Teaching Object-oriented Simulation in a Software Engineering Framework

Andrew J. Kornecki
Department of Computing and Mathematics
Embry Riddle Aeronautical University
Daytona Beach, FL 32114
phone: (904) 226-6888
daytona@erau.edu
fax: (904) 226-6678
korn@erau.db.erau.edu
http://faculty.erau.edu/korn/

1. Introduction
Simulation software has become an important element in the modern world. It touches all aspects of our life from space exploration to home appliances to leisure time. The method of software development has evolved through the years, adding organization and engineering discipline to computer programming. What used to work well for small programs developed by a single programmer, does not work for large applications developed by a team of software engineers. Simulation software, as any other software, has evolved from a piece-by-piece craft to an engineering creation with a defined structure and format.

Most undergraduate computer science and engineering programs offer an initial sequence of classes teaching the fundamentals of computer science and introducing practical knowledge of programming. The programming is being taught bottom-up, introducing language syntax, control structures, and building small programs (less than 100 lines of code). Subsequently, the material introduces subroutines and modularity, and more complex programs are presented. Traditionally, the courses have used procedural-oriented programming style as the building blocks for subsequent learning. It is no surprise that in each year object-oriented methodology (including its three main aspects: analysis, design, and programming) has been gaining more and more acceptance from the software industry. This acceptance is preceded by multidisciplinary research in the fields of simulation, artificial intelligence, computer science, and software engineering. Year after year new computer science and engineering programs produce graduates that join the ranks of software systems developers. Software industry representatives frequently complain that the newcomers lack skills in practical aspects of software engineering. Another deficiency of undergraduate programs has been caused by traditional approach to teaching procedural programming. In the last few years more programs started using C++ or Java for introductory computer science classes. However, in many cases the teaching methodology did not change and the objects are introduced as an afterthought. This paper describes a junior-level course in an undergraduate computer science program, designed to teach object-oriented design and coding, while introducing discrete modeling and computer simulation concepts. Additionally, the course provides a vehicle for development of small team class projects addressing critical aspects of the modern software development life cycle. The author’s experience with teaching the course may serve as a model for similar offerings in other colleges and universities.

Keywords: object-oriented methodology, discrete simulation, software engineering, education
the junior/senior year students have had difficulty facing problems requiring a different approach (top-down) and often a significantly different programming style.

The object-oriented paradigm has found its way into computer science curricula. Several academic programs have introduced object-oriented programming, reporting both accomplishments and problems [1, 2, 3]. A major pitfall of teaching object-oriented programming, as the primary instructional objective, is the existence of a large number of software artifacts outside academia, requiring students to have a thorough knowledge of the conventional approach. It is the premise of this paper that object-oriented paradigm may be introduced into computer science and engineering curriculum in a natural way while supporting modeling and simulation activities.

The paper concentrates on teaching development of software for discrete simulation using object-oriented techniques and methodologies in the framework of a modern software engineering process. We shall discuss software engineering issues related to object-oriented methodology. The main body of the paper describes and discusses implementation of undergraduate Modeling and Simulation course focusing on object-oriented software development.

2. Object-oriented Analysis and Design–Software Engineering Issues


The analysis phase must result in a "...complete, consistent, readable, reviewable by interested parties, and testable against reality..." description of the problem [5]. This phase allows the developer to identify key objects, their properties, functionality, and the relationships between them. The emphasis is on objects interaction via the mechanism of message passing. In the initial design phase critical strategic decisions must be made about the system (selection of platform and tools, architecture, communications). The approach supports system decomposition and allows developers to proceed with a series of deliberately selected partial design prototypes, each of them modeling important aspects of the design while advancing the system functionality. This spiral development process [6] eliminates rigidly applied waterfall life-cycle.

Evolutionary development, or prototyping, distinguishes object-oriented approach from a traditional structure-oriented methodology. It combines coding, testing and integration. The series of implementation prototypes, of increasing functionality, resulting from successive refinement, gradually converges into a final product. Evolutionary implementation must first achieve the basic functionality of the application. If an enhancement of functionality or a modification is required, the implementation must reflect the design, with spiraling back to the design. The performance issues are usually addressed when the system achieves appropriate level of functionality conforming to the requirement specification document [7]. When such approach is used in the classroom the instructor’s challenge is to keep students from "hacking the code" rather than to maintain the formal project structure. It is critical to enforce proper project documentation and keep track of the design (and often requirements) changes when the functionality of the system is being enhanced. This is where a disciplined software engineering approach is used.

