Pair Programming

Pair Programming

- Every line of production code is written by two people working together at the same keyboard
  - No boss; at any time the co-pilot or navigator can take over for the pilot or driver
  - Should switch roles frequently, every 20-30 minutes as a rough guideline
- Driver
  - Responsible for the code the pair is working on; e.g. types the code
- Navigator
  - Watches what the driver is doing and makes corrections and suggestions
  - Can take over when the driver gets stuck or wants to contribute

Programming

- Communication between the driver and navigator as coding and testing takes place
  - Explain intent of code
  - Driver can concentrate on algorithm while the copilot concentrates on syntax, variable names, types
- Co-pilot is not a passenger
  - Can take over
  - Is the code going in the right direction?
  - Is there another simpler approach?
  - What is the next step?
  - Was anything missed?
  - Where is that variable declared?
  - Is the array big enough to hold the data?

Protocol

- In XP each programmer signs up for his or her tasks for the week and pick pairs (ask someone to pair up) for that week
  - You can use two weeks if you wish for this project depending on how often you meet
- XP Rule: **If someone asks you to pair with him or her and you can, then you must say yes**

Isn't this inefficient?

- We have to do X and Y. Julio can write X in 3 hours and Patrice can write Y in 3 hours.
  - If we work in parallel then we can finish in 3 hours.
  - If we work in pairs then we finish in 6 hours.
- Sometimes this is true, but mainly only when typing is the most time consuming part of programming… there are other benefits to pair programming:

Pair Programming Myth

- Working in pairs is usually more productive than working alone
- While pairing with Patrice, Julio learned how to do task Y which ends up being useful when pairing with Paco on task Z
  - Paco ends up learning bits of Y as well
  - Work goes quickly on Z
- Research: Two Heads are 1.7 times better than One
  - Better design, fewer defects, reduced dependency on individual team members, better tech skills, better team communications, more enjoyment

Benefits of Pair Programming

- Real-Time Code Reviews
  - Catching syntax errors (minor gain)
  - Catching semantic errors as they occur (major gain)
    - Consider how these would otherwise be caught
  - Avoid "cognitive dropouts"
    - Forgetting a whole class, boundary condition, case
  - Catch invalid assumptions
    - Oh, I can't do it that way?

Benefits of Pair Programming

- Avoiding Distractions
  - Forces you to be prepared and not waste time or you'll be wasting your partner's time and nobody will want to pair with you
- Team spirit
- Managing for Two
  - High confidence if both agree on approach
  - If disagreement, forces discussion
    - If dissent, stop briefly, discuss, move forward in small steps; can always refactor later

Reminder: XP Principles

- Keep these in mind while pair programming
  - Simple Design
  - Testing First
  - Refactoring
  - Collective Code Ownership
- By maintaining simple design and small steps, dissent should not be too severe
- If you still can't decide, just pick one randomly; could be equally valid approaches

Longer-Term Benefit to Pairing

- Knowledge and Information Migration
  - Other programming pick up your good programming habits
  - Sharing of techniques
  - Compared to memetic evolution; best techniques survive, others die off
  - More people know how the same code works; avoids the closeted specialist and everyone becoming dependent on the specialist

Practical Issues

- Pairing is scary
  - Takes some time to become comfortable with pair programming
  - But quite a bit different from a traditional code review
- Third Alternative
  - When an issue requires discussion, stop typing and both developers work on a way to overcome the obstacle via design, thinking
- The Pair
  - Think of a couple – each pair has an identity
  - It will be similar with your pairs, roles will change depending upon the pair

Practical Issues

- Groups of Pairs
  - If multiple pairs, ideally working in the same room
- Odd Man Out
  - Use a rotating trio; two pair up while one does something independently
    - Catch up on homework, design, take a break, etc.
    - Third member could listen in and contribute
    - Should be fresh and available to rotate in

Practical Issues

- Reading Each Other's Signals
  - Be aware of each other's needs, e.g. when the copilot wants to take over. The less conscious the transitions, the less impact on a pair's production
- Giving up the "Wheel"
  - As the driver, listen to your copilot
  - Can ask the copilot to wait if you're on a roll, but the copilot probably has a good reason to want to take over
  - If copilot doesn't ask to drive you might need to initiate the switch

Practical Issues

- Font Size
  - If a member prefers a larger size, use it
- Code Formatting
  - Be consistent in some code format
  - Team must agree on a style, stick to it, and forget about it
- Revision Control
  - Use it
  - Whoever checks the code in last must resolve any conflicts between the two versions

Practical Issues

- Mixing and Matching
  - There will be people you prefer to pair with, which is fine, but don't let it become an exclusive subgroup.
  - Mixing up pairs lets ideas flow through the groups in pairwise fashions
  - Frequent common pairing is OK, but don't stop pairing with other team members
- Scheduling
  - Find a weekly time to pair and treat it with the same time commitment as a class
  - Reduces conflicts, doesn't require recurring effort to schedule, makes estimation more accurate

Does all production code need to be written in pairs?

- Ideally, yes
  - Possible in a full-time work environment
- Not as feasible in a classroom environment
  - Different schedules may make it hard to meet consistently
  - In this class you should strive for pair programming
    - If not feasible to complete the project, it is permissible to write code individually BUT this code must be reviewed by another team member before being checked in and accepted as production code

Final Word

- Pairing is like skiing
  - You can read about it, but until you get out on the slopes, you don't really have a feel for it.

Pairing Exercise (if time)

- On paper, write a test for a piece of code that is supposed to find the largest element in an array, using the pair programming techniques we just described.
- Write code to make the test pass.