Improved Teaching Model for Software Architecture Course

Zhenyan Ji[1, a *], Jing Song[2,b]

School of Software Engineering, Beijing Jiaotong University,Beijing,China

School of Software Engineering, Beijing Jiaotong University,Beijing,China

[a]zhyji@bjtu.edu.cn, [b]14126135@bjtu.edu.cn

Keywords: Software Architecture; Case driven; Teaching Model; Improved; Course

Abstract. Research on software architecture is a hotspot in the field of software engineering. To introduce the valuable engineering experiences and theoretical knowledge, a software architecture course is offered widely in colleges and universities. In our school, we originally adopted traditional approach, first introduced theoretical knowledge and then gave a couple of examples to explain it, to teach our students the course. Due to students' limited experience in developing software projects, the students feel difficult to understand the course contents. To reach better teaching effect, we propose case driven teaching approach, introduce cutting-edge industry seminars, and improve course assignments and grading. Each improvement is described in detail in the paper. Course evaluation results prove that students' satisfaction increases explicitly after the teaching model is improved.

Introduction

Since the late in the 1990, research on software architecture has become a hotspot in the field of software engineering [1-4]. More and more IT people have come to realize the significance of research on software architecture and its importance to the design and development of software systems. A lot of research and practical work has been conducted, and valuable experiences and knowledge has been accumulated [5]. Software architecture courses are widely offered in colleges and universities at home and abroad to introduce the valuable engineering experiences and theoretical knowledge [6,7].

Overview of Software Architecture Course

Overview of Software Architecture Course. As one of 37 National Pilot Software Engineering Schools, our school also offer Software Architecture course. The goal of Software Architecture course is teaching students modern design methods of software architecture and making it possible for students to grow into a good system architect within 5 or 8 years after graduation.

Our software architecture course consists of three dimensions, knowledge related to software architecture design, object-oriented design principles and design patterns. After learning the course, students should master the theoretical knowledge and apply them to solve practical engineering problems.

Pedagogy for Software Architecture Course. In our school all the courses adopt bilingual teaching. It means that all the course material such as reference books, PPTs, course assignments, lab materials and exams are in English and courses are lectured in English or Chinese. The motivation of bilingual teaching is to improve students ' ability of English listening, speaking, reading and writing. This ability is vital for a software engineer.

We originally adopted traditional approach to educate students the course. The teachers first introduced theoretical knowledge to students and then gave a couple of examples to explain it. The course had run several rounds and the course effect was not ideal. Due to students' limited experience in developing software projects, they felt difficult to understand principles and knowledge about architectural design. Additionally, bilingual teaching increased students' difficulties to understand the course contents. How to make the principles and patterns easier to be understood by students became our urgent task.

Improved Teaching Model

Analysis of Software Architecture Course. Our Software Architecture course consists of three dimensions, architecture design, OO principles and design patterns.

Architecture design includes three modules:

(1). A*rchitectural design*

In this module, the related concepts and principles of architectural design are introduced.

(2). *Quality attributes*

Firstly, the module introduces the concepts of quality attributes, such as performance, maintainability, modularity, extensibility and so on. Then how the quality attributes affect system architecture design is explained.

(3). A*rchitectural patterns*

In this module, several modern architectural patterns, such as C/S, B/S, Repository, Multi-layer architecture, SOA and so on, are introduced. The module also explains how to utilize architectural patterns to design software architectures.

OO principles dimension mainly introduces five OO principles [8]:

- SRP: Single Responsibility Principle
- OCP: Open Closed Principle
- LSP: Liskov Substitution principle
- DIP: Dependency Inversion Principle
- ISP: Interface Segregation Principle

and six package design principles[9]:

- The Reuse-Release Equivalence Principle
- The Common-Reuse Principle
- The Common-Closure Principle
- The Acyclic-Dependencies Principle
- The Stable-Dependencies Principle
- The Stable-Abstractions Principle.

This dimension also explains how to design software with the principles.

Design patterns dimension introduces 23 classic design patterns.

The course contents are abstracted from excellent software engineering experiences. The contents are very practical. If we adopt traditional way to teach, introduce theory first and then give a couple of examples. Students usually feel hard to deeply understand the principles and patterns so that feel difficult to apply them. How to decrease the difficulty to learn the course?