3. Object-oriented Programming–Implementation Issues

The real world consists of entities (or objects) with their identity, attributes, behavior, and associations. It is fairly natural to model the real world this way. Objects are also excellent building blocks for conceptual modeling. Teaching object-oriented concepts as a part of a modeling and simulation course allows the instructor to introduce them in a natural way (as opposed to "forced" introduction of classes in introductory C++ or Java language course). Since the class libraries serve as a backbone of the object-oriented software development, we use pre-defined libraries supporting discrete simulation, graphics, and animation.

In the late 60’s Simula, a language supporting simulation [8], introduced concepts of classes (integrating data and functions) and inheritance. The attempts to build "pure" object-oriented tools is exemplified by Smalltalk [9] and Eiffel [10]. The alternate direction of object-orientation focused on modification of the existing languages. The most popular C++ was developed in early 80’s [11]. In this category, we may list Common Lisp Object System [12], Object Pascal [13], and Ada95 after a face-lift from the 83’ version.

And then there is Java – developed by Sun Microsystems as a "simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language" [14]. With exception of simple data types, everything else is treated as objects. Java, due to the Internet bindings and growing application base, has gained many supporters in the development community.

The fact of using a specific programming language by no means indicates that the programming style will adapt accordingly. We have seen programmers writing Byzantine control structures in LISP, strug-
gling with the PROLOG goal oriented concept, and writing perfect procedural code in C++. It is very difficult to change one's programming style. It is therefore important to teach the software developers about various styles and tools. It is also important to give them early hands-on experience with a realistic project providing a vehicle to practice object-oriented approach.

4. Course Implementation

The Modeling and Simulation course, in addition to teaching discrete simulation, has two major advantages. It does teach object-oriented programming and also introduces team project allowing students to revisit the elements of software development life-cycle. The course is offered in the junior year of undergraduate computer science program. The program follows the guidelines of the ACM's Curricula [15]. Before registering for the course, students are exposed to computer science fundamentals including programming, data structures, and elements of software engineering, and subsequent courses on organization of programming languages and computer graphics. The useful topics introduced by these courses are: programming styles, software development life-cycle, and interactive interface. Also, the students have a sound background in discrete math, calculus, and probability and statistics [16].

4.1. Goals and Objectives

The Modeling and Simulation course introduces the basic aspects of modeling and simulation applied to real-life problems. The aviation and aerospace focus has been dictated by the specifics of the domain-centered curriculum [17], with other domains that may be easily substituted for aviation. The course introduces statistical models, queuing theory, random variate generation, simulation languages, object-oriented analysis, design and implementation, design and analysis of experiments, verification and validation of a simulation model. The course concentrates on discrete systems, representing real-world phenomena with multiple concurrent entities interacting within some physical environment.

Simulation tools like SIMULA, CSL, SIMSCRIPT, GASP, GPSS, SIMPAS, SIMON, Arena, or ProModel have been successfully used in industrial and scientific applications [18]. Their common feature is that they allow user to concentrate on simulation study, rather than write thousands of lines of supporting code in a general purpose high-level language. In addition to supporting a simulation project with its statistical implications, they include various utilities for controlling the simulation time, model building, and often a user-friendly graphic interface. They differ in the extent of "coding" required to imitate simulated system. As the programming aspect is important in computing curricula, we selected MODSIM simulation language developed by CACI [19]. MODSIM is a multi-platform product running both in UNIX and Windows environment. It is a complete high-level object-oriented language with a rich library of predefined objects supporting discrete simulation and graphic interface.

The primary objective of the course has been to give students a hands-on experience with object-oriented programming. In the process students have an opportunity to identify basic paradigms and concepts in system modeling and simulation, understand the sequence of computer simulation activities (problem statement, data acquisition, object-oriented system analysis and model design, simulation experiment, verification, validation, documentation) in the context of the software development life-cycle. They develop discrete computer simulation programs utilizing event and process oriented approach with a time scheduling mechanism. The course allows students to learn and appreciate the application of simulation techniques and methods, particularly in an aviation and aerospace domain. Additionally, the course introduces mathematical and statistical models, simulation languages, and gives brief review of queuing applications.

