Baby Steps Push Challenge

February 2021

Peter Kofler, 'Code Cop'
@codecopkofler
www.code-cop.org

Peter Kofler

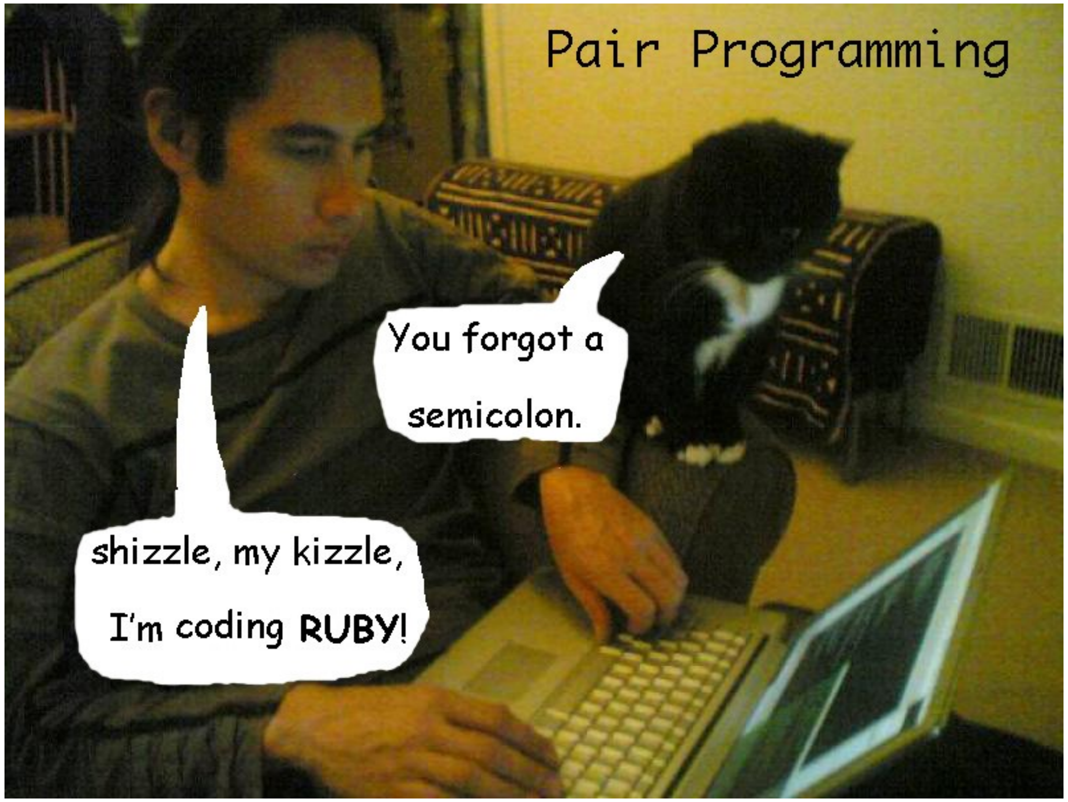
- Ph.D. (Appl. Math.)
- Professional Software Developer for 20+ years
- “fanatic about code quality”
- Independent Code Quality Coach

I help development teams with

- Professionalism
- Quality and Productivity
- Continuous Improvement

Mentoring

- Pair Programming
- Programming Workshops
- Deliberate Practice, e.g. Coding Dojos



Developing Quality Software Developers

Working in small increments

- What are the benefits of working in small increments?
- In which areas do we try to work in small increments?

Small increments everywhere

- Stories
- TDD (TCR)
- Unit Tests
- Refactoring
- Integration
- Delivery

Small increments in TDD?

- Get feedback sooner, if your idea works.
- Use the safety net (provided by tests) as soon as possible.
- Cleaning up a little new code is easier.
- Faster to go back a small step (Ctrl-Z).

