

OBE-Oriented Teaching Design for Architectural Design Module in Introduction to Software Engineering Course

Yadong Gong

School of Computer Science and Software, Zhaoqing University, Zhaoqing, 526061, China

Abstract: Architectural design is a key bridge from requirement analysis to software structure design in *Introduction to Software Engineering* course. Yet traditional instruction often leaves students knowing concepts but unable to divide modules, draw clear structure charts, or define interfaces properly. To address this problem, this paper proposes an AI-empowered OBE teaching design. Centered on expected learning outcomes, it defines knowledge, ability, and quality objectives, and aligns them with graduation requirements. Case projects are used to guide students through requirement understanding, function decomposition, module division, structure chart drawing, interface analysis, and design evaluation. Meanwhile, AI tools support case generation, requirement clarification, scheme comparison, feedback, and learning diagnosis, while students must record AI use, revision reasons, and personal judgments. The teaching process follows six stages: case introduction, knowledge construction, method demonstration, group practice, achievement evaluation, and continuous improvement. This design shifts teaching from concept transmission to competence development and from result-only grading to process-based assessment, helping improve students' design ability, engineering awareness, teamwork, and responsible tool use.

Keywords: Outcome-Based Education, Introduction to Software Engineering, Architectural Design, Teaching Reform.

1. Introduction

Introduction to Software Engineering is an important foundational course for computer-related majors [1-2]. It helps students understand the software development process, develop software engineering thinking, and build preliminary engineering practice abilities. In the teaching content of this course, architectural design plays a key role in connecting requirements analysis with detailed design [3-4]. It is the stage in which students move from “understanding and analyzing a problem” to “designing a feasible software solution.” Through architectural design, students need to determine the overall software architecture, divide system functions into modules, clarify module responsibilities, design interface relationships, and consider software quality attributes such as maintainability, extensibility, and reliability. Therefore, the teaching quality of architectural design directly affects students' understanding of software structure and their ability to carry out engineering-oriented software design. It is also an important entry point for cultivating students' software design competence.

However, in traditional teaching practice, the teaching of architectural design is often dominated by teacher explanation, PowerPoint demonstration, and after-class assignments. Students may be able to remember the definition, tasks, principles, and basic methods of architectural design, but they often encounter difficulties when facing real or semi-real project requirements [5]. For example, some students divide modules only according to surface-level functions without considering cohesion and coupling; some draw software structure charts with unclear hierarchy and unreasonable control relationships; some fail to describe module interfaces accurately; and others do not know how to transform a data flow diagram into a software structure chart. These problems show that the main contradiction in architectural design teaching is not that students do not know the concepts, but that they cannot effectively transform requirements into a reasonable software structure. In other words, there is still a

gap between conceptual understanding and design application.

The outcome-based education approach provides an important idea for improving this situation. OBE emphasizes student learning outcomes and requires teaching objectives, teaching activities, evaluation methods, and continuous improvement to be organized around the achievement of abilities [6-7]. For the teaching of architectural design, this means that the focus should shift from “what knowledge the teacher has taught” to “what design tasks students can complete after learning.” Meanwhile, the rapid development of artificial intelligence technology has brought new opportunities for software engineering education. AI tools can support teachers in generating project cases, task sheets, evaluation rubrics, and common error lists. They can also help students analyze requirements, check missing functions, compare alternative module division schemes, and revise design documents. More importantly, AI can assist teachers in collecting and analyzing learning process data, thus making it possible to evaluate not only final design results but also students' design process and improvement trajectory. Therefore, AI should not be regarded as a substitute for students' design work, but as a teaching assistant, learning partner, and diagnostic tool that promotes deeper thinking and iterative improvement.

