

# **Strong Pair Programming**

@leedscodedojo

# Pair Programming Basics

*a.k.a... 2 devs 1 keyb*

# Pair Programming basics...

- Driver is at the keyboard, writing code
- Observer is sat next to the driver...
  - Reviews code typed in
  - Considers strategic direction
  - Suggests improvements / alternative approaches
  - Identifying potential problems
  - Act as a safety net
  - Two heads are better than one
- The driver and observer switch roles *frequently!*



# Pair Programming Benefits

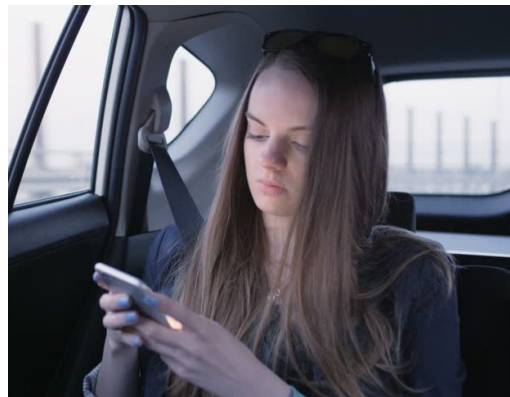
- Increased code quality
- Less time spent on code review stage
- Wider understanding of the solution in the code
- Less defects
- Helps to skill up junior devs
- Improves communication and team building

But...

- May take longer
- Harder when distributed
- Benefits are a bit subjective

# Pair Programming Anti Patterns

- One person coding, the other person just watching (or on their phone)
  - Observer is not engaged
  - Not reaping the benefit of pairing



# Pair Programming Anti Patterns

- Driver/Person at the keyboard is not communicating, just typing away
  - Solution is not shared with observer
  - No communication benefit
  - Dangerous when an expert is pairing with a junior dev

# Pair Programming Summary

Whoever has the  
keyboard has most  
control



# Strong Pair Programming

*"For an idea to go from your head into the computer ...  
it MUST go through someone else's hands"*

[Llewellyn Falco](#)





# Strong Pair Programming

- The Observer becomes the Navigator!
  - High level commands come from the navigator
- The Driver executes the instructions from the navigator
  - Low level commands being entered onto the keyboard by the driver

This style of pair programming is all about:

- Increasing communication
- Increasing collaboration and engagement



# Driving Tips 1: Trust your navigator

- Trust the navigator knows what they are telling you
- If you don't understand ***what*** they are telling you - ask!
- KEY BIT: If you don't understand ***why***, trust them, go with it. Have a chat about it once you have finished executing their current thought or idea on a particular section of code

## **Driving Tips 2: Get comfortable working without a complete understanding**

- You will learn as you go
- You might not know the language, OS, editor, code, or even the problem space you are working in....
- It's ok, you will soon!

# **Navigating Tip 1:** Give the next instruction to the driver the instant they are ready to implement it

- Manage a todo list of next things in your head
- Keep the driver in a state of flow
- Manage the big picture details so the driver can stay focused on the code they are typing

## Navigating Tip 2: Use as high a level of abstraction as possible

- But adjust your instructions to suit the driver..
- Higher level of abstractions free up the navigator to think of the next step without being bogged down by syntax/implementation

Lower level	Higher Level
Highlight code, refactor -> extract method, call it "getX()"	Extract that code to a method
Type: <code>newList = list.sort();</code>	Sort the list alphabetically
Type: <code>"public void testNullInput() ..."</code>	Create a test to make sure it works with null input

**Driver: “What if I have an idea I want to implement?”**

Great!

Switch places and become the Navigator.

# What's the strongest material known to man?

1. ~~Graphene~~
2. ~~Buckypaper~~
3. ~~Metallic glass~~
4. ~~Dyneema~~
5. ~~Lonsdaleite~~
6. ~~Wurtzite Boron Nitride~~
7. DIAMONDS!!!

# The Diamond Kata...

Simply print an ASCII diamond defined by the size passed in....

```
printDiamond(3)
```

```
  *  
 * *  
*   *  
 * *  
  *
```

```
printDiamond(4)
```

```
  *  
 * *  
*   *  
 *   *  
*     *  
 *     *  
  *   *  
   *
```



# The Diamond Kata...

Create a diamond made up of letters of the alphabet...

```
printDiamond('E')
```

```
  A
 B B
C   C
D     D
E       E
D     D
C   C
 B B
  A
```

```
printDiamond('C')
```

```
  A
 B B
C   C
 B B
  A
```

**Breaking down the problem: start small, iterate, gain confidence.**

```
printDiamond('C')
```

A  
B  
C

AA  
BB  
CC

A  
BB  
CC

A  
BB  
CC

A  
B B  
C C

A  
B B  
C C  
A  
B B  
C C

A  
B B  
C C  
C C  
B B  
A

A  
B B  
C C  
B B  
A

**All ideas presented here are from Llewellyn Falco's blog...**

<https://llewellynfalco.blogspot.com/2014/06/llewellyns-strong-style-pairing.html>

[@LlewellynFalco](#)

[llewellynfalco.blogspot.com](http://llewellynfalco.blogspot.com)