Article

Research and Practice of Cultivating Students' AI Literacy in the Software Engineering Course

Xiaoyan Hou[1,*], Tingting Zhang[1] and Xinwei Zhang[1]

[1] School of Computer and Artificial Intelligence, Guangdong University of Science and Technology, Dongguan, China

* Correspondence: Xiaoyan Hou, School of Computer and Artificial Intelligence, Guangdong University of Science and Technology, Dongguan, China

Abstract: AI tools have become deeply embedded in the software development process, and software engineering education is shifting from "teaching methods" to "teaching judgement". Taking the theoretical course Software Engineering as its basis and WorkBuddy as the AI assistant used uniformly across the course, this paper constructs a full-process cultivation path of "students draft first, AI reviews, students revise, teacher verifies" across six stages: requirements analysis, architectural design, database design, detailed design, coding implementation and system testing, and turns AI literacy into observable revision behaviour that is included in the assessment. Practice shows that this path improves students' ability to discriminate and correct AI output without increasing class hours.

Keywords: AI Literacy; Software Engineering Course; Teaching Reform; Human-AI Collaboration; Project-Driven Learning

1. Introduction

Generative artificial intelligence is spreading rapidly through the software development process. From requirement elicitation and system modelling to coding implementation and test-case writing, mature AI assistants are already available at every stage. The competence required of software engineering graduates has accordingly shifted from "can complete a development task independently" to "can work with AI and remain accountable for its output". This raises new demands for university software engineering education: students must not only master the methods of software development, but also be able to judge, correct and govern AI output - that is, to possess AI literacy [1].

The Software Engineering course systematically introduces structured and object-oriented development processes, and is taught mainly through methodology lectures and project modelling. Students' use-case diagrams, function module diagrams and E-R diagrams are usually marked once by the teacher, without iterative review; the common problems are that students can draw diagrams but fall short on standards, and that logical relations are misplaced. The introduction of AI tools makes a way out possible: at every modelling stage students can obtain immediate feedback, forming a loop of "students draft first, AI reviews, students revise". In teaching, however, it is common for students to adopt AI-generated diagrams directly without discrimination, and to lack the ability to correct the non-standard parts of AI output [2].

On this basis, this paper takes WorkBuddy as the AI assistant used uniformly in the course and, around six stages, namely requirements analysis, architectural design, database design, detailed design, coding implementation and system testing, constructs a full-process embedded path for cultivating AI literacy, so as to train the new type of software development talent required by enterprises.

Received: 19 September 2026
Revised: 07 November 2026
Accepted: 22 November 2026
Published: 26 November 2026

Copyright: © 2026 by the authors. Submitted for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

It focuses on two questions: how the connection points between AI tools and modelling tasks should be designed; and how students' AI literacy can be assessed through observable revision behaviour. Its significance lies in embedding AI literacy cultivation into an existing course rather than setting up a new one, thereby providing a reusable teaching design reference for similar theoretical courses.

2. The Connotation of AI Literacy and the Orientation of the Course

2.1. The Three-Layer Structure of AI Literacy

AI literacy is not limited to being able to use AI tools; it is a competence structure composed of three layers: operation, judgement and responsibility [3].

The operation layer refers to the ability to invoke tools to complete concrete tasks, including the ability to turn a vague requirement into an executable instruction; it has the lowest threshold and depreciates most easily as tools iterate. The judgement layer refers to the ability to evaluate AI output independently: to identify factual errors, non-standard points and design defects, and on that basis to decide whether to adopt, modify or reject; it relies on the user's own domain knowledge and cannot be supplied by the tool. The responsibility layer refers to the awareness of being accountable for the boundaries of use and for the final outcome, including academic integrity and the ability to justify every design decision [4].

Among the three, the judgement layer is the core: the operation layer is continuously lowered by vendors, the responsibility layer belongs to general academic norms, and only the judgement layer must rest on subject-specific knowledge, which is precisely the part that the Software Engineering course can and should undertake.

2.2. Course Orientation: Design Principles as the Yardstick for Judging AI

The core knowledge of Software Engineering - the completeness of requirements analysis, high cohesion and low coupling in module partitioning, the normal-form requirements of data modelling, the logical consistency of detailed design and the coverage of test cases - happens to constitute a ready-made set of evaluation criteria. Only when students master these principles do they have a basis for examining AI output; if the principles remain at the level of memorised concepts, students can only passively accept AI suggestions.

Accordingly, the role of AI tools in this course is not a shortcut to faster drawing and coding, but "a reviewer who is always present": the diagrams, code and test cases produced by the tool are material to be reviewed, and the students' task is to discriminate, revise and justify according to design principles. The cultivation goal is correspondingly translated into three observable behaviours: whether the student can point out where a suggestion violates a design principle, whether the student can explain the reason for adopting or rejecting it, and whether the student can keep the model internally consistent after revision.

