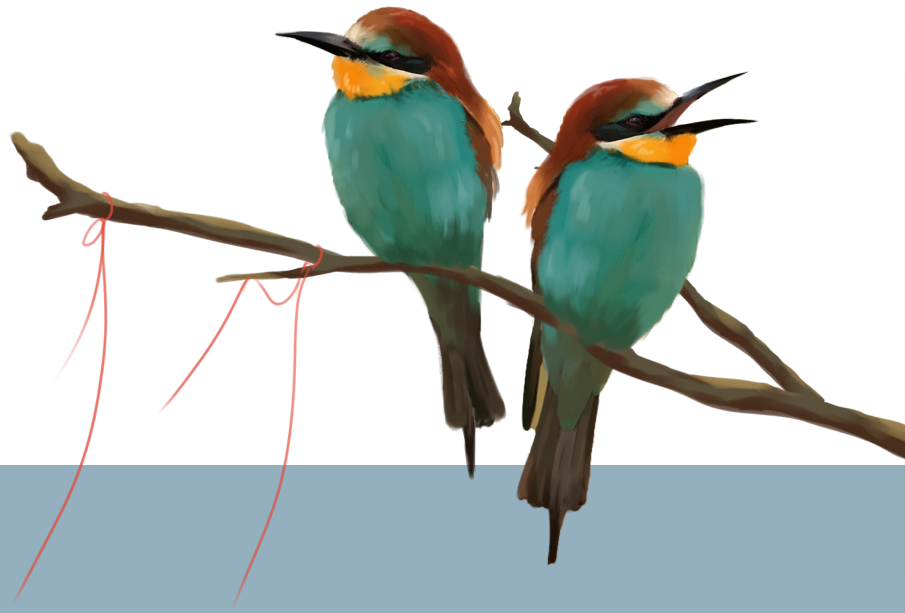
Australian National University

COMP2100/6442
Software Design Methodologies / Software Construction

Pair Programming



Bernardo Pereira Nunes

Slides based on https://martinfowler.com/articles/on-pair-programming.html

Outline

> Pair Programming
> How to pair?
> Styles
> Time Management
> Things to avoid
> Benefits
> Challenges

Pair Programming

> Two people write code together on one machine

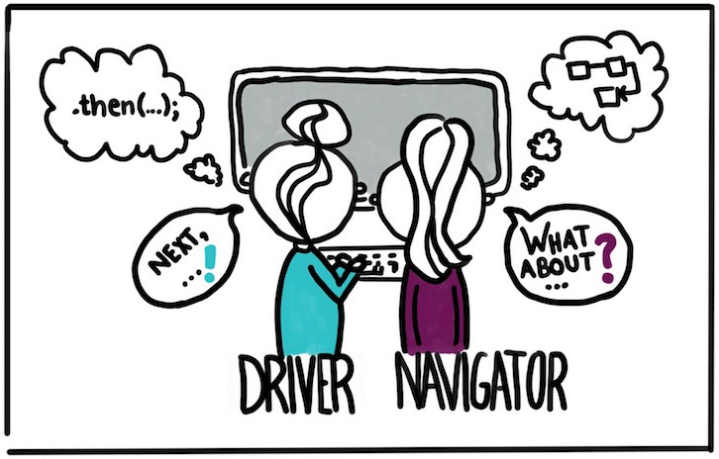
> Collaborative & Communicative

> It is not only about coding

>> It is also about planning and approaches that lead to better solutions and high-quality software

How to pair?

> Driver and Navigator



>> The **Driver** is the person at the keyboard, focused on completing the tiny goal at hand, ignoring larger issues for the moment and explaining what is being done. Driver is tactical.

>> The **Navigator** observes the driver and reviews the code on-the-go, gives directions and shares thoughts. The navigator also has an eye on the larger issues, bugs, and makes notes of potential next steps or obstacles. Navigator is strategic.

Common flow

> Start with a reasonably well-defined task.

> Agree on one tiny goal at a time (possibly based on unit test, a commit message, or written on a sticky note).

> Switch roles regularly to keep the energy level up and both learn and understand things better.

> As navigator, avoid the “tactical” mode of thinking, leave the details of the coding to the driver.

Styles

> PING PONG

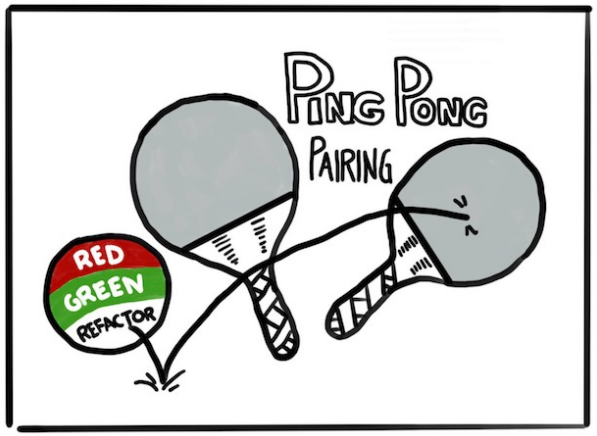
>> Uses the Test-Driven Development approach

>> Tasks are clearly defined and implemented following the TDD approach

> Example

Ping: Developer A writes a failing test
Pong: Developer B writes the implementation to make it pass. Roles are inverted.

