SUTD Leap

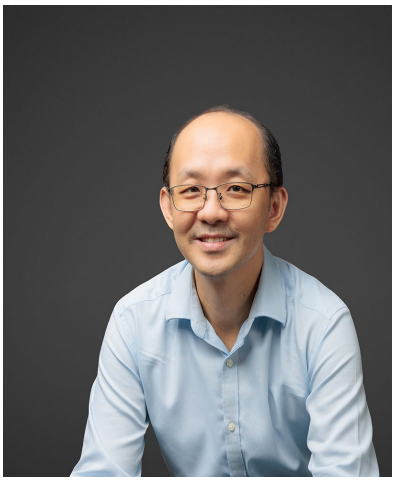
[RE]IMAGINE EDUCATION

A weekly Design.AI newsletter

BEYOND THE SHORTCUT: HOW CODEHINTER RESHAPES PROGRAMMING EDUCATION

Senior Lecturer Oka Kurniawan, DAI, ISTD

In Conversation with Dr. Nachamma Sockalingam, OSP



Dr. Oka Kurniawan
Senior Lecturer

https://www.sutd.edu.sg/profile/oka-kurniawan/

How can we truly help students strengthen their coding skills when AI tools like ChatGPT can generate answers in seconds? This week's feature explores this question with Dr. Oka Kurniawan, who shares insights from his work at SUTD, including the development of *CodeHinter*—an AI tool designed to guide, not shortcut, the learning process.

Oka makes a compelling case: effective learning depends not just on access to AI, but on when and how it's used. Just as we don't give calculators to students learning basic arithmetic, coding learners need space to build problem-solving fluency before relying on generative tools.

Through thoughtful analogies, tool comparisons, and data-driven insights, this piece challenges us to rethink AI's role—not as an answer machine, but as a scaffold for deeper learning.

Editor of Newsletter:
Dr. Nachamma Sockalingam, Office of Strategic Planning

Hi Oka. We know that you teach programming at SUTD. Tell us about the latest developments in teaching programming using GenAI such as ChatGPT? Would you recommend the use of these tools in learning programming?

Teaching programming is one of the most impacted discipline with the rise of AI The reason is the availability data, opensource codes and public GitHub repositories. OpenAI from its earliest models include Codex which is a model trained using data from GitHub public repositories. The latest OpenAI models perform very well in many areas related to coding from answering questions about a computer code, generating code, as well as fixing errors of a computer. However, the performance depends on the programming languages since all GenAI models depend on the data that is trained. Usually, it performs very well for Python and Java which are the two most popular programming languages. Even with this, GenAI still hallucinates and the more complex the problem is, the more GenAI performs poorly.

"Should we use these tools in learning programming?
The answer is it depends."

If it helps students to learn, then yes. Otherwise, we should not. This means that it depends on when and how we should use these tools. For example, should students use ChatGPT to generate code in their first-year programming course. My opinion is "No". The purpose of foundation programming course is to strengthen their problem solving plus programming skills and not just to produce a working code. So, when students simply use these AI tools, it may shortcut these learning moments and that is the reason that it should be avoided. On the other hand, students who are already proficient with their coding skills in the later years are encouraged to use these tools simply because it can truly help to increase their productivity.

Let us consider an analogy. In the case of Primary 1 students learning addition, subtraction or even multiplication, we should not be introducing calculators yet. The reason is that performing these operations manually repeatedly strengthens their arithmetic fluency and it has impact on their higher order thinking. Similarly with programming.

"Students must be proficient with their programming skills and problem solving first before resorting to these tools."

You have developed an AI tool called CodeHinter to support learners in programming. Tell us about that and how it compare to ChatGPT?

CodeHinter is an AI-based tool developed to build debugging skills. It is developed as a Visual Studio Code extension, an Integrated Development Environment (IDE) that is commonly used by programmers and those learning programming.

CodeHinter has a chat feature similar to ChatGPT but it has more. The main design of CodeHinter revolves around this one single button called “End-to-End Test”. We designed CodeHinter to follow common debugging framework.

Programmers can upload their codes into CodeHinter, and with a single button can launch a series of test for any syntax and logical errors. If there are syntax errors, CodeHinter will stop and help the learners to fix that error first. If there is no syntax error, the tool will continue to check if there is any logical errors.