Based on the above considerations, this paper focuses on the architectural design knowledge unit in the course *Introduction to Software Engineering*. Guided by the OBE concept, supported by AI technology, and organized through case-based teaching, this study designs the teaching objectives, teaching strategies, and teaching process of architectural design, aiming to explore an effective path for improving students' software structure design ability. The general reform idea of this paper can be summarized as follows: OBE is used to define ability-oriented learning outcomes, case projects are used to carry learning tasks, and AI is used to support teaching implementation, learning feedback, evaluation, and continuous improvement. Through this design, architectural design teaching is expected to move

from knowledge transmission to ability cultivation, from result-oriented assessment to process-oriented evaluation, and from experience-driven teaching to data-supported continuous improvement.

2. Teaching Objectives

The teaching objectives of the architectural design knowledge unit should be established according to the talent cultivation goals of computer-related majors, the graduation requirements of engineering education, and the course objectives of *Introduction to Software Engineering* [8]. Architectural design is a key stage connecting requirement analysis and detailed design in the software life cycle. Its teaching should not only help students understand basic concepts and principles, but also guide them to complete practical design activities, such as identifying system functions, dividing modules, drawing software structure charts, and explaining interface relationships. Therefore, under the guidance of the OBE concept, the teaching focus should shift from “what the teacher teaches” to “what students are able to accomplish” [6]. The expected learning outcomes of this unit can be described as follows:

(1) Knowledge objectives. Students should understand the position and role of architectural design in the software development process, and master its main tasks, basic procedures, design principles, heuristic rules, software structure charts, and data-flow-oriented design methods. They should be able to explain key concepts such as abstraction, stepwise refinement, information hiding, module independence, cohesion, and coupling, and understand how these concepts influence software maintainability, extensibility, reusability, and reliability.

(2) Ability objectives. Students should be able to analyze a given requirement specification, identify system functions and data flows, conduct functional decomposition and module division, draw a standardized software structure chart, and describe module responsibilities and interface relationships. They should also be able to evaluate and optimize a design scheme according to criteria such as requirement coverage, structural rationality, high cohesion, low coupling, clear interfaces, and ease of maintenance. In addition, students should be able to use AI tools to assist in requirement sorting, function checking, and scheme comparison, while making independent professional judgments rather than directly accepting AI-generated results.

(3) Quality objectives. Students should develop engineering standard awareness, software quality awareness, teamwork awareness, and responsible use of modern tools. In group design tasks, they should learn to communicate design ideas, cooperate in module division, present design evidence, and revise design schemes based on feedback. When using AI tools, students should record the prompt content, AI suggestions, adopted parts, modification reasons, and personal judgments, so as to form a critical, reflective, and accountable attitude toward AI-assisted learning and engineering design [10].

(4) Correspondence with graduation requirements. The above objectives correspond to several graduation requirement indicators of computer-related majors. Understanding architectural design principles and methods supports the requirement of applying engineering knowledge; analyzing requirement specifications and identifying system functions supports problem analysis ability; drawing software structure charts and explaining design decisions supports the

ability to design and develop solutions; using AI tools for assisted comparison and diagnosis supports the ability to use modern tools; and completing group discussion, design presentation, peer review, and document writing supports teamwork and communication abilities. Through this correspondence, the objectives of the architectural design unit become observable, assessable, and continuously improvable learning outcomes, providing a clear basis for teaching activity design, process evaluation, and subsequent teaching improvement.

3. Teaching Strategies

Under the concept of Outcome-Based Education, the teaching of architectural design should shift from a content-centered model to a learning outcome-centered model. In the architectural design unit of *Introduction to Software Engineering* course, the overall teaching strategy can be summarized as “outcome-oriented, case-driven, AI-assisted, collaboratively completed, and process-evaluated.” Specifically, all teaching activities should be organized around the expected learning outcome that students are able to complete an architectural design scheme for a given software project. Teachers should no longer arrange instruction simply according to textbook chapters, such as concepts, principles, diagrams, and methods. Instead, they should design teaching tasks by considering what kind of software structure design students are expected to complete at the end of learning. Therefore, teaching objectives, classroom activities, learning resources, AI tool use, and assessment methods should all serve the achievement of this outcome. In this way, architectural design teaching can better reflect the OBE principle of backward design, namely, determining learning outcomes first and then designing teaching activities and evaluation standards accordingly.

