Learning Support System to Facilitate Redesigning for Understanding Software Design Patterns

Hiroki OOE[a]*, Tomoko KOJIRI[b], Kazuhisa SETA[c]
[a]*Graduate School of Science and Engineering, Kansai University, Japan*
[b]*Faculty of System Science and Engineering, Kansai University, Japan*
[c]*Graduate School of Science, Osaka Prefecture University, Japan*
*k773152@kansai-u.ac.jp

Abstract: Design patterns are good designs in object-oriented programming and are generated experientially by predecessors. We propose a learning method of understanding design patterns by transforming a program with a design pattern into that without one (alternative solution). We also develop a support system that encourages learners of generating appropriate alternative solution. Experimental results proved that the proposed method was effective for a deep understanding of design patterns.

Keywords: experiential knowledge, design patterns learning support, generating alternative solution, vicarious experience support

1. Introduction

Experiential knowledge is knowledge acquired through experience [1]. Experiential knowledge addressed in this study is design patterns that are a collection of good object-oriented designs. Design patterns are produced through generation of various programs by our predecessors. To obtain design patterns, it is necessary to understand not only their meaning but also the appropriate and inappropriate conditions to apply. For grasping such conditions, to follow predecessors' generation process which is to transform programs with design patterns into those without design patterns (alternative solution) is effective. However, it is difficult for learners to modify a program into a reasonable one.

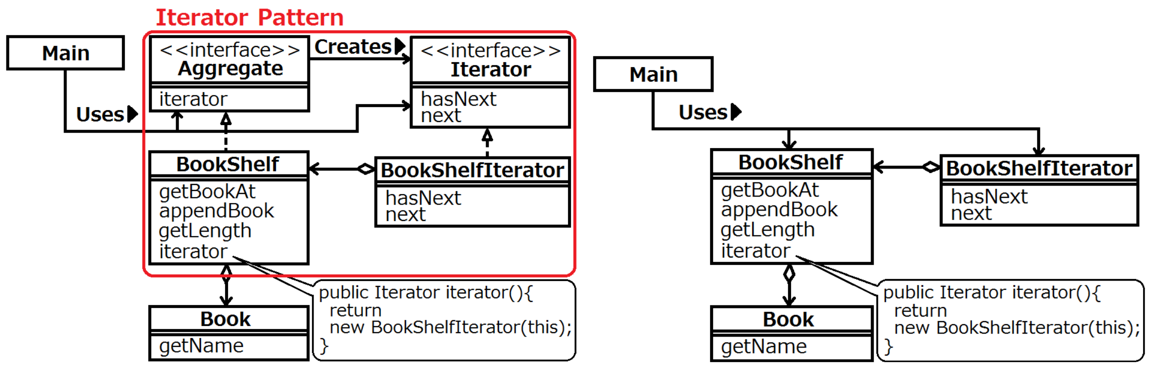
We propose a learning method of reliving predecessors' experience by producing alternative solutions. Additionally, we construct a support system to encourage learners of generating alternative solutions. Based on the method, learners can develop the ability of acquiring experienced knowledge, even if new design patterns are given.

Many studies support the learning of programming [2]. However, they cannot consider the conditions for applying the structure. Stephen introduces target problem to learners and makes them derive the design pattern by themselves as predecessors did [3]. However, in this method, some learners may not be able to derive the design pattern. In this study, in order for all learners to consider predecessors' trial and error easily, the learning method is proposed in which transforming the class diagram of the program using design patterns into alternative solution. We have constructed a learning support system in which the class diagram of the program can be transformed. In addition, the advising mechanism for supporting generation of the alternative solution is introduced.

2. Framework of Design Pattern

Design patterns are produced by experts of programming through their experiences of designing programs repeatedly for similar problems. Design patterns are structures that contain the benefits of object-oriented design, but alternative solutions do not. Therefore, for acquisition of design patterns, to generate alternative solutions that reduce the benefits of object-oriented design is effective to understand the advantages and disadvantages of the design patterns.

Example of alternative solution is shown in Figure 1. The class diagram in Figure 1(a) is the example of using the *Iterator* pattern [5]. The characteristic of this program is that *Main* class does not refer to the method in concrete class, such as *BookShelf* and *BookShelfIterator*, but uses the method in the *Aggregate* interface and *Iterator* interface. In this structure, even if adding or changing another concrete class having the same functionality as concrete class, then a program can be updated without changing the *Main* class. The example of its alternative solution is shown in Figure 1(b). This program does not introduce interface and *Main* class refers to the concrete class directly. In this structure, the change of the concrete method affects to the *Main* class. Therefore, this alternative solution reduces the reusability and flexibility of the original design.



```
public Iterator iterator(){
  return
  new BookShelfIterator(this);
}
```

```
public Iterator iterator(){
  return
  new BookShelfIterator(this);
}
```

(a) Class diagram of the solution using the Iterator pattern

(b) Class diagram of good alternative solution of (a) without using the Iterator pattern

Figure 1. Example of solution

In this study, learners transform the class diagram to the program structure of the same meaning as alternative solution of the design pattern. To make learners generate good alternative solutions, we have proposed a support system which evaluates the learner's class diagram of alternative solution and generates advice, if necessary. The system presents class diagrams of target design patterns that learners want to learn. Learners change the class diagram and design alternative solution without the design pattern. The system evaluates an inputted class diagram as the good alternative solution. The system gives advice to promote to the derivation of an appropriate alternative solution if the learners' alternative solutions are inappropriate. Learners change the class diagram by referring to advice and finish learning if they generate a good alternative solution.