3. Teaching Design for Embedding AI Tools into Each Stage of the Course

The course uses one continuous project as the carrier and embeds AI tools into six modelling stages. Every stage follows the same four steps - students draft first, AI reviews, students revise, teacher verifies. Students first work independently to form a baseline, then ask questions in the role of a reviewer to obtain suggestions, then judge and revise item by item while recording the reasons, and finally the teacher verifies the result [5].

3.1. Requirements Analysis Stage

Students independently draw a first draft of the use-case diagram from the project background, identifying actors, use cases and their relations. The draft is then submitted to AI for review, with prompts focusing on three kinds of questions: whether the business process is fully covered by use cases, whether the association between actors and use cases is reasonable, and whether the relations among use cases, especially include and extend, are expressed accurately. AI can often point out erroneous relations, but it also often

proposes extensions detached from business reality; students must compare the items one by one, mark them as adopted or rejected, and state the reason.

On this basis, non-functional requirements are supplemented in four dimensions: performance indicators, security and permissions, availability and maintainability, and data consistency. The literacy outcome of this stage lies in the awareness of requirements completeness and in the ability to discriminate against suggestions that look reasonable but do not fit the business; the criteria are whether the use-case diagram covers the main business paths and whether the non-functional requirements are measurable.

3.2. Architectural Design Stage

Students draw the function module diagram, dividing system levels and module responsibilities, and submit it to AI, asking it to evaluate the diagram item by item against high cohesion and low coupling. Typical problems exposed in practice are: partitioning by role or by front and back end, which leads to duplicated modules; modules with the same name appearing in one diagram; coupling that is too high, with unrelated modules placed under the same parent; and inaccurate module naming that matches the function description poorly [6].

Students judge and revise according to AI's comments so that the module partitioning conforms to the design logic of the project, and record the change in coupling before and after the adjustment. The literacy outcome lies in judging output by design principles rather than replacing one's own design decisions with AI conclusions.

3.3. Database Design Stage

Students complete a first draft of the E-R diagram and submit it to AI for review from three aspects: whether entity identification misses key business objects, whether the types of relations between entities are correct, and whether redundant relations exist. AI also judges whether the diagram satisfies the third normal form and points out transitive dependencies that can be removed.

Field design is a finer review point: whether naming is consistent, whether data types and lengths match the business, whether primary key, foreign key and uniqueness constraints are complete, and whether nullability is reasonable. AI performs well on details, but tends towards over-normalisation and neglects query performance; students must make trade-offs according to business access patterns, and the literacy outcome rests on the critical application of normalisation theory.

3.4. Detailed Design Stage

Students draw activity diagrams and sequence diagrams for the main functions of the system, so as to further clarify how the functions are implemented. The review focuses on whether the decision branches of the activity diagram cover exceptions and whether the swimlanes are consistent with module responsibilities, and on whether the order of object interaction in the sequence diagram conforms to the business logic, whether synchronous and asynchronous messages are chosen appropriately, and whether the activation periods of the lifelines match the actual calls.

The key at this stage is not whether a single diagram is right or wrong, but consistency across diagrams: whether a use case can find its corresponding realisation in the sequence diagram, and whether module responsibilities are reflected in the swimlanes of the activity diagram. AI is good at pointing out gaps in syntax and completeness, but poor at discovering semantic conflicts across documents. Here the teacher guides students to check the one-to-one correspondence among the use-case diagram, the module diagram, the E-R diagram, the activity diagrams and the sequence diagrams. The outcome lies in the rigour of design logic and the awareness of traceability.

3.5. Coding Implementation Stage

Students use code completion to break through coding bottlenecks and automatic commenting in Cursor to improve readability. The course, however, sets hard constraints: AI-generated code must be read line by line and its key logic explained in natural language; comments must explain design intent rather than restate the statement; and at

least three substantive modifications must be made to AI code, such as adding boundary checks, optimising queries and completing exception handling. The teacher samples the depth of understanding through a code defence, designating fragments at random for on-the-spot explanation [7].

The AI literacy outcome of this stage is to develop students' code-reading ability and awareness of standards, matching what current employers look for: strong logical thinking and the ability to revise AI code rather than merely generate it.

3.6. System Testing Stage

Students write a first draft of test cases from the requirements and the detailed design, and then use AI to supplement boundary values, exception paths and combination scenarios. AI enumerates boundary conditions efficiently, but produces redundant cases that cannot be executed or are semantically duplicated.