Case Driven Teaching Approach. To improve the teaching effects, we design a case driven approach to teach the course.

With the case driven approach, we tell a real life case to introduce a knowledge point. Then model the real life case with UML and implement the real life case with Java programming language. After students understand the case, we give students two or three program cases. Students need study the programs by themselves. Then a couple of students are invited to explain the cases with the knowledge point learned.

We take Decorator design pattern, for instance. The intent of Decorator pattern is as the below:

The decorator pattern can be used to extend (decorate) the functionality of a certain object statically, or in some cases at run-time, independently of other instances of the same class, provided some groundwork is done at design time. This is achieved by designing a new decorator class that wraps the original class[10].

In traditional way, we began with the intent when we introduced Decorator pattern. Students felt very difficult to understand the intent and the design of the pattern because the description is highly abstract. Then we improve our teaching approach, and we begin with a real life story about making a pizza. The story is as the below:

Before we learn the Decorator pattern, we can think of making a pizza. When making pizza, you will get a new kind of pizza if you topping pizza with a different combination of toppings. With different combinations, we can get an enormous amount of pizza kinds. It means that a pizza class has enormous amount of subclasses. Each time when we create a new combination, we need create a new subclass. Too many subclasses become a big problem. How to solve this problem? In fact, we can create an original class for pizza base and create a decorator class for each kind of topping. To get different pizza objects, we use different decorator classes to wrap the original class just like we use different combinations of toppings to topping pizza. Then the number of subclasses is decided by the number of toppings instead of the number of topping combinations. The number of subclasses is decreased dramatically. This is achieved by designing decorator classes that wrap the original class.

With the story, we introduce Decorator pattern. The story is interesting and intuitive. With the story, students can catch the intent and design of the pattern easily. Then we introduce the abstract intent from Wiki to deepen students' understanding. After students completely understand the intent, we model the above story with UML class diagram and explain it to students thoroughly. Then we compare the class diagram of the case with the one of the general Decorator pattern to deepen students' understanding of Decorator pattern design. Next, the case class diagram model is implemented with Java programming language. This step can help students to establish a mapping between Decorator pattern and its code implementation. Finally, students need do some practice. Two or three carefully selected examples, which include the design and code implementation, are given to students to study. Later, a couple of students are invited to explain the examples and the teacher can judge if students understand the knowledge point well. After this is done, the teacher supplements and summarizes the lecture.

Table 1 compares the traditional teaching approach and our case driven approach. From the table, we can see that traditional teaching consists of two stages and relies on the teacher's lecturing. The interaction between the teacher and students is mainly question-and-answer. The case driven teaching approach consists of seven stages. More active interactions are performed between the teacher and students. The approach can inspire and cultivate students' cognitive abilities, increase students enthusiasm to learn the course, and deepen students understanding of knowledge points.

Table 1. Comparison of traditional teaching and case driven teaching

No. of stages	**Traditional Teaching Approach**	**Case Driven Teaching Approach**
1	Explain the knowledge point	Tell a real life story to lead to a knowledge point
2	Teacher give a couple of examples	Explain the knowledge point
3		Model the real life story with UML
4		Implement the story model with Java programming language
5		Give students 2 or 3 examples to study
6		Invite a couple of students to explain the above-mentioned examples.
7		Supplement and summarize the lecture

Introduction of Cutting-edge Industry Seminars. To broaden students' view and introduce the newest technology development, we invite experts from leading IT company to give two seminars. Students can not only get first-hand information from leading IT companies but also establish contacts with them. Students can also learn IT industry's expectations for them. For example, cloud computing is a hot technology and excellent architecture, which develops rapidly recently. It's important for students to understand architectures of cloud computing. To make students understand the cutting-edge technology, we invited IBM technical experts to give a seminar on cloud computing. Students expressed their strong interest in the seminar. They asked a lot of questions not only about cloud computing but also about IBM recruitment-related issues.

Assignment Improvement. Because the course is given to the third-year students, it's important to cultivate students' research ability. Besides the original knowledge point practicing assignments, we introduce two assignments about writing research reports on hotspots in this area. Architecture description languages and architecture evaluation methods are two hotspots in this area. We require students to collect, analyze and summarize the materials about the two topics and write research reports. The assignments can not only cultivate students' research ability and the ability to write a scientific report, but also broaden students' scientific view.

