ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202605_9(5).0019

Exploration of Experimental Teaching Reform in Object-Oriented Programming Based on the "Three-phase Progressive and Multi-dimensional Assessment" Model

Xingjian Liang[1,*], Fang Xie[2]

[1]School of Computer Science and Engineering, Sichuan University of Science & Engineering, Yibin 644000, China

[2]School of Foreign Languages, Sichuan University of Science & Engineering, Yibin 644000, China

[*] Corresponding author

Abstract

In response to prevalent issues in current Object-Oriented Programming (OOP) experimental teaching, such as the disconnection between experimental content and engineering practice, low student learning enthusiasm and initiative, simplistic traditional assessment methods, and the impact of AI code generation tools, a "Three-phase Progressive and Multi-dimensional Assessment" experimental teaching model was proposed and implemented. This model systematically restructured experimental activities into three organically connected phases: "pre-class foundational exploration, in-class incremental practice, and post-class in-depth review and reflection." It incorporated a competitive mechanism based on an Online Judge (OJ) platform and a multi-dimensional formative assessment covering the entire process. Teaching practice indicates that this model can effectively enhance students' analytical and problem-solving abilities. While boosting learning initiative, it also provides an actionable pathway for addressing the challenges posed by AI technology and deepening the reform of practical teaching in engineering education.

Keywords

Object-Oriented Programming; Experimental Teaching; Teaching Model; Three-phase Progressive; Multi-dimensional Assessment.

1. Introduction

Object-Oriented Programming (OOP) is a core fundamental course for computer science and related disciplines. Given its highly practical nature and significant demands on students' programming proficiency, the corresponding experimental teaching component is designed not merely for mastering syntactic rules, but more importantly, for equipping students with the ability to identify and solve problems [1]. However, practical experimental teaching often faces issues such as outdated and conventional content [2-3], a lack of student initiative and low engagement [2,4-5], as well as challenges posed by the advancement of AI code generation technologies. Experimental projects are frequently disconnected from engineering practice, making it difficult to stimulate in-depth learning interest. The rapid proliferation of AI code generation tools further exposes the limitations of the traditional code-result-oriented teaching and assessment paradigm. The interplay of these factors constrains the intended role of experimental sessions in cultivating students' computational thinking, complex problem-solving skills, and software engineering practice capabilities. Consequently, systematic reform of OOP experimental teaching is imperative.

ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202605_9(5).0019

2. Analysis of the Current State of OOP Experimental Teaching

With the rapid advancement of AI technologies and the evolution of educational concepts, the traditional experimental teaching model reveals systemic inadequacies in several key areas, including experimental content design, student engagement, the impact of AI technology, and assessment mechanisms. The following provides a detailed analysis of the current situation from these four aspects.

2.1. Outdated and Conventional Experimental Content

Currently, in the experimental teaching component of this course at most institutions, the dominant approach largely relies on a theory-textbook-matched experimental system centered on knowledge verification. Constrained by the limited practical class hours—typically around 16 periods—the design of experimental projects tends to favor verification-oriented tasks with fixed structures and singular objectives [4]. The primary goal is to verify and consolidate theoretical knowledge points. For example: designing a "Student class" to verify and practice the basic syntax of classes and objects; using relationships among objects like "Computer", "CPU" and "Memory" to grasp concepts of composition and aggregation; and utilizing inheritance hierarchies such as "Person", "Student" and "Teacher" to understand inheritance and polymorphism. Under this experimental model, which follows a "knowledge-point mapping" design logic, instructors typically demonstrate standardized designs and explanations in class, after which students can only engage in imitative code reproduction. While this process helps consolidate syntactic knowledge, it fails to foster the development of systematic object-oriented thinking and modeling capabilities in students. Experiments gradually devolve into mechanical coding exercises. Student engagement remains low, with most participants aiming merely to complete the task. Consequently, the practical component significantly deviates from the course objective of cultivating practical problem-solving abilities, and it is ineffective in supporting the development of students' competencies in analysis, design, and implementation within complex engineering scenarios.

2.2. Low Student Interest and Lack of Resilience

On one hand, students generally exhibit weak intrinsic motivation for programming practice. The development of programming competency relies on sustained and in-depth practice. However, some students devote severely insufficient time to after-class practice and lack adequate accumulation in coding[6]. Consequently, when faced with logically complex tasks that are only slightly beyond the basics, they are prone to feeling daunted and tend to give up. In experimental assessments, these students often aim merely to meet attendance requirements and complete basic tasks, showing little willingness or effort to engage with more challenging assignments.

