INFORMATION
SCIENCE
TECHNICAL
REPORT

NAIST-IS-TR2007009
ISSN 0919-9527

An Experiment to Explore Practical Training for Object-Oriented Design Novices

Izuru Kume, Naoya Nitta and Yasuhiro Takemura

May 2007

NAIST

〒 630-0192
奈良県生駒市高山町 8916-5
奈良先端科学技術大学院大学
情報科学研究科

Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0192, Japan

Technical Report

An Experiment to Explore Practical Training for Object-Oriented Design Novices

Izuru Kume, Naoya Nitta and Yasuhiro Takemura

kume@is.naist.jp, n-nitta@konan-u.ac.jp, takemura@mxw.mesh.ne.jp

May 7, 2007

Graduate School of Information Science
Nara Institute of Science and Technology

Abstract

Project-based training is effective for practical software engineering education. Our goal is to establish a practical training method in software engineering education that is easier to be introduced by higher education institutions with limited human resources and budget, and gives similar educational effects to real software projects. In our past research, we proposed a framework of practical training using real scale open source software to teach students who have knowledge about object-oriented design techniques but no experience to apply their knowledge in practice. In this paper, we try to apply this framework to teach students who have knowledge about object-oriented programming but don't understand object-oriented design. We have developed a practical training process for such students and tried it for several students. In this paper, we show the experimental result of the trial.

1 Introduction

Recently many higher education institutes tend to pursue practical software engineering education. Behind the tendency there is a reflection that traditional computer science education focuses too much on theory and little on practical software engineering [5]. Our research goal is to establish a practical software engineering education method that is easy for many higher education institutes with limited human resources and budget to introduce. In this paper we propose an approach of practical training to effectively motivate students who are novice at software design to study object-oriented design. We also explain an experimental result that suggests the educational effects of our approach.

Teaching object-oriented design in higher education institutes seems not so successful compared to object-oriented programming. In fact, there is a shocking research result that only 9% of graduating students can make a " reasonable " design [2]. Nowadays, many higher education institutes introduce project-based software engineering courses which will naturally include practice of software design [4, 1]. Suitable teaching materials well explain design concepts and principles such as encapsulation [6]. However, the current education methods assume a certain level of students' skill [1] or a tight connection to particular companies which support educational software projects [4]. Otherwise, they sacrifice the practical reality for students' understanding [6]. Many higher education institutes need a means of teaching design novices without any special connection to supporting companies.

In this paper, we propose an approach of one-day practical training for design novices to understand the importance of design adequacy through their practice in software maintenance. We design the training process so that we can intuitively compare an inadequate design with adequate one. We expect that the comparison makes them recognize the importance of software design and give motivation to study object-oriented design. In our past paper [3], we proposed a framework of practical training that enables students learn from their own failure. In this paper we apply the framework to the design of our practical training for design novices. In the rest of this paper, we explain the practical training frame work for design education in section 2. In section 3, we explain a practical training approach to motivate design novices to teach object-oriented software design. We design the

training by applying the framework. In section 4, we show an experimental result to examine the educational effects of our approach.

2 Framework of Practical Training

2.1 Aims

The framework in our past paper [3] aims at designing a practical training in a style of " learn by a failure experience in software design ". There are three requirements for a successful training in this style. First, students should understand that their failure would introduce a serious trouble in a real project. Second, they should understand that they could not cope with the trouble in any ad hoc means. Third, it should be possible for the students to evaluate the adequacy of their design or compare it with adequate one. We can satisfy the first and second requirements by a teaching material in a practical scale. Our framework proposes an idea to edit a teaching material from a practical scale of open source software. To satisfy the last requirement is not so easy because students often lack of the ability to compare or evaluate design adequacy. Our framework proposes a common process model to tackle this problem.

2.2 Editing Teaching Materials

Students, who have never experienced a practical scale of software project, tend to believe that they can manage the whole process of software development [1]. Therefore, we need our teaching material to have certain degree of complexity and size. However, excess in complexity and size might it impossible for students' practice. We should balance its complexity and size at an appropriate size. In the past paper [3] we proposed an idea to help teachers edit a teaching material from an existing open source software product in a practical complexity and scale. The key of our idea is to introduce annotation tags that denote design rationale or principles. We assume that annotators understand the original software well enough for its annotation. It is still an open problem how to make annotations of unknown software.

