Pair Programming

/thoughtworks

Agenda

What is it?	03
Benefits	06
Approaches	08
Challenges	16
Let’s discuss	18

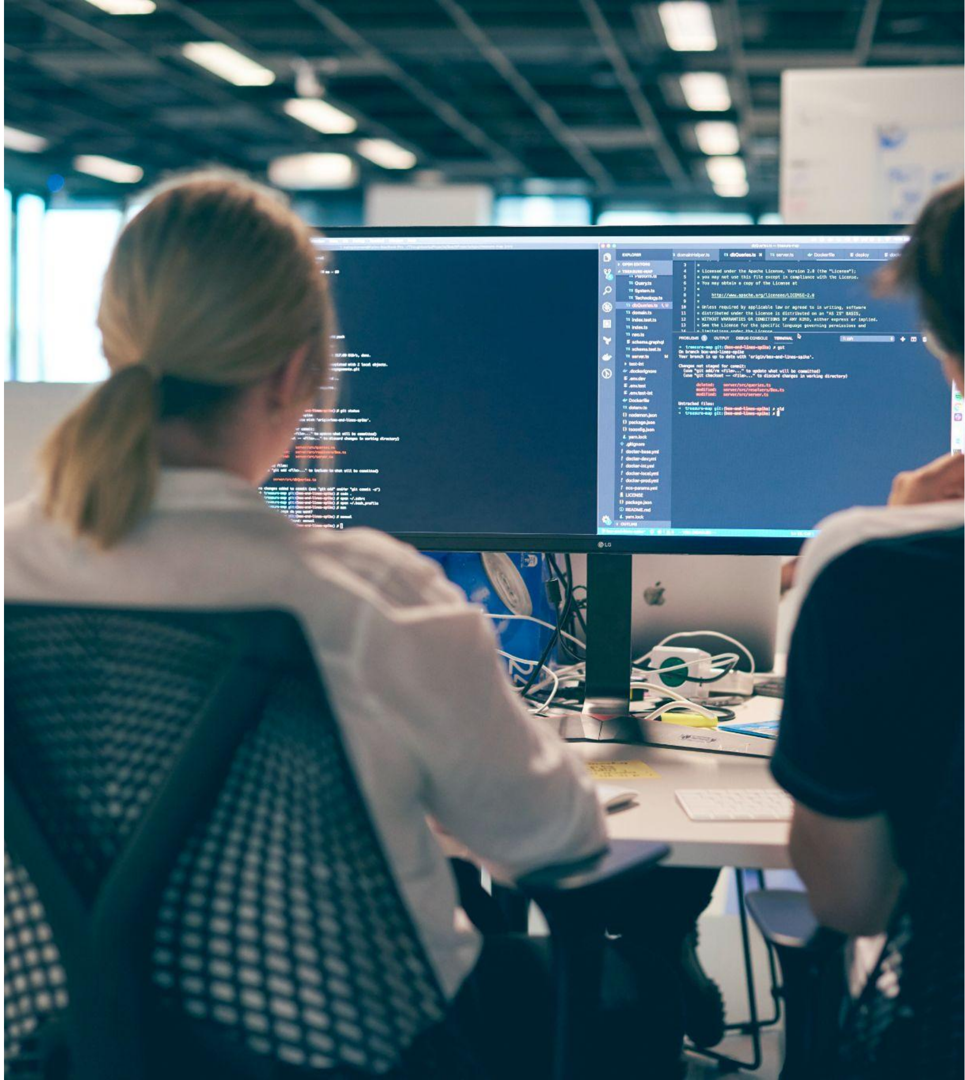
What is it?

/thoughtworks

What is Pair Programming?

“All code to be sent into production is created by **two people working together** at a **single computer**. Pair programming increases software quality without impacting time to deliver. It is counter intuitive, but two people working at a single computer will add as much functionality as two working separately except that it will be much higher in quality. With **increased quality** comes **big savings later** in the project.”

http://www.extremeprogramming.org/rules/pair.html



Misconceptions

You have to do pair programming if you're doing an agile process.

“This is utterly false. 'Agile' is a very broad term defined only in terms of values and principles… commonly observed practice, not a prescription.”

Pair-Programming halves the productivity of developers.

“My flippant answer to this one is:
‘that would be true if the hardest part of programming was typing’.”

It's only worth pairing on complex code, rote* code yields no advantage.

* rote = routine

I think there is a point to this (…), yields little opportunities for pairing to make a difference. Except this: writing boring rote code is usually a sign that I've missed an important abstraction (…) Pairing will help you find that abstraction.

