Pair Programming

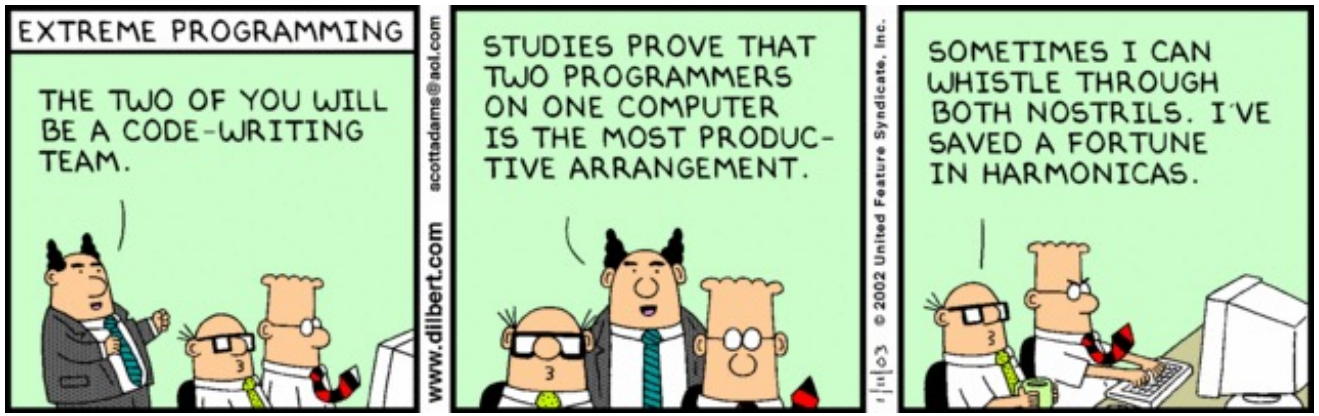
An in-depth analysis

Ermanno Scanagatta
github.com/ermannos

What is Pair Programming?

Two devs writing code on one machine in a collaborative way

It's not that easy, rules matter!



Pair programming styles

Style defines ROLES of each developer and RULES that drive the development

- **Traditional driver / navigator style**
- **Ping pong**
- **Strong-style pairing**

Traditional driver / navigator style

Driver

- the person that has the keyboard.
- focused on small goals, ignoring the big picture
- tactical view

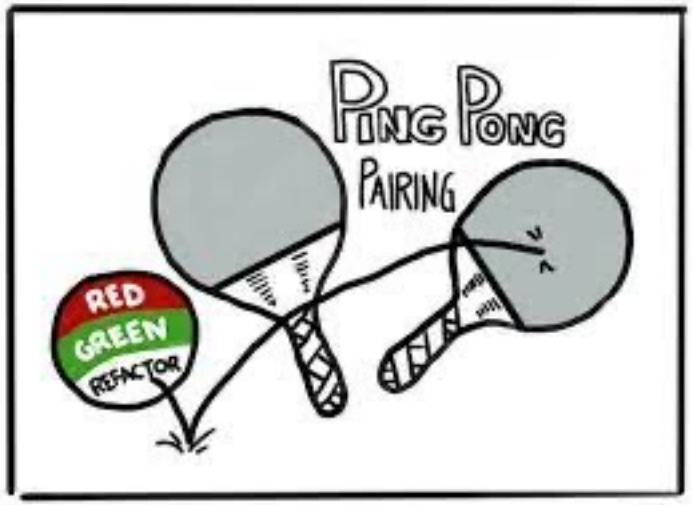
Navigator

- an observer, reviews the code while driver is typing
- share his toughts with the driver
- keeps the eye on the big picture (issues, bugs, obstacles)
- strategical view

Ping pong style

Ideal when you have a clear task that can be done with TDD

- **Dev 1**: writes a failing test
- **Dev 2**: writes the implementation to make it pass
- **Dev 1 & Dev 2**: make code refactoring together
- *swap devs and start again*



Strong-style pairing

Driver

- the person with the keyboard
- totally trusts the navigator
- should be confortable with uncomplete understanding

Navigator

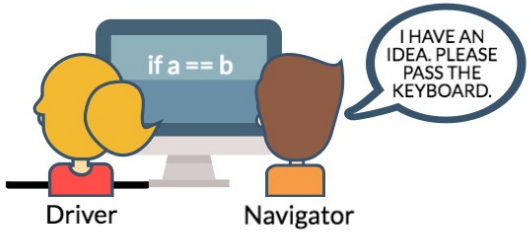
- gives the next instructions to the driver when they are ready to be implemented
- talks in highet level of abstraction the driver can understand

"For an idea to go from your head into the computer
it MUST go through someone else's hands" (Llewellyn Falco)

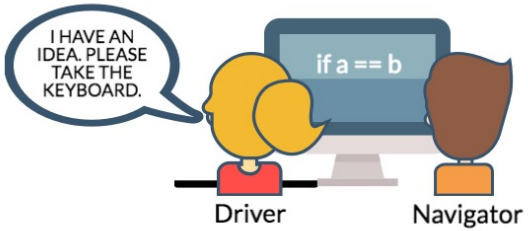
Common problems (in traditional pairing)

- **Navigator disengagement**
- **Fighting for the keyboard**
- **Expert/novice pairing**
- **Unexpressed thoughts**

Most of them are solved with strong-style pairing



TRADITIONAL



“STRONG STYLE”

Benefits

Keep the WIP low

Knowledge sharing

Reflection

Code review on-the-go

Two modes of thinking

Fast onboarding

Keeping focus

Collective code ownership

Time management

- **Pairing can be hard, breaks are crucial to tank energy (not for other work)**
  - do frequent short breaks (ex: 5' every 30')
  - take a long break (ex: 15") every 2 hours
- **Between two breaks work focused and without interruptions**
  - decide on what to work
  - avoid phone calls and distractions
- **Measure time**

Remote pairing

- **Choose a screen-sharing tool that allows to take control of other dev machine**
- **Video on please**
- **Consider using a sketching tool or a whiteboard**
- **Switch computers frequently (less lag when writing)**
- **Always take into account the human part**

Challenges

- Pairing can be intense
- Flow interruptions (calls/meetings)
- Different skill levels (technical or business knowledge)
- Power dynamics
- No time for yourself
- Context switching due to rotation
- Vulnerability needed
- Convincing managers and co-workers

Conclusions

Pair programming will make our code better!

The end

Thank you!

Questions?

References:

- Agile Technical Practices Distilled - A Journey Toward Mastering Software Design (Moreira Santos, Consolaro, Di Gioia)
- On Pair Programming: *https://martinfowler.com/articles/on-pair-programming.html*
- Llewellyn's strong-style pairing: *https://llewellynfalco.blogspot.com/2014/06/llewellyns-strong-style-pairing.html*