2.3 Three Learning Steps

As we explained in subsection 2.1, making students compare or evaluate software designs need a trick. Less experienced developers often can't evaluate their design,

while they can evaluate it when they experience a trouble at a later maintenance activity, especially if the trouble could be prevented by a more adequate design. Our idea is to replicate a failure scenario upon our training process. Because such a scenario is a little complicated, the process model should separate the step for obtaining a design from the one for its evaluation.

In order to reflect the above idea, the training process contains three steps: (1) briefing, (2) trial, and (3) evaluation. Often, the trial step represents a design activity, and the evaluation step represents a later maintenance activity. At the briefing step, students are given all necessary information to follow a training scenario. At the trial step they obtain a design to be evaluated. At the evaluation step, they practice the " later activity " in the scenario. They can then evaluate the design through the result of their practice. Originally, we got the idea in considering a training approach for those students who have learned object-oriented design techniques but lack of their application in a real software development project [3]. The training aimed at cultivating the students' ability to design with foresight. The process model is also useful for the aim of this paper in spite of the difference in the groups of assumed students and in the educational aims.

3 Practical Training for Object-Oriented Design Novices

3.1 Assumed Students and Aim

In this paper we study an approach of practical training to motivate novices at software design to study object-oriented software design. We have the following assumptions about students: (1) Students have learned object-oriented programming language(s) but have no or little experiences to use them in practice. (2) They have never learned software design. (3) They may be able to have implicit object-oriented models of object-oriented programs in their mind, but have no ability to describe it as model artifacts. Our aim is to make such students understand that adequacy of software design influences the amount of required efforts at later maintenance activities. Unlike the past case in [3], we don't have to teach how a design influences the activities.

3.2 Requirements

Compared to the students we assumed in the past paper [3], our students have less experience in object-oriented programming. In addition, they have no experience of object-oriented design. From these assumptions it follows that we should not expect their ability of object-oriented modeling. Hence we should avoid using abstract and conceptual modeling terms alone in the practical training. Instead we should explain necessary design concepts, the influence of designs on later maintenance efforts in terms of the program code included in the teaching material.

One of the main reasons why the students have never studied software design is that they have never felt its necessity in their programming experiences. From our experiences, those students who learn programming by themselves often write small programs with one or little number of functions. Small programs require no design to develop. In addition, they seldom maintain their programs. Therefore, we can motivate the students to study software design through a practice of developing a large program or maintaining an existing software product. We select the latter from the consideration of the students' programming experiences

and skills.

3.3 The Teaching Material

We edit a teaching material from an existing network game system in whose users compete for their typing skills each other through the internet. The system is developed in the laboratory of one of the authors for an educational purpose. It has a so-called client-server architecture using Servlet technology. The server part is written in Java while the client part in JavaScript. In addition, the whole product includes informal documents of the requirement specifications and graphics to be displayed. As is usual in Web applications, it has simple software architecture which helps us understand by relating the server side behaviors to the client side displays. Therefore, it is easy to understand and suitable to use as our teaching material of software maintenance.

We use the Java source code and the informal specification documents for our teaching material. The code has about 2 thousands of LOC, a suitable size for the task by the students. We add design artifacts to the material so that our students can understand the source code during the limited period of the trial step. The artifacts include class diagrams and use case diagrams, and textual explanations of the Java classes. The teaching material also includes four extended versions of the original game system (Figure 1). Two versions, say [A1] and [B1], extend the system by introducing the same new play mode. The former [A1] is redesigned in an ad hoc fashion, while the latter [B2] in an adequate fashion by taking advantage of abstraction by a common super class. Other two versions, [A2] and [B2], further extend [A1] and [B1] respectively by adding more play modes. The difference of design adequacy between [A1] and [B1] becomes apparent when we compare the additional efforts of the further extension to obtain [A2] and [B2] respectively.