Case-based project teaching is an important carrier for implementing architectural design knowledge. In actual teaching, teachers may select cases that are close to students’ campus life or professional practice, such as a student performance analysis system, an online course selection system, a library management system, or a campus second-hand trading platform. These cases should not be overly complex, but they should include basic requirement descriptions, business processes, data inputs and outputs, functional boundaries, and possible quality requirements. Through such cases, students can experience the complete transformation process from requirement understanding to software structure design. For example, when designing an online course selection system, students need to identify functions such as student login, course inquiry, course selection, course withdrawal, timetable generation, and administrator maintenance, and then further consider how these functions should be decomposed into modules and how the modules should interact with one another. In this process, the case project becomes a practical carrier of architectural design knowledge, helping students understand that software structure design directly affects system maintainability, extensibility, reusability, and subsequent detailed design.

AI tools can provide effective support before, during, and after class, thereby improving the efficiency and quality of teaching implementation. Before class, teachers may use AI to assist in generating case materials, requirement descriptions, task sheets, evaluation rubrics, lists of common errors, and classroom discussion questions [9]. This can reduce repetitive preparation work and help teachers provide

richer and more diverse teaching resources. During class, students may use AI tools to sort out requirements, check whether important functions are missing, compare different module division schemes, and identify possible problems in software structure diagrams. For instance, after students propose an initial module structure, they can ask AI to analyze whether there are unclear responsibilities, excessive coupling, or missing interfaces. After class, teachers may use AI to analyze students' architectural design documents and summarize common problems, such as unreasonable module hierarchy, incomplete interface descriptions, weak correspondence between requirements and modules, or insufficient explanation of design principles. However, AI should be positioned as a teaching assistant, learning partner, and diagnostic tool rather than a substitute for students' independent design and judgment.

To avoid improper use of AI, it is necessary to establish clear AI usage norms and a critical evaluation mechanism. Students should be required to record their use of AI in the architectural design report, including the prompts they used, the suggestions provided by AI, the parts they adopted, the parts they rejected, and the reasons for their modifications. This requirement can help students transform AI use from simple answer generation into a process of comparison, judgment, revision, and reflection. At the same time, teachers should examine whether students truly understand their own design schemes through classroom questioning, group defense, document review, and random explanation of module responsibilities and interface relationships. For example, if a group presents a software structure diagram generated or optimized with AI assistance, students must be able to explain why certain modules are divided in that way, what design principles are reflected, and how the scheme satisfies the original requirements. Therefore, the key to AI-empowered architectural design teaching is not to enable students to obtain answers more quickly, but to guide them in developing engineering judgment and critical thinking while using modern tools.

Process-based assessment and continuous improvement should be integrated into the whole teaching process. The evaluation of architectural design should not focus only on the final report, but should also pay attention to case discussion, classroom exercises, group design, staged submission, achievement presentation, peer review, and post-class revision. The assessment content may include requirement coverage, rationality of module division, standardization of software structure diagrams, analysis of coupling and cohesion, completeness of interface descriptions, team collaboration, and proper use of AI. Teachers can design an evaluation rubric corresponding to these indicators so that students clearly understand the expected learning outcomes and quality standards before completing the task. Meanwhile, assessment results should be used as an important basis for teaching improvement. By analyzing students' design documents, classroom performance, AI usage records, and peer evaluation results, teachers can identify common learning difficulties and adjust subsequent cases, exercises, teaching explanations, and evaluation criteria. In this sense, assessment is not the end of teaching, but a mechanism for promoting students' design revision and teachers' continuous improvement.

