From invisible struggle to visible learning

When I joined The Hong Kong Polytechnic University in September 2023, I was assigned to teach **COMP1012: Programming Fundamentals and Applications**, a large undergraduate course enrolling roughly **250–350 Year 1–2 students**. This was my first substantial experience teaching such a large, highly diverse cohort in Hong Kong. The class included students from fashion, design, applied physics, linguistics, building sciences, engineering, data science, and applied biology, many with little or no prior coding experience; over ten students had **special educational needs**.

What struck me was not only the scale, but the vulnerability within it. Many students were not struggling for lack of ability or motivation; the language, assumptions, and examples through which computing is often taught did not align with their backgrounds. Some copied code without understanding it. Some were too anxious to ask "basic" questions. Some did not yet see why programming mattered to their discipline. Some believed computing was only for naturally technical students. In such a class, these students could become silent, invisible, and disengaged.

I found this deeply important at a human level. I did not want my teaching to reward only the already confident. I did not want students to leave as if they did not belong. I aimed to move them from uncertainty to confidence, from imitation to reasoning, and from passive task completion to meaningful creation[1]. This commitment became the foundation of my teaching philosophy and led me to develop a pedagogy built around five interconnected principles: **cognitive visibility**, **translational explanation**, **adaptive support with scaffolded challenge**, **output conversion**, and **iterative, evidence-informed refinement**.


Students' Course Project: Smart Elderly Home, Industrial Centre, PolyU


Students' Project Workshop: AIoT and Intelligence and Robotics (AIR), PolyU

I. Cognitive visibility: helping students see their own thinking

One of the first things I noticed was that many students focused on producing the final answer without understanding the reasoning that produced it. In programming, however, the reasoning process is often the real learning[2,3]. If students cannot see how logic unfolds, they cannot debug independently. If they cannot explain their own code, they may complete one task but struggle as soon as the problem changes.

I therefore began making student thinking visible. Before students wrote code, I asked them to trace loops and program logic line by line on paper. Before asking them to code a triangle pattern, for example, I first asked them to predict what each row should look like, what pattern repeated, and where a loop was needed. In labs, I asked students to explain code aloud in everyday language so I could check whether they understood the logic, not only the syntax. I

also used flowcharts and live coding with intentional errors so that debugging became a normal part of reasoning rather than a sign of failure[2].

One student, **CHAN Ho Tsun (from Year 2, Civil Engineering)**, reflected: **"The hands-on coding in tutorials really clicked for me; especially when you made us trace through Python loops line-by-line on paper before typing. That forced me to see where variables actually change, not just guess. Compared to other courses, you were low-pressure but firm on fundamentals; like making us explain our code out loud during labs."**

His words captured exactly what I hoped to achieve: less guessing, more seeing. Through this approach, hidden misconceptions became easier to identify and correct, and quieter students could participate because reasoning, not only final answers, was made visible and valued.

II. Translational explanation: making computing accessible across disciplines

In a classroom this diverse, I could not assume that all students shared the same technical vocabulary or examples. A concept that feels intuitive to an engineering student may feel distant to a student from fashion, design, linguistics, or applied biology. I therefore practised **translational explanation**: the computing concept remained rigorous, but the entry point changed[4,5].

When explaining a loop, for instance, I described it not only in abstract programming terms, but as repeating an action until a task is completed. For engineering students, I linked it to a sensor taking readings every second. For a design student, I compared it to repeated refinement in an iterative creative process. For linguistics students, I related it to repeated rule application in language patterns. For students from other non-technical backgrounds, I used familiar examples such as checking every item on a shopping list or repeating steps in a daily routine. The concept stayed the same, but the pathway into it became more accessible.

A student, **WU Yuxuan (from Year 1, Data Science)**, wrote: **"It is clear that the slides are well prepared, with not only clear information but also many details to enhance the learning experience, such as graphs for visualisation, analogies, and examples for illustration. Other than the knowledge you taught me, the most impressive thing about you is the attitude and passion you bring to me."**

This approach helped students from different backgrounds gain clearer access to computing concepts. Beginners became less intimidated by technical language, and students from non-computing disciplines began to see programming not as something alien, but as a tool that could serve their own fields. This same philosophy later informed my work on **Computerised Adaptive Testing (CAT)**, where explanations and question pathways could be adapted in more learner-responsive ways.

III. Adaptive support and scaffolded challenge: challenging students without overwhelming them

One of the hardest balances in teaching beginners is deciding how much challenge to give. If the course is too easy, students do not grow. If it becomes too difficult too early, some give up before realising they are capable[1]. In a large class, this challenge is magnified because students arrive with very different levels of preparation, confidence, and academic identity.

I addressed this through **frequent low-stakes quizzes, guided examples, structured labs, milestone-based assignments, and regular consultation opportunities**[6]. I also designed projects so that students could begin by modifying code frameworks before moving toward

more independent work. Instead of facing a blank page immediately, they first learned to understand, analyse, and improve key sections of code. I also worked deliberately to create a safe-to-fail environment through patient error analysis, anonymous feedback opportunities, and visible normalisation of struggle[7].