3.4 The Training Process

The training process aims at helping the students to make relative evaluation of [A1] and [B1]. As we explained in section 2.1, it is difficult for design novices to relatively evaluate the two designs by themselves. However, they can easily

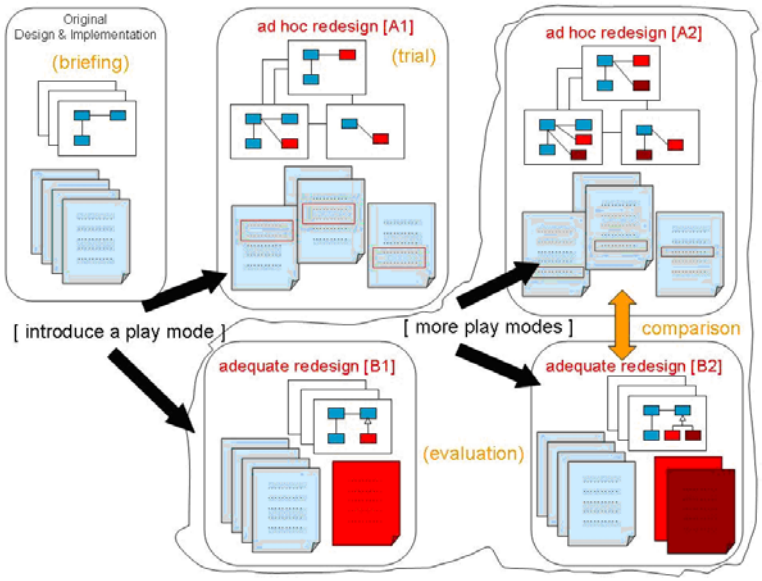

Figure 1: Maintenance Scenario

compare the efforts for further changes. The comparison demonstrates the effects of the two designs. Therefore, we should design the training process to effectively compare of the efforts. At the same time, we should take account of the fact that the students are new to object-oriented software design. In the training process, we show the students the five designs, the original design, and its four extensions, step by step in the software maintenance scenario instead of showing all at once.

At the briefing step we take time to explain the original game system. In addition we actually run the system to explain the server side behaviors. We also teach the students the meaning of UML class diagrams for the sake of UML novices. At the trial step, we explain the first specification change to add to the game system a play mode which allows users to train their typing skill instead of a real competition. We show [A1] as an example of re-design and its implementation. Students try to understand the design and to make sure of the implementation of the new mode. We don't expect that the students can evaluate the given design [A1]. It suffices if they can perform further re-design by themselves by following [A1] at the next evaluation step.

At the evaluation step, we explain further addition of another play mode to the students, and assign them a task to re-design and re-implement [A1]. The

redesign is not so difficult for the design novices because they follow the former example [A1]. Then we ask the students to count the number of program code lines required for the re-implementation. In order to save the training time, we give the students [A2] as a " complete version " of further extension of [A1], and make them examine the changes in its program code. We expect that through the examination the students can accurately estimate the maintenance efforts which they should have done to add the new mode. We also expect that the students feel as if it would be the estimation of their own maintenance task.

After that we show the students [B1] and [B2] as an alternative of [A1] and [A2] respectively. The version [B1] introduces a proper class hierarchy in order to abstract out the variations of play modes. The design makes it easy to add any number of play modes without changing any of the existing code. In fact, we can accomplish the extension of [B1] to [B2] only adding a class to implement the second play mode. The students also examine the changes in the source code from [B1] to [B2]. We expect that they can intuitively understand that extending [B1] to [B2] is much easier than [A1] to [A2], thus they can understand the important role of object-oriented design in software maintenance.

4 Experimental Result

4.1 Aim and Expectation

To validate the educational effects of our approach, we aim to examine the teaching material and the process of our practical training by questionnaires. As for the educational effects, we expected that students can compare the extension efforts from [A1] to [A2] to those from [B1] to [B2]. We expected, however, that they can't explain why the difference arises because they did not learn the theoretical concepts required for the explanation. We are interested in how difficult the students feel for the tasks given in the training process in order to arrange the balance between the reality of the training and the lack of students' competence. We are also interested in the improvement of students' motivation and understanding during the training process.

