A graduate course in object-oriented analysis based on student-generated projects

ISSE-TR-94-102

Bo Sanden
George Mason University
Fairfax, VA 22030-4444
bsanden@gmu.edu

March 1, 1994

Abstract

This paper describes the experience from several offerings of a Software Requirements course based on the Object Modeling Technique by Rumbaugh & al. The paper describes the organization of the course, which includes a semester project carried out by groups of 3-4 students. Descriptions of 20 such projects are given, most of which were based on ideas from the student teams.

1 Introduction

This paper relates experiences with a graduate course in object-oriented analysis according to the Object Modeling Technique (OMT) [Rumbaugh]. In addition to regular homework exercises, the course includes a project where teams of 3-4 students use OMT to analyze and specify a system of their own choosing. Several examples are cited in an appendix. This element of the course emphasizes *actional* knowledge in that students must tackle a non-textbook problem. It calls for invention on the part of the students. Prospective employers desire that students acquire such actional knowledge and learn to appreciate the importance of innovation and invention [Denning].

Acknowledgement: Thanks to the numerous students in my Software Requirements class for their creative project ideas.

1.1 Institutional setting

George Mason University in suburban Washington D.C. has more than 21,000 students, 5,500 of whom are master's students, 500 doctoral students and 2,000 extended studies students. Of a total of 1,400 faculty, 643 are full-time. There are 35 master's programs, 12 doctoral programs and 8 certificate programs.

The School of Information Technology and Engineering (SITE) houses departments in Computer Science, Information and Software Systems Engineering (ISSE), Systems Engineering, Electrical and Computer Engineering, Operations Research and Engineering and Applied and Engineering Statistics. The emphasis is on graduate education: While four BS programs produce a total of 220 graduates yearly, seven MS programs – CS, ECE, IS, OR, Software Systems Engineering (SWSE), Statistics, and Systems Engineering (SE) – produce 250 masters yearly. A school-wide doctoral program in Information Technology and Engineering graduated 13 PhDs in 1992. Graduate courses are geared to students with full -time jobs and meet once a week for a 2 hour 40 minute evening lecture.

1.2 Software Systems Engineering Program

The Software Systems Engineering MS program belongs to the ISSE department and was started in the fall of 1989 based on the experience from the Wang Institute [Fairley]. It consists of six core courses plus four electives (two of which may be replaced by a thesis). The core courses are: Software Construction, Software Design, Software Requirements and Prototyping, Formal Methods and Models, Software Project Management and Software Project Laboratory. Each core course is offered every fall and spring semester. A number of software-engineering electives are offered, including Object-oriented Software Development, User Interface Design, Advanced Software Design, Testing and Quality Assurance and Software Engineering Economics. Further electives are provided by the other MS programs in SITE. The program is further discussed in [Sanden 1993] and [Ammann 1994].

In addition to the lectures, a typical course generally includes weekly assignments, midterm and final exams, and a group project. The project provides an element of realism, where the course material must be applied to something that has not already been formulated. The class size is typically around 40 resulting in about 12 teams of 3-4 students.

The projects are limited by the time frame of a semester course. Occasionally, it is possible for a project to span more than one successive class, such as the Requirements, Design and Project Lab classes, but this is the exception rather than the rule since the students do not move as one contingent from one class to the next. Thus, it is normally impossible to form the same teams in more than one class.

1.3 Software Requirements course

The Software Requirements course uses Davis's general text [Davis] and specializes in either object-oriented analysis (OOA) or prototyping. The OOA specialization has been offered several times based on the Object Modeling Technique (OMT) [Rumbaugh]. Earlier, the course was offered once based on the text by Coad and Yourdon [Coad]. While the projects outlined here could be analyzed based on the Coad and Yourdon notation, our experience is that Rumbaugh's book is superior in terms of coverage, examples and exercises. Although alternative texts have emerged, [Embley & al.], [Coleman & al.], etc., there has been no reason to change the adoption.

