BLUFFER'S GUIDE TO PAIR PROGRAMMING PEDAGOGY

There is far more to Pair Programming than simply asking your students to work in pairs. In this helpful guide, Alan O'Donohoe, computing teacher, leader of exa.foundation and an advocate of Pair Programming, wants to convince you to try it in your own teaching, so he has provided you with a range of strategies and advice you can use to great effect in your own classroom to unleash the great learning potential within this approach

Why should I consider using Pair Programming?

From 2010 to 2015, I embarked upon my conversion from ICT teacher to Teacher of Computing. During this transition period, I experimented with a range of teaching and learning strategies, including techniques like rubber duck debugging, Sabotage, and pair programming. I developed a handful of teaching strategies that I believe significantly improved the manner in which I had become accustomed to teaching. These strategies didn't just guarantee more successful outcomes for my students, I felt they also liberated me from the most onerous and stressful activities, like debugging students' code.

This meant I could more objectively evaluate the learning taking place and consider how I might plan the most effective teacher interventions. Of all the strategies that I tried, I found that pair programming had the largest impact on supporting pupils learning how to programme, design algorithms, and think like a computer. Many other teachers have also successfully converted to pair programming pedagogy.

Richard, teacher: "I tried it yesterday... Wow! What a difference it made to a difficult class. With a couple of notable exceptions, the class were engaged, discussing, and most of all programming, differentiated problems according to the ability of the higher level student."

What about traditional teaching methods?

When I first started teaching in the early 1990s, I adopted a teaching paradigm that firmly placed me as the 'expert in the classroom'. When I started teaching Computing I soon realised that this was unsustainable. My GCSE classes were asking me questions that I had not yet learned the answers to myself, and they increasingly relied on me to debug their coded solutions for them. At first, it inflated my ego and self-confidence to know that my classes were heavily dependent upon my knowledge and experience, this resulted in a constantly high demand on me to support them – but soon I found this tiring and stressful. There were some students I could never remain a few steps ahead of.

Through pair programming, my students developed greater independence – relying less on support from their teacher, they collaborated more among themselves using each other as a resource, and their levels of engagement were far higher than

Pair programming has a large impact on supporting pupils learning how to programme

PAIR PROGRAMMING CHALLENGES:

1. Create a program that will ask the user to enter two separate numbers, then print the output like this, 'You entered two numbers n1 and n2.', where n1 is the first number, n2 is the second number.

2. Create a program that will ask the user to enter two numbers, then add them together and print the output like this, 'You entered two numbers n1 and n2. The sum of these two is n3', where n3 is the sum of both.

3. Create a program that will ask the user for two numbers, then multiply them together and print the output like this, 'The product of n1 and n2 is n3'.

4. Create a program that will ask the user for two numbers, then divide the largest number by the smallest and print the output like this, 'When n1 is divided by n2, the answer is n3' where n1 is the largest number, n2 is the smaller number and n3 is the answer.

5. Create a program that will ask the user to enter two numbers, then ask the user if they are to be added, multiplied, subtracted, or divided. It will then perform the calculation for the user as above. Add some validation to your coded solution so that your program will only accept integers from the user input.

any other previous strategy I'd tried. I discovered there were also significantly fewer bugs to solve, partly because there were always two pairs of eyes on the coded solution, so they stopped asking me to debug their solutions.

It's worth stating that I didn't experience the positive benefits straight away – far from it. I needed to persist, observe, and reflect on what was happening, I had to relinquish some traditional control in order to allow my students to develop in confidence, and let go of some long held beliefs and habits of mine which had become embedded.

Won't the teacher become redundant?

Not at all, the role of the teacher develops into a far more useful one. I know that developing paired programming within my own pedagogy had a positive transformation on pupil learning outcomes. Previously, I fell into the trap of thinking that I needed to have mastered all of the Computing curriculum and the entire programming language before I could teach it effectively. I used to believe that the only path to my students' enlightenment was through me. However, in order for our students to grow and develop, then part of that process must require the teacher to create conditions in which students can flourish without direct intervention from the teacher.

What is the best way to try pair programming?

I'd recommend you start as soon as possible, which will allow you to develop the strategy in your own class. In terms of ability, pair like with like so that the stronger, more confident students are paired together and the weaker students are also paired. Explain that each pair will only develop the code on one computer throughout. One partner drives (with the keyboard and mouse) and the other navigates (vocal instruction and feedback). Then every five minutes or so, tell them to swap roles. Once you've started this with your classes, you'll spot ways that you can improve and develop the use of the strategy. You'll be surprised how few demands a class make on you once pair programming has become established with a teaching group.

In the rest of this article I describe some of the 'pair programming' strategies that I used, which I sorely wished I'd discovered earlier on, plus we'll include some further reading. If you wish to harness the full potential of pair programming in your classroom, here are some guidelines to follow:

Clearly define and enforce roles (Driver & Navigator)

For paired programming to work effectively, it's important to clearly define the expectations of each role. For a fixed interval of time, the 'Driver' uses the keyboard and mouse, while being guided by their 'Navigator'. The navigator acts as both coach and mentor, guiding

Pair like with like for the best results

Ask students to swap roles during challenges

the driver along the right path. While the responsibilities of driving can be more stressful and intensive than navigation, partners swap roles every five minutes to share the burden and opportunities. Partners either swap seats every five minutes, or simply pass the controls over to the navigator.

