Using Model-Driven Development Tools for Object-Oriented Modeling Education

Seiko Akayama[1], Kenji Hisazumi[2] Syuhei Hiya[1], and Akira Fukuda[3]

[1] Graduate School of Information Science and Electrical Engineering, Kyushu University, Fukuoka, Japan,
[2] System LSI Research Center, Kyushu University, Fukuoka, Japan
[3] Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka, Japan

Abstract. Model-driven development (MDD) can help verify the accuracy of models and generate source codes, which allows a programmer to minimize the development time required to evaluate the software so that he or she can focus on the modeling process. Thus, modeling should be taught with MDD because it allows students to acquire modeling skills in a short period of time. We conducted a course to teach UML modeling to two groups. The first group used the MDD tool, while the second did not. We elucidate the advantages of each case with and without the use of the MDD tool. Based on our results, we propose the effective use of MDD tools in UML modeling education.

Keywords: UML (Unified Modeling Language), MDD (Model-driven development), object-orientation, Modeling, learning support

1 Introduction

Object-oriented modeling is widely used during embedded software development and is taught in many universities. Modeling ensures good quality and productivity during software engineering [1]. Thus, software development is shifting from manual programming to model-driven development (MDD) [2].

However, it is difficult to teach modeling. In the early stages of the learning process, students have questions such as "Why do we need modeling"? "How do we model"? and "Is this model equivalent to the specification"?

MDD is used to verify the accuracy of models and generate source code. This allows a programmer to minimize development time so that he or she can focus on the modeling process. Therefore, modeling should be taught with MDD, because it allows students to acquire modeling skills in a short period of time.

Previous studies based on the research on MDD and education divided into three categories: studies on MDD education [3][4], such as meta-model and mechanisms of MDD; studies on developing MDD in system development exercises [5][6][7]; and studies on software modeling education using MDD [8][9].

One issue that arises when we use the MDD method for modeling education is that students tend to neglect the quality of the model because they focus on completing the functional aspects that can be evaluated with the MDD [9].

The purpose of this study is to demonstrate the effectiveness of MDD tools for modeling. From these results, we propose the effective use of MDD tools in UML modeling education. Note that, in this paper, tools with the ability to automatically generate codes from UML models are called MDD tools.

In this paper, we discuss the following research questions.

When using MDD tools in modeling education for novices,

RQ1 what is the difference in the process of creating the model?
RQ2 what is the difference in model quality?

Further, model quality is of three types (syntactic, semantic, and pragmatic) [10].

We conducted a course to educate two groups in UML modeling. The first group members used the MDD tool, while the second group did not. We clarify the advantages of each case with and without the use of MDD tools. From these results, we propose the effective use of MDD tools in UML modeling education.

2 MDD Tools

There are some modeling education studies using BridgePoint as an MDD tool [5][8][9]. However, to the best of our knowledge, when learners create a model using BridgePoint, they require a long time to learn the action language that is required to define the state actions. Therefore, learners find it difficult to focus on creating state machine diagrams and class diagrams. In order to define actions easily, we have developed a Domain-Specific Modeling (DSM) language for modeling education using the social DSL platform "clooca [11]."

This platform allows making class diagrams and state machine diagrams. A class diagram consists of classes and relations, while state machine diagrams consist of states (including an initial state), event transmission states, events, and actions.

In order to collect the change history of the model, a model repository is created. By storing models in the model repository when a new or additional changed model is created, learners can see the past versions of models later. The model repository is an additional function of the MDD tool.

3 Experiment

3.1 Experimental Description

We performed experiments on subjects divided into two groups. The first group used the MDD tool (experimental group) and is called the with-MDD group, and the second group did not use it (control group) and is called the without-MDD group.

For the with-MDD group, we prepared an MDD tool that can generate code, and group members were allowed to check their operations at any time. In contrast, in the without-MDD group, we prepared the same tool, but group members were allowed to check operations only once at end of the exercise.

3.2 Experimental Procedure

The number of subjects was 12, and the subjects were well-versed with JAVA and UML notation. The educational items used in the experiment are listed in Table 1.