4.2. Programming Assignments

The first part of the course uses sequence of four individual programming assignments building object-oriented design and implementation skills. The four programs, each with roughly two weeks delivery period, are:

- text-based airline scheduling operations
- airline crew assignment with text menu

Table 1.
• simulation of en-route air traffic control operations with text output
• graphic-based simulation program to imitate en-route air traffic control operations

The objective of these assignments is to get students familiar with object-oriented development environment using the library of pre-defined objects. After using primitive text output, file and related I/O manipulation, the subsequent programs introduce interactions with the user, dynamic simulation, random numbers, statistical data manipulation, and graphic user interface. The programs are designed in such way that the students can re-use objects created in the earlier assignment and create new program by adding functionality to the existing code – in line with the evolutionary software development. MODSIM, the selected development platform, supports clear distinction between the front (planning, requirements, design) and back (coding, testing) phases of development by separate DEFINITION and IMPLEMENTATION program components. This allows the developer to do the design defining all objects with their attributes and methods (describing the interfaces between objects) in the DEFINITION part, deferring the coding to the IMPLEMENTATION part.

The students, familiar with Personal Software Process concepts [20], are required to collect data regarding the size and effort. Table 1 shows the data collected in the Spring 1999 offering of the class. The code size averages 0.65KLOC, with about six objects and twenty methods. The total time spent on the project over two weeks averages c. 18 hours.

The individual students’ data, shown in Figure 1, are supporting theory that careful planning and design reduce time spent on coding and testing. The collected data show that the students spending less time

![Figure 1](image)

<table>
<thead>
<tr>
<th>Students who spent:</th>
<th>Code/test average</th>
<th>Plan/design average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less than 6 hours code/test</td>
<td>4.5 hours</td>
<td>6.3 hours</td>
</tr>
<tr>
<td>Between 6 and 12 hours code/test</td>
<td>8.8 hours</td>
<td>5.9 hours</td>
</tr>
<tr>
<td>Over 12 hours code/test</td>
<td>21.5 hours</td>
<td>4.8 hours</td>
</tr>
</tbody>
</table>

Table 2.
4.3. Team Project

The second part of the semester is used for a team project divided into two phases.

The team of 4-6 students prepares project plan with specifically assigned roles and responsibilities schedule and the task assignment. Each team meets for a session with the client (course instructor) on a Project Launch meeting. The session is designed to identify specific goals and objectives, give a feedback on the project plan, etc. The subsequent 2-3 weeks period is used to submit a Mid Phase report containing software requirements and early design. This part also includes analysis of input data, design of the experiments, and selection of the performance measures. The Final Phase of the project deals with the implementation, testing, simulation runs, and output data analysis. A record of the team activities includes meeting attendance, time, problems discussed, problems resolved, tasks assigned, and the areas of responsibility.

As in real life the project has its client. The course instructor plays this role of the future simulation user. Frequent consultations with the client are required. The project often involves additional activities such as collecting data, interviewing the domain specialists, reading research papers, or exploring the feasibility of implementation by writing and executing small prototype programs. The projects are designed to use simulation to solve a specific problem. Typical problems include simulation of airport runway system, airport passenger terminal operation, en-route air traffic control sector, aircraft maintenance facility, airport luggage operations, etc. The objective of the project is to analyze the current hypothetical situation and proposed better solution by change of the system structure or parameters. Two project phases, each of 2-3 weeks duration, represent the following steps in a simulation study:

Mid Phase:
• Problem Statement
• Conceptual Model Building
• Design of Object-oriented Discrete Simulation
• Analysis of Input Data

Final Phase:
• Evolutionary Implementation of Object-oriented Discrete Simulation
• Graphic Output
• Simulation Experiments
• Analysis of Simulation Runs

The project is completed with demonstration of working simulation, presentation of results, and the report posted on Internet.

4.3.1 Mid Phase

The phase focus is on description of the problem, operations of the system to be simulated, simulation objective, measures of system behavior, identification of the system environment and available data sources. The system is described in terms of objects with their attributes, behavior, and interactions. The system parameters, decision variables and performance measures are identified and the DEFINITION part of the program is completed. The design supports the concepts of reusability, since the students not only use the available rich standard MODSIM library of objects, but also use the existing code examples with objects serving similar purpose. The reuse principle allows the students to create inherited classes and produce working prototype of significant size in a relatively short time. A new challenge is to explain the reuse concept to the students, who until now are being told that plagiarizing is a bad thing to do. The side-effect of this phase is application of software engineering techniques such as project planning, requirement collection, and system specification. Also a significant educational objective of this phase is hands-on experience with object-oriented design.

Activities in this phase are reflecting a conventional simulation class. It focuses on analysis of pre-defined data prepared by instructor reflecting the timing relationships for system-critical simulation entities (e.g., passengers, aircraft). The type and parameters of the distributions representing the input data set (including building histograms, computing statistics, hypothesis testing, etc.) form a base for the numerical values of program parameters.