On the other hand, students' ability to independently handle program errors is notably deficient. Programming is inherently an iterative process of debugging and correction. Yet, many students easily experience anxiety and frustration upon encountering runtime errors, finding it difficult to calmly and systematically localize and analyze the issues. Their problem-solving strategies often remain superficial, relying on seeking direct help rather than engaging in self-directed investigation. This prevents the debugging process from effectively transforming into a crucial learning stage for deepening understanding and honing logical thinking, thereby further diminishing the potential for gaining a sense of accomplishment and interest through practical work.

ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202605_9(5).0019

2.3. The Impact of AI Code Generation Technology on Traditional Programming Instruction

In recent years, the proliferation of various AI code generation tools has posed a fundamental challenge to the traditional programming teaching model based on writing code manually and step-by-step debugging. It has significantly altered the behavioral patterns and cognitive pathways through which students complete programming practice tasks. When confronted with experimental assignments, some students tend to regard AI tools as instant answer generators, directly describing problem requirements to the AI to obtain complete or extensive code segments. This approach leads them to bypass essential learning phases such as independent problem analysis, algorithmic design, syntax consultation, and even trial-and-error debugging. As a result, they fail to engage in the necessary object-oriented thinking exercises and cultivation of rigorous code logic through the experimental process.

2.4. Singular Experimental Assessment Method, Failing to Gauge Authentic Learning Outcomes

Currently, the assessment of experimental teaching predominantly employs an outcome-oriented summative evaluation model[7]. This model relies primarily on scoring the final code submitted by students and its execution results, while severely neglecting students' cognitive activities, level of effort, and problem-solving strategies during the experimental process. Given the high replicability of programming code, and in the absence of effective process supervision and academic integrity constraints, students may resort to mutual plagiarism, downloading solutions online, or directly using AI tools to generate code to obtain the correct answer, thereby circumventing the independent thinking and skill training that should have taken place. Furthermore, under the practical constraints of a high student-to-teacher ratio and large class sizes, it is difficult for instructors to conduct continuous observation and precise evaluation of each student's experimental process, which further widens the gap between assessment results and genuine competency. Therefore, a singular assessment focused solely on the final code not only fails to accurately distinguish students' true programming ability and level of thinking but also objectively encourages utilitarian, surface-level learning behaviors. This undermines the role experimental teaching should play in cultivating students' rigorous academic attitude, independent thinking skills, and engineering practice ethos.

3. The Concept of the "Three-phase Progressive and Multi-dimensional Assessment" Experimental Teaching Model

This paper proposes the "Three-phase Progressive and Multi-dimensional Assessment" experimental teaching model. By elaborately designing the three phase of pre-class, in-class and after-class learning, it constructs a spirally ascending ladder for ability training. The model integrates competition mechanisms and reflective sessions, and requires teachers to evaluate students’ completed learning outcomes across all three stages. In this way, it can effectively address the aforementioned current situations and existing problems, and substantially improve the quality of experimental teaching.

The "Three-phase Progressive and Multi-dimensional Assessment" model aims to restructure the experimental teaching process. It systematically deconstructs the traditional single-session experimental activity into three interlocking, spirally ascending phases: "pre-class foundational exploration, in-class incremental practice, and post-class in-depth review and reflection." Multi-level formative assessment is embedded throughout. In the pre-class phase, students complete foundational tasks using online resources or AI tools and document their solution approaches and programming rationale. The instructor conducts an initial diagnosis of learning readiness and provides feedback based on this. During the in-class phase, students engage in

ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202605_9(5).0019

incremental development in the lab based on their pre-class work, with the instructor providing on-site guidance and Q&A. A second round of process assessment is conducted via an Online Judge (OJ) platform. In the post-class phase, students are required to systematically organize their solutions for the incremental tasks and conduct a structured review and reflection of the entire experimental process. The instructor then provides a third, comprehensive evaluation based on the depth of reflection and quality of the summary, thereby forming a closed-loop teaching and assessment system that covers the entire experimental cycle.

The core advantage of this model lies in its use of structured process management to address learning challenges in the AI era and its progressive task design to foster the step-by-step development of student competencies. First, it makes the learning process visible and assessable. By focusing the three-phase assessment on students' thinking processes, level of effort, and iterative progress rather than solely on the final code, it effectively curbs opportunistic behaviors such as over-reliance on AI-generated or plagiarized code. Second, the progressive "foundational-incremental-reflective" design ensures the consolidation of basic knowledge, stimulates inquiry and collaboration through incremental tasks, and compels students to integrate and solidify knowledge through the mandatory reflection phase. This design not only enhances the logicality and systematicity of experimental teaching but also aligns better with the gradual nature of engineering competency development. It facilitates a shift in instructional focus from code implementation to process management and reflective enhancement in the context of widespread AI technologies adoption, targeting the cultivation of higher-order abilities.