4.2 Questionnaire and Process

We prepared three questionnaire sheets and ask the students in training to fill them at different steps in the training process. Several questions are repeated in the questionnaire sheets so that we can grasp the progress of students' motivation and understanding. All questions are divided into two categories. The questions in the first category examine students' understanding. The questions in this category are about the game system, the training scenario, and the general object-oriented design concepts. The questions in the second category examine students' attitude and motivation. There are several questions for other purposes than that of this paper. We don't mention about them in the following. After the end of the experiment, we also had a free discussion.

Five students participate in the experiment as its subjects. They attended lectures of C programming and Java Programming. One of them experiences Java programming. Four of them attended a lecture of UML. At the beginning of training, we asked the students to fill the first questionnaire sheet. Most of its questions are preliminary and are not mentioned except one question that asks them their knowledge and skill of object-oriented software design. We went on to the briefing step, and then to the trial step. At this point, we asked them to

fill the second sheet in order to ask: (1) the students' understanding about the game system and about the maintenance scenario to introduce a play mode, (2) the clarity of our explanation, and (3) their motivation. Then we went on to the evaluation step. Just after the completion of this step we asked the students to fill the third questionnaire sheet. The last sheet contains the same questions to the second sheet and has additional questions about the further changes in the design and implementation of the game system. Then all the participants had a free discussion in a relaxed atmosphere.

Table 1: Result of the first questionnaire sheet

Question	Average	Standard Deviation
1. Do you enjoy observing the runtime result of the program you make?	4.4	0.5
2. Do you need more effort to study object-oriented design?	4.6	0.5
3. Do you have an unambiguous vision of your goal of your study?	2.2	0.4
4. Are your teachers and class mates friendly?	4.2	0.4
5. Is it important for you to study object-oriented software?	4.0	0.9
6. Do you know about object-oriented design well, and do you have asufficient skill of object-oriented design?	1.2	0.4
7. Do you want to study more about object-oriented desin, and to improve your skill of object-oriented design?	4.8	0.4

4.3 Results of Questionnaires

In this subsection, we show the results of the questionnaires. First, we compare in Figure 2 the results of the questions to the second and the third questionnaire sheets. In the following, we refer to the number of the question in the second questionnaire sheet for each question. As for the questions about understanding, questions of number 1, 3, and 4 ask about the specification, architecture, and class responsibility respectively. The rest questions ask about details of the design and

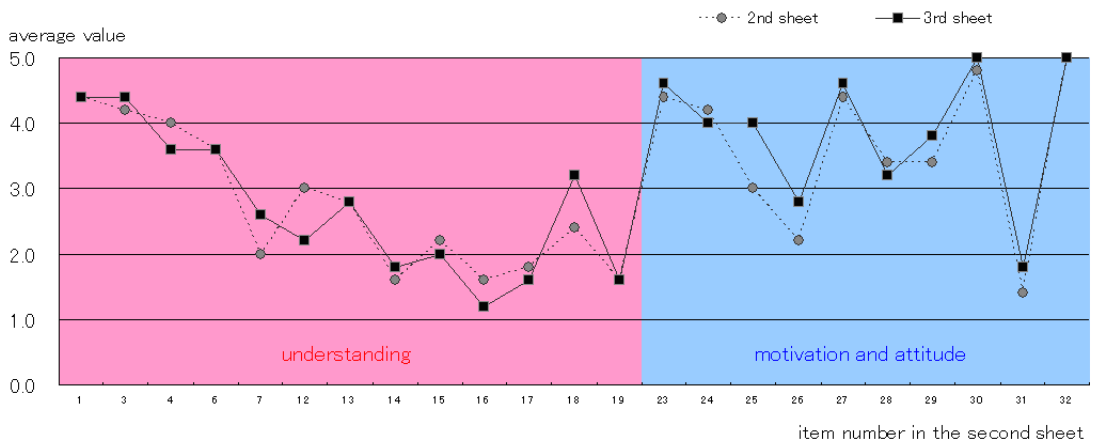

Figure 2: Comparison between the result of the 2nd sheet and that of the 3rd sheet

implementation of the system. In particular, number 18 and 19 ask about the design in terms of general design concepts. From the answers, we can see that the students understood the system in terms of its specification, architecture, and individual classes, but not the design of class clusters and the implementation. The training has no significant effects in this point.

