Learning by Doing: Practice and Experience in the Teaching of Master of Software Engineering Students*

LI Wen-jun, CHAO Hong-yang, CHANG Hui-you
(Software School, Sun Yat-sen University, Guangzhou 510275, China)

Abstract: Software industry oriented education faces more challenges than traditional research and discipline oriented education. To meet these challenges, the Learning by Doing teaching pattern plays an important role in the software industry oriented education. Practice and experience in the teaching of Master of Software Engineering (MSE) students in the Software School of Sun Yat-sen University are presented in the paper. From our viewpoint of MSE education, the success comes from a good trade-off between Learning by Doing lectures and theoretical lectures. Furthermore, feedback and communication are critical and requisite for the successful application of Learning by Doing teaching pattern.

Key words: Learning by doing; software industry oriented education; MSE teaching
CLC number: TP31 Document code: A Article ID: 0529-6579 (2007) S2-0156-05

1 Introduction

Established in 2001 as one of the first 35 demonstrative software schools approved by the National Ministry of Education, the Software School of Sun Yat-sen University has educated more than 1,000 Master of Software Engineering (MSE) students, including about 300 part-time MSE students. After graduation, these students have a strong record of attaining employment and income levels well above average. In 2006, a team of MSE students from the Software School of Sun Yat-sen University won the top winner of the 2006 "IBM Cup" of Chinese Higher Education SOA Application Contest. In this half-year long contest concerning the new theory, technology and tools for Service-Oriented Architecture (SOA) applications, they defeated about 400 competitors including teams of undergraduate, postgraduate and even Ph. D candidate students from both Computer Science and Software Engineering.

The success and achievements of the students are the natural evolution of new education patterns applied in the Software School, a software industry oriented education that educates students for application, engineering and multi-orientations. To meet the new challenges of software industry oriented education, the Learning by Doing teaching pattern advocated by Roger C. Schank[1, 2] plays an important role in the education practice in the Software School of Sun Yat-sen University.

In this paper, the practice and experience of the teaching of MSE students in the Software School of Sun Yat-sen University are presented. The rest of the paper is organized as follows: Section 2 discusses the challenges and opportunities for software industry oriented education. Section 3 and 4 introduce the MSE teaching practice in the Software School of Sun Yat-sen University and present the experiences we have learned. Finally, Section 5 concludes the paper and outlines directions for future work.

2 Challenges and Opportunities for Software Industry Oriented Education

Software industry oriented education faces more challenges than traditional research and discipline oriented education. Since there are only five years of experiences of demonstrative software school in China, the research of teaching for application, engineering and multi-orientations is still immature. These challenges come from students, instructors and tutors, and the software industry. But the challenges are also accompanied with many opportunities.

2.1 Challenges from Students

As a young discipline, the recruited students of software school are not quite as well as students majoring computer science and engineering, both at the undergraduate level and the postgraduate (MSE) level.

For MSE students, they come from different un-

*收稿日期: 2007-10-28
作者简介: 李文军 (1966年生), 男, 教授; E-mail: lnslwj@mail.sysu.edu.cn

dergraduate background. More than 35% MSE students in the Software School of Sun Yat-sen University majored in a bachelor degree out of computer science and engineering discipline. Many of them lack of a systematic and substantial training as a computer scientist or a software engineer.

But usually these students are more diligent and assiduous due to the poor discipline backgrounds and aspiration for expectative career lives. The various discipline backgrounds also make it easy to be educated for multi-orientations. For example, the top winner team in the SOA Application Contest consists of five students, three of them have strong backgrounds of computer science and engineering, while the other two of them comes from Electrical Engineering and Management Engineering background.

2.2 Challenges from Instructors and Tutors

The teachers of Software School come from both academic and software industry background. This will lead the advantage over traditional academic education of computer science and engineering. But a teacher with an academic background is usually acquaintance with traditional research oriented teaching. He is good at the training for academic research and theoretical foundations. But he may be strange to engineering and application education due to the lack of practical engineering experience.

On the other hand, an instructor or tutor with a software industry background is accomplished in engineering and applications. But sometimes he may not be equipped with sufficient teaching skills and systematic knowledge body.

Much effort should be put into the communication and intercourse between the instructors and tutors from different background, so as to make these two complemental teaching resources benefit from each other.

