How to Teach Software Modeling

Tetsuo Tamai
Graduate School of Arts and Sciences
The University of Tokyo
3-8-1 Komaba, Meguro-ku
Tokyo 153-8902, Japan
tamai@acm.org

ABSTRACT

To enhance motivation of students to study software engineering, some way of finding balance between the scientific aspect and the practical aspect of software engineering is required. In this paper, we claim that teaching multiple software modeling techniques from a unified viewpoint is a good way of obtaining the balance and attracting the students' interest as well.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*computer science education*; D.2.1 [**Software Engineering**]: Requirements/Specification—*modeling*

General Terms

Design

Keywords

software modeling, software engineering education, UML

1. INTRODUCTION

Software engineering education at universities faces a common problem; that is regular students do not usually have experience of developing software for practical use and thus are not motivated for software engineering aiming at high quality software production by a project team or a persistent organization. Software projects conducted by students simulating real scale software development may help enhance students' motivation, although it requires a lot of efforts to prepare such projects and manage them.

Another way of solving this problem is to teach those who already have real experience in industry. In our case, there are currently five Ph. D. students under the author's supervision who are working at companies as well as doing research in our lab. As a by-product, interactions between the part-time students and the other regular students stimulate each other, particularly enlightening the regular students to practical software issues. However, too much emphasis on practicality may bring negligence to science and technology and may generate anti-intellectualism. A good balance between the scientific aspect and the practical aspect of software engineering should always be pursued.

In our view, teaching various software modeling techniques is a good way to achieve balanced software engineering education. It is needless to say that *model* is a key concept and *modeling* is an essential skill in software engineering. There are a variety of modeling techniques; some are intuitive and quite accessible to novices, while some are highly sophisticated and attract theory oriented students and researchers.

In this paper, we would like to show that it is effective to teach multiple modeling techniques from a unified viewpoint. It is based on our experience of teaching software engineering courses at several universities in Japan. Recently, the author published a textbook on software engineering, specifically focused on software modeling (unfortunately, it is written in Japanese)[1]. The book covers the whole area of software engineering, including design, testing and evolution but the modeling part has a role of attracting interests of intelligent students, who may not have much experience in developing real scale software systems. It also gives a consistent viewpoint penetrating through various techniques employed in different stages of software engineering.

2. MODELING TECHNIQUES

In software engineering, models are used for various purposes, e.g. life cycle model, process model, project model, product model, quality model, domain model, requirements model, design model, object model, data model, etc. In the following, we basically focus on requirements and design models but most of the discussions will hold for other kinds of models.

Teaching modeling is almost equal to teaching abstraction. Models are constructed through capturing the crucial properties and structure of the target, abstracting away irrelevant details. Thus, learning how to model is a good training for mastering abstraction.

2.1 Graph Representation of Models

Many software models are represented with diagrams. Wide acceptance of UML symbolizes the trend that diagrams are often preferred to textual languages. Among many types of diagrams, graph structured diagrams are by far the most widely used. The reasons may be as follows.

1. A most fundamental way for human mind to understand the world is by regarding it as consisting of a set of conceptual units and a set of relations between them. Conceptual units can be naturally illustrated with boxes or circles or whatever closed figures and relations can be illustrated with lines or arrows connecting such figures, corresponding to vertices and edges of graphs, respectively.

2. It is easy to draw graph structured diagrams by hand or with drawing tools.

3. Concepts and algorithms of the graph theory are available and often useful in analyzing models represented by graphs. A typical example is reasoning on transitive relations by tracing along paths of graphs. Also, the concept of subgraph is highly useful in decomposing higher-level models or clustering lower-level models.

Accordingly, a number of models share the same structure of graphs. Table 1 shows graph structures of some typical models.

Table 1: Graph structures of typical models

model	vertex	edge
Data flow	process	data flow
ER	entity	relationship
State transition	state	transition
JSD	process	data stream connection state vector connection
Activity	activity	control flow
Petri net	place, transition	fire and token flow

2.2 Commonality and Difference between Models

It is pedagogical to let students notice the common structure shared by a number of models. However, the apparent resemblance often causes confusion. Such confusion can be observed not only in software modeling graphs but in many diagrams found in daily newspapers, magazines, reports, proposals and other documents. It is often the case that one vertex denotes a type of things and another denotes quite a different type on the same diagram or one type of edges co-exist with edges with different meaning. Thus, it is important to make students consciously aware the differences between different models. We often experience that when we let students draw data flow diagrams who appear to have understood the data flow model perfectly, the diagrams turn out to be something like control flow graphs.

To show the difference, it is instructive to categorize models represented by graphs. Basically, there are two categories.

1. Static models:

   An edge connecting vertex A and vertex B represents a relation between A and B. When the edge is undirected, it means "A and B are in some relation" and when directed, it means "A has a relation with B". Typical examples include entity relationship model, class diagram and semantic network.

2. Dynamic models:

   An edge from vertex A to B denotes a move from A to B. The edge in this case is always directed. There are two subcategories:

   (a) The case where a view of control moves from A to B. Examples are control flow model and state transition model.

   (b) The case where data or objects flow from A to B. Examples are data flow model, work flow model, and transportation flow model.

Static models and dynamic models may not be easily confused but confusion between different dynamic models are often observed, e.g. data flow and control flow or state transition and activity transition. Since graphs are intuitively understandable, their semantics are apt to be understood ambiguously or misunderstood.

3. UML

UML diagrams can also be viewed in terms of graph structures. Table 2 shows graph structures of five UML diagrams.

Table 2: Graph structures of UML diagrams

diagram	vertex	edge
class diagram	class	generalization, composition, association
state machine	state	transition
activity diagram	activity	control flow
collaboration diagram	object	message flow
sequence diagram	message anchor point	message flow

It is usually not desirable to teach UML per se. UML is a collection of miscellaneous diagrams and its specification is continuously changing. For the pedagogical purpose, UML had better be regarded as a catalogue of analysis and design know-how collected around diagrammatic representations. Diagrams should be selected according to the policy of how to teach modeling methods.

Each UML diagram contains overly rich constructs, which sometimes blur the essential property of the model. For example, the activity diagram is essentially a control flow diagram but it also includes a notation for data flow description. From the stance of emphasizing differences between various models, it is not appropriate to include such ad hoc constructs. By the same token, the collaboration diagram (, renamed to "communication diagram" in UML 2) is explained to have the equivalent semantics as the sequence diagram. But if that is the case, significance of the collaboration diagram is considerably limited. The author prefers to regard it as showing collaboration relations between objects, integrating a set of different sequence diagrams.

4. CONCLUSION

Software modeling is important by itself but teaching modeling in the software engineering course has at least two additional meanings. One is to give a bird's-eye view to the whole software engineering through the standpoint of modeling technology. The other is to attract interest of good students who may not have much experience in developing a real-scale software but possess intelligence and will to attack complexity of modern software construction.

5. REFERENCES

[1] T. Tamai. *Foundations of Software Engineering*. Iwanami Shoten, Tokyo, Japan, 2004. in Japanese.