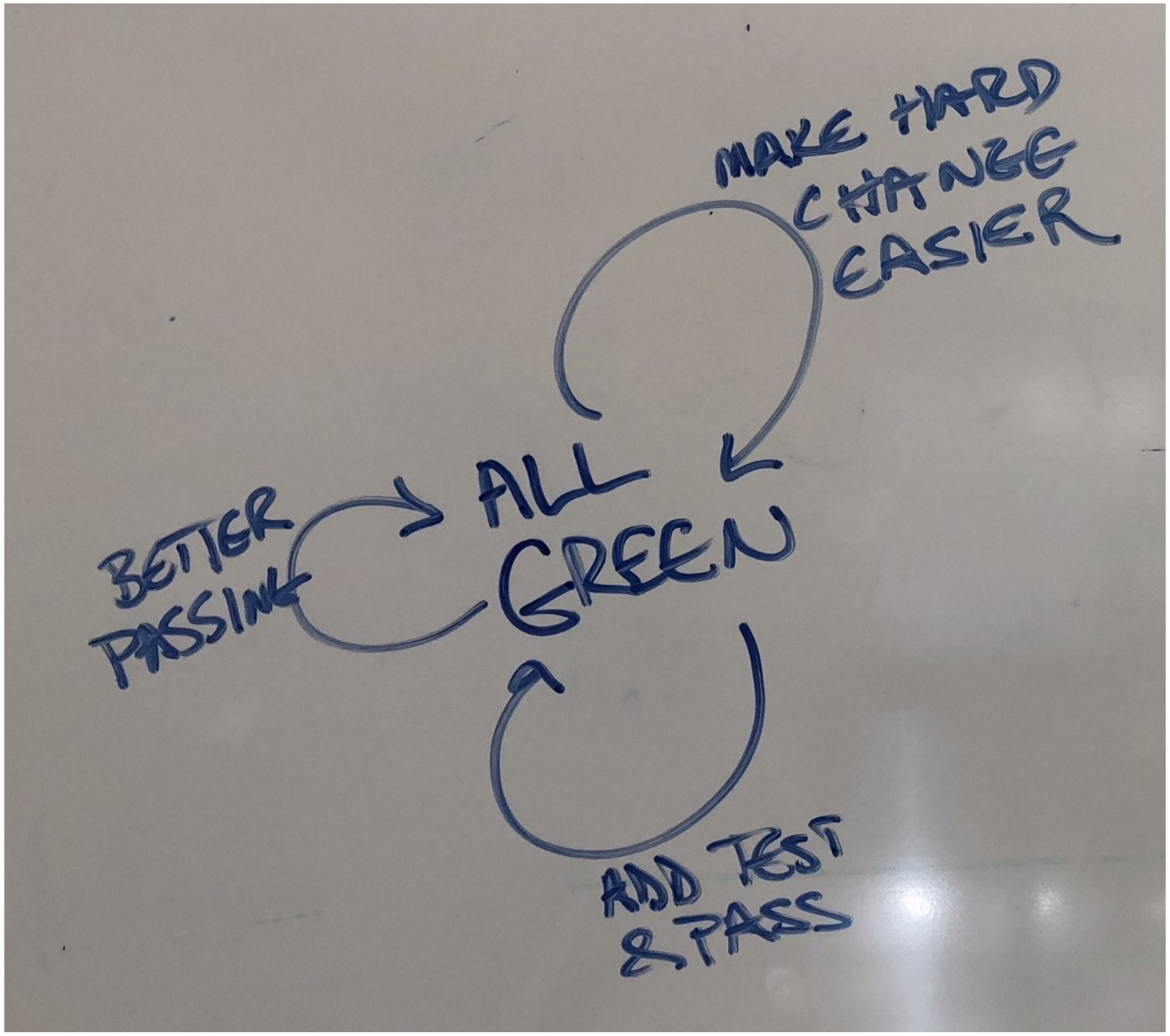
How to make smaller steps?

- Focus on one idea;
  introduce one concept at a time.
- Chose another test if you need to write
  a lot of (untested) code to make it pass.
- Try a smaller part of problem,
  - e.g. reduce problem dimension,
  - e.g. reduce inputs (hard-code others).

(we talked about this before)

Use Baby Steps

Smaller Increments in TCR!



How to make even smaller steps?

- Use constants (fake it).
- Add a passing test to get started
  (which asserts the wrong thing at first).
- Improve passing test/passing code a bit.
- Prepare code for changes
  (“make hard change easy”).
- An endless stream of tiny changes.