OMT includes an *object model* and a *dynamic model*. Objects are capsules of data (state) and operations. The object model describes the attributes (state variables) and operations of each object and the associations between objects. The dynamic model consists of concurrent processes and describes how the state of each object changes as a result of events. The dynamic model is based on Statecharts [Harel]. The object model is considered the primary model. The dynamic model is subordinate and created by endowing each relevant object with "life". OMT also includes a *functional model* based on data flow diagrams. This model is rather weakly integrated with the two others. The experience from this course has shown it largely unnecessary.

1.4 Project structure

The project setup is intended to resemble a real-world situation where a development team proposes a new product according to a rather broad agreement with a commissioner played by the instructor. Students are encouraged to think of topics early, based on a list of successful earlier projects. The lecture material provides further examples. The process starts with

a one-page synopsis submitted by the team. The instructor gives written feedback on the project synopsis based on the experience of earlier projects. It is important to ensure that the project can be done with a reasonable effort and result in meaningful object and dynamic models. Once an agreement on the product has been reached, the group undertakes a system analysis based on OMT. The solution can be regarded as part of the software requirements document.

Because of the large class size, formal interaction between teams and instructor is limited. In general, there is one meeting with each team, held at a point where working documents of an object model and a dynamic model are available. The meeting is usually quite intensive and very productive. It tends to generate a large amount of feedback as well as ideas for refinement and extension. Each meeting typically runs 1-1.5 hours. The team meetings replace about 2 regular lectures. A few teams tend to request one or two additional meetings. Most other communication between team and instructor is handled via electronic mail.

At the end of the semester, each team presents its project to the class. To make this reasonably efficient, one or two students from each team does the presentation usually within a 20-30 minute slot. An earlier "tag-team" approach, where each member presented their contribution was abandoned. That approach made it very difficult to stick to a presentation schedule. Because of the equal emphasis given to all parts of each project, much time was spent on aspects that turned out to be uninteresting. With about 12 teams, the presentations take 2 lectures. Although the experiences of teams are quite interesting and the presentation provides valuable examples in the use of the modeling technique, there is a lack of incentive for the students to attend the presentation. Attendance usually drops considerably compared to the regular lectures. Since it is difficult to actually test the students on the contents of each others' project presentations, it has been difficult to make attendance at the presentations reflect on their grades.

1.5 Evaluation

The idea of a collective project grade for each team was abandoned for a couple of reasons. Once completed, most projects tend to fall within a relatively narrow quality range. Also, a project grade might unfairly favor a weak student in a good team. Instead, the individual

student's contribution to the team effort is weighed into his/her final grade together with the exam results. The meetings with each team usually gives the instructor a fairly good impression of everybody's contribution. In addition, every student submits a confidential peer evaluation after the completion of the course. The peer evaluation tends to reveal cases where an individual has contributed little or nothing to a project. Reports of an outstanding contribution by some student are also common. Since some students are ill at ease with evaluating their peers, it has to be accepted that they turn in a bland evaluation reflecting an equal contribution by everybody. Even if this does not reflect the real contributions, it suggests that the team has worked well together. In spite of these efforts, it is not always easy to measure the individual contribution, so the grades tend to be dominated by the exam results except when there is reason to believe that the contribution to the project was considerably different from average.

2 Experience with different project ideas

A number of representative project ideas are outlined in the appendix. As a general observation, few projects result in both a significant object model and a significant dynamic model. To ensure a non-trivial dynamic model, students are encouraged to look for problems that involve significant series of events over time. This excludes many typical data processing projects that might have led to interesting object models. The object model is most useful when either there is a non-trivial database or an exact definition of concepts is necessary. In the *robot golf course mower* problem (2) [1], the various areas in a golf course give rise to an interesting generalization structure. A similar structure is found in the *aircraft carrier* problem (16).

A class of systems with disappointing dynamic models are those largely based on feedback. These include the *fish tank* (7), the *home heating system* (9) and *home security system* (3) and the *car auto-pilot* (8). While these systems may seem dynamic, they tend to decompose into multiple objects each with a single state and a regular action of the type "take sample and adjust output".

