Curriculum and Teaching Methodology (2025)
Clausius Scientific Press, Canada

DOI: 10.23977/curtm.2025.080810
ISSN 2616-2261 Vol. 8 Num. 8

Exploring the Software Design Patterns Course Based on the Outcome-Based Education (OBE) Philosophy

Xianshuang Zhao

Guangzhou College of Applied Science and Technology, Zhaoqing, Guangdong, China

Keywords: Software Design Patterns; Outcome-Based Education; Software Development; Modular Teaching

Abstract: To address practical challenges in teaching "Software Design Patterns" courses-including complex content, disconnect between theory and practice, and implementation difficulties-this study explores an OBE (Outcome-Based Education) approach. Through discovery-based exploration with classroom implementation as the ultimate goal, we developed implementation-oriented projects that integrate software design patterns with commercial software development. These projects enable students to learn the advantages of modular programming through software development, cultivating design pattern thinking. By adopting OBE principles in teaching "Software Design Patterns," students gain precise understanding of design patterns' value in commercial software development, enhancing learning engagement and improving course effectiveness.

1. Introduction

"Software Design Patterns" is a course offered in the Computer Science and Technology program at higher education institutions, designed to help students standardize code and structural design in daily software development [1]. It aims to cultivate students' ability to write standardized code and conduct systematic architectural design for system frameworks [2]. However, as the course content leans heavily on theoretical and abstract concepts, many students struggle to comprehend its complex structural design principles, making it generally challenging for them to grasp.

Guangzhou Institute of Applied Technology has introduced the "Software Design Patterns" course in its Computer Science and Technology program since 2023, following the computer science curriculum framework. This outcome-based course adopts an industry-aligned educational design approach [3], enabling students to better apply theoretical knowledge in real-world software development environments. The program not only enhances students' understanding of software design principles but also builds practical foundations for their future careers, effectively bridging academic learning with industry needs.

2. OBE Philosophy

Outcome-Based Education (OBE), first proposed by Spady et al. in 1981, is an outcome-oriented pedagogical approach designed to equip students with the ability to apply acquired knowledge

effectively in engineering practice [4]. This framework rapidly gained traction as a mainstream educational reform concept in countries like the UK and France. The American Society for Engineering Education (ASEE) adopted it as a benchmark for integrating theory with practice. The OBE philosophy emphasizes combining instructional design with practical teaching to enable students to flexibly apply knowledge and achieve tangible outcomes. This requires implementing backward design principles [5] in teaching practices, where outcomes are planned in reverse to ensure alignment with educational objectives and help students attain desired learning outcomes.

In the design and implementation of educational curricula, classroom instruction remains an indispensable core teaching framework. It transcends mere one-way knowledge transmission to create a structured learning environment. Within this framework, educators meticulously curate and logically organize knowledge systems that form the bedrock of students' professional competencies. These foundational elements provide essential theoretical support and intellectual preparation for their future career success. Specifically, well-structured classroom teaching helps students develop clear disciplinary frameworks. When confronting complex, ever-changing workplace challenges, this systematic learning creates a cognitive map that prevents "missing the forest for the trees," offering reliable analytical foundations for problem-solving. More importantly, only by truly understanding the core principles and logical underpinnings taught in class can students grasp the "why" behind each operational step and technical decision during practical applications. This marks the critical leap from "knowing what" to "understanding why." Such deep comprehension empowers students to apply theories flexibly, adapt autonomously, and innovate boldly in practice. It also guides them to verify, reflect upon, and critique learned theories through hands-on experience, forming a virtuous cycle where theory and practice reinforce each other. Finally, the fundamental purpose of classroom teaching--that is, the realization of the organic integration and mutual promotion of theory and practice, so that students can truly achieve the excellent ability to apply what they have learned and use it to promote learning--is realized naturally in this process.