4. Teaching Procedure Design

The instructional process of architectural design is

organized around six progressive stages: case introduction, knowledge construction, method demonstration, group practice, achievement evaluation, and continuous improvement. Under the guidance of OBE, each stage is not arranged simply according to the order of textbook knowledge, but according to the expected learning outcomes that students should finally achieve. In this teaching unit, the core outcome is that students can transform a given requirement specification into a relatively reasonable software architecture, including functional decomposition, module division, software structure chart, module responsibility description, interface relationship description, and basic design quality analysis. Therefore, the teaching process follows a path from conceptual understanding to method application, from teacher demonstration to student practice, and from initial design formation to iterative optimization. AI tools are introduced throughout the process to support case preparation, classroom interaction, scheme comparison, learning diagnosis, and feedback generation, so that teaching activities can better reflect students' actual learning status and design difficulties.

In the first stage, case introduction is used to help students understand the engineering significance of architectural design. At the beginning of the class, the teacher presents a simple and familiar project case, such as a student performance analysis system. The case includes basic requirements such as score entry, score statistics, performance ranking, grade distribution analysis, and report generation. On this basis, the teacher provides two different software structure schemes: one scheme divides modules only according to interface pages, while the other scheme divides modules according to data input, business processing, statistical analysis, and report output. Students are guided to compare the two schemes from the perspectives of maintainability, extensibility, clarity of module responsibility, and coupling between modules. During the discussion, AI can be used to quickly generate possible maintenance scenarios, such as adding a new statistical indicator or supporting multi-class comparison, so that students can observe how different structures affect later system modification. Through this case comparison, students gradually realize that architectural design is not a formal drawing task, but an important engineering activity that directly influences software quality.

In the second stage, knowledge construction is conducted to help students master the basic tasks and principles of architectural design. The teacher explains that architectural design is mainly responsible for determining the overall software structure, decomposing the system into modules, defining module responsibilities, designing module interfaces, and determining the basic organization of data and control flow. On this basis, key design principles such as abstraction, stepwise refinement, information hiding, module independence, high cohesion, and low coupling are introduced. Instead of explaining these principles only as abstract definitions, the teacher combines them with positive and negative examples. For example, a module that simultaneously completes data input, statistical calculation, and report output is used as a negative example of low cohesion; two modules that frequently share internal data structures are used to explain the problem of high coupling. AI can assist the teacher in generating additional counterexamples and in-class questions, but the teacher should guide students to make judgments based on software engineering principles. In this way, students can understand

the engineering meaning of design principles through concrete structural schemes rather than remaining at the level of memorizing concepts.

In the third stage, method demonstration focuses on the mapping from data flow diagrams to software structure charts. The teacher first takes a transform-oriented data flow diagram as an example and demonstrates how to identify the input flow, transform center, and output flow. Then the teacher explains how to map the input part into input modules, the transform center into processing modules, and the output part into output modules, gradually forming a software structure chart with clear hierarchical relationships. During the demonstration, the teacher emphasizes that the structure chart should reflect calling relationships, module hierarchy, and the direction of data transfer, rather than simply copying the data flow diagram. After that, a transaction-oriented data flow diagram is introduced. The teacher guides students to identify the transaction center and different transaction paths, and then explains how to design a main control module and several transaction processing branches. AI tools may be used to compare alternative module division schemes or to check whether some functions are omitted, but the final judgment must be made by students according to design principles. This stage helps students establish a clear transformation path from requirement models to software structure models, which is one of the key abilities in architectural design learning.

In the fourth stage, group practice is arranged to enable students to complete the architectural design of a case project. Students are divided into small groups and each group receives a requirement specification, such as an online course selection system, a library management system, or a campus second-hand trading platform. Based on the given requirements, students need to complete functional analysis, module decomposition, software structure chart drawing, module responsibility description, interface relationship description, and design principle analysis. In this process, students are encouraged to use AI tools as learning assistants. For example, they may ask AI to help identify possible missing functions, compare two module division schemes, or generate a checklist for interface description. However, students must record their AI usage in the design report, including the prompts used, the suggestions given by AI, the parts adopted, the parts rejected, and the reasons for modification. The teacher observes group discussions, asks targeted questions, and provides timely guidance when students show problems such as unclear module hierarchy, overlapping module responsibilities, or direct dependence on AI-generated answers. Through group practice, students move from “understanding the method” to “applying the method”, while AI support improves the efficiency of feedback and the quality of design iteration.