As for the category of motivation and attitude, question number 24 asks the students' estimation of maintenance efforts to change a program code in general. The result shows that they averagely recognize its difficulty. However, that the result has not improved through the evaluation step is contrary to our expectation. We will discuss this issue in the next section. Question 23 asks about the clarity of our explanation in the whole training process. The result guarantees that the students really understand the maintenance scenario. Questions 25 and 27 ask if the training help the students understand the importance of software design and the importance to explain software design respectively. Question 26 asks if the training helps them understand the concept of information hiding. Since all of these results have improved, we can expect that the training was useful to make them recognize the importance of software design but it has little effects to understand design principles, as we expected.

Question 28 and 29 ask about the students' attitude toward this training pro-

cess. The results don't show their significantly positive attitude. From the comments in the free discussion, we guess that they were so tired by the training process and by filling up the long questionnaire sheets. Question 30 and 32 ask about their motivation to study object-oriented software design and to increase their skill, respectively. Combined the results with those of questions 25 and 27, we can conclude that our training process improve their motivation to learn object-oriented software design. Question 31 asks about their self confidence in their knowledge and skill of object-oriented design. The result shows the lack of their self confidence.

There are significant improvement in questions 18 and 25. Question 18 asks the students if they understand that a design process includes decision about data representation. The result is not so positive but significantly improves, because the students actually compared two ways of data representation in [A2] and [B2], we guess. The significant increase in question 25 shows the effect of the evaluation step to make the students recognize the importance in software maintenance, we guess.

Table 2: Result of the second questionnaire sheet

Question	Average	Standard Deviation
1. Do you understand the purpose of the original software system? Describe the purpose in brief.	4.4	0.5
2. Count the number of the major functions of the original system, and give a brief explanation for each of the functions.		
3. Do you understand the collaboration between browsers and the server in the original system?	4.2	0.4
4. Can you explain the responsibility of the class DataOfBeginningToCombat in the original sytem? Explain the responsibility briefly.	4.0	1.1
5. Count the number of major classes in the original system, and give a brief explnation for each of the functions.		
6. Do you understand the timing in the collaboration to call the method DataOfNextWord.doPost() in the original system? Explain it briefly.	3.6	1.4
7. Do you understand the runtime process of the method call "CombatManager.getInstance() .setYourHPAndGetMyHP(mySessionID, nYourHP);" in DataOfNextWord.doPost() in the original system? Explain the process briefly.	2.0	1.1
8. Do you understand that the task just shown aims at extending the system's function?	4.0	1.1
9. Can you explain the additional function in order to introduce the concept of "play mode"? Explain the function briefly.	4.0	1.1
10. omitted.		
11. How many classes in the original system do we have to change their implementation according to the extension to [A1]?		
12. Do you understand the kinds of the data passed during the collaboration between DataOfBiginningToCombat and CombatManager in the original system and [A1]? Explain them briefly.	3.0	1.3

Question	Average	Standard Deviation
13. Do you understand the process for DataOfBeginningToCombat in [A1] to obtain usrs current play mode? Explain the process briefly?	2.8	0.4
14. As for Question 13, do you understand the representation of the play mode obtained by DataOfBeginningToCombat? Explain its representation briefly.	1.6	0.8
15. Do you understand how the play modes are used in the program code of [A1]?	2.2	1.2
16. Do you understand the program statements which represent the usage of play mode data in DataOfBeginningToCombat in [A1]? Point out the statements.	1.6	1.2
17. Do you have any idea about in which way you present a play mode in [A1]? Explain the presentation briefly.	1.8	1.6
18. Do you understand that software design includes decision about presentation of data.	2.4	1.1
19. Do you understand which matters are included in software design except for presentation of data.	1.6	0.8
20. Do you grasp the cases to apply hiding to classes? Point out the classes to which encapsulation is applied.	1.2	0.4
21. Can you guess what kind of future changes on the original game system will become easier by encapsulation? Explain it briefly.	2.4	1.0
22. Can you point out where in the original system you should apply encapsulation? Point out where.		
23. Were the explanations given by the teacher clear?	4.4	0.5
24. Do you think that changing a program code in general is difficult?	4.2	1.0
25. Is this training useful for you to understand the importance of program design?	3.0	0.0
26. Is this training useful for you to understand encapsulation?	2.2	0.7
27. Does this training make you recognize the importance of explanations of program design?	4.4	0.8