2.3 Challenges from Software Industry

The software industry is an industry with flexible and volatile theories, methodologies, technologies and tools. Students are required to master the up to date software tools and development environments to solve real world problems. Meanwhile, they have to accumulate theoretical foundations to have a comprehensive understanding of the theories, principles and patterns behind those tools and development environments.

Furthermore, multi-orientations graduated students majoring in software engineering along with different backgrounds, such as enterprise management, financial management, electrical engineering and applied mathematics, etc., have a good reputation in the human resource market.

Software industry oriented education has to keep the balance between theoretical foundations and engineering applications. This is one of the major differences between software school and career or vocational education.

3 Practice in the Software School of Sun Yat-sen University

The Software School of Sun Yat-sen University aims to educate high level software engineering human resources for the social and economic development of the Pan-Pearl River Delta region.

3.1 Motivation

To educate MSE students for application, engineering and multi-orientations, each student of the Software School of Sun Yat-sen University will cultivate 3 accomplishments, 4 levels and 6 abilities.

(1) The three accomplishments are Innovation, Collaboration (Team Player) and Professional Dedication. These accomplishments are considered as the basic elements for a successful software engineering professional.

(2) The Four Levels indicate various knowledge and skills mastered by MSE students, including Senior Programmer, System Analyser and Designer, IT Project Manager, and IT Enterprise Strategy Decision Maker.

(3) The Six Abilities cover the ability to problem analysis and solving, the ability to communication and collaboration, the ability to initiative knowledge acquirement, the ability to effective project management, the ability to practical engineering, and the ability to international competition.

3.2 Curriculum Design

There are two years of education for full-time MSE students. In the first year, students concentrate on intensive course study. In the second year, they dedicate to practice in the software industry and writing master thesis. The education for part-time MSE students will last for three years.

The required subjects for MSE students cover the fundamental engineering and management theories, technologies and skills, including: Theoretical Foundations of Programs, Object-Oriented Technology and Method, System Analysis and Design, Distributed Computing, Software Architecture, and Software Project Management.

Students are also required to study some of the following elective subjects: CMM in Practice, Modern Network Protocols and Programming, Advanced Communication Technology, Data Warehouse and Data Mining, Information Security, Embedded Systems and Applications, Computer Animation and Game Program-

ming, Information Engineering Supervision, and Lectures on Innovative Software Technologies.

Differed from traditional lecture and test based teaching, most of the MSE courses incorporate the Learning by Doing teaching pattern. But to balance the theoretical foundations and practical engineering applications, a good trade-off must be done for different courses. The following sections present two typical courses in the practice of MSE teaching in the Software School of Sun Yat-sen University.

3.3 Object-Oriented Technology and Method

Object-orientation is the predominant technology and methodology for contemporary software development. Based on the Java platform, the course Object-Oriented Technology and Method introduces the fundamental concepts and different aspects of the theory, methodology, technology and tools of object-oriented development.

The course is divided into three parts. The first part introduces basic concepts of object-oriented programming and design, including type system, structured control structures, objects and classes, encapsulation and information hiding, data abstraction, inheritance, polymorphism (including overloading, generics and runtime polymorphism), inner class and package, assertion and exception handling, and object containers.

The second part focuses on practical application related subjects, such as input and output streams, object persistence and O/R mapping, Graphical User Interface (GUI) and event-driven programming, multithreading with different thread models, and networking over various protocols.

While some design patterns emerge in the second part (For example, the Decorator pattern in I/O streams, the Iterator pattern in the collection framework, the Composite and Observer patterns in AWT/Swing), the third part present object-oriented design principles and design patterns in detail. For full-time students, some innovative technologies are also appended in the course, such as Aspect-Oriented Programming (AOP), Application Framework, or Object-Oriented Refactoring.

According to the Learning by Doing paradigm, four or five mini projects are designed step-by-step to facilitate the digest of lecture knowledge and to promote the practical ability. For example,

(1) A simple personal income tax calculation project is conducted to demonstrate the difference between traditional structured programming and object-oriented programming. Students will also learn the following knowledge and skills by doing: coding conventions (including the documentation style annotation, proper naming of identifiers, indention of source codes, etc.), data abstraction, encapsulation and information hiding, and simple business modeling based on UML.

(2) An agenda service program with character-based command line user interface is used to illustrate the separation of the presentation tier and the business logic tier, the abstraction of object classes, the application of Single Choice Principle to achieve representation independence using polymorphism, the deployment of the Abstract Factory pattern to manage future changes, and the usage of exception mechanism to handle exceptional conditions.