Table 1. Outline of the class.

		without-MDD	with-MDD
day 1	1st hour	Object-orientation, review of UML, What is MDD?	
	2nd hour	How to use the MDD tool (clooca).	
	3rd hour	Basic exercise 1: Creating the right-turning and stopping model.	Basic exercise 1: Creating and running the right-turning and stopping model.
	4th hour	Basic exercise 2: Creating the line trace model.	Basic exercise 2: Creating and running the line trace model.
day 2	5th hour	Review of the basic exercises, Description of the integrated exercise	
	6th hour 7th hour 8th hour	Integrated exercise: Creating the auto transport system model Running of the model at the end of exercise	Integrated exercise: Creating and running the auto transport system model

3.3 Exercises

The integrated exercise was aimed at developing an auto transport robot for a fictitious transportation company. The development objective was to develop a robot vehicle using LEGO Mindstorms NXT (Figure 1). The automated operations were transportation, forwarding, and out-of-service. These three operations were affected by the presence or absence of delivery destinations or cargoes. A wall detector (sonar sensor) monitored the delivery destinations while a bumper (touch sensor) detected the forwarding destination. The task of the robot was to trace a black line along a course using a line monitor (light sensor) to make stops at the delivery destination, forwarding destination, or the garage. The robot had to behave appropriately at each point and deliver the cargo.

4 Results

4.1 What is the difference in the process of creating the model?

Typical examples of class diagrams of the with- and without-MDD groups are shown in Fig. 2 and Fig. 3.

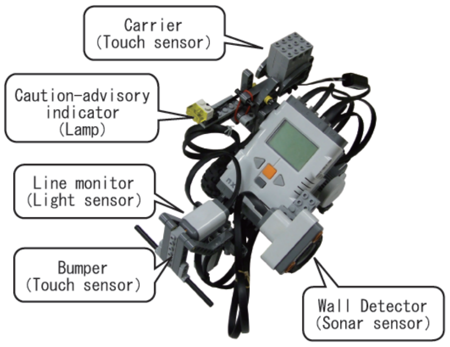

Fig. 1. Auto transport robot used in the integrated exercise.

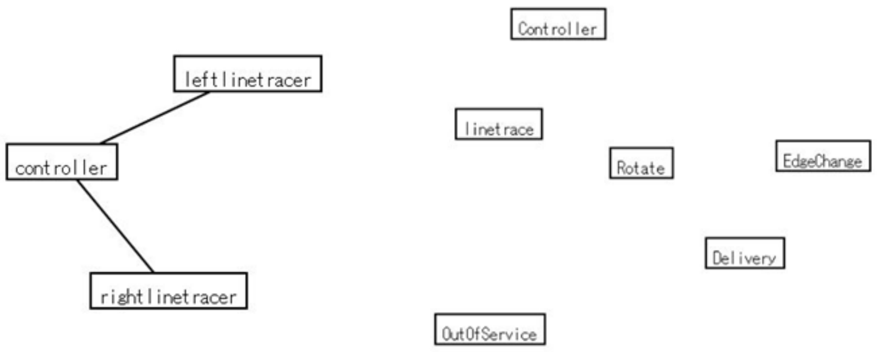

Fig. 2. Examples of class diagrams of the with-MDD group.

Subjects of the with- and without-MDD groups were respectively labeled A-F. Table 2 summarizes the number and kind of class categories.

In the without-MDD group, the subjects had classes related to only services (examples: transportation, forwarding, and out-of-service) with the exception of a class for controlling the whole system. In contrast, the with-MDD group had numerous classes related to functions (examples: line trace, edge change, and rotation), and there were four class diagrams which did not include a class related to services.

When subjects created the integrated exercise model, four subjects reused the basic exercise model in the with-MDD group; however, no subjects reused models in the without-MDD group. In order to analyze the process of creating the model, exercise time divided into three: first, middle, and last. Table 3 illustrates the number of classes added, modified, and deleted at the time of the first, middle, and last.

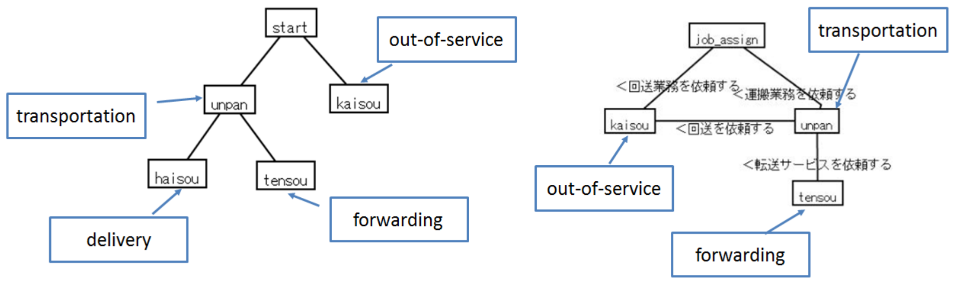

Fig. 3. Examples of class diagrams of the without-MDD group.