3. Mechanism for Supporting Alternative Solution Generation

The system holds the data for class diagrams of the problems and class diagrams of good alternative solutions. Good alternative solutions can be expressed by the difference between class diagrams of the problem and class diagrams of alternative solutions. An alternative solution is created by adding/deleting objects/relations from the given problem. Modification of existing objects/relations is equivalent to adding modified objects/relations after deleting the original one, so it can be represented by a combination of addition and deletion. In this study, the system holds alternative solution data as added/deleted object and type of its change, such as addition or deletion. In addition, a good alternative solution is intended to reduce the benefits of object-oriented design. The system must hold benefits of corresponding alternative solution with class diagram data.

Alternative solution data for Figure 1(b) is shown in Table 1. Since it reduces the reusability and flexibility, "reusability and flexibility" are described as the target benefits of object-oriented design. Additionally, it deletes the relation between *Main* class and *Aggregate* interface, so such deletion and its type are noted as ID I. In the same way, all differences are listed.

Table 1. Alternative solution data of Figure 1(b)

Benefits of object-oriented design	ID	Changed object	Type of change
Reusability and flexibility	Ⅰ	Relation（Depend : Uses）: Main class⇒Aggregate interface	Deletion
	Ⅱ	Relation（Depend : Uses）: Main class⇒BookShelf class	Addition
	Ⅲ	Relation（Depend : Uses）: Main class⇒Iterator interface	Deletion
	Ⅳ	Relation（Depend : Uses）: Main class⇒BookShelfIterator class	Addition
	Ⅴ	Class : Aggregate interface	Deletion
	Ⅵ	Class : Iterator interface	Deletion
	Ⅶ	Relation（Depend : Creates）: Aggregate interface⇒Iterator interface	Deletion
	Ⅷ	Relation（Implement）: BookShelf class⇒Aggregate interface	Deletion
	Ⅸ	Relation（Implement）: BookShelfIterator class⇒Iterator interface	Deletion

The alternative solution inputted by the learner is also transformed to this form. The system compares the lists of problem and alternative solution in the system with the learner's alternative solution and generates advice that helps learners to derive correct alternative solutions if the learner's diagram is inappropriate.

The advice is derived depending on the types of inappropriateness in the learner's class diagram. Learners who generated inappropriate solutions do not understand one of the following factors:

- Factor 1. Benefits of object-oriented design that should be reduced from the given class diagram
- Factor 2. The way of changing the class diagram to reduce benefits of object-oriented design

If learners add/delete unnecessary classes/relations, they might not understand factor 2. Such learners can generate correct class diagram if the system points out existence of the unnecessary addition/deletion. However, if the learner does not perform the necessary addition/deletion, they might stumble in both factors 1 and 2. Therefore, the system first tells learners the benefits of the object-oriented design and the part of the inappropriate solution that they should specifically address. The system advises a correct change to let them understand factor 2 if the learner still cannot generate a correct class diagram. The advice templates are shown in Table 2, where <Aspect> corresponds to the benefit of object-oriented design, and <Object> represents class or relation to be added/deleted.

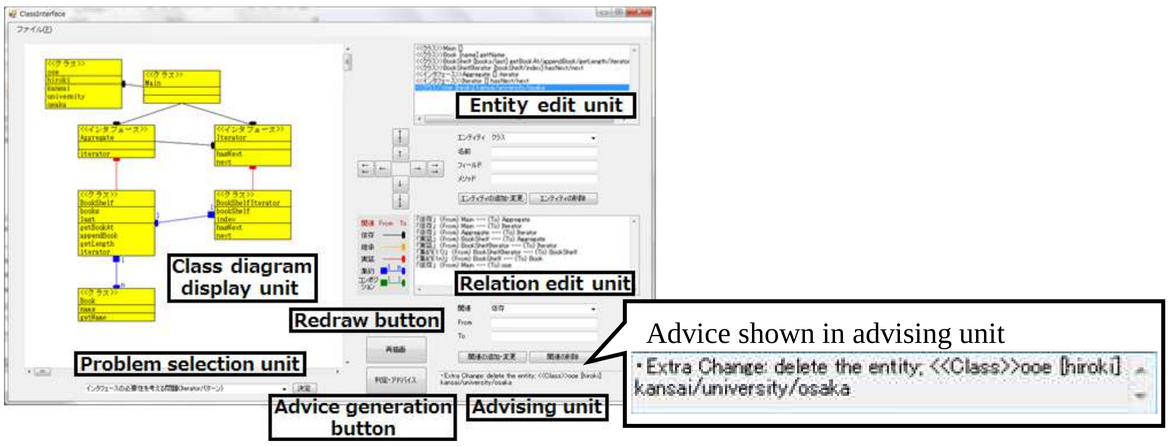
Table 2. Advice template

Type of inappropriateness	Advice template
Lack of necessary addition	First advice: "Focus on <Object> for reducing <Aspect>." The Second advice: "Add <Object> for reducing <Aspect>."
Lack of necessary deletion	First advice: "Focus on <Object> for reducing <Aspect>." Second advice: "Delete <Object> for <Aspect>."
Extra addition	"Extra Change: delete <Object>."
Extra deletion	"Extra Change: add <Object>."