4. Specific Implementation of the "Three-phase Progressive and Multi-dimensional Assessment" Experimental Teaching Model

To ensure the effective implementation of the "Three-phase Progressive and Multi-dimensional Assessment" teaching model, this course has undergone a systematic restructuring in terms of experimental procedure, task design, supporting platforms, and assessment mechanisms. Using the three organically connected phases: pre-class, in-class, and post-class, as the framework, the specific implementation plan is introduced below.

4.1. Pre-class Phase: Foundational Task Exploration and Preliminary Assessment

The core objective of the pre-class phase is to guide students in completing preliminary knowledge preparation and practice, thereby laying a foundation for in-depth classroom instruction. The instructor publishes foundational tasks in advance on the online teaching platform (e.g., the course website or a learning management system). These tasks are closely aligned with the core knowledge points of the current experiment, aiming to verify fundamental concepts and syntax. For instance, when learning "Classes and Objects," a preparatory task is set as "Implement a complex number class with methods for addition, subtraction, multiplication, and division." For the topic "Inheritance and Polymorphism," a task might be "Design a student class, create an array of student objects, and implement sorting by student ID." The instructor must clearly communicate that the formal experimental content and assessment will involve incremental development based on these foundational tasks. This encourages students to complete the foundational tasks with purpose and strive to master them, rather than merely submitting code through plagiarism, copying, or direct AI generation to fulfill the requirement perfunctorily.

While publishing the foundational tasks, the instructor also releases instructional videos detailing the experimental design rationale and coding process. If students lack the prerequisite

ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202605_9(5).0019

knowledge to complete the tasks independently, they can refer to these videos or utilize AI tools for assistance. They are required to write a concise programming rationale documenting their design logic, understanding of key code segments, and any problems encountered. By reviewing the submitted code and rationale documents via the online platform, the instructor achieves an initial diagnosis of the pre-class learning status and collects common issues, providing a basis for targeted intervention during the in-class phase.

4.2. In-class Phase: Incremental Practice and Real-time Evaluation

The in-class phase serves as the core component for deepening abilities and conducting comprehensive training. The teaching activities revolve around incremental tasks—extensions and elaborations based on the pre-class assignments, either logically or functionally, prompting students to apply acquired knowledge to solve more complex problems. For example:

Building upon the "Complex Number Class" preparatory task, an in-class task could be designed as: "On the basis of the implemented complex number class, add methods to calculate the modulus and argument, and override the equals() method to enable equality comparison for complex numbers."

Building upon the "Student Class" preparatory task, an in-class task could advance to: "Design a 'Fruit' class and its subclasses, implement multiple sorting (using comparators) based on attributes such as weight and price, and manage the fruit collection with CRUD (Create, Read, Update, Delete) operations."

To stimulate learning motivation and simulate the efficiency pressures encountered in real-world engineering practice, this phase introduces an ACM-style competition mechanism based on an Online Judge (OJ) platform. Students are required to submit code for the incremental tasks on the OJ platform within a set time limit. The system provides real-time judgment (correct/incorrect feedback) and generates a ranking based on submission attempts and problem-solving time. The instructor's role shifts from a lecturer to a facilitator and guide, focusing on personalized Q&A, addressing common issues through focused explanations [6], and guiding students to progress from merely making the code work to critically thinking about how to make the code more optimal and robust.

4.3. Post-class Phase: Structured Review and Experimental Report Assessment

The post-class phase aims to facilitate the integration and consolidation of knowledge, enabling the critical transition from "having done" to "having learned" and ultimately to "knowing how to learn." Students are required to complete a structured experimental report, which must include at least the following three parts:

(1) **Rationale and Implementation Annotations:** Students must add design-process descriptions and detailed comments to the code of the incremental task completed in class, clearly explaining the logic behind key code sections, class structure design, and important methods.

(2) **Problem and Debugging Summary:** Students must systematically document all compilation errors, runtime errors, and logical errors encountered throughout the entire experimental process, detailing the troubleshooting steps, resolution strategies, and the underlying principles involved.

(3) **Reflection and Transfer Proposal:** Students should summarize the core insights gained from the experiment and any remaining points of confusion. Furthermore, they are to attempt to transfer a design pattern or problem-solving method learned in this experiment to a self-proposed, similar new scenario, providing a brief description.

By evaluating this review report, the instructor conducts the third assessment, focusing particularly on the depth of student reflection, the systematic nature of their summaries, and

ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202605_9(5).0019

their nascent ability to transfer and apply knowledge. This completes a closed-loop, multi-dimensional evaluation cycle for a single experimental activity.

5. Analysis of Practical Effectiveness

After one semester of implementing the experimental teaching reform in the "Object-Oriented Programming" course, this model has yielded preliminary positive outcomes.

Student learning initiative has significantly increased. Due to the close linkage between in-class tasks and pre-class preparation, students' proactiveness and the quality of their preparation have noticeably improved. The ACM-style competition format has transformed the classroom atmosphere from passive listening to active exploration, making students the true protagonists of their learning. The iterative debugging process on the OJ platform has greatly enhanced students' problem-solving abilities and debugging skills.