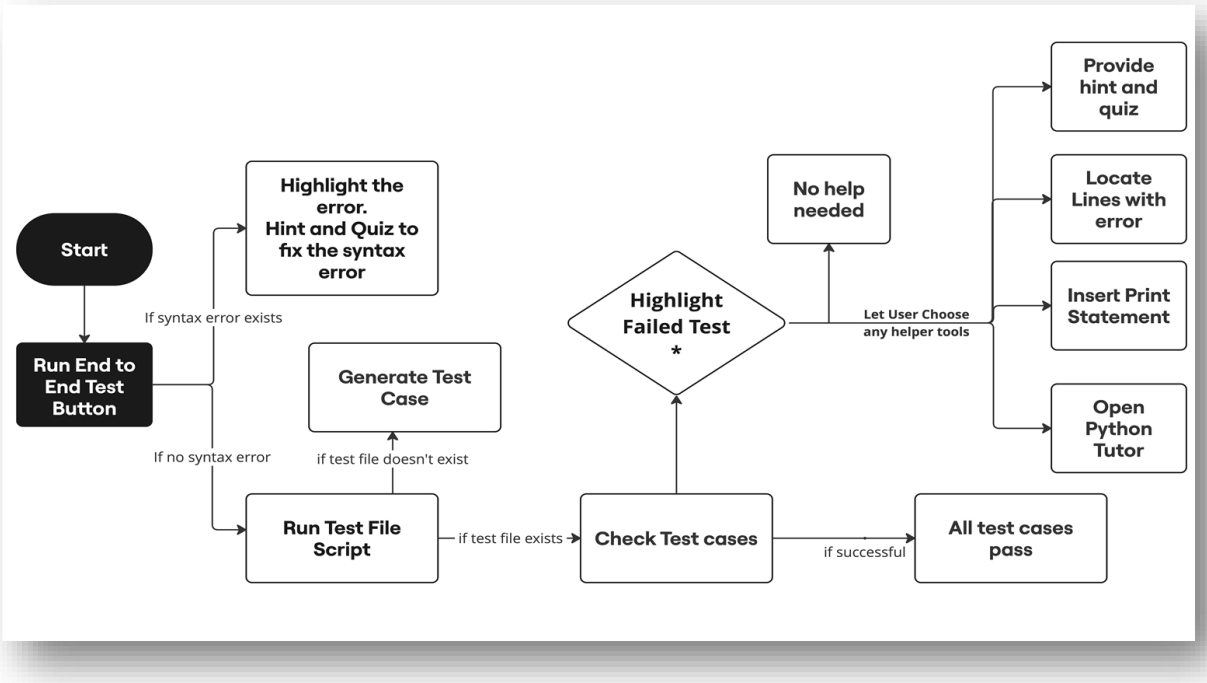


Figure 1: Flow of users interaction with CodeHinter. The process begins with users pressing the single ‘End-to-End Test’ button, which checks for syntax and semantic errors. Users are then guided to complete the missing debugging steps. If test cases fail, the tool provides options to help users resolve the bugs. The asterisk indicates the state of the screenshot in Figure 2.