It can also be followed by refactoring. This way you follow the "Red - Green - Refactor" approach: Write a failing test (red), make it pass with the minimum necessary means (green), and then refactor.



Styles

> PING PONG

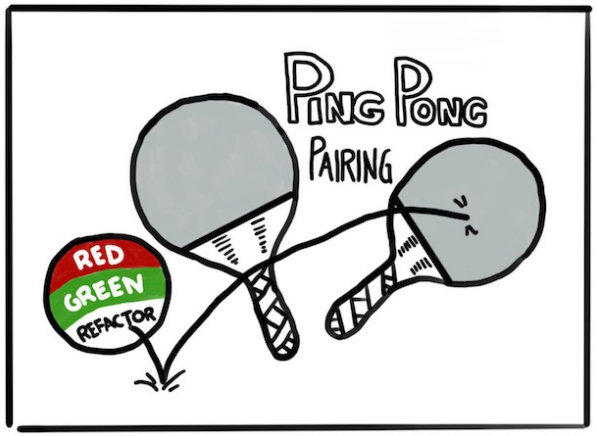
>> Uses the Test-Driven Development approach

>> Tasks are clearly defined and implemented following the TDD approach

> Example

Ping: Developer A writes a failing test
Pong: Developer B writes the implementation to make it pass. Roles are inverted.

It can also be followed by refactoring. This way you follow the "Red - Green - Refactor" approach: Write a failing test (red), make it pass with the minimum necessary means (green), and then refactor.



It's not just about coding

> Understand the problem

Read and discuss the problem before coding. Clarify any potential misunderstandings with the product owner.

> Come up with a solution

Brainstorm potential solutions (together or not)

*if research is required, follow a divide-and-conquer approach. After an agreed time, return and share/discuss your findings.

> Plan your approach

For the solution you chose, what are the steps you need to take to get there? Is there a specific order of tasks to keep in mind? How will you test this? Document your decisions.

Two people catching misunderstandings or missing prerequisites save a lot of time.

Documentation is also part of pair programming (write-and-review or both together).

Time Management | Pomodoro

> Breaks the work down into chunks of time

>> [25, 60] min separated by short breaks [5,10] min

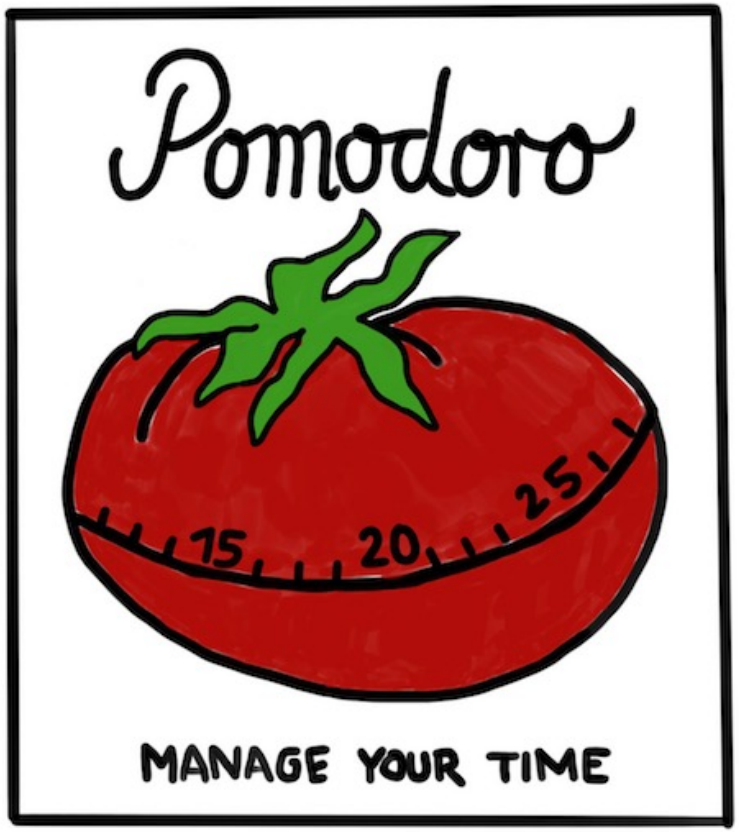
> Pairing can be exhausting – use a timer to remind you to switch roles | Plan your day: 8h of pair programming is not realistic (consider emails, meetings and other commitments).

