviewers (web interface)
- Reviewers view diff
- Potential back and forth communication, with the goal to reach an agreement
- Finally, when the reviewers reply “lgtm”, you can submit your code!

My Personal Experience with Code Review

- It helped me get quickly familiar with software development at Google, and taught me a lot new knowledge and practices.

- However, it will not always completely prevent errors.

Takeaways from Code Review

- As a developer, achieving the functionality is not the only goal.
  - Readability
  - Maintainability and extensibility
  - Security
- Programming is never a lone work
  - Involves constantly learning from others
  - Communication everywhere

Questions?

References

- https://www.youtube.com/watch?v=sMql3Di4Kqc

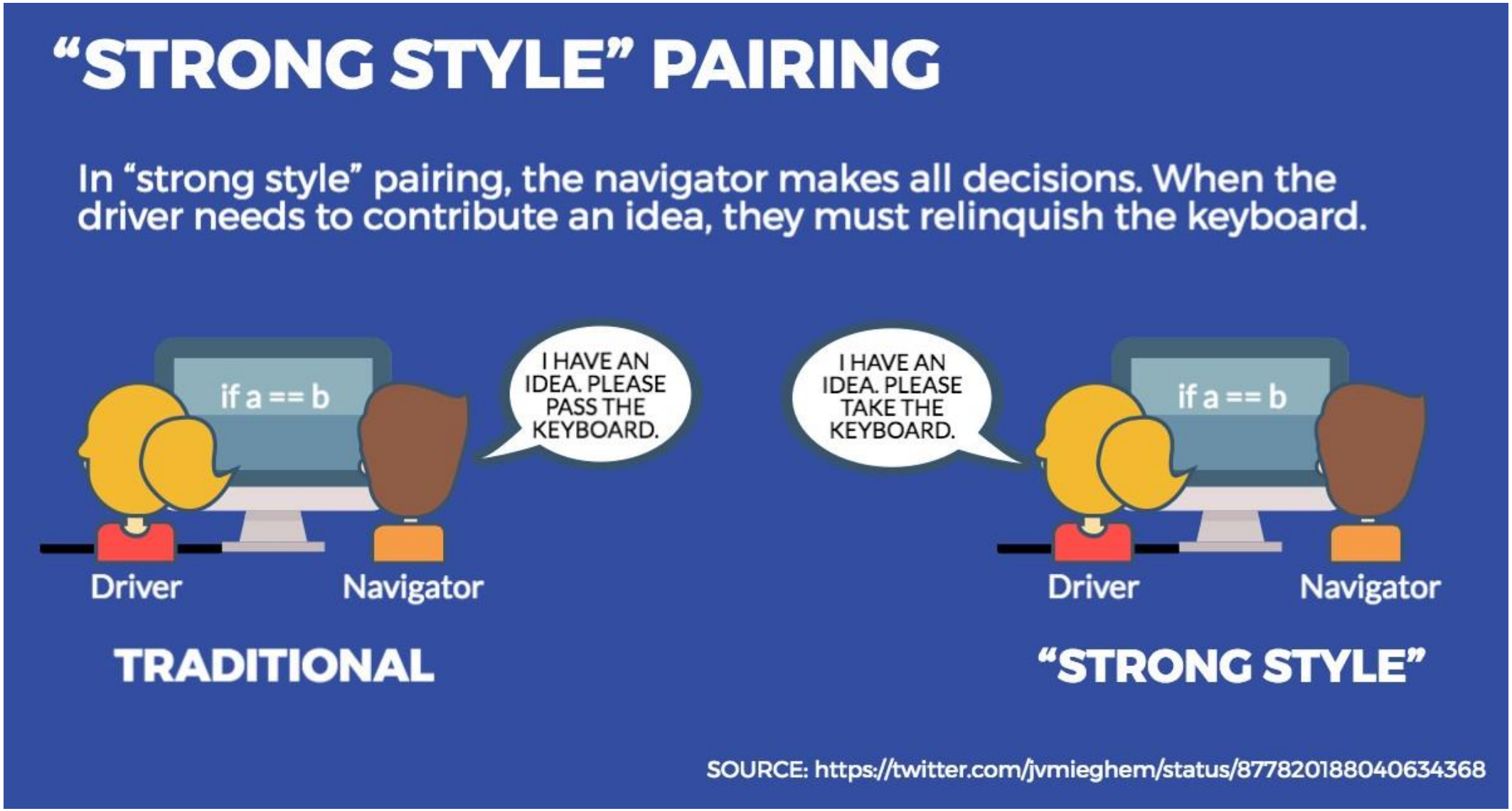
Pair Programming is… *Programming in Pairs*

- Two people collaborating **synchronously** on the **same unit of code**, usually in person and with only one keyboard input.