A student, **GUAN Fuyin (from Year 1, Applied Mathematics)**, reflected: **"The design of the course project demonstrates his accurate understanding of students' learning stages. Since we were still new to programming and lacked the ability to write complete programs independently, he provided the full main code framework in advance. Instead of demanding that we code from scratch, he only required us to understand, analyse, and modify key sections."**

This confirmed something important for me: scaffolding does not reduce rigor; it makes rigorous learning possible. Students were challenged without being overwhelmed. Beginners gained confidence, hesitant students became more willing to ask for help, and stronger students still had room to extend their work.

IV. Output conversion: helping students turn coursework into contribution

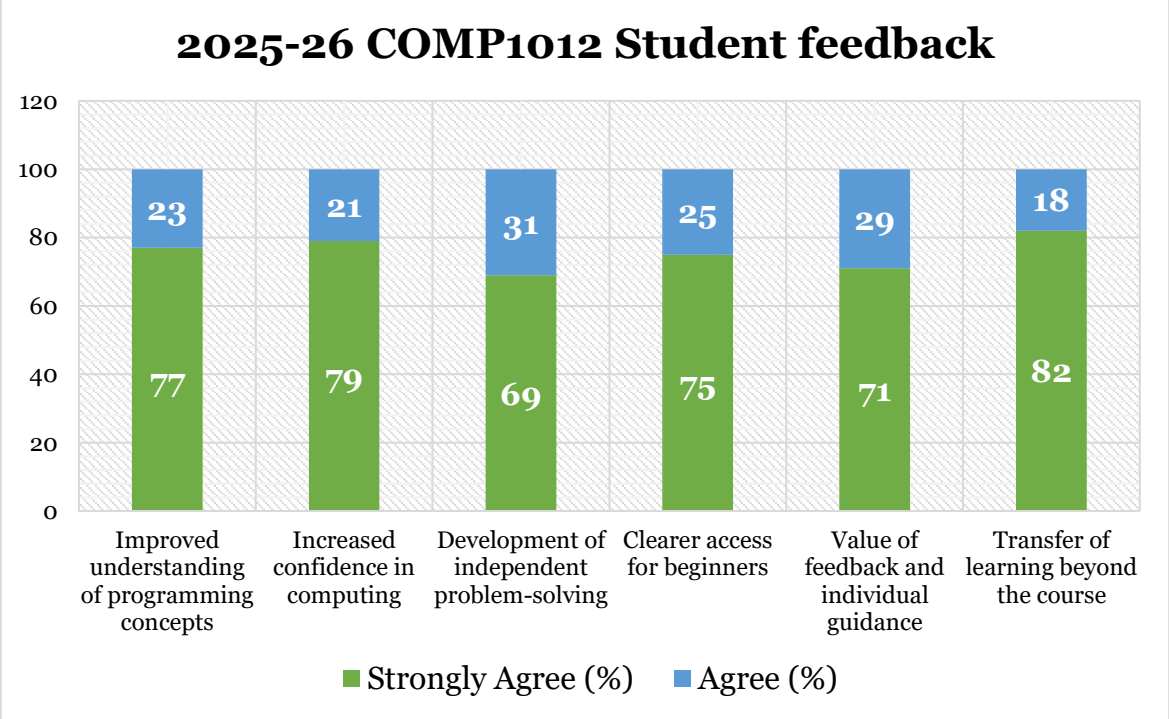

Figure 1. Perceived Impact: Agreement Levels on Course Outcomes

Over time, I became convinced that coursework should not end with grading. A classroom task can become a prototype, a conference paper, a competition entry, a patent idea, a startup concept, or the beginning of a research journey. Many undergraduates, especially in large foundational classes, have never imagined that their coursework could carry value beyond the classroom. I wanted them to experience computing not only as examinable knowledge, but as something generative[8,9].

I therefore began framing coursework as a starting point for real contribution. I encouraged students to choose projects connected to their own interests and disciplines, and I mentored them beyond the normal boundaries of the course. A student from building sciences could see

programming through smart monitoring and safety analytics; a student from design through interaction and creative systems; a physics student through sensing and modelling; and a biology student through data interpretation and applied analysis. For stronger projects, I helped students refine ideas, strengthen the technical work, and explore publication or competition pathways.

This led to outcomes that were meaningful not only in achievement terms, but in identity terms. One student project was developed into a paper accepted at **IEEE ICCSP 2026 in India**, where it won a **Best Paper Award**. Another project was submitted to the **CTM Big Data + AI Competition in Macau**, where the student team won **first prize** among around 800 teams.

A student, **RUAN Haozhe (from Year 2, Applied Physics)**, wrote: **"Another distinctive aspect of his teaching is that he encourages students to innovate and build projects based on their own interests, rather than strictly limiting us to a fixed project scheme. He also supports students in further developing strong projects, including helping them explore opportunities to publish their work in conferences or book chapters."**

For me, what mattered most was not only that student received recognition, but that they began to see themselves differently: not just as learners completing a requirement, but as contributors, researchers, innovators, and sometimes mentors.

