Teaching Software Modeling and Design

Hassan Gomaa
Department of Computer Science
George Mason University
Fairfax VA 22030, USA
hgomaa@gmu.edu

Abstract. **This paper describes my experience with teaching courses on software modeling and design to undergraduate and graduate (Masters and PhD) students, in addition to in-depth short courses to industry. The undergraduate course is an introductory software engineering course, which includes lectures on software modeling and design. The Masters course is a detailed course on software modeling and design. The PhD and advanced Masters courses are advanced courses on software modeling and design in the areas of software product line engineering and real-time design. The in-depth industrial courses are courses that cover essentially the same material in overview (1 or 2 day) or more detailed (4 day) formats.**

Keywords: **software modeling, software design, UML, requirements modeling, use case modeling, analysis modeling, static modeling, dynamic modeling, design modeling.**

I. Introduction

This paper describes my experience with teaching courses on software modeling and design to undergraduate and graduate (Masters and PhD) students, in addition to in-depth short courses to industry. It briefly describes the contents of each course, how the teaching of each course has evolved, the teaching approach of "learning by doing", and lessons learned.

My teaching in this area started with a Masters level course in software modeling and design, followed later by advanced specialized graduate software modeling courses in the areas of software product line engineering and real-time design. In parallel, I also taught several industrial courses in these areas. Most recently, I have taught an undergraduate course in software engineering, which includes introductory lectures on software modeling and design. Teaching these courses also led me to write books on software modeling and design [2, 4-7].

II Graduate Courses on Software Modeling and Design

The first courses I taught on software modeling and design were to graduate students in the Masters of Software Engineering program at the Wang Institute in 1986-87, and to graduate students in the Masters and graduate certificate programs in Software Engineering at George Mason University from Fall 1987 onwards [1]. The programs at Mason have a mix of full-time and part-time students. Because of the large number of part-time students, all graduate courses are taught as evening courses, meeting for a three hour class once a week.

A. Course Structure and Evolution

Although the courses on software requirements and design pre-dated UML, they followed a software modeling approach based on Structured Analysis and Structured Design, with the design modeling emphasizing the design of concurrent systems [2]. *There was also an advanced real-time design version of the course, which was taught to* PhD and advanced Masters students [3].

An early decision on course structure was for the course to cover important design concepts, provide an overview of various design methods, and follow this by focusing on one design method in particular. The goal is for students to understand typical design issues that arise in designing a software system, and to work in teams to get experience of applying the design method to a real-world problem.

From 2000 onwards, the courses were changed to be based on the Unified Modeling Language (UML) [4]. The UML-based course taught requirements modeling focusing on use case modeling, analysis modeling consisting of both static and dynamic modeling, and design modeling with emphasis on the design of concurrent and distributed systems. From 2011 onwards, the course was changed again to be a more general software modeling and design course [6] using the UML 2 notation, with a common approach for the requirements modeling and analysis modeling sections of the course. This is followed by design modeling, which addressed the design of different types of software architectures, such as client/server software architectures and service-oriented architectures by first considering the design patterns that these architectures are based on.

The UML-based software modeling and design method taught in each course starts with requirements modeling and progresses through to detailed design in Pseudocode. The lectures are supplemented with detailed case studies of applying the modeling method.

B. Team Projects

The goal of the team project is to encourage students to "learn by doing". Throughout the semester, students work in teams on a hands-on design exercise, in which they

apply the design method to a real-world problem, such as a supermarket check-out system or an inner-city traffic management system. Students use a commercially available UML modeling tool for documenting their analysis models (with static models using class diagrams, dynamic interaction models using sequence diagrams, and finite state machine models using statecharts) and design models (both static and dynamic).

Student teams also have two tutorials with the course instructor during the semester, which replace regular lectures, during which each team meets separately with the instructor. The first tutorial is to review the team's draft analysis model and the second to review the team's draft design models. The instructor provides each team with feedback, which they can use to revise the model before submitting the revised version for grading. Most students find these tutorials very useful. In addition, students have the option of resubmitting the analysis model, after it has been graded, to address the comments made by the instructor, before submitting the design model.

III ADVANCED GRADUATE COURSES ON SOFTWARE MODELING AND DESIGN

I also teach two PhD and advanced Masters courses on software modeling and design in the areas of software product line engineering and real-time design. Courses on real-time modeling and design have also progressed from the pre-UML modeling approaches [3] to a UML-based method for concurrent and real-time design [4]. The course focuses on design concepts for real-time systems, such as concurrent tasks, priority based scheduling, real-time scheduling algorithms such as rate monotonic scheduling, distributed control with concurrent state machines, subsystem and component design. This course has been revised recently to follow the revised material provided in [7].The course follows the same overall structure as the introductory graduate class described in Section 2, thus covering the design method in considerable detail and having a team project on the design of a real-world real-time problem.

The second advanced course is on software product line (SPL) design [5] with UML. This course covers SPL concepts and addresses feature modeling in considerable detail as the key modeling approach for differentiating between commonality and variability among the SPL members of a software family. The relationship between use cases and features are covered. The steps in analysis and design modeling also address commonality/variability analysis and how variable and optional features are realized in analysis and design. The course follows the same overall structure as the introductory graduate course of covering the SPL design method in considerable detail and concurrently running a team project in which students work on applying the design method to a SPL problem.

IV INDUSTRIAL COURSES ON SOFTWARE MODELING AND DESIGN

The industrial courses are courses that cover essentially the same material as the graduate courses but use a different course structure. Courses are either overview (1 or 2 day) courses or more detailed (4 day) intensive courses. The longer courses include time for industrial students to work on a design problem. I have taught three different industrial courses, on Software Modeling and Design, Real-Time Software Modeling and Design, and SPL Modeling and Design. All three courses use the UML notation.