Object-oriented design thinking has been deepened. The incremental tasks compelled students to continually review and modify their class designs, transforming their understanding of concepts like encapsulation, inheritance, and polymorphism from theoretical knowledge to practical application. Post-class reflection further consolidated these design principles.

The use of AI tools has become more rational. Students gradually realized that AI tools cannot directly provide perfect answers that meet complex incremental requirements. Their value lies more in offering conceptual references and aiding debugging. Consequently, the pattern of usage has shifted from plagiarism to collaboration.

A comparative analysis of the course grades for two cohorts of software engineering majors at the university which the author works in revealed a significant improvement in the excellence rate for the 2024 cohort (the experimental group implementing the reform) compared to the 2023 cohort (the control group without the reform). Specifically, the 2023 cohort had a final exam paper excellence rate of 1.6% and a comprehensive course grade excellence rate of 4.3%. In contrast, the corresponding rates for the 2024 cohort rose to 9.6% and 10.2%. This represents an increase of 8.0 percentage points in the final exam paper excellence rate and 5.9 percentage points in the comprehensive course grade excellence rate, indicating the reform's positive effect on promoting overall student learning outcomes.

Practice has shown that the "Three-phase Progressive and Multi-dimensional Assessment" experimental teaching model, by restructuring the teaching process, strengthening formative assessment, and organically integrating competition and reflection mechanisms, has effectively stimulated students' intrinsic motivation, systematically cultivated their problem-solving and system design skills essential for the AI era, and enhanced the quality and rigor of experimental teaching. It provides an actionable pathway for the reform of experimental teaching in programming courses.

6. Conclusion

Faced with the problems existing in the traditional experimental teaching of Object-Oriented Programming, including rigid experimental content, insufficient students' learning interest and autonomy, a single evaluation method, and the practical impact brought by AI code generation technology, this paper systematically proposes a "Three-phase Progressive and Multi-dimensional Assessment" experimental teaching model on the basis of an in-depth analysis of the above issues. The model deconstructs experimental teaching activities into three organically connected, spirally ascending phases: "pre-class foundational exploration, in-class incremental practice, and post-class in-depth review and reflection." By integrating OJ platform competition, multi-phase process assessment, and a mandatory reflection mechanism, it constructs a closed-loop teaching and assessment system. This model not only effectively

ISSN: 2637-6067 DOI: 10.6918/IJOSSER.202605_9(5).0019

mitigates the academic integrity crisis posed by AI tools but also transforms them into aids for promoting deeper learning. Through progressive task challenges, interest stimulation via competition, and the solidification of knowledge through reflection, it tangibly enhances students' core competency in OOP and their comprehensive abilities. This exploration provides a valuable contribution to cultivating software talents equipped to meet the demands of the intelligent era.

Acknowledgements

The authors sincerely acknowledge the financial support from the following projects: the Sichuan University of Science & Engineering Teaching Reform Research Project “Exploration of Experimental Teaching Reform for Programming Courses” (Grant No. JG-2306), and the Sichuan University of Science & Engineering Graduate Teaching Development Project “Software Architecture Course Case Library” (Grant No. AL202305).

References

[1] Sun Yuan. Exploration and Practice of Teaching Mode Reform in Object-Oriented Programming Course[J]. China Educational Technology & Equipment, 2025, (06): 65-69.

[2] Gao Zhiyuan, Li Wei. Reform and Exploration of Experimental Teaching for the C++ Programming Course in Undergraduate Institutions[J]. Modern Business Trade Industry, 2025, (04): 258-260.

[3] Cao Manman. Innovation and Reform of Experimental Teaching for Computer Programming Courses under the Background of Emerging Engineering Education[J]. Computer Education, 2024, (05): 64-69.

[4] Chen Jun. Exploration and Practice of an Online-Offline Blended Teaching Model Based on the OBE Concept—Taking the Fundamental Programming Course as an Example[J]. Information & Computer, 2024, 36(22): 233-235.

[5] Zhu Jingwen, Xie Maoqiang, Zhang Shenglin. Exploration of the Experimental Teaching Model for C/C++ Programming in Software Engineering Major[J]. Computer Education, 2024, (08): 208-212.

[6] Li Wenfeng, Fan Haiju, Tian Hongjuan, et al. Experimental Teaching of Programming with a Tiered-Advancement Approach Combining "Teaching-Practicing-Competing"[J]. Computer Education, 2024, (01): 204-208.

[7] Jia Jinfang. Reform and Exploration of Experimental Teaching for the C++ Programming Course[J]. Computer Knowledge and Technology, 2024, 20(08): 146-148+164.