V. Iterative evidence-informed refinement: improving teaching through student evidence

Because classes are large and diverse, I rely on more than intuition. I evaluate and refine my teaching through course performance data (Figure 1), eSFQ results, anonymous feedback, consultation patterns, student testimonials, and output-based evidence; project quality, publications, and competition achievements. I view this as quality assurance and part of my development as an educator.

When students were confused, I revisited it next class with clearer wording, simpler examples, or different analogies. When many made similar mistakes, I treated it as evidence my explanations or scaffolding needed improvement. This created a cycle of listening, redesigning, testing, and refining[10,11].

This process contributed to my professional growth. For example, 2024 feedback called for more hands-on, industry-relevant projects. I used this to redesign later opportunities. In 2025, I created industry-connected AIoT projects giving students practical programming with robotic systems, humanoid robots, and AIoT platforms. Student feedback helped me become a more responsive, reflective, and professionally mature educator.

A student, **YANG Kam Yung (from Year 2, Fashion and Textiles)**, wrote: "**What sets Professor Aquil apart is his passion for teaching and his relentless drive to improve his methods. He never simply repeats the same lectures term after term; instead, he actively seeks student feedback, experiments with new activities, and refines his explanations based on what works**."

That reflection means a great deal to me because it captures the kind of teacher I want to become: one who does not simply deliver content, but listens, learns, and grows in response to students.

Overall Teaching Philosophy

My teaching philosophy is **distinctive** because it begins with a human concern: in a large and diverse computing classroom, some students may silently decide they do not belong. My response is not to lower standards, but to redesign teaching so that reasoning is visible, concepts accessible, challenge sustainable, and learning generative.

It is **actualisable** because it is embodied in concrete, scalable practices: tracing code on paper, explaining logic aloud, drawing flowcharts, using intentional debugging, designing low-stakes quizzes, providing structured project frameworks, adapting examples across disciplines, offering consultation, and mentoring projects toward publications, competitions, innovation, and startup pathways.

Most importantly, **it is beneficial for student learning** by fostering stronger conceptual understanding, greater confidence, better independent problem-solving, and a clearer sense of how computing connects to their fields and futures. Some students move from anxiety to achievement. Some move from coursework to research. Some move from beginner to mentorship. In each case, the most meaningful outcome is transformation, not just performance.

At its heart, my pedagogy began with a concern that some students would become invisible. My work as a teacher has been to make learning more visible, humane, and possible. I want students to leave my courses not only with stronger technical skills, but with greater confidence, clearer intellectual agency, and a belief that they, too, can contribute. For me, the most meaningful measure of teaching is not simply that students complete a course, but that they begin to see themselves differently because of it; in this large and diverse foundational class, no student withdrew, and I regard that not merely as retention, but as evidence that students felt supported enough to stay, grow, and believe they belonged.

References

[1]Bandura, A. (1997). Self-efficacy: The exercise of control. W. H. Freeman
[2]Collins, A. (1987). Cognitive Apprenticeship: Teaching the Craft of Reading, Writing, and Mathematics. Technical Report No. 403.
[3]Venables, A., Tan, G., & Lister, R. (2009). A closer look at tracing, explaining and code writing skills in the novice programmer. Proceedings of the Fifth International Workshop on Computing Education Research Workshop, 117–128. https://doi.org/10.1145/1584322.1584336
[4]CAST. (2024). Universal Design for Learning Guidelines 3.0. CAST.
[5]Moreno, R., & Mayer, R. E. (2002). Verbal Redundancy in Multimedia Learning: When Reading Helps Listening. Journal of Educational Psychology, 94(1), 156–163. https://doi.org/10.1037/0022-0663.94.1.156

[6]Black, P., & Wiliam, D. (1998). Inside the Black Box: Raising Standards through Classroom Assessment. Phi Delta Kappan, 80(2), 139–148.
[7]Wood, D., Bruner, J. S., & Ross, G. (1976). The Role of Tutoring in Problem Solving. Journal of Child Psychology and Psychiatry, 17(2), 89–100. https://doi.org/10.1111/j.1469-7610.1976.tb00381.x
[8]Biggs, J. (1996). Enhancing Teaching through Constructive Alignment. Higher Education, 32(3), 347–364. https://doi.org/10.1007/bf00138871
[9]Miller, R. (2012). [Rev. of A Guide to Authentic E-learning - By Jan Herrington, Thomas C. Reeves, and Ron Oliver]. Teaching Theology & Religion, 15(2), 202–203. https://doi.org/10.1111/j.1467-9647.2012.00798.x
[10]Hattie, J., & Timperley, H. (2007). The Power of Feedback. Review of Educational Research, 77(1), 81–112. https://doi.org/10.3102/003465430298487
[11]Sadler, D. R. (1989). Formative assessment and the design of instructional systems. Instructional Science, 18(2), 119–144. https://doi.org/10.1007/BF00117714