The overview courses are lecture oriented. The longer courses have problem sessions built into the schedule for students to work in small teams on a design problem. In some courses, students also work in the evenings on the design problem

It is definitely the case that students on the four-day courses learn a lot more about software modeling and design, particularly through applying what they have learned to a design real-world problem.

V UNDERGRADUATE COURSE ON SOFTWARE MODELING AND DESIGN

In the past five years, I have also developed an undergraduate software engineering course that features software modeling and design concepts. The course follows an iterative software life cycle model, so that introductory modeling concepts can be covered in the requirements and design parts of the course. Use case modeling is used for the requirements phase. Simple analysis and design modeling is used for the design phase. Software testing covers different testing approaches but also covers model-based testing based on use cases, which are used for integration and system testing.

Students also work in teams on the different phases of solving a real-world problem starting with developing use case models, a simple class diagram with entity classes, and sequence diagrams for object interactions. The software architecture is described in terms of components with provided and required interfaces. Since students have no knowledge of concurrency or user interface design, the design assignment for the student project is for a sequential server, such as a hotel reservation server. The

user interface is simulated by creating a file with a sequence of inputs that represent online user inputs for making hotel reservations, checking in and checking out. Students carry out integration testing to test object interactions and use case based system testing to test the complete system using the simulated user interface.

The undergraduate course is the most challenging for teaching modeling concepts. Most students come to the software engineering course having previously only taken computer science courses in object-oriented programming and data structures. Some students have difficulties with understanding the more abstract modeling concepts. However, developing a simple hotel reservation system starting with use cases and progressing through to an implemented working system is an interesting experience for many students.

VI Lessons Learned

General lessons:

a) Although giving a survey of different modeling and design methods is useful, teaching one method in detail is necessary for students to understand the intricacies of software design.
b) Having a real-world project to work on, "learning by doing", is important for reinforcing what students learn. This applies to both undergraduate and graduate students.
c) From a teaching perspective, a big challenge is to make sure that the lectures "are in sync" with the project, so that the lectures are taught before the concepts are needed for the project.

Graduate courses:

a) Students who appreciate the modeling courses most are typically part-time graduate students who work in industry or full-time graduate students who have previously worked in industry. Many of these students have worked on large scale software development projects and understand that software modeling is a valuable skill for software development.
b) Graduate students are capable of working on team projects much more independently than undergraduate students who need much more help.

Undergraduate courses:

a) Undergraduates tend to find it more difficult to appreciate software modeling concepts, although working on a software project helps them understand how these concepts can be used in practice.
b) Keeping lectures in sync with the project is a particular challenge for undergraduate projects, since they span requirements through implementation. It also means that project grading needs a quicker turnaround so that students can benefit from the comments in time for the next phase of the project.
c) As undergraduate students are less mature than graduate students, they need to have each concept explained very clearly, emphasized more than once, and reinforced with examples.
d) It is essential to provide rubrics to students for each assignment.

Industrial courses:

a) The industrial courses are typically attended by employees who need to understand software modeling and design for the projects they are working on and so are the most motivated to learn.
b) Students get the most benefit from the intensive 4-day course, since they apply what they have learned by solving design exercises.

VII Conclusions

This paper has described my experience with teaching courses on software modeling and design to undergraduate and graduate (Masters and PhD) students, in addition to in-depth short courses to industry. It has briefly described the contents of each course, how the teaching of each course has evolved, the teaching approach of "learning by doing", and lessons learned. Teaching these courses also led me to write books on software modeling and design [2, 4-7].

Although I developed all these courses, they have been taught by full-time and adjunct software engineering faculty at George Mason on a regular basis. For new course instructors, I provide all the course material to them and mentor them over the semester on a regular basis.

Teaching material for the courses on *Software Modeling and Design* and *Real-Time Software Design* are

available on the George Mason University web site [8]. The material includes course syllabus, sample course schedule, presentation slides for each lecture, and course assignments.

REFERENCES

1. P. Ammann, H. Gomaa, J. Offutt, D. Rine and B. Sanden, "A Five year Perspective on Software Engineering Graduate Programs at George Mason University", Proceedings SEI Conference on Software Engineering Education, San Antonio, TX, January 1994.
2. H. Gomaa, "Software Design Methods for Concurrent and Real-Time Systems", Addison-Wesley SEI Series in Software Engineering, ISBN 0-201-52577-1, 1993. (Also translated into Chinese by Pearson Education Asia Ltd. and Tsinghua Press, 2003).
3. H. Gomaa, "Courses on Software Design Methods for Real-Time Systems", Proceedings SEI Workshop on Real-Time Systems Education", Daytona Beach, April 1996.
4. H. Gomaa, "Designing Concurrent, Distributed, and Real-Time Applications with UML", Addison-Wesley Object Technology Series, ISBN: 0-201-65793-7, 2000. (Also translated into Chinese by Beijing University of Aeronautics and Astronautics Press, 2004).
5. H. Gomaa, "Designing Software Product Lines with UML: From Use Cases to Pattern-based Software Architectures", Addison-Wesley Object Technology Series, ISBN: 0-201-77595-6, 2005.
6. H. Gomaa, "Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures", Cambridge University Press, ISBN: 9780521764148, 2011. (Also translated into Chinese by China Machine Press, 2014).
7. H. Gomaa, "Real-Time Software Design for Embedded Systems", Cambridge University Press, ISBN: 9781107041097, 2016.
8. H. Gomaa, "Teaching Material for Software Modeling Courses", http://mason.gmu.edu/~hgomaa/