Reference: https://martinfowler.com/bliki/PairProgrammingMisconceptions.html

Benefits

Why do pair programming? Awareness of all its benefits is important to decide when you do it, how to do it well, and to motivate yourself to do it in challenging times.

/thoughtworks

Helps keep focus

Ask for explanations from each other to stay on track.
Make plans together by thinking about each step to reach your goal.

Higher quality product

Immediate feedback.
Refactoring is part of pairing.
Fewer defects.
Less churn.

Knowledge sharing, reduced silos

Share knowledge on technology and domain on a daily basis and prevents silos of knowledge.

Simplicity of design

A pair can stop their partner from going down a rabbit hole, or suggest a simpler or better solution.

Fast onboarding of new team members

Since pairing facilitates knowledge sharing it can help with the onboarding of new team members.

Reflection

Pairing helps reflect more, not just on the code, but also on the user story, the value, potential solutions, etc.

Collective Code Ownership

Consistent pairing makes sure that every line of code was touched or seen by at least 2 people.

Continuous Integration

Waiting for a code reviews delay integration, which delays feedback/QA, increases conflicts, prohibits refactoring, increases risk.

Approaches

Before we get to the challenges that come in a package with all the benefits, let’s talk about a few common approaches to pair programming.

/thoughtworks

The Roles: Driver and Navigator

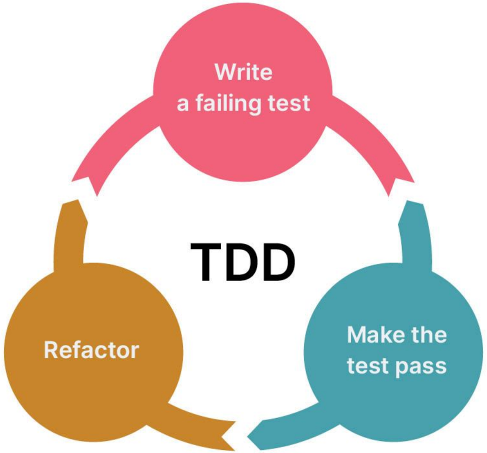
These classic pair programming role definitions are the basis for a lot of other approaches as well.

- Player / Coach
- Captain / First Mate

- Gain shared understanding of story, agree on high level plan and tasks
- Start with a reasonably well-defined task
- Agree on one tiny goal at a time - defined by a unit test, or defined by a commit message, or written on a sticky note, or…
- **Driver:**
  Has the *keyboard*; is focussed on completing the current tiny goal, ignoring larger issues for the moment; narrates what they are doing while doing it
- **Navigator:**
  Code reviewer; giving directions, sharing thoughts; has an eye on the *larger issues*, bugs, potential next steps > brings them up after the task is done

Ping Pong

This technique embraces **Test-Driven Development** and is perfect when you have a clearly defined task that can be implemented in a test-driven way.



- Ping: Developer A writes a failing test
- Pong: Developer B writes the implementation to make it pass.
- Developer B then starts the next Ping, i.e. the next failing test.
- Each Pong can also be followed by refactoring the code together, before you move on to the next failing test. This way you follow the "Red - Green - Refactor" approach: Write a failing test (red), make it pass (green) and refactor.

Pair Development

This approach is particularly great for **exploratory** tasks, when both pairing partners don’t have a good idea of possible solutions yet, or both are using the technology at hand for the first time.

- Get to a common understanding of your task.
- Define a list of questions that you need to answer in order to implement a solution.
- Split up to research and explore.
- Separately search the internet to answer a question
- Read up on a concept that is new to both of you
- Use your pairing partner as a sparring partner to discuss and share what you have found.
- Make sure you sit next to each other, even when working separately.

Pomodoro

The pomodoro technique is a **time management** method that can be applied to almost all of the pairing approaches. It can help you to stay focused and to remember to take breaks to avoid mental fatigue. It also facilitates to regularly switching keyboard control.

1. Decide on the task to be done.
2. Set a timer (traditionally to 25 minutes)
3. Work on the task.
4. End work when the timer rings and put a checkmark on a piece of paper
5. If you have fewer than four check marks, take a short break (3–5 minutes), then go to step 2.
6. After four pomodoros, take a longer break (15–30 minutes), reset your checkmark count to zero, then go to step 1.

“Strong Pairing”

This technique can be used for **knowledge transfer**.