Question	Average	Standard Deviation
28. Do you want to participate this training again in future?	3.4	0.5
29. Do you eager to participate in this training.	3.4	1.0
30. Do you think it is important for you to study object-oriented design?	4.8	0.4
31. Do you know object-oriented design well or do you have a good skill of object-oriented design?	1.4	0.5
32. Do you want to improve your understanding and skill of object-oriented design?	5.0	0.0

Table 3: Result of the third questionnaire sheet

Question	Average	Standard Deviation
1. Do you understand the purpose of the original software system? Describe the purpose in brief.	4.4	0.5
2. Count the number of the major functions of the original system, and give a brief explanation for each of the functions.		
3. Do you understand the collaboration between browsers and the server in the original system?	4.4	0.5
4. Can you explain the responsibility of the class DataOfBeginningToCombat in the original sytem? Explain the responsibility briefly.	3.6	1.0
5. Count the number of major classes in the original system, and give a brief explnation for each of the functions.		
6. Do you understand the timing in the collaboration to call the method DataOfNextWord.doPost() in the original system? Explain it briefly.	3.6	1.4
7. Do you understand the runtime process of the method call "CombatManager.getInstance() .setYourHPAndGetMyHP(mySessionID, nYourHP);" in DataOfNextWord.doPost() in the original system? Explain the process briefly.	2.6	1.4
8. One or more classes are added at the first extension version [B1]. Do you understand their responsibilities? Explain them briefly.	3.6	0.5
9. One or more classes are added in [B1]. Do you understand their responsibilities? Explain them briefly.	3.8	0.4
10. How many classes in the original system do we have to change their implementation according to the extension to [B1] to [B2]?		
11. Did you expect that you would need some changes on data representation in DataOfBeginningToCombat in [A1] when you would further introduce the "competition mode"? Answer by (yes) or (no).	5.0	0.0
12. As for Question 11, can you explain the reason why you need to change the code? Explain it breifly.	4.0	0.6

Question	Average	Standard Deviation
13. In [A1] there are other classes whose implementation shoud be changed according to the further introduction of the "competition mode". How many in the classes can you find? List up them.		
14. As for Question 13, can you explain what changes on the responsibility and collaboration of the changed classes? Explain them briefly.	2.2	1.0
15. As for Question 13, can you implement the necessary changes? Explain your implementation design briefly.	2.8	1.2
16. As for Question 13, do you have clear guidelines to find which part in the program code you should change.	3.4	0.8
17. Do you tnink changing a program is more difficult than you thought before you actually looked for the parts that should be changed?	3.2	0.7
18. Do you understand the kinds of the data passed during the collaboration between DataOfBiginningToCombat and CombatManager in the original system and [B1] and [B2]? Explain them briefly.	2.2	1.0
19. Do you understand the process for DataOfBeginningToCombat to obtain usrs current play mode in [A1] and [B1]? Explain the process briefly.	2.8	0.7
20. As for Qusetion 19, do you understand the representation of the play mode obtained by DataOfBeginningToCombat? Explain its representation briefly.	1.8	0.7
21. As for Question 20, do you understand how the obtained data is represented in [A2] and [B2]? Explain both cases of the representation.	1.6	0.8
22. Do you understand how the play modes are used in the program code of [A1] and [B1]? Explain it briefly.	2.0	1.3
23. Do you understand how the play modes are used in the program code of [A2] and [B2]? Explain it briefly.	1.8	1.2
24. Do you understand the program statements which represent the usage of play mode data in DataOfBeginningToCombat in [A1] and [B1]? Point out the statements.	1.2	0.4