3. Teaching exploration of "software design patterns"

In the field of instructional design, numerous case studies have successfully implemented the OBE (Outcome-Based Education) philosophy across various courses, achieving remarkable teaching outcomes. For instance, Professor Sun Xia has applied OBE principles to language instruction [6], while Professor Chen Xiangqing has adapted them for the "Market Research Techniques" course [7]. The author has further applied OBE concepts to teach "Software Design Patterns" to computer science students.

3.1 Defining the objectives, significance, and final learning outcomes of the "Software Design Model" course within the professional framework

The Software Design Patterns course in computer education aims to equip students with practical application skills. By mastering these principles, learners can develop optimal software architectures during development processes. This foundational knowledge enhances their effectiveness in software development and maintenance. When entering corporate environments, students applying these patterns in real-world projects will significantly boost design efficiency while ensuring smoother software upgrades and future maintenance.

In alignment with the specific objectives and expected outcomes of the software design course, and adhering to the Outcome-Based Education (OBE) teaching philosophy [8][9], this course has developed clear and measurable learning outcomes through thorough research and meticulous planning:

3.1.1 Knowledge outcomes

Through course study, students will systematically master several common theoretical patterns in software design and their core principles. The Decorator pattern emphasizes dynamically extending functionality without altering the original object structure. By breaking down features into independent decorator classes and combining them through composition, it effectively avoids the proliferation of subclasses caused by deep inheritance hierarchies. The Factory pattern introduces specialized factory classes to encapsulate object creation logic, separating instantiation processes from business code. This significantly enhances code flexibility, maintainability, and scalability. The Adapter pattern uses adapter classes as an intermediate layer to convert incompatible interfaces into client-desired formats, resolving system collaboration issues caused by interface mismatches. The Singleton pattern ensures a class has only one instance, providing a unified global access entry point. By limiting instantiation frequency, it prevents resource waste, making it suitable for scenarios requiring strict control over object quantities. The Publish-Subscribe pattern decouples message publishers and subscribers, establishing an efficient asynchronous communication mechanism that improves system responsiveness and module independence. These design patterns collectively form essential principles and practical tools in object-oriented software architecture.

3.1.2 Competency Outcomes

Through this course, students will develop a systematic understanding of software design patterns and ultimately acquire the core competence to analyze and flexibly apply them in complex software design. This ability manifests in three dimensions: First, in software framework design, students will transcend isolated pattern recognition by integrating them as foundational elements of holistic architecture. They will demonstrate proficiency in applying creational, structural, and behavioral patterns to create highly cohesive yet loosely coupled frameworks that ensure optimal scalability and maintainability. Second, in logical thinking, students will move beyond mechanical application of theories to deeply analyze the inherent needs and contradictions within business scenarios. They will pinpoint design pain points in real-world localization and personalization challenges, evaluate trade-offs, and propose viable solutions-showcasing exceptional analytical and decision-making capabilities. Finally, in hands-on practice, acquired design patterns will translate into tangible code productivity. Students will implement design concepts through coding, developing critical modules and core logic in concrete system architectures to ensure precise execution of design principles. In summary, this course aims to cultivate software engineers capable of both strategic top-level design and meticulous problem-solving in practical implementations.

3.2 Design the teaching content of the course based on the learning results

Software design patterns are highly theoretical, involving abstract concepts and principles that students often struggle to connect directly with real-world applications. To effectively bridge the gap between theory and practice in classroom instruction, teachers prioritize incorporating concrete case studies into their teaching. Through these vivid, real-world examples, instructors can transform abstract theories into tangible concepts, helping students better grasp the core principles of design patterns. This approach not only sparks students' interest in learning but also enhances their hands-on skills and problem-solving abilities, making the curriculum more engaging and practical. By truly integrating theory with practice, this method achieves twice the result with half the effort, delivering exceptional educational outcomes.

