Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

TEACHING THE CLASSICS OF SIMULATION TO BEGINNERS (PANEL)

Ingolf Ståhl (Moderator)

Stockholm School of Economics
SE-113 83 Stockholm, SWEDEN

Henry Herper

Otto-von-Guericke University
D-39106 Magdeburg, GERMANY

Raymond R. Hill

Wright State University
Dayton, OH 45435, U.S.A.

Catherine M. Harmonosky

Penn State University
University Park, PA 16802, U.S.A.

Joan M. Donohue

University of South Carolina
Columbia, SC 29208, U.S.A.

W. David Kelton

University of Cincinnati
Cincinnati, OH 45221-0130, U.S.A.

ABSTRACT

In order to get more people to use and understand simulation, improved teaching of simulation to beginners is important. The panel members share their experience in teaching the classic systems of simulation, used for several decades, to novice students.

1 INGOLF STÅHL

Although discrete simulation is a very powerful tool for the analysis of many different problems in industry, the usage of simulation is indeed surprisingly low, in comparison with its potential. It is my main belief that the greatest problem with the spread of simulation is that the teaching of it is so limited. Only a small percent of business and engineering students get enough schooling in simulation to be able to appreciate the fundamental advantages and limitations of this tool. I therefore think that an improvement in the teaching of simulation to beginners is very important. Hence, the purpose of this panel is to try to give people who in the nearby future will be teaching simulation to novices in the simulation field some ideas of how this can be done by giving examples of how this has been done in recent years.

I shall start by trying to explain the title of the panel: Teaching the Classics of Simulation to Beginners. Let me start with "beginners". When I talk about beginners, I am not referring to students of computer science, but to business students and to engineering students who study areas like production, logistics, supply chain management, etc.

We here mainly talk about a first, and usually only, simulation course that they get, corresponding to 10 – 12.5 percent of a student work year. There can also be other types of introductions to simulation. From my own experience, one possibility is a ten-hour **part** of another course, e.g. focused on operations research or computer methods, giving almost the same knowledge in a simulation system as the full course, but not dealing with a real project. Another alternative is a four-hour rapid introduction to simulation modeling, leading to simple service system models.

We envisage the student in the future to become mainly an intelligent buyer of simulation services, but also able to produce a first simulation prototype that in some cases is developed further by a computer scientist, but in other cases used directly in a "quick and dirty" fashion for solving an urgent problem.

We are finally also very interested in the growing trend of teaching simulation at the high school level, since it is important to get students early into simulation.

The word "classics" can, as we will hear from our panel, be interpreted in several different ways. It can refer to simulation systems that in various forms have been taught to beginners for two decades or more. Since we in the Educational track, also have a sister panel, called *Simulation Textbooks, Old and New*, and we like to have a connection between these two panels, we can see the "classics" as the simulation systems that have dominated the major text books during the last decades.

Obviously there is a plethora of alternatives as regards software to be used for teaching beginner. For example, at the last WSC there were over a dozen software products, which the vendors could claim were suitable for teaching beginners. Obviously this panel can discuss only a very limited number of alternatives. Furthermore, having each member of the panel represent one software product would turn the focus of the panel in the wrong direction, making it a panel arguing the merits of different types of software. The task of this panel is rather to show e.g. how the same type of software can be used in different ways for the same purpose of getting beginners into simulation.

This focus has the effect that certain new simulation systems are not discussed in this panel. This refers especially to systems that have a focus on animation in the sense that animation was part of the software from the very first beginning. Hoping that there can be also future panels of this kind, we have, in our choice of which group of software to start with, for this year chosen to concentrate on the "classics". It should also be noted that in the educational track there are full papers that discuss the use also of other systems for the education of beginners.

At an early stage of the planning this panel, I sent a list of questions to the panellists. Among these were the following:

1. What kind of students do you teach or have you been teaching? If different groups, is there a difference in teaching approach as regards different students?
2. What simulation system are you teaching? Have you earlier taught other simulation systems to beginners in simulation?
3. Do your students do project work? If so, how big part of the course is this? Do they then use a larger version of the software system than the (almost) free student version?
4. How much general simulation theory is involved in your course, e.g. random numbers, input analysis, output analysis, verification and validation?
5. Which textbook or textbooks do you use and have you been using?
6. How many classroom hours is the course? Do the students spend more hours on this kind of course than on other courses?
7. In what kind of facilities does your teaching take place? Do you use a projector? Have the students got access to a computer during class? Do they do their work on their own PCs?