Question	Average	Standard Deviation
25. Do you understand the program statements which represent the usage of play mode data in DataOfBeginningToCombat in [A2] and [B2]? Point out the statements.	1.2	0.4
26. Do you have any idea about in which way you represent a play mode in [A1] and [B1]? Explain it briefly.	1.6	0.8
27. Do you have any idea about in which way you represent a play mode in [A2] and [B2]? Explain the presentation briefly.	1.6	0.8
28. Compare the extension from [A1] to [B1] with that of [B1] to [B2]. Can you decide which of the two extension cases requires you more efforts?	4.6	0.5
29. Consider the case that you add a new play modes to [B2], for example a mode to pose handicaps on more skilled player to compete with a less skilled player. Can you accomplish the addition by changing [B2] in a similar way that you obtain [B2] from [B1]? Please ansewer by [yes], [no], [you don't know].	5.0	0.0
30. As for Question 29, can you explain why do you think so? Explain the reason.	3.2	1.0
31. As for Question 26, compare the required efforts to change [A1] to [A2] with those to change [B1] to [B2]. The two changes differ at the selection of the way of representation of the play modes. One selection makes the changes in the program code easier then others. How many of the pairs of such contrasting changes in the both program codes can you find? Make a brief explanation for each of the pairs.		
32. Do you understand that software design includes decision about presentation of data.	3.2	1.2
33. Do you understand which matters are included in software design except for presentation of data.	1.6	0.8
34. Were the explanations given by the teacher clear?	4.6	0.5
35. Do you think that changing a program code in general is difficult?	4.0	1.1
36. Do you understand what you should do at the evaluation step?	4.2	1.2

Question	Average	Standard Deviation
37. Is the task posed on you at the evaluation step difficult?	4.0	0.9
38. Is this training useful for you to understand the importance explanations of program design?	4.0	0.6
39. Is this training useful for you to understand encapsulation?	2.8	1.0
40. Does this training make you recognize the importance of program design?	4.6	0.5
41. Do you want to participate this training again in future?	3.2	0.7
42. Do you eager to participate in this training.	3.8	1.0
43. Do you think it is important for you to study object-oriented design?	5.0	0.0
44. Do you know object-oriented design well or do you have a good skill of object-oriented design?	1.8	0.4
45. Do you want to improve your understanding and skill of object-oriented design?	5.0	0.0

5 Conclusion

Acknowledgments We thank all students who spent several hours to participate in our experiment as its subjects.

References

[1] David Coppit and Jennifer M. Haddox-Schatz. (2005) Large Team Projects in Software Engineering Cources. Proceedings of SIGCSE '05 (St. Luis, Missouri, February 2005), ACM Press, 137-141.

[2] Anna Eckerdal and Robert McCartney and Jan Eric Mostrom. (2006) Can Graduating Students Design Software Systems?. Proceedings of SIGCSE '06 (Houston, Texas, March 2006), ACM Press, 403-407.

[3] Kume, I., and Nitta, N. and Takemura, Y. (2006) A Method for Creating Teaching Materials of Practical Object-Oriented Methods. Proceedings of ICCE2006 (Beijing, Nov-Dec 2006), IOS Press, 75-78.

[4] JShen-Tzay Huang, and Yi-Pei Cho, and Yu-Jen Lin Schwartz. (2006) Exploring the Cognitive Apprenticeship Approach for Teaching Introductory Software Engineering. Proceedings of IEEE Conference on Software Engineering Education and Training Workshops.

[5] Lutz, J. M., and Bagert, D. (2006) Software Engineering Curriculum Development. IEEE Software, 23, 6, 16-18.

[6] Michael R. Wick and Daniel E. Stevenson and Andrew T. Phillips. (2004) Seven Design Rules for Teaching Students Sound Encapsulation and Abstraction of Object Properties and Member Data. Proceedings of SIGCSE '04 (Norfolk, Virginia, March 2004), ACM Press 100-104.