A class of non-trivial dynamic systems that cannot be handled satisfactorily with OMT

[1] The numbers in parentheses refer to the project descriptions in the Appendix.

are control systems involving multiple processes sharing resources. These include the *jukebox* (4), where a money insertion process and a playing process share the request queue object, the *automated switchyard* (15), the *aircraft carrier* (16), the *automated parking garage* (12) and the *automated lubrication service* (17). OMT provides no obvious way to represent and reason about such resource-contention problems. [Sanden 1994]

A class of problems includes user interfaces of the same general description as the ATM in Rumbaugh. These are easily modeled on the ATM. A difficulty with the state model in this kind of problems is that the lack of structure may conceal errors. We have found that the Jackson diagram notation [Jackson], [Sanden 1994] is a more explicit although much more demanding notation that allows you to easily verify regularities such as: "each transaction resulting in dispensed money produces a receipt". (In the Rumbaugh solution no receipt is printed if the subtransaction where money is dispensed is followed by a second subtransaction that is cancelled.)

3 Conclusion

Several offerings of the Software Requirements with an object-oriented specialization have shown that most students easily come up with project ideas and appreciate the freedom to choose for themselves. Only a few teams in each class tend to lack the necessary imagination, and do a variation of an earlier project. That way, a rich set of examples is built over time. It has not been the experience that the stronger students necessarily have an easier time coming up with project ideas; the difference instead shows in the resulting models.

Paradoxically, it is easier in many ways for the instructor to deal with several different problems rather than many solutions to a single one. These slightly different solutions are hard to keep apart, and there is a temptation to inappropriately steer all teams towards one preconceived solution.

Readers are encouraged to try this approach in their own teaching, using some of the examples given in the appendix as inspiration for the students. Although the experience related here is with graduate students, there is no particular reason why the same approach cannot be used in upper-level undergraduate teaching. While graduate students might be expected to draw on their work experience for project ideas, this seldom happens in reality.

A similar approach to the semester project has been tried in Software Design, the Software Project Laboratory and Advanced Software Design. These courses were based on the *entity-life modeling* approach to concurrent software design [Sanden 1994]. The experience from those classes is equally positive.

4 References

Ammann,P. Gomaa,H. Offutt,J. Rine,D. Sanden,B. A five-year perspective on Software Engineering graduate programs at George Mason University in *Software Engineering Education, Proceedings, 7th SEI CSSEE Conference, San Antonio, Jan. 1994*, Springer Verlag, 473-488

Coad,P. Yourdon,E. *Object-oriented Analysis*, Yourdon Press, Prentice-Hall 1990

Coleman,D. Arnold,P. Bodoff,S. Dollin,C. Gilchrist,H. Hayes,F. Jeremaes,P. *Object-oriented Development: The Fusion Method*, Prentice-Hall 1994

Davis,A.M. *Software Requirements - Objects, Functions, & States*, 2nd Ed. Prentice-Hall 1993

Denning, P. Designing new principles to sustain research in our universities, *CACM* 36:7 (July 1993), 99-104

Embley,D.W. Kurtz,B.D. Woodfield,S.N. *Object-oriented Systems Analysis*, A Model-driven Approach, Yourdon Press 1992

Fairley,R.E. "A post-mortem analysis of the Software Engineering programs at Wang Institute of Graduate Studies", *ACM Sigsoft Software Engineering Notes*, 13:2 (April 88), 41-47

Harel,D. Statecharts, a visual approach to complex systems in *Science of Computer Programming 8*, North Holland, 231 -274 (1987)

Jackson,M.A. *Principles of Program Design*, Academic Press, New York 1975

Sanden,B. An Ada-based, graduate Software Engineering curriculum at GMU, ASEET Symposium, Monterey, CA, Jan 1993

Sanden,B. *Software Systems Construction with Examples in Ada*, Prentice-Hall, 1994

A Appendix: Project topics

A number of representative project ideas are outlined below. The descriptions are brief; they are not intended as sufficient project descriptions but rather to suggest what kind of projects work. Students should instead be encouraged to come up with ideas from their own experience.