Some of these questions are answered in the prepared statements below. Others might be answered in the following general discussion. Since these questions are focused on helping other teachers by giving some idea of "best practice", I will, in order to try to also stimulate a more "controversial" debate, hopefully also with the audience, finally present some points of what I, based on the experience of teaching simulation to over 5000 beginners over two decades, have come to regard to be **suitable criteria for simulation software to be used in simulation education** for beginners. Do you others think that these criteria are reasonable? Which others should be included?

A. Ease of Learning

A1. The learning should not presuppose any preknowledge of programming,

A2. The system should help the students to focus on modeling and experimentation, and not on syntax detail. Students should not have to learn a new concept every time that a new and different thing shall be done.

A3. The simulation language should be fun to learn. Students should be able to do interesting things after a very short period of learning, e.g. after one classroom hour, produce some non-trivial simulation programs.

A4. When students frequently make the same mistake, one must always consider the alternative of changing the system or language instead of forcing them to learn strange features. The system should not be bound to compatibility with earlier versions.

A5. The system must provide most necessary statistics automatically. The novice does not know what kind of statistics is of interest.

A6. It should be possible to cover the system **completely** in a pedagogical manner, with many examples etc., in a small book (e. g. 400 pages) at a low price. This limitation also ensures that all teachers can master the system.

A7. The simulation system should facilitate the teaching in computer labs as well as self-studies in front of the student's own computer.

A8. To facilitate learning, in particular self-studies, the system itself must be supplemented with a great many program examples, tutorial lessons and help pages.

A9. When being projected on the screen by a projector, e.g. in a PC lab, it is important that all important aspects on the projected screen picture are readable by the students. One must avoid having a lot of small details on the computer screen picture.

A10. The system should make it very easy to define, and redefine, an empirical random distribution, e.g. by a number of pairs of value and frequency.

B. Ease of Input

B1. The main form of input should be in form of a Graphical Users Interface, where one from a menu of symbols chooses the (building) blocks of the program. The choice of symbols should be done using either a "drag-and-drop" or a "point-and-click" method.

B2. The number of symbols in the symbol menu should be strictly limited.

B3. For inputting the operands of a block, one should be able to click on an individual block in the block diagram to open a dialog for inputting the operands of this block. In

order to diminish the need of a manual, this dialog should reveal the syntax of the block operands.

B4. It should also be possible to input the program as text, by using a simple editor. The length of the program should be short.

C. Ease of Reading Output

C1. It must be easy to read and understand the output. The system should not provide a lot of advanced output that the novice would find confusing.

C2. The system should provide an easy-to-read program listing, which is clear and compact, allowing for short comments. This listing is essential for making it easy for the teacher to correct and mark the student programs.

C3. One should also be able to complement the program listing with a program logic diagram, directly obtainable from the program in text format to make it easy for students to study, discuss and document the logic of a program.

C4. The output should contain graphs and histograms that are clear and easy to understand.

C5. A simple form of animation, facilitating program verification as well as an understanding of how the program works, is essential

D. Ease of Doing Replications and Experiments

D1. To encourage replications, it should be very easy to make replications of the runs by just one command, easily available in the GUI.

D2. It is also desirable that the system can automatically carry out a statistical analysis of these repeated runs, e.g. of confidence levels.

D3. It is also desirable to have some form of very simple optimization.

E. Safe Programming

E1. The system should minimize the risk of the student making logical errors. Students should not run into surprises and unexpected errors due to not having learnt the full system.

E2. The system should have an extensive error trapping system with as clear error codes as possible

E3. The system must have some simple, very easy-to-learn, system for debugging and program verification, e.g. in the form of block based animation.

A system that was developed according to these criteria, and which has been used in high school education of simulation, will be discussed by our next panel member.

2 HENRY HERPER

Computer Science, in Europe often called Informatics, has during the last few years developed into a subject taught in Secondary Schools, both in Europe and the US. In the process of selecting the contents of this subject as regards the general education of high school students, it has often proved difficult to decide on which of the many areas of Computer Science that are most suitable to include. Modeling and simulation are among the areas of basic, but applied, Computer Science that are then competing for a place in the curriculum.