Avoid misconceptions about pair programming

Some teachers remain to be convinced about the learning potential of pair programming. At first, it seems ludicrous requiring students to share computers when there are enough for every student to work on their own in isolation. Some teachers worry that their students might only produce half as much code. Well, if you're really lucky, your students might generate even less than half as much code! If you're not sure why this would be a good thing, read back over this paragraph again.

Share challenges and sub-challenges

Each lesson would start with one large, shared challenge for the group that I designed to address particular learning objectives, but decomposed into smaller sub-challenges. For example, if I identified that students needed to develop their experiences of file handling, I would design a task that required a solution that relied on file handling to some extent in the success criteria. Often the challenge would be an extension or development of a previous programming challenge.

Plan challenges with graduated difficulty

Since the students are likely to have different backgrounds, experiences, and levels of competency in programming, I chose to design challenges and sub-challenges that initially required all pupils to develop a solution to a relatively simple problem. The first sub-challenge may be an extension of a recently introduced concept. These sub-challenges would then increase in degree of challenge. Most pupils would be expected to develop a more challenging solution, and I would also describe a much more challenging extension that I expected only a small minority of the group could achieve within the available time.

Plan your seating carefully

The seating plans I used initially, paired pupils with each other according to their position in the register alphabetically. However, once I had some acquired some assessment and progress data, for example from the MCQs we were using, I was then able to pair students with peers where there was a small achievement differential. The first time

I tried this, I paired students from extreme ends of the achievement spectrum – but I found this to be counter-productive and led to other problems. It was only after trying this that I decided pairs matched by ability was a much more successful strategy.

Provide documentation and supporting resources

If there are particular resources that your class will find useful, share links and references to these so that the navigator can refer to them when appropriate. You might develop a resource bank to support all challenges that would include supporting documentation, links to similar solutions, and code snippets. If there were resourcing gaps, I often asked my students to search for resources and then add them to a shared document for everyone's benefit.

Teacher Interventions

During the pair programming sessions, I would typically visit each pair, speak to both partners, and initially praise them on something e.g. their use of comments, their efficient use of code. Then I would also suggest one thing I would like them to improve. This process allowed me to track the progress of different pairs, to spot any problems ahead, and to plan any whole class interventions.

Timed intervals

Typically students are instructed to work in each role for five minutes and then swap roles. I then planned whole class interventions every 15 minutes or so as required, to remind them about certain points, rules, to remind them about the characteristics of great navigators, or to share some solutions and approaches I had observed. The stopwatch feature on my phone allowed me to set an alarm at the end of each block. You could tell students to create a Scratch project that alternates between 'Driver (right) Navigator (left)' and 'Driver (left) Navigator (right)' every five minutes and plays a sound to remind partners to swap roles.

FURTHER READING:

- James Franklin's page on Pair Programming: **helloworld.cc/2D16pHH**
- CSTeaching Tips: **helloworld.cc/2DkuMgq**
- Laurie Williams' page on Pair Programming: **helloworld.cc/2D2a3jG**
- Pair Programming in a box: **helloworld.cc/2Fso8WO**
- Pair Programming Wiki: **helloworld.cc/2mjRyg9**
- Pair Programming video: **helloworld.cc/2D4qpZi**
- exa.foundation resources: **helloworld.cc/2DOHh3l**

Pairing students at opposite skill levels was not successful

> " Some teachers worry that their students might only produce half as much code. If you're really lucky your students might generate even less than half as much!

Show & tell

There is terrific learning potential in sharing coded solutions with groups. After 20 or 30 minutes, I instruct the current driver to stand up and sit down with another navigator. Then each visiting driver has three minutes to review the code in front of them, suggest what works well to that navigator and what they might consider to improve their work. Then, when the visitors return to their original partners, they have lots to tell each other about what they have seen and heard from the others. The next time you repeat this, make sure to send the other partner to visit to keep it equitable.

Espionage

When there was an uneven number of students in the class, I introduced espionage. I identified a sensible student and asked them to choose any pair they would prefer to work with. Then, in addition to driver and navigator, I gave this group another role – 'spy'. When it was their turn to be spy, they had to wander the room stealthily looking for ideas to steal from other pairs, and then report back to their partners what they found while spying on other pairs.

Encourage and celebrate effective navigator behaviours

It's unlikely that you'll need to remind drivers what they are supposed to be doing, however being an effective navigator requires great skill. During my intervention I focus on positive behaviours I see navigators display. We had a poster, that the class helped me create, that lists the key behaviours of an effective navigator that included: be positive and encouraging; use clear, helpful language; look ahead to spot hazards and obstacles; look for ideas elsewhere; make suggestions... (HW)