A.1 Airport parking system

The system controls the access to the different parking lots at an airport. Cars are identified upon entry to the airport premises and assigned parking spaces depending on the duration of intended travel, etc. They are charged when they leave the area. Frequent airport users may charge their parking fees to accounts with the airport. Airport vehicles, shuttle buses and drop-off/pick-up vehicles are excluded.

The object model includes a visit class with the subclasses parking-visit and drop-off/pick-up visit. A parking-visit has the subclasses frequent-user-visit and infrequent-user -visit. A frequent-user-visit is associated (many to one) with a frequent-airport-user class. The dynamic model includes a state diagram of a visit beginning in a state outside and including events such as enter airport domain, enter legal parking space, enter illegal parking space. In the case of illegal parking, events such as get warning, receive citation, etc., follow.

A.2 Robot golf course mower

A robot lawn mower automatically mows the various playing areas at a golf course. A database describes the topology of the golf course including the various cuttable areas of each playing area: tee, fairway, green, rough and fringe. Each area has a reference point. The mower navigates between these reference points along separately defined pathways. Within each area, the positions of known obstacles to the mower are also stored. The mower has the capability to identify new obstacles that it encounters.

A generalization hierarchy describing different kinds of areas is prominent in the object model. The class area has attributes such as reference point and operations such as contains position(p) which returns true iff p is within the area. The subclasses of area are: cuttable, obstacle and pathway. Further down in the hierarchy, a green class has the attribute cut orientation reflecting the direction the green needs to be cut next time to achieve a desired cutting pattern.

A Jackson diagram [Jackson], [Sanden 1994] was used in the dynamic model of mower behavior, which is conveniently seen in terms of nested iterations: The life of the mower is an iteration of work shifts. Each shift includes an iteration over playing areas, and within each area, there is an iteration over the cuttable areas. Cutting is represented as an iteration over cutting path segments, and transportation between areas is an iteration over route segments.

A.3 Home security system

The security system is used to protect a family home and its grounds. It monitors the home and grounds for intruders, detects smoke and fire and tracks legal home occupants.

In addition to the obvious model of the house as an aggregation of its physical parts, the object model defines a class entry with the two subclasses illegal entry and legal entry. These subclasses have associations with the outside door, windows, etc., showing that the door is the only legal entry point.

The object model of the security system itself includes an example of multiple inheritance in that a telephonic device is both an access unit (used to change system parameters) and an alarm unit (used by the system to report a breach of security, a fire, etc., to the authorities).

The object model also defines the creatures that the system must deal with. The creature class has the subclasses intruder and non-intruder. Non-intruder has the subclasses human and pet, where human has subclasses reflecting whether this human non-intruder is authorized to manipulate the security system.

A.4 Jukebox

This project specifies the software necessary to run an ordinary jukebox. Customers are allowed to insert coins and select songs from a number of independent stations, placed perhaps at each table in a diner. Selected songs are entered in a request queue. The jukebox mechanism retrieves one song at time from the queue, operates an arm that retrieves and mounts the appropriate compact disc from storage (if it is not already mounted), and plays the song.

A.5 Package exchange system

The automatic package exchange system allows users at different locations to send letters and parcels to each other. A pneumatic technique may be used for transportation. The system consists of customer-service stations where customers send and receive packages, tubes and switches.

If the system is operated on the basis of "package switching", that is, packages are buffered at each switch before being forwarded, the problem gives rise to a textbook state model of a single package. If the system is instead based on circuit switching, the problem becomes one of resource sharing, where a package must obtain exclusive access to a series of tubes (and switches) before being sent. With each package needing simultaneous exclusive access to multiple resources, some technique must be employed to prevent deadlock [Sanden 1994].

A.6 Traffic control

The system controls the traffic lights at one or more road intersections. The object model introduces and defines concepts such as lane group which is a group of lanes that have a green light at the same time. (Each traffic lane may belong to one or more lane groups.) The order in which different lane-groups get to go is represented by means of the one-to-one association turns-green-before between lane-groups.

A.7 Fish tank

The system automatically monitors a fish tank, performing tasks on a daily, weekly and monthly basis. It controls food delivery, light level, water temperature, water level, and pH level and effectuates water cleaning through a filter. The user sets initial controls and modifies settings as necessary. The system sounds an alarm and displays appropriate messages when values fall outside set limits.