Improvement of Grading. Originally, a student's final score consists of three parts (see Table 2). Written exam takes up 50%, and assignments take up 45% and attendance rate 5%.

Table 2. Comparison of original Evaluation and improved one

Original Grading		Improved Grading	
Written exam	50%	Written exam	30%
Assignments	45%	Assignments	45%
Attendance rate	5%	Active participation	20%
		Attendance rate	5%

To grade students' performance more objectively, encourage them to attend classes actively and cultivate their thinking ability, we improve the course grading. Grade of active participation is introduced and the percentage of written exam is decreased.

Results of Course Evaluation

We have done course evaluation, students' satisfaction survey, before and after course improved. After we adopt the above-mentioned teaching model to teach our course, the students' course satisfaction increases explicitly. Fig.1 shows that the average score of lecturing satisfaction was 8.3 before teaching model improved and 9.6 after improved. The average score of content satisfaction was 8.4 before improved and 9.8 after improved. The average score of assignment satisfaction was 8.5 before improved and 9.6 after improved. The average score of grading satisfaction was 8.8 before improved and 9.2 after improved.

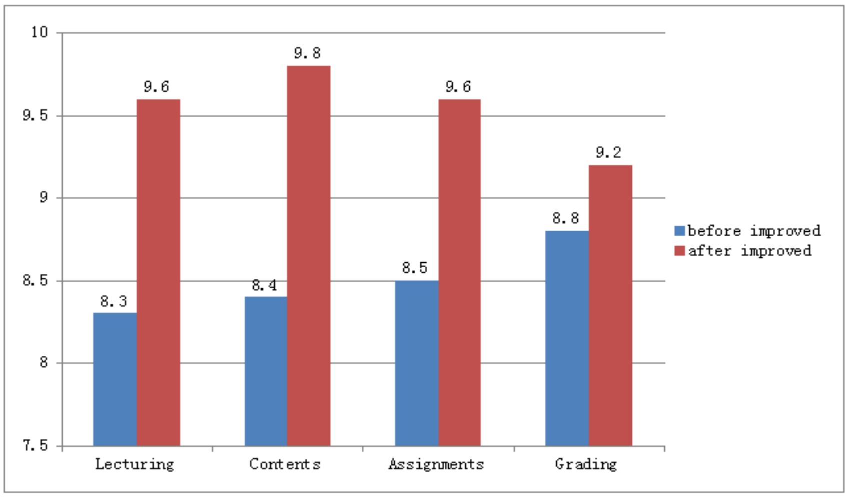

Figure. 1 Comparison of course evaluation results

Conclusion

This paper presents our improved teaching model for Software Architecture course. With the model, we propose case driven teaching approach, introduce cutting-edge industry seminars, and improve course assignments and grading. Course evaluation results prove that students' satisfaction increase explicitly after the teaching model is improved.

References

[1] L. Bass, P. Clements, and R. Kazman. Software Architecture in Practice. Addison Wesley, second edition, 2003.

[2] D. Budgen. Software Design. Addison Wesley, second edition, 2003.

[3] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. A System of Patterns. John Wiley & Sons, 1996.

[4] P. Clements, F. Bachman, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. Documenting Software Architectures: Views and Beyond. Addison Wesley, 2003.

[5] Mary Shaw, Hans Van Vliet,Software Architecture Education Session Report, in WICSA'05: Proceedings of Working IEEE/IFIP Conference on Software Architecture - WICSA , pp. 185-190, 2005.

[6] Remco C. De Boer, Rik Farenhorst, Hans Van Vliet, A Community of Learners Approach to Software Architecture Education,in CSEE&T'09: Proceedings of Conference on Software Engineering Education and Training, pp. 190-197, 2009.

[7] Patricia Lago, Hans van Vliet, Teaching a Course on Software Architecture, in CSEE&T'05: Proceedings of Conference on Software Engineering Education and Training, pp. 35-42, 2005.

[8] Information on http://en.wikipedia.org/wiki/SOLID

[9] Information on http://en.wikipedia.org/wiki/Package_principles

[10] Information on http://en.wikipedia.org/wiki/Decorator_pattern