In the fifth stage, achievement evaluation is carried out through classroom presentation, peer evaluation, and teacher feedback. Each group presents its architectural design scheme in class, including requirement understanding, module division logic, software structure chart, key interface relationships, and the use of AI tools. Other groups conduct peer evaluation according to a rubric provided by the teacher. The rubric focuses on requirement coverage, rationality of software structure, module independence, clarity of interface description, standardization of design documents, teamwork, and appropriateness of AI use. The teacher then gives comments on common strengths and weaknesses, especially problems such as incomplete requirement coverage, confused

structure chart hierarchy, excessive coupling between modules, and insufficient explanation of design basis.

In the sixth stage, continuous improvement is conducted based on the evaluation results and learning evidence collected during and after class. After class, the teacher collects student design documents, peer evaluation results, classroom observation records, and AI usage records to analyze common learning difficulties. These data are used to revise case materials, adjust classroom exercises, improve evaluation criteria, and strengthen weak teaching links in the next round of instruction. Therefore, evaluation is not merely a final score after teaching, but an important basis for students to revise their design schemes and for teachers to continuously improve instructional design.

5. Conclusion

This paper proposes an AI-empowered OBE teaching design for architectural design in the course *Introduction to Software Engineering*, aiming to solve problems such as the separation of theory and practice, weak student design ability, and delayed evaluation feedback. Guided by learning outcomes, supported by case projects and AI tools, the design integrates teaching objectives, strategies, processes, evaluation, and continuous improvement into a closed loop. It emphasizes students' ability to transform requirements into reasonable software structures through case introduction, knowledge construction, method demonstration, group practice, achievement evaluation, and iterative improvement. This approach helps improve students' software design ability, engineering practice ability, teamwork, communication, and appropriate use of modern tools. In future practice, it can be extended to requirement analysis, detailed design, software testing, and other software engineering modules, while being continuously optimized through authentic project cases and learning data.

References

- [1] Sommerville, I. (2016). *Software engineering* (10th ed.). Pearson.
- [2] Bourque, P., & Fairley, R. E. (Eds.). (2014). *Guide to the software engineering body of knowledge (SWEBOK Guide)* (Version 3.0). IEEE Computer Society.
- [3] Bass, L., Clements, P., & Kazman, R. (2021). *Software architecture in practice* (4th ed.). Addison-Wesley.
- [4] Garlan, D., & Shaw, M. (1993). An introduction to software architecture. In V. Ambriola & G. Tortora (Eds.), *Advances in software engineering and knowledge engineering* (pp. 1–39). World Scientific. https://doi.org/10.1142/9789812798039_0001
- [5] Lethbridge, T. C. (2000). What knowledge is important to a software professional? *Computer*, 33(5), 44–50. <https://doi.org/10.1109/2.841783>
- [6] Biggs, J. (1996). Enhancing teaching through constructive alignment. *Higher Education*, 32(3), 347–364. <https://doi.org/10.1007/BF00138871>
- [7] Felder, R. M., & Brent, R. (2003). Designing and teaching courses to satisfy the ABET engineering criteria. *Journal of Engineering Education*, 92(1), 7–25. <https://doi.org/10.1002/j.2168-9830.2003.tb00734.x>
- [8] ABET. (2022). *Criteria for accrediting engineering programs, 2023–2024*. <https://www.abet.org/accreditation/accreditation-criteria/>

- [9] Zawacki-Richter, O., Marín, V. I., Bond, M., & Gouverneur, F. (2019). Systematic review of research on artificial intelligence applications in higher education: Where are the educators? *International Journal of Educational Technology in Higher Education*, 16(1), 39. <https://doi.org/10.1186/s41239-019-0171-0>
- [10] Kasneci, E., et al. (2023). ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and Individual Differences*, 103, 102274. <https://doi.org/10.1016/j.lindif.2023.102274>