(3) An on-line testing system based on an item library along with GUI and Rational Database (RDB) emphasis on event-driven programming, object persistence and object relation mapping (O/R mapping), the concept of programming in the large, and the 3-tier or n-tier design of software architecture.

(4) Case study on a real-world open source object-oriented project, such as JHotDraw or various Sudoku game programs. Students are required to inspect the documents and source codes of the projects, and then propose pros and cons of the design and coding convention of these projects.

(5) Refactoring an object-oriented program (such as various open source Sudoku game programs) based on an object-oriented refactoring approach or even on an aspect-oriented refactoring approach.

During the lab-time, inter-student communication is encouraged thoroughly in the same laboratory. Communication between the instructor and students will be compensated with the help of the Internet (website, blog, email and BBS). A discussion lecture-time will be arranged for each project, in which the instructor and students will propose and discuss different software quality factors of several typical experimental reports.

Since this subject is the fundamental course for MSE students with weak discipline background of computer science and engineering, more lecture-time has been arranged to teach theoretical foundations. The ratio of lab-time and lecture-time is about 1:4. Students must spend a lot of time on the projects after school.

3.4 Distributed Computing

Distributed computing plays an important role in the software industry and dominates the enterprise application development. The subject Distributed Computing in the Software School of Sun Yat-sen University covers different levels of distributed computing, including socked-based networking with TCP/UDP and other Internet protocols, distributed objects with RMI and CORBA, component models with J2EE/EJB, and

Service-Oriented Architecture (SOA) based on Web Services and Grid Services.

Each level of distributed computing is studied in a Learning by Doing paradigm. After the instructor present the theoretical aspect of the technology, students learn the detailed knowledge by conducting a mini project. The ratio of lab-time and lecture-time is about 2:1. The instructor will spend one lab-time to discuss the design and implementation of the project in the laboratory, and spend another lab-time to present and discuss typical experimental reports with students.

(1) For socket-based networking, a mini project of the construction of an HTTP 1.0 server and the design and implementation of a simple HTTP client will help the students learn the TCP protocol and use it to implement the HTTP protocol. Different thread models are also discussed in this project, including thread-per-request, thread-per-connection, and thread pooling. An alternation of the project is to shift from the design and implementation of a HTTP client and server to a FTP client and server.

(2) For distributed objects, a simple pilot project conducted on the Java RMI platform will help students accommodate to the design of the interfaces of distributed objects and the implementation of distributed objects. The project may be an evolution of a project in the Object-Oriented Technology and Method course, but re-developed in a distributed approach.

(3) A further project will be done on the CORBA platform to demonstrate the independence of programming languages based on the interfaces of distributed objects defined by the OMG Interface Definition Language (IDL), the design of QoS policies of Portable Object Adapter (POA) to manage objects on the server side, the application of Dynamic Invocation Interface (DII) to send a request dynamically from the client side[3].

(4) In the fourth project, students are asked to implement a special type of resource pooling, i.e. instance pooling, and evaluate the performance of the implemented pooling mechanism in an empirical approach. From this project, students not only learn how to design and implement a resource pooling mechanism, but also study how to conduct an empirical experiment and collect and analyse experimental data.

(5) For learning distributed component models, students are asked to re-design and implement the second project based on J2EE/EJB platform. In this project, students can experience the difference between distributed objects and component models, and master the common services provided by the component container.

(6) Case study on an existing SOA solution, from which students can discuss the problems and solutions in Enterprise Application Integration (EAI) and e-commerce/e-government applications.

Since the condensed lecture-time concentrate on the theoretical aspects and the most important technology, students have to learn related details by themselves. For example, the instructor only present the concept of Quality of Service (QoS), and the principles and architectures of POA, concerning no details on the programming of POA. Meta-Learning (Learn How to Learn)[4] is an importance accompanist of Learning by Doing. To promote Meta-Learning, the instructor should provide sufficient hints and materials to students.

4 Experiences Learned from the Teaching of MSE Students

In the five years of teaching of MSE students in the Software School of Sun Yat-sen University, some experiences have been learned from the practice of Learning be Doing in software industry oriented education.

4.1 Lecture-Time vs. Lab-Time

Facing the challenges from the software industry, software industry oriented education should keep the balance between theoretical foundations and engineering applications. This is a balance between something stable and something volatile. From our experiences, the success comes from a good trade-off between lecture-time and lab-time in different subjects.