- The rule: "For an idea to go from your head into the computer it MUST go through someone else's hands"
- In this style, the navigator is the person much more experienced with the setup or task at hand, while the driver is a novice (with the language, the tool, the codebase, ...)
- The experienced person stays almost always in the navigator role and guides the novice.
- While this technique borders on micro-management, it can be a useful onboarding tool to favor active “learning by doing” over passive “learning by watching”.
- This style is great for knowledge transfer, but shouldn’t be overused. Keep in mind that your goal should be to easily switch roles after some time.

More at:
https://llewellynfalco.blogspot.de/2014/06/llewellyns-strong-style-pairing.html

Housekeeping

Scheduling

- Consider regular, overlapping pairing hours for everyone
- Agree with your pair on how many hours you are going to pair at the start of the day.
- Check if you have meetings during the day and plan for that. If you have any meetings and you need to leave your pair, make sure things can continue without you.
- Remember to take breaks
- Give a heads up if you're going to be out the next day
- Solo work on Fridays?

Remote pairing

- Offer your remote partner the best **audio experience** you can manage: Look for a quiet area and use a good headset. "Push to speak" can also help.
- Make sure to **switch computers regularly**, so that each of you has a chance to work on their machine. It might be exhausting to work on a remote computer for the whole day, especially if there is a network lag.
- **Adjust font size & colors** to work for both people.
- **Avoid scrolling** in a long file, this can be very difficult to follow if there is network lag. Try to use short keys to open different parts of the file or use the collapse/uncollapse functionality instead.

Things to consider

- **Stay focused.** Avoid personal phone calls, texts, or unrelated emails while pairing. Take care of that before or after pairing. You may need to check Slack while pairing.
- **Watch out for micro-management** mode, because it doesn’t leave room for the other person to think and is a frustrating experience.
- **“5 second rule”** - when the navigator sees the driver do something, wait at least 5s before saying something - the driver might already have it in mind, then you are interrupting their flow
- **Communication** is important. Give/receive feedback. If something isn’t working for you, the other person won’t know unless you tell them. Be open to receiving feedback.
- **You do not have to be an expert in everything.** You will need to expose vulnerabilities, but will grow in the process. Have trust and empathy for your partner, who will do the same.
- **Success is not about one person.** You may be able to go faster on your own, but the whole team needs to come along and collaboration is important.
- **Don’t be afraid to type on the other person’s computer.** Also make use of drawing tools.

Pair Switching

- **Rotate** every x days (whatever works for the team)
- **Share context.** The anchor of the story should take time to explain the context, details from previous conversations, what has been done so far, what is left.
- **Give the new person time** to catch up. Let the new person take time to read the story, look at the existing code, ask questions, and share ideas before starting.
- **Don’t pair with the same person** too often. This doesn’t help share context. Address reasons for avoidance, be aware of preferential partners.
- **Speak up** if you have a preference for a story or person during pair switches. You may benefit from person with extra knowledge on a topic, or be interested to work on a certain topic/technology. Maybe you need to work alone that day.

Challenges

While pair programming has a lot of benefits, it’s also not easy. The following pages list some of the common challenges teams experience, and some suggestions how to cope with them.

/thoughtworks

Exhausting

- Take scheduled breaks
- Limit pairing to fixed, team-agreed hours, max. 6 hours per day
- Switch roles often

Intense interpersonal collaboration can be hard

- See this as an opportunity to improve your mentoring and coaching skills
- Schedule regular feedback sessions with your team members and give feedback on pairing

Challenging to schedule around meetings

- Define core pairing hours without meetings
- Check your calendars together at the beginning of your pairing session to make sure there's enough time to pair
- Rely on PO/PM, or non-pairing team members to run interference for the team

Differences between pairers

- Start your pairing session with the question: "How do we want to work together?"
- Be aware of how you like to work/are efficient (but also don't be closed off to other approaches)

Perceived slowdown vs. delivery pressure

- Make the benefits of pairing visible to your team and your stakeholders
- Understand more time upfront will result in higher quality and less defects/churn later

Pairing exposes vulnerabilities

- It can be hard to expose what you don't know. Showing vulnerability takes courage but leads to innovation and change.
- Have empathy for others. Build an environment of trust on the team.

No time for yourself

- Don't pair 8 hours a day, agree on pairing hours with your team
- When you feel like as a pair, you don't have the collective knowledge to approach a problem, split up to read up and share back, then continue implementation

Context switching with pair rotations

- Find a balance between rotation on stories and the possibility for a new pairing partner to get enough context on the story and contribute properly
- Anchor should share context with new pair

Let’s discuss!

To pair? ✓

Or not to pair? ⊗