Traditionally, continuous simulation has played an important role in the Computer Science curriculum in high schools. Research has shown that discrete simulation, on the other hand, has been very little used, mainly since no suitable tools have been available for this kind of simulation. Against this background, teachers at the Stockholm School of Economics and the Magdeburg University have in cooperation developed a family of discrete simulation systems for education. The basis of this was the micro-GPSS simulator, developed by Ingolf Ståhl in Sweden. For this simulator, we in Germany have created a Windows based Development and Experiment Environment. This simulation tool has been successfully used in Germany, both for Computer Science teacher training and for the student education in secondary schools.

For example, in a course called *Introduction to Modeling and Simulation*, given in some German high schools, the students first learn to produce abstract models as excerpts from the real world. They then select a level of abstraction for the model that corresponds to the problem to be studied. These models are then implemented on the computer with a simulation language. In all phases of the model formation process, the model must be verified and validated. This is only possible, if the students know the real system well. The results from the simulation run are then prepared and visualized. A further important goal of the course is to have the students learn how to interpret the simulation results and to evaluate them critically. For the total course around 30 classroom hours are available. This puts special demands on the simulation tool to be used.

2.1 Demands on Tools for Simulation Education in Secondary Schools

Models of server systems are especially suitable for learning simulation technology. For the implementation of these systems, different classes of simulation tools are available. An extensive evaluation of simulation tools that can be used in education is presented in Herper and Ståhl (1999).

For the simulation of service systems, a discrete events oriented simulation language is especially suitable. This would, due to the high degree of abstraction as regards the commands, allow for the implementation of models for many different areas of application. The goal of the educational process is not training in a specific simulation product or tool. Students have to learn the basic techniques for the development of the simulation models.

For a simulation language to be used in the education in high schools there are several demands: An Integrated De-

velopment Environment with a Graphical Users Interface and a Help system is a prime requirement. An interface to an animation system is necessary for the presentation of results. The time requirement for learning the basic elements of a simulation language should preferable be only a few hours. In this way, more time will be available for e.g. the experimentation with the model. The basic principles of the simulators' work must have such transparency that the students can understand the process. The simulation language should be available for education free of cost.

2.2 Teaching Simulation with WinGPSS

A prerequisite for the successful introduction of discrete simulation into the high school curriculum is the availability of well-trained teachers. We have hence focused on getting future teachers to learn how to get their future students interested in discrete event simulation. How can then this best be done? For the teaching of simulation in Magdeburg in Germany, GPSS has a long tradition as language for basic education. Although the students of computer science learn GPSS/H, we have for the education of teachers in Computer Science in the high schools used WinGPSS, since this can then also be used in their teaching in the high schools.

Since 1995 a Micro-GPSS-based simulator has been used in the education in some German high schools and in the education of teachers. WinGPSS was developed on the basis of the experience from this use of micro-GPSS

One of the advantages of this GPSS-based simulator is that it uses only 22 block types. Experience has shown that out of these only 15 are really needed for the education in high schools. Already with 3 block types the students are able to create a simple model and with 5 block types a simple service process can be modeled. The Integrated Development Environment allows for the simultaneous presentation of the block diagram and the text based model.

One of the drawbacks with the learning of classic GPSS, like GPSS/H, is that several blocks have many operands with a great number of possible combinations and default values. This frequently causes the beginner to make errors. In WinGPSS, there are, besides much simpler operand syntax, also dialog windows for the input of the operands with a description of each operand. The main syntax and characteristics of the operands are presented in the dialogs and the correctness of the input is, at least partially, checked. Furthermore the student has access to a Help system with examples. In this way, the frequency of beginners' errors has been substantially reduced.

A further basic problem in the education of beginners is the difficulty of understanding parallel processes. To alleviate this problem, we have introduced a block animation option, by which the movements of the transactions through the block diagram of the model are visualized, without the need for additional programming or model description. This animation is controlled by a trace file and is implemented as *post run animation.* In this way, the simulation process can be analyzed several times. In this type of animation, the transactions will be presented with a number based on the order of generation. Furthermore, this animation will constantly present a type of block statistics, which presents both the number of transactions that currently occupy a certain block (both in symbols and numbers) and the total number of transactions that have visited the block (in numbers).