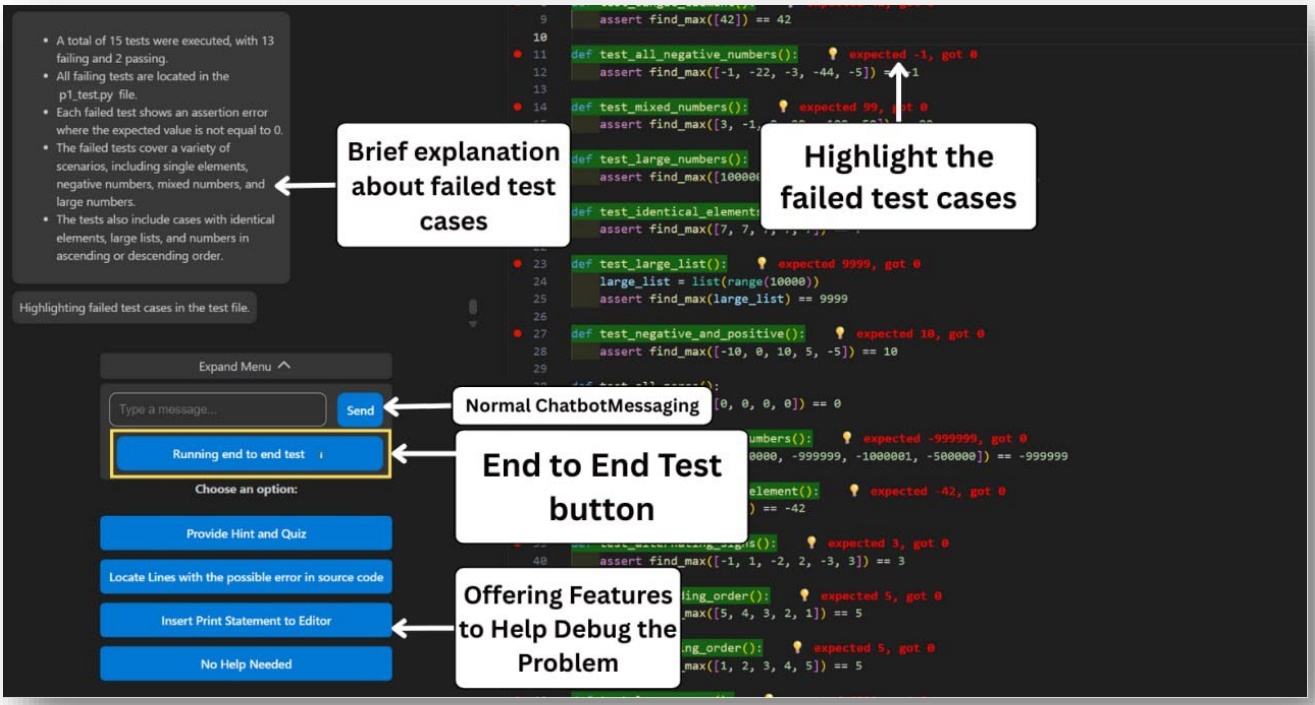
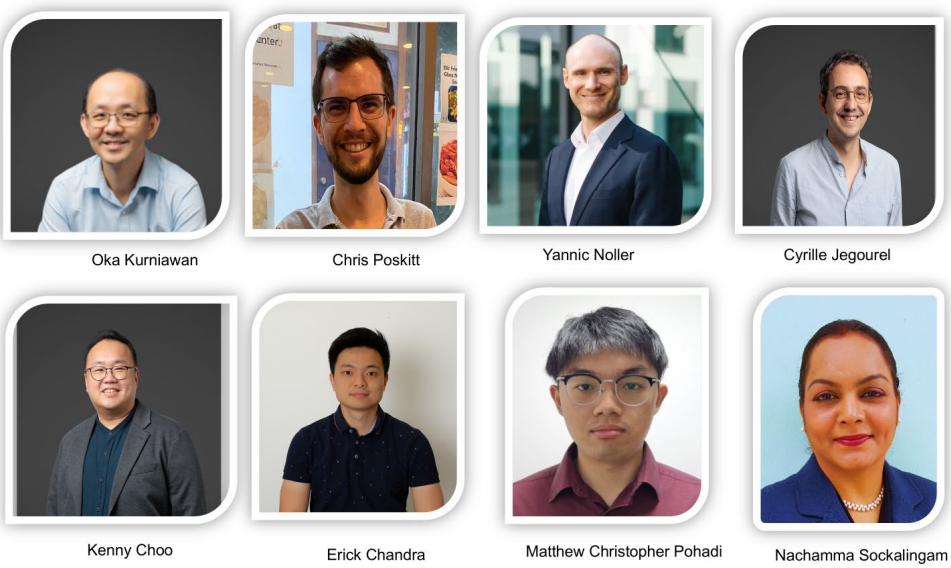


Figure 2. 'End-to-End Test' feature, the main feature of CodeHinter. The screenshot showed the state when users encounter failed test cases (Figure 1, marked by *), highlighting the expected value and actual output on the text editor while providing a brief explanation in the chatbot.

CodeHinter is developed by a team of SUTD and external faculty members with researchers, and is funded by Ministry of Education Tertiary Research Fund (MOE-TRF 2023)



CodeHinter Team

When CodeHinter detects any logical errors, it will prompt the users to check what should the next step be. In this case, CodeHinter gives options and freedom to users depending on their ability and learning objectives. Users can choose to ask CodeHinter to provide a hint by creating a quiz.

CodeHinter will then generate one correct solution to fix the error and some distractors as alternative options. In this way, students learn to think critically on how to fix the errors instead of simply blindly changing the code. Users can also choose to ask CodeHinter to locate the errors.

Research shows that locating errors are one of the biggest hurdles of debuggers. Once the errors are located, programmers usually are able to fix the bug. Users can also ask CodeHinter to insert “print” statements into the code to evaluate the values of the variables at that point. This feature is intended for novice programmers who may not be familiar with some of the debugger tools available in IDE. Inserting a print statement is one of the simplest way to debug their code. Lastly, users can simply choose to fix the code on their own without help from CodeHinter.

This way, we give users agency to choose the actions that best fit their situation and learning conditions when using CodeHinter.

Have you tested out CodeHinter against ChatGPT? What were your findings?

We are currently in the process of publishing our studies in comparing CodeHinter and ChatGPT. We studied the intrinsic motivation of users when using CodeHinter as when they use ChatGPT to debug some logical errors in an existing code.