- Often one person is the *driver* and the other is the *navigator.*
  - Swapping roles with frequency is *strongly* encouraged.

- Why do people and organizations embrace pair programming?
  - Improves quality of resulting code. Reduces time spent being stuck on small stuff.
  - Promotes sharing of knowledge and improves ability to communicate about code.
  - Increases confidence in and enjoyment of programming.

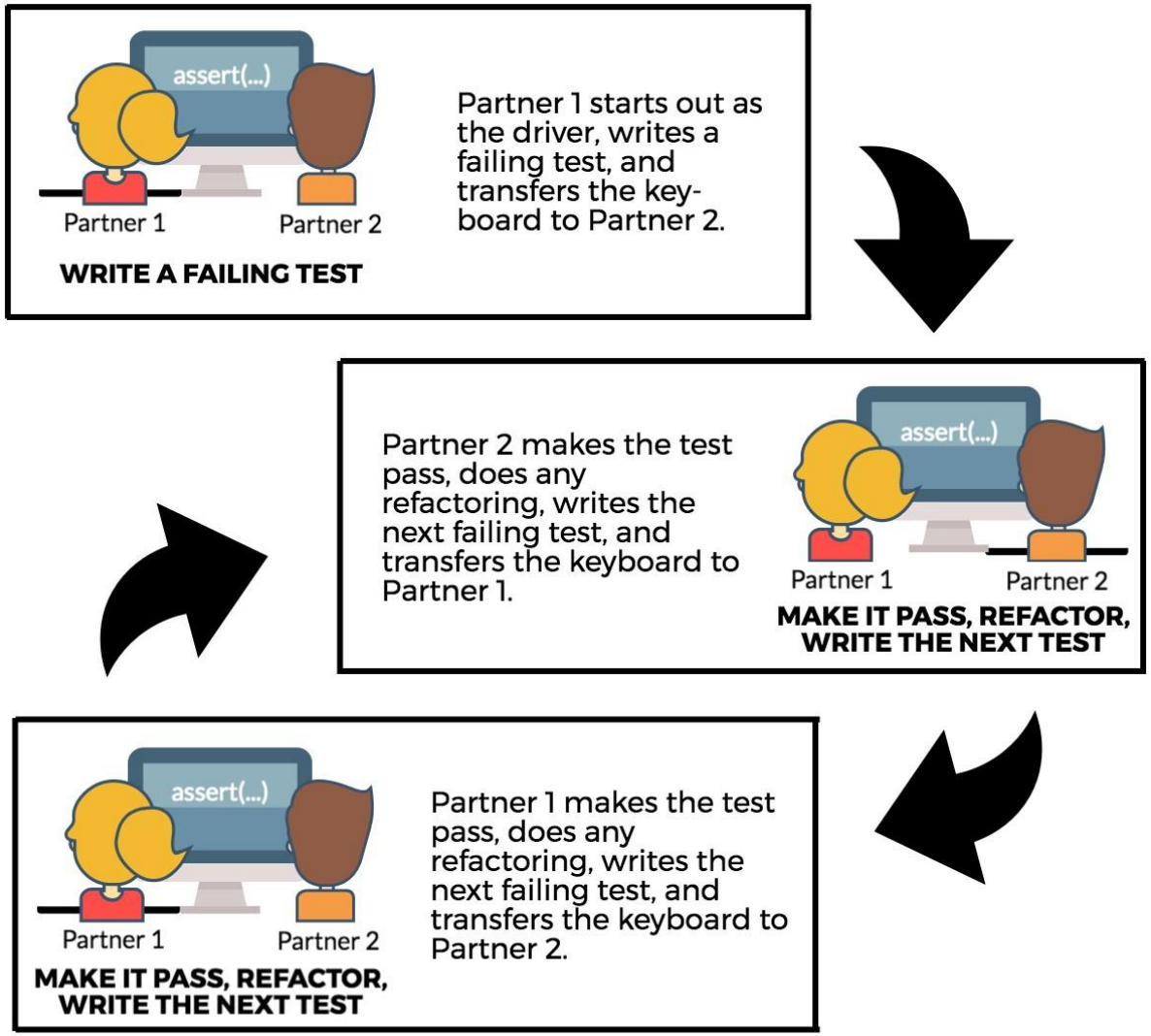
Pair Programming Guide by Weblab –
https://medium.com/@weblab_tech/pair-programming-guide-a76ca43ff389



