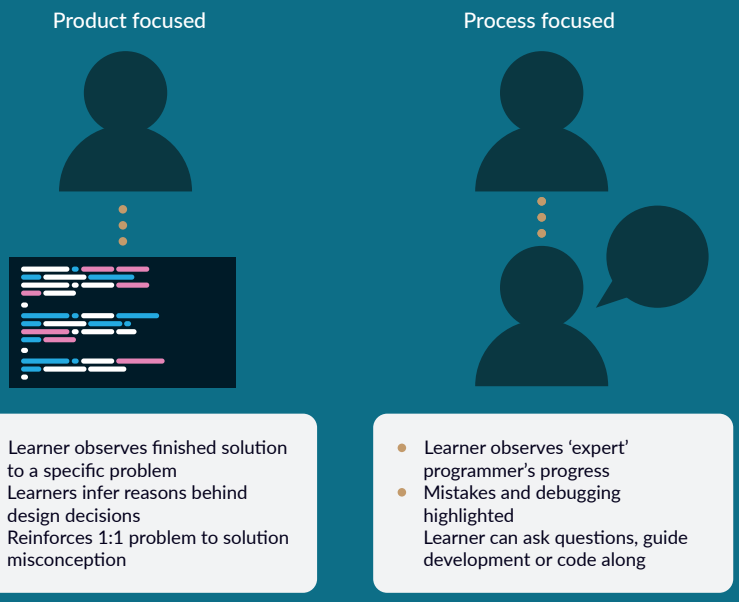
National Centre for Computing Education

Raspberry Pi

Pedagogy Quick Reads

Live coding: using the thought processes of a programmer to bring coding to life

When learners read static, completed programs, they aren't exposed to the troubleshooting that has already taken place to get to that end product; this is known as being **product focused**. Live coding is when a teacher develops the solution to a problem in front of the class for learners to follow, which is known as being **process focused**.



Bringing programming to life

Novice programmers can often look at a finished program and have the misconception that it has been written from top to bottom and that a skilled programmer always knows exactly what they are doing and can just write out what they need without making any mistakes. As any programmer or even a writer knows, this is not the case.

Live coding demonstrates to learners the incremental nature of programming. It shows that problems are decomposed into small sections that are programmed, tested, and debugged, before the next stage is worked upon. It models good programming practice and shows learners that a plan for a program is formulated and followed, rather than a solution formed on an ad hoc basis.

Bringing programming to life is essential to show learners that program development is non-linear. The code moves around and changes as a solution is developed. It models how programs should be frequently tested to debug them quickly. It also shows learners how to solve common errors that may occur when using a new concept.

Key benefits

According to the literature,[1, 4] the key benefits of live coding are that it:

- Reduces cognitive load, through collaboration
- Makes the process of learning programming easier to understand for novices
- Helps learners understand the process of debugging
- Exposes learners to good programming practices

Good practice when live coding:

- Select an appropriate programming challenge to teach a new concept, consolidate learning, or address misconceptions
- Talk to your learners and ask them questions
- Narrate your inner monologue
- Make (and fix) mistakes, either planned or accidental
- Slow down to give your learners time to process
- Show learners that code isn't written from top to bottom in a linear form; it moves around as it is developed
- Be visible: let learners see your face, don't turn your back for too long
- Pause to write things on the board: draw diagrams, work things out
- Use the largest font possible (without losing view of the full line of code)
- Break the code into small chunks (decompose) and use subgoal labelling while forming the solution

Strong links with worked examples:

Live coding helps novices learn by observing an expert programmer working through a problem, and so it has strong links to the concept of worked examples. Further information can be found on our Quick Read (**ncce.io/qr02**).

Cognitive apprenticeships

The idea of cognitive apprenticeships was introduced by Collins et al.[2] in 1987. They believed that "teaching methods should be designed to give students the opportunity to observe, engage in, and invent or discover expert strategies in context".

At the modelling stage of cognitive apprenticeships, an expert shows learners how to carry out a task, which "requires the externalization of usually internal (cognitive) processes and activities".[2] In live coding, an educator develops a program in front of a class while highlighting their choices, decisions, mistakes, and debugging strategies.

Cognitive apprenticeships (cont.)

Coaching (an aspect of cognitive apprenticeships) is where learners are given a challenge that is slightly too much for them to handle but are supported through the solution through feedback and modelling. Live coding is a great example of a coaching strategy, guiding learners through a task that would usually be unattainable.

Slowing down to get the best results

Live coding is very different to reading solutions on a worksheet or in a textbook. Those examples show a final, polished solution without any insight into how the programmer has made decisions about their code. The *Role of Live-coding* [1] paper states that "when students begin to learn programming, usually they don't have a good idea about where to start".

If you write your solution in front of learners it forces you to slow down, which helps you think about what you are doing and enables learners to follow your process. It is important that you don't simply copy and paste the solution from one tab into a new window; this defeats the purpose and your learners may get lost very quickly. You could write some notes about how you solved the problem and keep these on your desk as a prompt.

Learners benefit from following the process of your work, as it keeps them engaged in finding the solution. This is another reason why slowing down is important. You can chunk the demonstration and have sections where learners watch and sections where they code. It is important that they don't miss key things while they are typing, so monitor their progress as you carry out your session.

You can also provide video recordings of your sessions to help learners who may need a recap or learn at a different pace. If you decide to record your live coding session, make sure you stick to the live coding principles and don't create a step-by-step tutorial instead.

Predicting testing and debugging

When carrying out a live coding session, it is important that it doesn't become a tutorial that leads learners to the perfect solution on their first attempt. The learners are part of the journey. The best way to engage them is to ask them to make predictions about the program before it is run.

Wilson's *Teaching Tech Together* [3] emphasises the importance of making mistakes while live coding. Mistakes should be "embraced" because they allow learners to see that programmers don't get it right first time and often have to review and fix their work to find a solution.

When live coding, you should plan intentional mistakes but should also be confident when making unintentional mistakes. Intentional mistakes should link to common errors or learner misconceptions in order to target and alleviate them. You should also continually test your program. This helps learners see this as a natural way to program and teaches them to frequently test their own work.

When making intentional mistakes, encourage learners to predict what will happen, before running the code. Doing so will help learners suggest strategies to fix those errors. Miller et al.[5] discovered that "students who predict are significantly more likely to correctly report the outcome of a demonstration". Outcomes were improved whether their prediction was correct or incorrect. Therefore, asking prediction-focused questions while live coding is an important part of the process.

References

[1] Halverson, E., Halverson, R., Patel, J., Raj, A. (2018) Role of live-coding in learning introductory programming. *ACM*. 13, 1–8. Available from: https://doi.org/10.1145/3279720.3279725

[2] Brown, J.S., Collins, A., Newman, S. E. (1987) *Cognitive apprenticeship: teaching the craft of reading, writing and mathematics*. BBN Laboratories, Cambridge, MA., Centre for the Study of Reading, University of Illinois. Report number: 403.

[3] Wilson, G. (2009) *Teaching tech together: how to create and deliver lessons that work and build a teaching community around them*. Abingdon, Taylor & Francis. Available from: https://teachtogether.tech/#s:performance-live

[4] Sands, P. (2019) Addressing cognitive load in the computer science classroom. *ACM Inroads*. 10 (1), 45–51. Available from: https://dl.acm.org/doi/10.1145/3210577

[5] Chu, K., Lasry, N., Mazur, E., Miller, K. (2013) Role of physics lecture demonstrations in conceptual learning. *Physical review physics education research*. 9 (2), 1–5. Available from: https://journals.aps.org/prper/pdf/10.1103/PhysRevSTPER.9.020113