4. Prototype System

We implemented the proposed system using Microsoft Visual C++. The user interface is presented in Figure 2. The class diagram is displayed at the class diagram display unit. At the beginning of learning, the class diagram of the problem is shown, and learner's solution is displayed during the learning. The problem selection unit holds a list of learnable design patterns and learners can select one from it.

The entity edit unit represents a list of entities that exist in the class diagram. Learners can add and delete the entity from text fields. The relation edit unit represents a list of relations. Learners can also add, and delete them. The redraw button is used for reflecting the class diagram that is inputted in the entity edit unit and relation edit unit. The advice generation button is used for evaluating the class diagram created by the learner and for asking advice. When its button is clicked more than once, the former advice is replaced by the new advice. Figure 2 shows an example of the provided advice. This advice shows "deleting the unnecessary class *ooe* whose field is *hiroki* and whose methods are *kansai*, *university*, and *osaka*".



・Extra Change: delete the entity: <<Class>>ooe [hiroki] kansai/university/osaka

Figure 2. User interface

5. Experiments

We conducted an experiment to verify the effectiveness of the proposed methods. Subjects were six graduate and undergraduate students who had not learned design patterns, or who were unable to apply design patterns to the problem. First, subjects were asked to take a pretest to confirm their understanding of design patterns. The pretest has three questions: question 1 was used to create an alternative solution from the given class diagram with design pattern; question 2 asks the reason for created alternative solution in question 1; and question 3 asks the role of the focused class/relation in the design patterns. After explaining the manipulation method of the system, subjects were asked to create a class diagram of alternative solution using the system. After that, subjects took a posttest to verify the change of understanding of design patterns. The format of the posttest is the same as that of the pretest, but the target design pattern is different. Additionally, we prepared a questionnaire to evaluate the performance of this learning method.

Results of the pretest and the posttest are shown in Table 3, which presents the number of subjects for each situation. In the pretest, most subjects could not answer all the questions correctly. In contrast, all subjects were able to answer question 1 and numerous subjects were able to derive answers for questions 2 and 3 in the posttest. Since many subjects were able to understand the benefits of object-oriented design, our system is effective for learning design patterns.

Table 3. Results of pretest and posttest

Question	Result	Pretest	Posttest
1	Created appropriate alternative solution	1	6
	Created inappropriate alternative solution	4	0
	Could not create at all	1	0
2	Explained correctly	1	3
	Explained incorrectly	2	3
	Could not explain at all	3	0
3	Explained correctly	1	5
	Explained incorrectly	2	1
	Could not explain at all	3	0

Questions of the questionnaire and the results are shown in Table 4. They were asked to select one from 1 (strongly disagree) to 5 (strongly agree). The table shows the number of subjects who select each evaluation value for each question. From questions 1 and 2, it seems clear that subjects can create the solution, but that they cannot understand the meaning of created alternative solution. We should revise the advising mechanism to make learners understand the meaning of the created alternative solution. Results of questions 3 and 4 show that subjects were able to understand the effect of the learning through creating alternative solution. However, only half of them want to use this learning method. The complexity of manipulating our system makes subjects feel this learning method is inconvenient. Therefore, it is necessary to improve the system usability. Additionally, we need further evaluation to prove the effectiveness of our system with more subjects.

Table 4. Questions and results of questionnaire

Question	Evaluation				
	1	2	3	4	5
1. Did you understand the meaning of alternative solution which you created?	0	2	1	1	2
2. Did you think given advices were appropriate?	1	0	4	1	0
3. Do you understand the importance of creating alternative solution?	0	0	0	5	1
4. Do you want to create alternative solution in learning new design patterns?	0	2	1	2	1

6. Conclusion

In this paper, we proposed a design pattern learning support system by creating alternative solutions. In the future, the advising method should be improved to make explain the role of the target classes and relations deeply. Additionally, the system interface must be modified to edit the class diagram more smoothly.

References

[1] Polanyi, M. (2003). The Tacit Dimension, trans. Isao, T., *Thikuma Library* (in Japanese).

[2] Hwang, W., et al. (2008). A Web-based Programming Learning Environment to Support Cognitive Development, *Interacting with Computers*, 20(6), 524-534.

[3] Stephen, W. (2005). Teaching Design Patterns by Stealth, *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, 37(1), 492-494.

[4] Seta, K., et al. (2010). Presentation-based Collaborative Learning Support System to Facilitate Meta-cognitively Aware Learning Communication, *The Journal of Information and Systems in Education*, 9(1), 3-14.

[5] Erich, G., et al. (1994). Design Patterns: Elements of Reusable Object-Oriented Software, *Addison-Wesley Professional*.