> Work without interruptions and use breaks to take a break (avoid writing emails or other work, you need focus)

> Pair Rotations

> Change pairs regularly (every couple of days)

> Benefits: avoid knowledge silos, increase collective code ownership, get more code review on the go.



Things to avoid

> Do not use other devices (e.g., phone) or perform other tasks. Use the breaks if needed.

> Avoid micro-management mode:

>> "Now type 'System, dot, print, "...

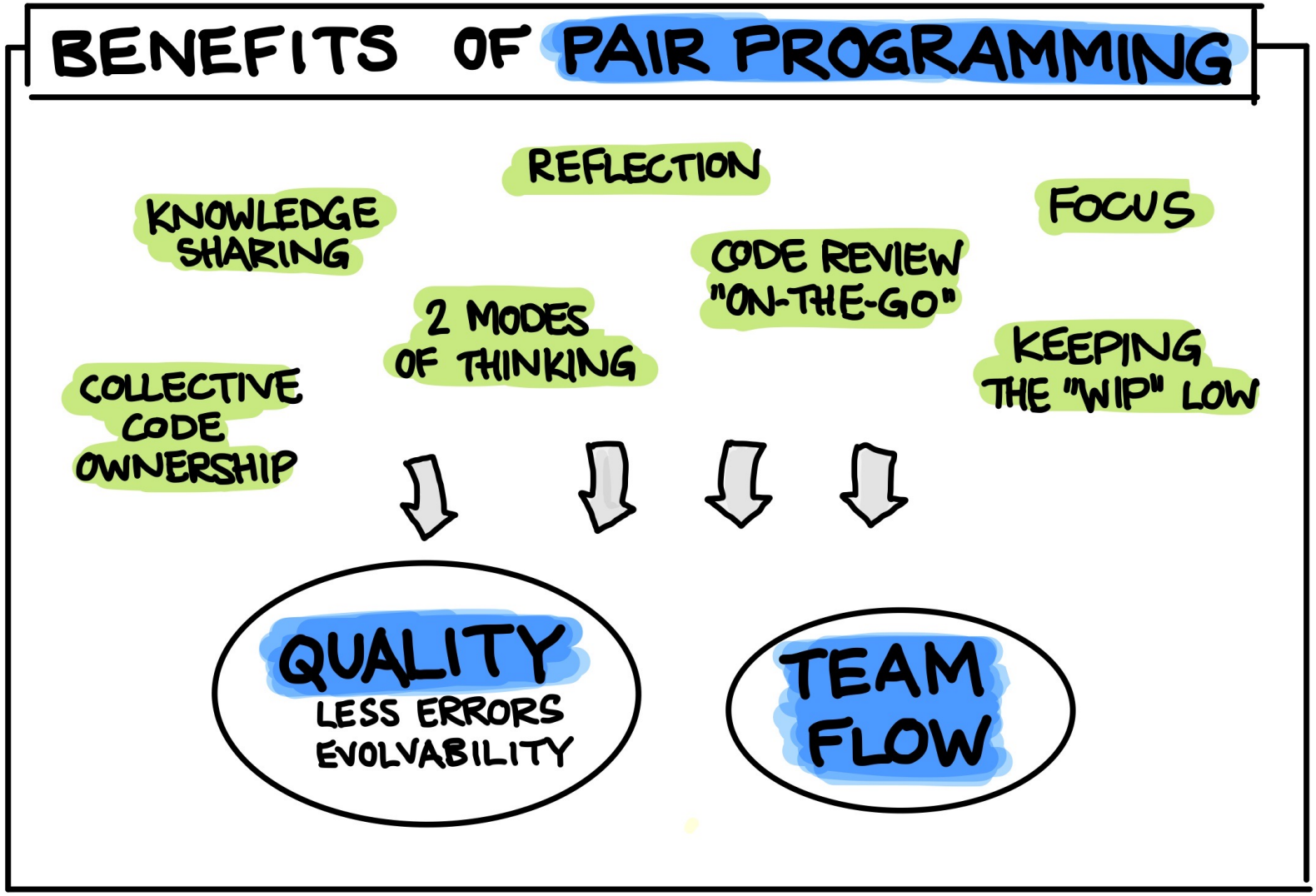
>> "Now we need to create a new class called..."

>> "Press command shift O..."

>> Apply the 5-second rule as a navigator. Wait at least 5 seconds before pointing to an error. It will help not interrupt the driver's flow.

> Watch out if you're "hogging the keyboard": Are you controlling it all the time, not letting your pairing partner do some typing as well?

> There is no "the right way" to pair programming – be flexible to get the benefits of pair programming.