We found that students that use CodeHinter have similar motivation level as when they use ChatGPT. However, the reason they choose CodeHinter is totally different. Users cited “efficiency” as the main reason to use ChatGPT. This is because they can get the solution faster by prompting ChatGPT. However, users cited “pedagogical support” and “fostering learning engagement” as their reason for using CodeHinter. In fact, the intrinsic motivation level of using CodeHinter is significantly higher under one category which is “perceived competence”.

This means that when they use CodeHinter, they are more motivated because they perceive they were able to fix the bug on their own. This is a positive result of using an AI-tool that is designed based on pedagogical perspective.

We have a few other interesting findings which we can share when the result is accepted and published.

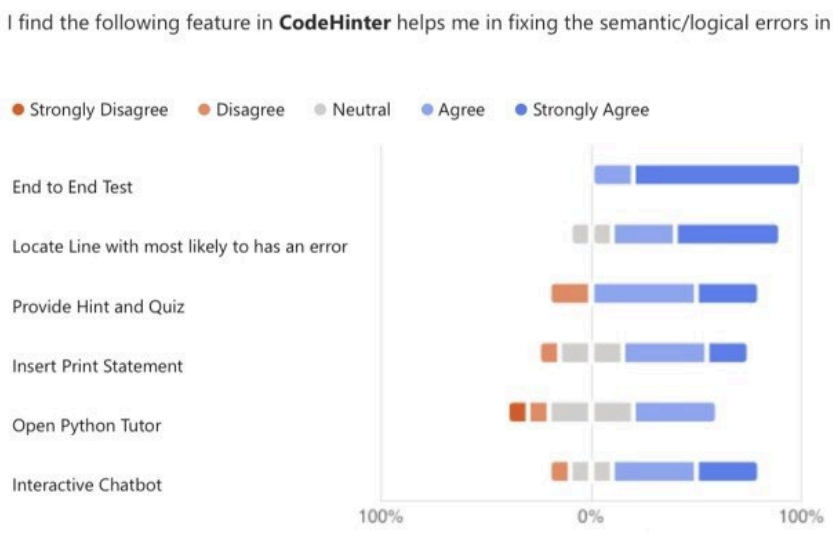


Figure 3. User perspectives on the usefulness of CodeHinter

What are your recommendations about teaching programming using ChatGPT or CodeHinter. How and when should students use these tools.

Firstly, we should identify the learning objectives. Based on this learning objectives, we should design our teaching strategies, which include what kind of tool is appropriate to achieve those learning objectives.

For first year programming where problem solving is best developed by writing their own code, we should teach students how to use the AI tools that does not shortcut their learning. Students who are mature will understand this and they will be willing to choose the best tools for their learning.

ChatGPT can still be used for first year programming for many other purposes, such as explaining a solution or instructor notes, generating questions to test their understanding, as well as become a Socratic partner where they can teach a particular concept to AI agent to help them learn.

What would be your next step in testing out or developing CodeHinter?

Currently, CodeHinter focuses on helping novice programmers fix logical errors which is currently not trivial for complex problem even for ChatGPT. However, this assumes that students have some working code at hand which they need to fix.

Our next step is to help novice programmers to scaffold their code writing from scratch given a problem description. We have a problem-solving framework called PCDIT which we use in class facilitated by instructors to help students lower their barrier in writing code from scratch.

The plan is to create a multi-agent AI system that can help novice programmers to work out the PCDIT framework to help them solve programming problems. In this way, we have a more holistic system from code creation to bug fixing for novice programmers.

Here are our publications that you can refer to.

1. Oka Kurniawan and Erick Chandra and Christopher M. Poskitt and Yannic Noller and Kenny Tsu Wei Choo and Cyrille Jegourel. 2025. Designing for Novice Debuggers: A Pilot Study on an AI-Assisted Debugging Tool. accepted in *Proceedings of the 24th Koli Calling International Conference on Computing Education Research.* https://arxiv.org/abs/2509.21067 .

2. Yannic Noller, Erick Chandra, Srinidhi Chandrashekar, Kenny Choo, Cyrille Jegourel, Oka Kurniawan, and Christopher M. Poskitt. 2025. Simulated Interactive Debugging. In *40th IEEE/ACM International Conference on Automated Software Engineering, ASE 2025*. https://arxiv.org/abs/2501.09694 .

In this week's spotlight, Dr. Oka Kurniawan shares how *CodeHinter*, a custom-built AI tool, is redefining how programming is taught at SUTD. Unlike generic tools like ChatGPT, CodeHinter is designed to promote deeper learning by guiding students through the debugging process rather than offering instant solutions. The discussion explores when AI should—and shouldn't—be used in coding education, why intentional tool design matters, and how scaffolding can turn AI from a shortcut into a powerful learning partner.