Teaching UML Modeling Before Programming at the High School Level

Cortland Starrett
Mentor Graphics Corporation (www.mentor.com)
Faith Christian School (www.fcslafayette.org)
cort@roxsoftware.com

Abstract
This paper describes experience teaching modeling at the high school level prior to teaching programming and embedded control. An implementation-independent form of UML modeling is being used to teach students to analyze various applications, systems and problem domains. The objective is to introduce the abstract thinking processes involved in modeling before introducing the more concrete thought processes involved in programming with frequently used textual imperative programming languages. The author reports on experience teaching abstraction and UML modeling to a class of 12th graders.

1. Introduction

1.1. Teaching Model Driven Development
Model Driven Development (MDD) and Model Driven Architecture (MDA) are growing in adoption within the software and electronics engineering industry. These practices represent an evolution of the level of abstraction at which engineers and programmers develop their applications. As lists of requirements grow longer and the complexity of systems expands, it is necessary to work at a higher level of abstraction to maintain and improve productivity. Each increase in level of abstraction raises productivity. Model Driven Development using the Unified Modeling Language (UML) represents a logical step upward in this progression.

Modeling with UML is taught at the professional level in corporations and is taught in many universities. It is often taught as an advanced topic or an elective. In virtually all cases in the university engineering and computer science environment, modeling is taught to students after they have already learned programming.

I am a full-time engineering manager with a background in embedded systems development and engineering tools. I have been modeling embedded systems since 1988 and practicing and teaching executable UML [1] since 1991. Until recently most of my teaching has been to industry professionals in the various fields of engineering development.

1.2. Questions
This paper explores the following questions: Should modeling be taught in high school or first year college? Is this feasible? How will students respond to modeling? How will it help them?

2. Methods

2.1. Course Material
In addition to engineering, I teach a computer science course at Faith Christian School, a K-12 grade parochial school of 633 students. In the first semester, this course covers as background binary mathematics, basic set theory, simple electronics, networking and computer history. Executable UML modeling and programming are taught in the second semester. Contrary to conventional practice, modeling is taught before computer programming.

The students in the class range in age from 16 – 18 years. None have learned programming. Each has been given a basic background for computer science.

As of the writing of this paper, the course described is still in session. Results are preliminary and fresh. Data modeling (class diagrams) control (state machines) and processing (action language) have been taught and practiced.
2.2. Laboratories with Lego Robots [2][3]

As part of learning to model, the students experience hands-on, embedded target experience using robots built with Lego. Executable UML models are translated into ANSI C code and cross-compiled for the embedded microcontroller used in the Lego kit. The resulting image is then downloaded into the programmable Lego brick where it controls the robot autonomously. A “sumo contest” is held in which student teams compete, pitting robot against robot. Since the hardware platform for the physical robot is kept constant for all teams, the competition is decided by the team with the best UML model.

3. Results

3.1. Teaching “Pre-Programmers” to Model

The students are learning to model instead of implement, thus learning to model before they implement. They are finding abstractions for elements in the application domain rather than jumping into implementation too quickly. They cannot jump into implementation; they do not know how yet!

I am finding that abstractions can be easier to draw when not confused with the constructs and vocabulary of concrete implementation. I am impressed as the students identify the classes, attributes, associations, and state machines involved in the following domains: automobile, MP3 player, football game, microwave oven, bicycle, sudoku, others.

Data modeling concepts such as generalization, associative classes and reflexive associations have come naturally. These students struggle more with syntax than with abstraction.

The students seem able to overcome the “blank sheet of paper syndrome” more easily than many of the groups of more mature students I have taught. When beginning an assignment, these students easily find a place to start brainstorming a list of class candidates quickly.

UML modeling quiz results have been comparable to those of professional developers. The speed at which models are formed by pre-programmers is equal to that of experienced programmers.

4. Discussion

Learning an organized process of approaching systems analysis and programming helps students in at least two ways. By breaking the problem-solving process down into separate views, development is easier. By working at a higher level of abstraction, development is more productive. Factoring out the implementation details during systems analysis allows developers to focus on the application problem independently of the deployment technology. A student who learns to approach development tasks with modeling will be a better engineer, computer scientist or programmer.

5. Conclusions

Results from teaching young people modeling before they know how to program indicate that modeling can be taught and perhaps should be taught before programming for the reasons that follow.

5.1. Modeling is more generally applicable.
Modeling is not technology-specific and is useful to people in many disciplines. Modeling and abstractions are fundamental to analytical thinking and can be applied in many disciplines outside of engineering and computer science.

5.2. Modeling is methodical.
Modeling provides a method to help students approach problems and solutions step by step. It provides teachers a well-ordered sequence of steps to follow when teaching.

5.3. It is feasible.
The students in this class are grasping the core concepts of abstraction quickly and naturally. In fact, one finding is that “pre-programmers” may have an easier time with the high-level abstraction process than do programmers who already understand related but different concepts.

6. References