The object model includes an aggregation of devices and a generalization structure of various kinds of devices. The dynamic model consists of state diagram for food control, light control and chemical control. The control of water temperature, etc., is a feed-back system. Feeding and other functions operate based on the time of day.

A.8 Auto-pilot system for cars

A number of projects deal with automation of cars. The auto -pilot enables vehicle operation on a limited-access highway system. It monitors the vehicle position relative to the road and surrounding traffic, controls the speed, and monitors vital vehicle functions (fuel level, etc.).

These projects are usually disappointing from the point of view of OMT. The object model tends to reduce to the rather obvious aggregation of vehicle components (sensors, actuators, etc.). The dynamic model tends to reflect the feedback nature of the problem: the state diagram of each sensor tends to consist of one (or two) states and periodic actions.

A.9 Home heating system

The home heating system allows the inhabitants to preselect desired temperatures for different times of day. The system anticipates the beginning time for each desired temperature and starts heating or cooling in order to reach the target temperature at the desired time. The system also senses the arrival of an unexpected visitor and adjusts the temperature accordingly.

A.10 Automated tennis player

This is a robot that can be used as the automatic opponent of a human tennis player. The robot moves by means of turnable wheels and handles a racket in much the same way as a human player. By means of sensors in the robot's and the human's rackets, the robot calculates the trajectory of the ball and the move it must make to hit it. Based on a knowledge of its own parameters, it determines whether it can make it in time to return the ball.

This is clearly a fanciful project which requires a certain leap of faith in technology. In general, the experience with fairly unrealistic projects like this one is not worse than with more implementable ideas.

A.11 People mover

The people mover is an automated vehicle for the transportation of people in an environment such as an airport. The people mover navigates based on emitters placed at suitable distances along the pathways and can automatically carry out a traversal from a point A to a point B within its intended range. It has the ability to detect and avoid stationary and mobile obstacles.

As is often the case, the object model forces a clear definition of the central system concepts. The terminal building (or buildings and connecting spaces) is broken into areas. Each emitter is associated with an area and uniquely defines a single point. A route traveled by the mover consists of route segments with start and end at such a point.

The dynamic model includes a user-interface diagram similar to the ATM example [Rumbaugh]. (In these simple terminal systems, many states are conveniently associated with the screen that is currently displayed, such as a credit card screen prompting the user to insert a card.)

The dynamic model of the behavior of the mover is rudimentary since the technique for navigating around an obstacle had be largely left outside the scope of the project.

A.12 Automated parking garage

The automated parking garage consists of parallel aisles with two levels of parking stalls on either side. The aisles are connected by transportation roadways. (Other configurations are equally possible.) The garage is served by automated carriers. Each carrier picks up a car that has been parked by its driver in a designated entrance module and transports it to a free parking stall. Upon request, a carrier retrieve a car from a stall and moves it to a designated exit module, where it is picked up by its driver. There is a wait area for idle carriers.

The challenge lies in the management of the automated carriers. Multiple carriers are considered necessary for performance reasons, so the aisles and roadways must be managed as shared resources. The aisles are allocated to one carrier at a time, allowing the carrier the necessary time to load or unload. Carriers are not allowed to wait in the roadways, where they would impede each other's movements.

A.13 Automated store

Various examples of automated stores have been used. Some are similar to the people mover or the automated garage in that they involve either shopping or stocking robots that navigate the aisles while avoiding obstacles. A more realistic variation involves automated shopping as follows: Barcoded samples of various merchandise are displayed in the store. Each customer uses a hand-held electronic unit that scans barcodes and where the quantity ordered of each item is keyed-in. At a suitable point in the process, the hand-held unit creates an order to the storage facility, where the customer's order is assembled. Soon after the customer reaches the checkout, the goods are delivered. Provisions must be made for the customer to change the order by deleting items or reducing the quantities.

This problem gives rise to a dynamic model of a customer transaction basically consisting of item selection and checkout processing. A concurrent series of events starts with the order to the storage facility where items are picked and the order assembled.