For example, as the first subject for MSE students, Object-Oriented Technopgy and Method is specially important but difficult for those students without undergraduate background of computer science and engineering. Thus the lecture-time should be scheduled more then the lab-time considering students weak Meta-Learning ability.

When the Distributed Computing course is arranged in the next semester, students have strong foundations to conduct Meta-Learning. It is possible for the lab-time to be scheduled much longer than the lecture-time.

4.2 Project Design

Elaborately designed projects are critical to the application of the Learning by Doing teaching pattern. Instructors should put more effort into the design of projects than the preparation of lecture notes.

Each stage or step of the project must be clear and detailed enough for students to perform the practice. If it is possible, milestones should be set for the assess-

ment of each stage or step.

In the Learning by Doing pattern for software industry oriented education, doing is much more than programming. For elaborated projects, the approach of Learning by Doing may cover the practice of analysis and design, testing, empirical experiment, project management, documentation, discussion and argument, presentation and communication, etc.

4.3 Challenges for Instructors

The lab-time conducting Learning by Doing has more challenges than lecture-time for instructors. After the design of an elaborated project, the instructor must prepare carefully the possible problems and solutions arising during the practice.

Due to the flexibility and volatileness of the practical procedure, the instructor is required to be high professional and full accomplished.

4.4 Feedback in Learning by Doing

One of the requisite effort for the successful application of Learning by Doing is the feedback of each project. Without feedback, students may miss the critical knowledge when learning by doing, or even be lost in the practical technology details.

Students should have the opportunity to get the feedback from both the instructors and their classmates. During the practice hours, students have the opportunity to get feedback from the discussion between students and instructors. After the practice, some typical artifacts are asked to be presented in the lecture-time and discussed and argued by the instructor and other students. It is encouraged that typical artifacts will be put into the course website (For example[5]) with discussions and arguments by the instructor and other students.

4.5 Communication in Learning by Doing

Communication is another critical factor of the success of the application of Learning by Doing. From our experience, inter-student communication is more important than the communication between the instructor and students. Since the condensation of lecture-time, inter-student communication becomes an important source of Meta-Learning. Software School administrators and instructors should construct a good environment for inter-student communication. For example, grouping students or pair programming with different foundations and skills, arrangement of laboratory environment to facilitate intercourse, and policies to encourage inter-student communication.

Internet can be utilized to compensate the lack of instructor-student communication, including website, blog, email and BBS. A comprehensive teaching website[5] consists of course news, syllabus and lecture notes, abundant learning materials, experimental requirements and tools, comments on artifacts done by students, Frequently Asked Questions (FAQs), related state of the art technologies, and useful links.

5 Conclusions

In this paper, the practice and experience of the teaching of MSE students in the Software School of Sun Yat-sen University are presented. The Learning by Doing teaching pattern applied in the two typical software engineering course Object-Oriented Technology and Method and Distributed Computing is introduced in detail.

From our experience of the application of Learning by Doing, it is important for the instructors to decide a good trade-off and consequence between lecture-time and lab-time for different subjects. Feedback and communication also play critical roles in the successful application of Learning by Doing. This new teaching paradigm is a real challenge for instructors.

For further and sustainable application of Learning by Doing in the MSE education, a repository of practical projects are to be built by professionals from both academic and industrial background. The repository will consist of elaborated projects, including clear and detailed steps, reference artifacts, possible questions and solutions, etc. It is also expected that multiple software schools will collaborate to develop the repository.

References:

[1] SCHANK E C. Goal-Based Scenarios: Case-Based Reasoning Meets Learning by Doing[M] // David Leake. (ed), Case-Based Reasoning: Experiences, Lessons & Future Directions. AAAI Press/The MIT Press, 1996: 295—347.

[2] SCHANK E C. Revolutionizing the Traditional Classroom Course[J]. Communications of the ACM, 2001, 44(12): 21—24.

[3] LI W J, ZHOU X C, LI S X. Distributed Objects Technology[M]. Beijing: China Machine Press, January 2004.

[4] iCarnegie: Software Development Curriculum Powered by Carnegie Mellon[EB/OL]. http://www.icarnegie.com/

[5] Homepage of the Object-Oriented Technology and Method course website[EB/OL], Software School, Sun Yat-sen University. http://www.cs.sysu.edu.cn/~lwj/object-oriented/