Pair Programming
&
Mob Programming

An Introduction

Lean Bytes

The classic way: the lone coder

- Coders code on their own, in the zone, a code warrior on the edge of time.
- Other coders are not aware of the code they are writing.
- If the coder gets stuck then they may spend a long time trying to sort out an issue; they may be embarrassed to ask for help.
- It therefore hard to maintain and measure code quality, which can lead to a requirement of code reviews (or ought to).
- It can cause a lack of team ownership of code, leading to “well, I didn’t write this, so-and-so did”.
- Knowledge silos are created when only one developer knows the code. If a person leaves, they take their knowledge silo with them.
- It makes it hard for newcomers to the team to learn.

Pair Programming

- From XP (eXtreme Programming).
- Part of the technical implementation of Agile.
- Two-developers work on the task together.
- One of the pair can break off to do something else then come back later; This is especially useful during analysis tasks.

Ways to do it physically

- Co-located teams, sharing a keyboard, or with two keyboards.
- Remotely using Skype sharing a screen.
- Remotely using TeamViewer, or similar VNC product.
- Co-located teams work best in my experience.
- But pair programming helps keep remote workers from loosing focus or slacking off!

Ways to do it

- People take it in turn to *drive*.
- The *driver* has the keyboard, the *passenger* sits on their hands!
- Swap every 10 to 20 minutes (15 is good).
- If one person is less experienced, then it may be good to let them drive more; may help enhance the *knowledge transfer*.
- For longer tasks, one person in each pair can *swap-off* the task at the next day and another can come on to it. The next day, the other swaps-off.

Results – Benefits

- Knowledge transfer between developers is continuous. Makes it easier to get newbies up to speed.
- Greater sense of team-ownership of code.
- Code quality most often improves; bugs go down.
- Knowledge is not tied up in the mind of a single coder.
- Problems are solved faster and more efficiently due to constant exchange of ideas.
- Swapping-off and onto tasks increases spread of knowledge in team.
- It feels good; there's more human interaction, and people often feel more confident about the solutions they create.

Results - Downsides

- It can be hard to justify to management − as it seems like using twice as many resources to achieve the same goal.
- There's not a lot of evidence out there to support it; no pretty graphs to show management.
- If your test suite takes ages to run, then it can seem like quite a waste of time to do *pair test runner watching*.
- However, you can measure in your team using cycle time, burn down rate, and number of bugs from released code.
- I have experienced some developers who found it tiring to pair all the time; encourage some breaking off, or more swapping.
- I have also experienced a pair who conspired to do ill as neither were convinced with Agile principles. Once noted we could ensure they rarely paired together and frequently swapped. Also see Mob Programming….