Table 2. The number and kind of class categories.

	Categories of class	A	B	C	D	E	F	Average
with-MDD	Number of classes for controlling the whole system	1	1	1	1	1	1	1.0
	Number of classes related to services	0	2	0	0	2	0	0.7
	Number of classes related to functions	2	3	3	4	3	0	2.5
	Total number of classes	3	6	4	5	6	1	4.2
without-MDD	Number of classes for controlling the whole	1	1	1	1	1	1	1.0
	Number of classes related to services	4	4	3	3	4	3	3.5
	Number of classes related to functions	0	0	0	0	0	0	0.0
	Total number of classes	5	5	4	4	5	4	4.5

In the without-MDD group, the proportion of the number of added and deleted classes reduced in the last, while the proportion of the number of states added and modified increased in the middle and last. Thus, it is assumed that subjects fixed the class structure by adding and deleting classes at first and then designed the behavior model by adding and modifying states. On the other hand, in the with-MDD group, subjects also added and deleted classes in the last. Therefore, the with-MDD group changed the class structure until they reached the last. In the with-MDD group, subjects tended to modify a functional model and then created a static structure of the system. In contrast, in the without-MDD group, subjects created a new structure model and then created the behavior model. Therefore, when the subjects did not use MDD, they tended to create a model by implementing a top–down approach; however, when they used MDD, they did so using a bottom–up approach.

4.2 What is the difference in model quality?

Class diagrams Table 4 shows the number of errors in the class diagrams. Each subject created one class, and therefore, the maximum number of errors is six.

In six of the seven items, there were many errors in the with-MDD group. All members of the without-MDD group committed the error “One class has several

Table 3. Model creation history of the integrated exercise.

	First		Middle		Last	
	without-MDD	with-MDD	without-MDD	with-MDD	without-MDD	with-MDD
class addition	17.3 %	8.9 %	8.1 %	9.0 %	0.0 %	1.6 %
class modification	9.6 %	7.8 %	6.5 %	3.0 %	9.8 %	3.9 %
class deletion	3.9 %	4.1 %	1.8 %	0.5 %	0.0 %	3.3 %
state addition	44.0 %	34.3 %	69.9 %	71.4 %	68.4 %	49.0 %
state modification	10.0 %	32.9 %	6.5 %	13.5 %	20.9 %	35.2 %
state deletion	15.2 %	12.1 %	7.3 %	2.7 %	1.0 %	7.0 %

Table 4. The total number of classes with the error about the quality categories.

		without-MDD	with-MDD
Syntactic	Relations are not drawn	0	2
	No relations names	2	4
Semantic	Class names do not represent the responsibilities	0	2
	Relation names are inappropriate	0	2
Pragmatic	There are unnecessary classes	0	1
	One class has several responsibilities	0	1
	Several classes have the same functions	6	0

responsibilities." The model quality of the classes was high for the without-MDD group. However, no members of the without-MDD group could separate the functions of the class and no member created a function class.

We think that if the models were not executable, the subjects would have focused on operation models at a higher level of abstraction. It is difficult to model functions that have a low level of abstraction such as a running style.

State machine diagrams Typical examples of state machine diagrams of the with- and without-MDD groups are shown in Fig. 4. Table 5 shows the number of errors in the state machine diagrams.

The with-MDD group had "No states names" and "State names inappropriate" errors such as a or b.

Table 5. The total number of states with the error about the quality categories.

Model quality type	Error items	without-MDD	with-MDD
Syntactic	No states names	0	8
Semantic	Several states have the same state names	5	11
	State names are inappropriate	0	7

We evaluated the achievement rate for the integrated exercise as an evaluation function for measuring the semantic quality. Table 6 lists the achievement