Pair Programming Guide by Weblab –
https://medium.com/@weblab_tech/pair-programming-guide-a76ca43ff389

“PING PONG” PAIRING

In “ping pong” pairing, the “write a failing test”, “make it pass”, “refactor” loop of Test-Driven Development is used.



Pair Programming Guide by Weblab – https://medium.com/@weblab_tech/pair-programming-guide-a76ca43ff389

Pair Programming is Embraced in Many Methodologies

What Is eXtreme Programming?

eXtreme Programming is a software development method that favors informal and immediate communication over the detailed and specific work products required by many traditional design methods. Pair programming fits well within XP for reasons ranging from quality and productivity to vocabulary development and cross training. XP relies on pair programming so heavily that it insists all production code be written by pairs.

XP consists of a dozen practices appropriate for small to midsize teams developing software with vague or changing requirements. The methodology copes with change by delivering software early and often and by absorbing feedback into the development culture and ultimately into the code.

Several XP practices involve pair programming:

- Developers work on only one requirement at a time, usually the one with the greatest business value as established by the customer. Pairs form to interpret requirements or to place their implementation within the code base.
- Developers create unit tests for the code's expected behavior and then write the simplest, most straightforward implementations that pass their tests. Pairs help each other maintain the discipline of writing tests first and the complementary, though quite distinct, discipline of writing simple solutions.
- Developers expect their intentions to show clearly in the code they write and refactor their code and other's if necessary to achieve this result. A partner who has been tracking the programmer's intention is well equipped to judge the program's expressiveness.
- Developers continuously integrate their work into a single development thread, testing its health by running comprehensive unit tests. With each integration, the pair releases ownership of their work to the whole team. At this point, different pairings can form if another combination of talent is more appropriate for the next piece of work.

To learn more, see Kent Beck's book,[1] or consult the eXtreme Programming Roadmap at xp.c2.com, where a lively community debates each XP practice.

Reference

1. K. Beck, *eXtreme Programming Explained: Embrace Change*, Addison Wesley Longman, Reading, Mass., 2000.

Williams, Laurie; Kessler, Robert R.; Cunningham, Ward; Jeffries, Ron (2000). "Strengthening the case for pair programming" (PDF). *IEEE Software*. **17** (4): 19–25. CiteSeerX 10.1.1.33.5248. doi:10.1109/52.854064.

Tips for Pair Programming

- Switch roles frequently.
  - No hard rules but every 20-30 minutes or after an individual feature is completed.
- Start with writing a failing test where it's possible to.
- Listen more than you talk. Ask questions more than command.
- Divide the load of thinking about levels of abstraction.
  - Driver concerned with implementation details, compiling and correct code.
  - Navigator concerned with higher-level: code organization, timeboxing, tests and functionality to add next.
- "Yes, and…" – accept your partner's line of thinking and carry out their thought experiment.
- Take breaks. Don't be that person and code while your partner is gone.
- After a session: reflect on what worked, what didn't, and how you felt the process went.

Aside on Software Development Methodologies

- Methodologies are like programming languages and belief systems…
  - Humans canonize them, so they're inherently imperfect.
  - Teams have their own interpretation of how to prioritize rules.
  - There is no singular, obviously right methodology for everyone.

- **"Be wary of a Gary"** *(the pit preacher)*
  - Be especially critical of people who believe they alone have the only answer to everything
  - These people are often writing the books, blog posts, on speaker circuits, and so on

- **What's important is *thinking* about *your process* and *improving yourself* with *intent*.**
  - Don't get *too* meta, though!
  - Progress over process!
    - Process is often easier to wax poetic about than the actual work of making forward progress.