BENEFITS OF PAIR PROGRAMMING
REFLECTION
KNOWLEDGE SHARING
FOCUS
CODE REVIEW "ON-THE-GO"
2 MODES OF THINKING
KEEPING THE "WIP" LOW
COLLECTIVE CODE OWNERSHIP
QUALITY
LESS ERRORS
EVOLVABILITY
TEAM FLOW

(some) Challenges

> Pairing is exhausting – take breaks and use Pomodoro

> Intense collaboration can be hard – requires communication and interpersonal skills

> Interruptions by meetings (block times both can focus on PP)

> Different Skill levels

>> often leads to false assumptions on how much each of them can contribute, or frustrations because of differences in pace

>> both can contribute with insights, discussions can lead to better solutions

>> explain as a means to check your own understanding

> Power dynamics: junior/senior | non-men/men | career changers/folks with a CS degree …

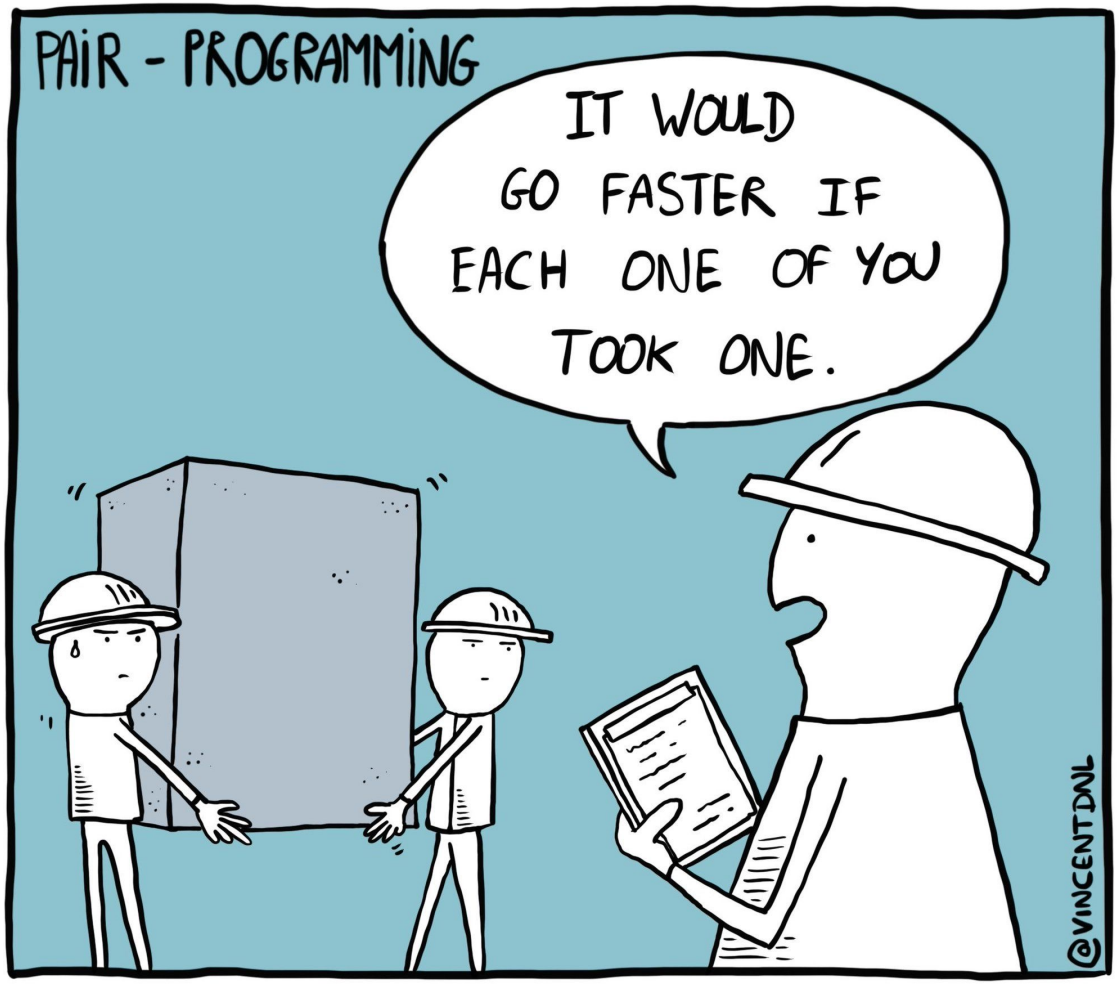
Examples:

>> One person is dominating the pairing session by hogging the keyboard and not giving room to their pairing partner.

>> One person stays in a teaching position and attitude all the time.

>> One person is not listening to the other one, and dismissing their suggestions.

Meme for today's lecture! Keep practicing!



Main References

[1] Pair Programming. https://martinfowler.com/articles/on-pair-programming.html