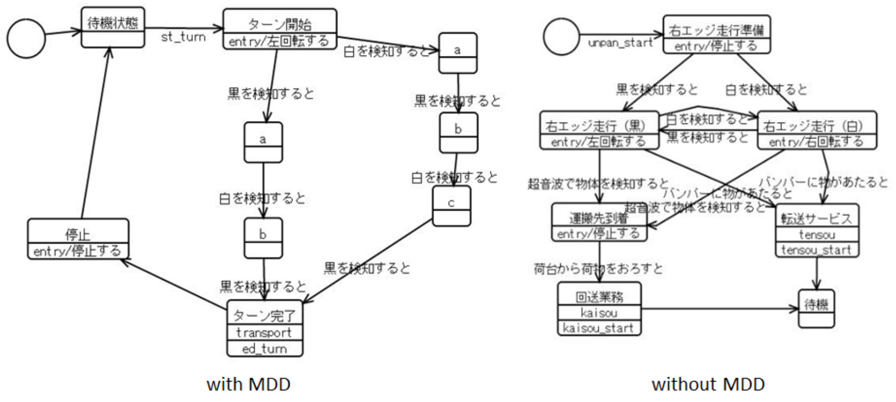

Fig. 4. Examples of state machine diagrams.

rates for the integrated exercise. The with-MDD group had a high achievement rate because they were able to test the models.

Table 6. Achievement rate for the integrated exercise.

	without-MDD	with-MDD
1) Trace a line	83%	100%
2) Detect a wall and stop	67%	67%
3) Detect unloading and send round	50%	83%
4) Stop at the garage	67%	67%
5) Trace the opposite line edge	50%	83%
6) Detect forwarding destination and invert	33%	67%
7) Restart the line trace	0%	67%
8) Complete all patterns	0%	17%

5 Discussion

Currently, many educational modeling courses apply a top–down approach. In contrast, successful MDD practice tends to be driven from the bottom–up approach. Therefore, Whittle et al. suggested that modeling should be taught with a bottom-up approach rather than a top–down one [12]. In this study, we assumed that by limiting the number of times operations can be checked and by automatic code generation, change in the approach (bottom–up and top–down) to development can be encouraged. We believe that it might lead to improved modeling skills by teaching bottom-up and top-down approaches on both sides.

6 Conclusion

We conducted a course to educate two groups in UML modeling. Subjects from one group used the MDD tool, while those from the other group did not. We verified the advantages of each case, with and without the use of the MDD tool. The results showed that when subjects did not use MDD, they tended to create models using a top–down approach and gave appropriate names to classes and states, and that when they used MDD, they tended to create models using a bottom–up approach and achieved a high achievement rate in the exercise.

Based on these results, we believe that by limiting the number of times operations can be checked and by automatic code generation, a change in the approach (top–down or bottom–up) to development of models can be encouraged. In the future, we intend to clarify the number and timing of optimal operation checks.

References

1. Lethbridge, T.C., Mussbacher, G., Forward, A., Badreddin, O.: Teaching uml using umple: Applying model-oriented programming in the classroom. In: 24th IEEE-CS Conference on Software Engineering Education and Training. (2011) 421–428
2. Liggesmeyer, P., Trapp, M.: Trends in embedded software engineering. IEEE Software **26**(3) (2009) 19–25
3. Gjosater, T., Prinz, A.: Teaching model driven language handling. Electronic Communications of the EASST **34** (2010) 1–10
4. Tekinerdogan, B.: Experiences in teaching a graduate course on model-driven software development. Computer Science Education **21**(4) (2011) 363–387
5. Burden, H., Heldal, R., Siljamaki, T.: Executable and translatable uml - how difficult can it be? In: APSEC'11. (2011) 114–121
6. Henry, S.F., Gardner, H., Boughton, C.: Executable/translatable uml in computing education. In: Proc. Sixth Australasian Computing Education Conference. (2004)
7. Khmelevsky, Y., Hains, G., Li, C.: Automatic code generation within student's software engineering projects. In: Proceedings of the Seventeenth Western Canadian Conference on Computing Education. WCCCE '12, New York, NY, USA, ACM (2012) 29–33
8. Starrett, C.: Teaching uml modeling before programming at the high school level. In: Proc. IEEE International Conference on Advanced Learning Technologies, IEEE Computer Society (2007) 713–714
9. Akayama, S., Kuboaki, S., Hisazumi, K., Futagami, T., Kitasuka, T.: Development of a modeling education program for novices using model-driven development. In: Proc. 2012 Workshop on Embedded and Cyber- Physical Systems Education, ACM (2012)
10. Lindland, O.I., Sindre, G., Sølvberg, A.: Understanding quality in conceptual modeling. IEEE Softw. **11**(2) (March 1994) 42–49
11. Technical Rockstars: clooca. `http://www.clooca.com`
12. Whittle, J., Hutchinson, J.: Mismatches between industry practice and teaching of model-driven software development. In: Models in Software Engineering. Volume 7167 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 40–47