4.3.2. Final Phase

After the feedback from the Mid Phase, the team continues with code development and testing of the accepted design. The idea is to introduce functionality one step at a time completing the IMPLEMENTATION. For a simulation study, modeling an element of the client domain, it is critically important to accomplish customer active participation in the development process. The software output needs to be presented in a manner easy to understand and to interact with. Enhancing the system functionality with graphics and animation is the most fun for students. The client (domain specialist) can interact with the system, change the system inputs and see the immediate effects of introduced changes. This activity constitutes an important part of the validation process. Often the simulation system acceptability depends on an appropriate graphical user interface. For the purpose of the educational objective, the Final Phase gives the students an additional opportunity to apply the concepts of reusability, inheritance, and get a hands-on experience in an event driven programming, where the system reacts to user input.
The completion of this phase produces fully functional simulation software. To complete the project students carry out the simulation experiment to solve the assigned problem formulated earlier as the simulation objective. They use the simulation experiment design, as developed in the Mid Phase, and replicate simulation appropriate number of times to obtain statistically significant results. The results need to be further validated by computing output statistics and the confidence intervals [21]. Upon completion of the simulation experiments students prepare the Term Project Report. The report can be viewed on the Web and includes: Introduction, Problem Statement, Simulation Objective, System Description, Object Model, Input Data Analysis, Simulation Code, User's Manual, Sample Output, Analysis of Experiments, Conclusions, and Supporting Data (logs, references).

Each team presents the project in class in a brief presentation. The presentation includes simulation objective, system description, program demonstration, and an outline of the simulation results. Each student submits, on e-mail, a confidential conclusion section identifying the problems encountered and both self- and peer evaluation.

5. Conclusions
Object-oriented methodology has established its mark on simulation software and seems to be here to stay. The critical issue is to educate new generation of software developers to understand the principles of discrete simulation concurrently with object-oriented methodologies and good software engineering practices.

Despite the apparent success of offering Modeling and Simulation course in a described format, evident by the comments from the industry and the graduating students, there is still room for improvement. More class material can be shifted from the theoretical aspects of simulation techniques (for example random number generation and testing, or elements of queuing theory) to instruction enhancing the concepts of object-oriented software development. This change is in response to the input from industries employing our graduates. There is evidence that the offering could be significantly improved if we could expand on real-life aspects of software development. The omission of these aspects is treated by the industry as a serious deficiency of the undergraduate education in computer science and engineering. On the other hand, the course shall keep the breadth of coverage to provide students with appropriate background for continuing their education.

The topics related to Object-oriented methodology have rich literature. However, a consistent answer to the most efficient teaching of object-oriented software development is still an open issue. Students cannot rely on a wealth of personal experience in the development of object-oriented simulation software in a team environment. Also the time-frame for development, limited to one semester only, and the part-time nature of the undergraduate student work calls for a more structured and disciplined process than in a conventional workplace.

The developed curricula and the class materials may be used as a blueprint for other undergraduate computer science, computer engineering, and future software engineering programs. The course materials, from the last course offering (Spring 1999), are accessible on the Web (http://faculty.erau.edu/korn/350s99/s350s99.html) and they can be adapted and modified for local hardware and software platforms.

While the paper was submitted and accepted, the CACI Product Company, the owner and developer of MODSIM, has been taken over by CompuWare. The new parent company claims that they will continue to support MODSIM for current users, but the future of this tool on the long range is uncertain. A mailing list related to this issue has been established at: http://www.cgroups.com/group/Modsim. However, the presented approach can be adapted using different application platform using developed libraries for C++ and Java supporting discrete simulation. Example of such may be SimPack Toolkit developed at University of Florida [22].

6. References

Dr. Andrew J. Kornecki was teaching and doing research at the University of Mining and Metallurgy in Krakow, Poland while earning MSEE (’70) and PhD (’74) degrees. Subsequently, he was on faculty of the Garyounes University, Benghazi, Libya and the University of Kentucky, Lexington, KY, before joining Embry Riddle Aeronautical University, Daytona Beach, FL. His expertise includes software development, computer organization, verification and validation, modeling and simulation, real-time and safety-critical systems, and computer performance analysis. He contributed to research on Intelligent Simulation Training System and served as a visiting researcher with the Operations Research Service of the Federal Aviation Administration. He has been involved in activities of the Society for Computer Simulation, including serving on SCS Board of Directors. Recently, he has been involved in research on safety critical system verification. He can be found on http://faculty.erau.edu/korn