Students must judge the validity of the supplemented cases: deleting unexecutable items, merging duplicates and adding rule scenarios specific to the business. Evaluation takes the coverage of requirements by test cases and the proportion of valid cases as the main indicators; the literacy outcome lies in the systematic nature of testing thinking and the awareness of checking the completeness of AI output.

4. Teaching Implementation and Effect Analysis

4.1. Implementation Approach

The course selects a project with a complete business chain as the continuous carrier; teams of four to six students advance stage by stage, each stage corresponding to one AI review and one teacher verification. WorkBuddy is used uniformly, which keeps the process record traceable and avoids inconsistent evaluation standards caused by differences in tools. The teacher provides prompt templates but requires students to adjust them themselves; the expression of the prompt is itself part of the operation-layer training.

4.2. Evaluation Method

Evaluation consists of three parts - diagram quality, revision record and defence presentation - weighted 40%, 40% and 20% respectively. The revision record is the key of this design: students must record, item by item, AI's original suggestion, their own judgement, the reason for it, and the result after modification. The record externalises what would otherwise remain an internal thinking process and becomes the most direct observable evidence of AI literacy.

4.3. Effect Analysis

Effects can be observed through two comparisons: first, differences in output quality under different teaching requirements, with indicators including missed requirements, module coupling, the number of normal-form violations, the number of code-standard warnings and test coverage; second, questionnaire and defence performance, examining the clarity with which students articulate design principles and the reasonableness of their revision reasons. The most visible change is not in the appearance of the diagrams but in the way students ask questions: from "draw me a use-case diagram" to "please check whether my use-case diagram misses exception flows" - which is itself a sign that judgement-layer literacy has taken shape.

5. Problems and Reflections

First, the risk of dependence. The more capable the tool, the more easily students skip thinking and adopt directly, producing "qualified output, absent understanding". The response is to add a defence and a "deliberately planted error" link to the assessment: the teacher provides in advance a review record containing erroneous suggestions and asks students to identify where the problems lie.

Second, unstable output. The same prompt may yield different suggestions in different rounds, and conclusions contrary to design principles may even appear. Teaching must therefore make clear that design principles are the final standard, that AI

suggestions are only for reference, and that cross-verification over several rounds is better than single adoption.

Third, the boundary of use. From the start of the course it must be made clear which stages AI may enter and which parts it cannot replace, such as the authenticity of requirements investigation and the attribution of responsibility for design decisions, and students must mark the degree of AI participation in their deliverables, so as to avoid disputes over academic integrity.

6. Conclusion

Based on the theoretical course Software Engineering, this paper embeds AI tools in the complete process of software project analysis and design, translates the three-layer structure of operation, judgement and responsibility into six executable teaching stages, and uses the revision record as the vehicle for literacy assessment. Practice shows that AI literacy cultivation does not require a separate course; the key is to redefine the role of the tool: not a ghost-writer, but a reviewer always present; not a shortcut that lowers the difficulty, but a lever that raises the demand for judgement.

Funding: This work was funded by the Higher Education Teaching Reform Project of Guangdong University of Science and Technology under Grant GKZLGC2025056, GKZLGC2025021, and GKZLGC2025392.

References

1. X. Xu and X. He, "Generative artificial intelligence empowering software engineering education and the improvement of students' ability," *Computer Education*, vol. 2025, no. 8, pp. 1–8, 2025.
2. Q. Wu, X. Cheng, G. Zhao, et al., "Opportunities and challenges of AIGC-assisted computer programming teaching," *Computer Education*, vol. 2025, no. 7, pp. 69–73, 2025.
3. K. Yin, "Artificial intelligence literacy: background, concept definition and constituent elements," *Library and Information*, vol. 44, no. 3, pp. 60–68, 2024.
4. S. Zhu, H. Li, D. Wu, et al., "Construction of artificial intelligence core literacy for minors: research on artificial intelligence technical specifications for minors (II)," *e-Education Research*, vol. 44, no. 6, pp. 15–21, 53, 2023.
5. J. Yang and Y. Qiang, "AI-enabled software engineering teaching reform under the OBE concept," *Computer Education*, vol. 2025, no. 8, pp. 46–51, 2025.
6. C. Xiao and S. Wang, "Application of generative artificial intelligence in the teaching of software design patterns," *Computer Education*, vol. 2024, no. 11, pp. 161–166, 2024.
7. S. Chen, Z. Chen, X. Fan, et al., "Code refactoring teaching in software engineering practice courses driven by generative artificial intelligence," *Computer Education*, vol. 2025, no. 6, pp. 213–217, 2025.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of Publisher and/or the editor(s). Publisher and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.