Small unit tests?

- Test only one thing at a time - test fails for only one reason.
- Small tests (usually) run faster, thus providing feedback sooner.
- Smaller methods are easier to understand.

How to write smaller unit tests?

- One assertion (of concept) per test.
- Split expected outcome by concern.
  - e.g. method saves data and sends email
- Use parametrised tests to avoid duplication.
  - First change existing test to use a single parameter (as refactoring).
- Extract shared test setup to reuse later (as refactoring).

Baby Steps in Refactoring?

- Small transformations are less likely to go wrong.
    - e.g. a single Rename
    - e.g. a single Extract Method
- Only small refactorings supported by IDE.
- Code is working all the time. We can stop whenever we need to.
- Sequence of transformations produce a significant restructuring.

Smaller Refactoring Steps #1

- Parallel Change
    - Expand: Add logic parallel to old one.
    - Migrate: Switch usage one by one.
    - Contract: Remove old logic.
- Useful for many refactorings which are not supported by IDE.

Smaller Refactoring Steps #2

- Branch by Abstraction
  - Use Parallel Change to hide logic you want to change behind abstraction.
  - Add new subclass for the new logic.
  - Switch all clients to new abstraction.
  - Remove abstraction (if inelegant).

- Instead of branching by source control.

Integrate Often?

- Know if your code integrates sooner.
- Run system tests to verify that parts work together.
- Smaller changes - less merge conflicts.
- Your changes are available for others sooner - we can collaborate on tasks.

How to integrate more often?

- Merge & Push whenever your tests are green.
    - IntelliJ Commit and Push: Ctrl+Alt+K
    - IntelliJ Push: Ctrl+Shift+K
- IntelliJ: Maybe Limited WIP plugin?
- Maybe use Limbo? (Constantly rebasing, committing and pushing your micro-changes.)
- Use a build notifier in the task bar.
- The build must not go red.

Deliver small increments?

- Smaller releases are less likely to fail.
- Rollbacks are smaller and faster.
- Get new features, experiments, bug fixes, etc. to your users sooner.
- A/B test some new features in early stages.

Hands-on Exercise

Coding Dojo Mindset

- Safe place outside work
- We are here to learn
- Need to slow down
- Focus on doing it right
- Collaborative Game

Assignment: Parrot

Refactor Parrot (Type Code)

- Parrot calculates speed of types of parrots.
- Replace Type Code with Subclasses

- Find a pair.
- Clone the repo, branch `cp-trunk`
- Create a branch with your team name.
- Push a change to the Readme to test setup

→Replacing Type Code

(participants' presentations)

https://refactoring.guru/refactoring/techniques

Replacing Type Code

- Replace Type Code with Class
- Replace Type Code with Subclasses
- Replace Type Code with State/Strategy
- Replace Subclass with Fields

https://refactoring.guru/refactoring/techniques

Challenge

Push Counter Challenge

- Leaderboard shows score of each team https://push-counter.herokuapp.com/
- Commit and push green increments to score points.
- Repeat: Push as often as possible!
- Try to perform maximum number of smallest steps and test runs.
- Upper limit as around 40 changes...

Don't Focus on Getting it Done. Focus on Doing It Perfectly.

Practice

Closing Circle

- What did you learn today?
- What surprised you today?
- What will you do differently in the future?

Peter Kofler
@codecopkofler
www.code-cop.org

Kata by

Emily Bache
@emilybache

CC Images

- gums https://www.flickr.com/photos/sunface13/427985982/
- Bruce http://www.flickr.com/photos/sherpas428/4350620602/
- pairing http://www.flickr.com/photos/dav/94735395/
- wants you http://www.flickr.com/photos/shutter/105497713/
- baby https://www.flickr.com/photos/11904001@N00/3983980813/
- hands https://www.flickr.com/photos/ninahiironniemi/497993647/
- dojo http://www.flickr.com/photos/49715404@N00/3267627038/
- parrot https://www.flickr.com/photos/crunchyfrog/125645513/
- tug https://www.flickr.com/photos/marine_corps/25686193850/