The course instructional design content project table is shown in Table 1. The project design is based on real-world software design scenarios, integrating application scenarios of various software

design patterns to deepen students 'understanding. Project 1's application scenario involves banks using the Adapter pattern during software upgrades to achieve interface conversion, addressing compatibility issues between legacy systems or third-party systems. Given the rapid iteration speed of software features during upgrades, developing new interface functionalities requires significant manpower, time, and costs. For legacy systems, minor modifications to existing interface data are needed to maintain stability and functionality. To save development time and costs, the Adapter pattern is utilized to create middleware that integrates legacy systems with new ones, avoiding redundant development. New features are progressively added to the new interface to achieve upgrade objectives. Key learning outcomes include understanding the application scenarios of the Adapter pattern, hands-on implementation of Adapter code, and enhanced practical skills. Project 2's application scenario involves using the Decorator pattern in coffee order system design to improve system robustness. For example, coffee's basic components include milk, while sugar and syrup are decorative additions. During design, milk can be repeatedly decorated with sugar or syrup. By leveraging the Decorator pattern's characteristics, modular decoration operations are adopted to enhance code reuse capabilities, improve code reusability, and shorten development cycles.

Table 1: Project Design for the "Software Design Patterns" Course

Project ID	project name	Knowledge Output	Achievements
1	Application of Adapter pattern in banking scenarios	The Adapter pattern shares characteristics with other real-world application scenarios, serving as a functional solution.	Master adapter application scenarios, hands-on skills, and the ability to apply knowledge flexibly
2	Decorator pattern in coffee Application in the order system of the orchid	The Decorator pattern is characterized by its applicability to real-world scenarios, serving as a design pattern that provides a flexible way to add functionality to existing code.	Master the application scenarios of decorators for developers, and develop hands-on skills to apply knowledge flexibly.

4. Conclusion

This study explores the teaching of "Software Design Patterns" based on the Outcome-Based Education (OBE) philosophy. By focusing on the course's ultimate learning objectives, it conducts backward design and meticulous refinement of instructional knowledge and skill development. The approach helps students understand how these design patterns can be applied in real-world software development scenarios, ultimately achieving the desired learning outcomes.

In the OBE-based curriculum model, the practical nature of software development is addressed through a design approach that integrates real-world scenarios with coding. This methodology provides students with systematic design processes across multiple application contexts, enhancing their critical thinking and development capabilities through software design pattern categorization. The selection and determination of course objectives, teaching methods, and instructional tools are centered around educational outcomes, aiming to improve students 'comprehensive abilities while addressing limitations in existing teaching models. This approach ultimately elevates the teaching quality of the "Software Design Patterns" course and stimulates students' learning engagement.

References

[1] Llanez G C, Vallejo P, Aguilar J. Design patterns applied in the development of serious games for cognitive-affective training [J]. Science of Computer Programming,2026,248103378.
[2] Nasrabadi Z M, Parsa S, Jafari S. Measuring and improving software testability at the design level [J]. Information

and Software Technology,2024,174107511.
[3] ReimanisD, IzurietaC. A study of behavioral decay in design patterns [J]. Journal of Software: Evolution and Process,2023,36(7).
[4] Poy O, Moraga Á M, Garc á F, et al. Impact on energy consumption of design patterns, code smells and refactoring techniques: A systematic mapping study [J]. The Journal of Systems & Software,2025,222112303.
[5] Sudha R, A. S C. Evaluation of Quality Attributes of Software Design Patterns Using Association Rules [J]. International Journal of Advanced Intelligence Paradigms,2021,19(1-2).
[6] Sun Xia, Cheng Hongbin. Teaching Reform of C Language Course Based on OBE Model [J]. Journal of Ningbo Institute of Education, 2016,18(4):16-18.
[7] Chen Xiangqing, Guan Qiuyan, Zheng Peiqiong. Teaching Reform and Practice of 'Market Research Techniques' Course Based on OBE [J]. Business Economics, 2015(11):150-151.
[8] Sanchez-Gordon S, Sánchez-Gordón M ,Yilmaz M, et al. Integration of accessibility design patterns with the software implementation process of ISO/IEC 29110[J]. Journal of Software: Evolution and Process,2019,31(1).
[9] Hussain S ,Keung J ,Sohail K M, et al. Automated framework for classification and selection of software design patterns [J]. Applied Soft Computing Journal,2018,751-820.