Mob Programming

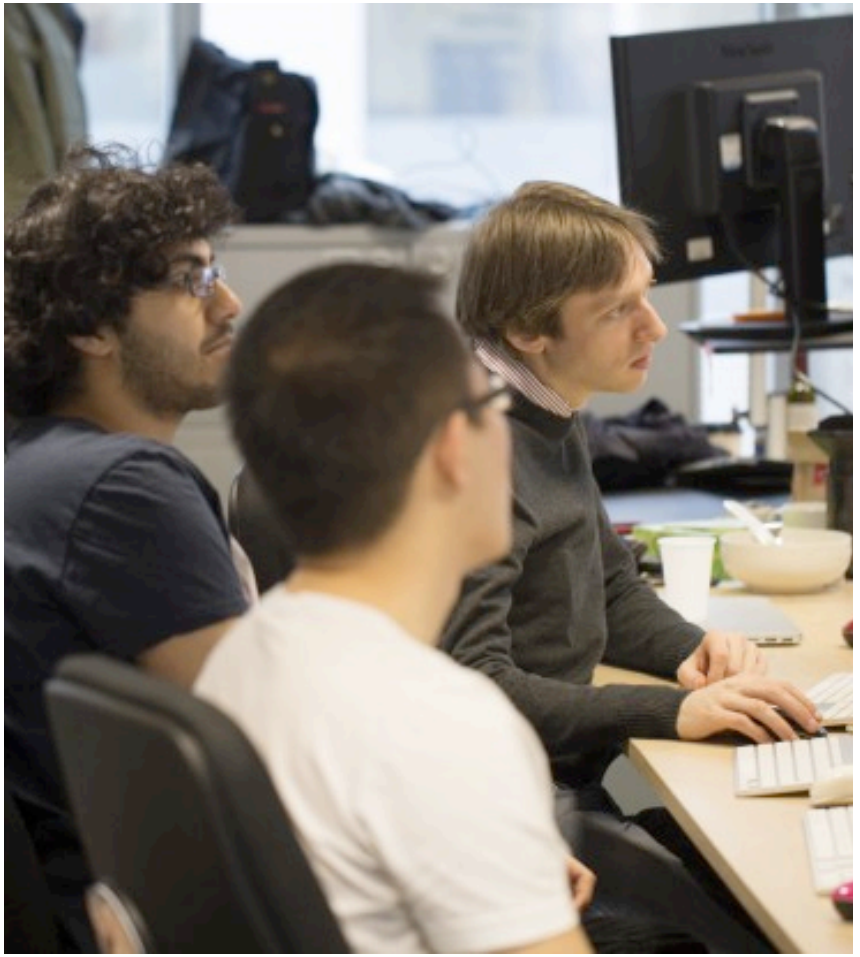
A Whole Team Approach

Illustration © 2012 - Andrea Zuill

mobprogramming.org **Twitter: @WoodyZuill**

Mob Programming

- Defined by Moses Hohman and Andrew Slocum in "*Extreme Programming Perspectives" (2003)*
- Recently popularised by Woody Zuill in *"Mob Programming: A Whole Team Approach" (2014).*
- Everyone shares a single keyboard!
- Developers take it in turns to *drive*.
- Variation: the driver can't contribute
- Co-location is pretty much a must.
- Developers (and POs, etc) are able to drop in and out freely.



Let’s watch a video from Woody Zuill

Results

- Total team-ownership of code. Ability to say "*we did this, we decided that*" with conviction.
- Enhanced code quality, far, far fewer bugs.
- Knowledge transfer to all team members (to at least some degree).
- Can help identify knowledge silos.
- In my experience it doesn't actually slow a team down, remarkably. Any slow down can be justified by resulting code quality.
- Eliminates conspiracies-to-do-bad; well, if the whole team agrees to the conspiracy, then you're pretty much sunk!

Mob programming - downsides

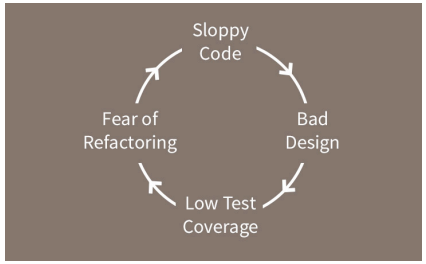
- Whatever you do, don't tell the FD! (at least not until you have proof it works).
- Remote workers are hard to integrate.
- Not that good for investigation and analysis.
- Not that good for devops/sysadmin tasks; pairing works better for this.
- Will most likely initially affect your throughput as it reduces your team's WIP limit to one.

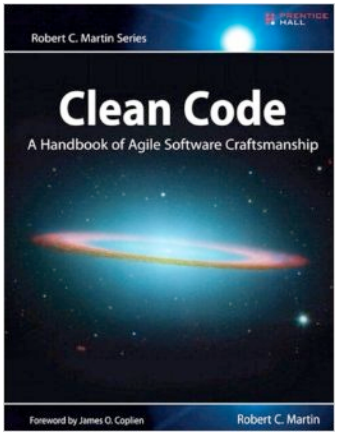
And not just programming…

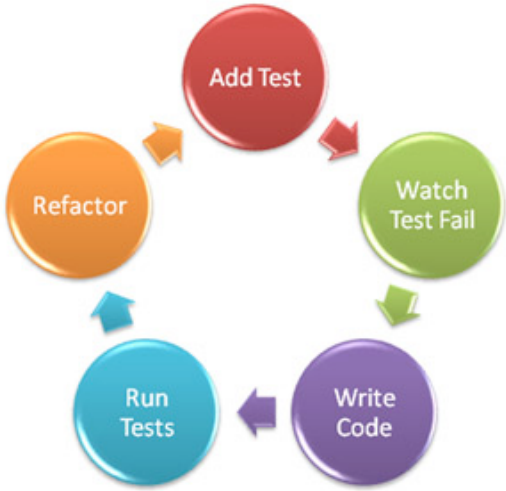
- Analysis
- Backlog grooming
- Testing
- Systems administration / DevOps
- Reporting
- Digging holes in the road (mob digging has been used for this for many years already).

And remember to also…

- Keep the stories short.
- Focus on the what, why and for whom, not the how.
- Do test-driven development.
- Write CLEAN code.
- Use design patterns.
- Refactor often.
- Nurture respect amongst your team members.
- Sit on your hands if you can't keep them off the keyboard.







Thank you.

07811 671 893

mike.harris@leanbytes.co.uk
http://leanbytes.co.uk
https://mbharris.co.uk
http://uk.linkedin.com/in/mbharris
https://github.com/mikebharris/

Lean Bytes