A.14 Video rental store

It is easy to imagine the automation of some highly structured retail operations. A video rental store may have number of terminals where customers browse a catalog and select tapes. The store is manned by robots that retrieve the desired tapes and deliver them to the customer positions. The same robots also service the return chutes. There is a special return chute for damaged tapes.

The dynamic model includes a user interface somewhat similar to the ATM [Rumbaugh]. A tape state diagram reflects events such as rent, return and return-damaged. A robot model includes the necessary steps to take upon return of a tape, etc. (Identify the tape based on its bar code, re-shelve it, etc.)

The robots are assumed to travel along a one-way track that passes by the tape shelves, the customer positions and the chutes. There is also a siding for idle robots.

A.15 Automated switchyard

In the automated switchyard, a computer controls the disassembly and reassembly of trains by means of remotely controlled switch engines. An incoming train is placed on a particular siding. Individual cars or blocks of more than one car are detached and moved by switch engines to a sidings where trains for different destinations are being assembled. The sidings are arranged in a tree structure. A resource contention problem arises when different switch engines need to travel over the same segments of rail and past the same turnouts to reach their respective sidings. [Sanden 1994]

A.16 Aircraft carrier

The aircraft carrier system deals with the management of aircraft on board the carrier. The movement of aircraft raises problems similar to those in the switchyard when the different aircraft need exclusive use of the same elevator between flight deck and hangar deck, etc.

The object model includes classes such as aircraft, aircraft type, pilot, and sortie, and an association certified between pilot and aircraft type. The different kinds of areas on-board

give rise to a generalization hierarchy similar to that in the robot lawn mower problem.

The aircraft dynamic model starts in a state parked and unmanned and covers the events involved in manning the aircraft, moving it to the launch site, take-off, landing and movement back to the parking area.

A.17 Automated lubrication service

This project envisions an automated facility that undertakes standard vehicle service on a number of cars placed in individual bays. A number of robot arms travel on rails in the ceiling to reach the various caps on the car engine (oil, radiator fluid, etc.) Additional arms operate from under the car and access the oil drain plug, etc. The arms have the ability to open a plug, take care of the draining liquid and re-close the plug, or open a cap, insert liquid, sense the filling level and re-close the cap. A database must know the position of each cap and plug relative to certain reference points in the car.

A.18 Home toilet

The home toilet project is unique in that it has no software connection but shows how an object model and a discrete -event dynamic model can be used to describe a well-known continuous system. Essentially, the toilet consists of a bowl, a tank, a filling pipe that allows water to enter the tank and a port that allows it to enter the bowl. The states reflect the level and movement of water in the bowl and tank respectively. For each one, there is an idle state where no water is moving, a filling state with a net inflow of water and an emptying state with a net outflow. Events include the movement of the trip lever handle to start the flushing cycle, and those defined by the hydrodynamics of the toilet itself: The water level reaches a certain minimum where the ball cock valve closes and a transition from emptying to filling occurs. A similar event exists for the bowl when the emptying stops and the bowl start filling up to its idle level.

A.19 Luggage tracking

The luggage tracking system operates either in an airport or for an airline. It is basically a database system keeping track of customers and their baggage. Data is captured when a piece of luggage is checked in, when it is loaded on an aircraft, is off-loaded, arrives in the baggage pick-up area, etc. The life of a piece of luggage reflecting these events gives rise to an interesting dynamic model.

A.20 Programmable remote controller for home electronics

The controller is a sophisticated remote that controls not a single device but a set of home entertainment devices such as a TV set, a VCR, a record player, etc. At a given time, a number of different devices can be registered with the device, which stores the operations of the device's original remote control. The device can store a number of different macros that can be recorded, edited and executed by the user.

The object model is essentially an aggregation showing the various buttons, displays, etc., of the controller. The dynamic model reflects the state of the controller as a whole. The user interface is non-trivial but sequential and reflects the normal, name and edit modes used for normal operation, for registering new devices and for macro editing, respectively. In the normal mode, there is concurrency between the manually keyed-in commands and the commands made by executing macros.