Compared to other GPSS versions, simplifications have been made e.g. as regards the block types dealing with the movements of the transactions. The new blocks GOTO, IF and WAITIF, with a simplified syntax, correspond better, both in terms and functioning, to similar concepts in common procedural programming languages.

It is important that the students in the validation phase, i.e. when testing the validity of the model, learn to interpret the produced results correctly to make comparisons with their own experience of the real system. After the construction of a model, the students learn to make experiments with the model. There are in WinGPSS special commands available that facilitate the execution and interpretation of a series of experiments.

If it is found desirable to have the simulation results visualized by an animation model in a manner that is closer to real system than with the block animation mentioned above, then there is an interface to Proof Animation of Wolverine Software. When working with the combination of WinGPSS and Proof, the students learn about the correspondence between the movements of the transactions in the simulation model and of the objects in the Proof model.

The experience from projects carried out this far in schools has been that students are well able to model service systems using the tools of discrete simulation. The simulation language GPSS has proved to be suitable as an introductory language, when there is a corresponding Integrated Development Environment available. The students have solved smaller simulation problems individually. Somewhat more complex problems were solved in small groups. The teacher has played the role as a consultant for the students, helping them e.g. to reduce the complexity of the model to such a size that the students can handle it. The students' intimate knowledge of the real system has proved to be a special advantage. The fact that the students can find a solution to a real problem as well as gain a deeper understanding of the whole system has proved highly motivating for them. It should finally be mentioned that the students have shown a high degree of creativity when visualizing the simulation results using animation tools.

3 RAY HILL: SIMULATION AS MODELING METHODOLOGY VERSUS SOFTWARE TUTORIAL

I want to focus on two themes: teaching the modeling process and conveying the need to learn the simulation "language" in its details and complexities.

Modeling is "the representation, often mathematical, of a process, concept or operation of a system, often implemented by a computer program." Simulation is the representation of the behavior of one system by another system. Simulation modeling allows one to gain insight into complex systems or processes via the exercising of a computer-based, generally stochastic computer model. The simulation modeler efforts are enhanced by the plethora of simulation packages now available. No longer must a simulation modeler also be a reasonable computer programmer (an intended exaggeration). With icon-based macro languages such as ARENA, AWESIM, EXTEND, and Micro-Saint, to name just four, literally anyone can create, execute, and examine a simulation model. This is both good and bad.

Icon-based simulation software enables anyone to conduct modeling and simulation. The up-side of this is the increased use of simulation for analytical purposes as well as the increased number of simulators. The down-side is the potential for "bad" analysis involving inexperienced simulators employing a model that runs correctly in the simulation environment. The easy-to-use simulation software approach makes it easy for a novice user to teach themselves the software with little thought regarding simulation design, analysis methodology, or modeling expertise. In the class I constantly re-enforce my view that simulation is applied statistics (this also keeps those nagging "too much statistics" comments off the course critique).

The teaching of simulation can easily can fall into the trap of teaching the software. Simulation modeling requires mastery of the modeling process and then the translation of a conceptual model into the simulation language. The trap of teaching software can be enabled by even the best software. For example, the ARENA flowchart view window has the look and feel of a flowcharting tool. This is very beneficial. However, a novice user may believe they are conceptually modeling a system or process when in fact they are programming. These icons request specific model data and generate underlying SIMAN code. (I, like David Kelton, also like Catherine Harmonosky's idea of handing in the .mod and .exp files with Arena assignments). Even textbooks fall into the trap, defining their "modeling process" via the development of the actual executable model. To learn the simulation modeling process, one must step away from the simulation software, generate some conceptual interpretation of the system or process, and return to the simulation software for a language translation phase.

A person's views are influenced by their experiences; I am influenced by experiences with my first simulation-specific language, SLAM. SLAM featured a set of icons used to define a conceptual model of a system or process. Once satisfied with the paper-based conceptual model, the SLAM icon information was translated into SLAM statements. Similarly, in my early simulations, built in FORTRAN, the simulation components were defined and the simulation processes fully detailed before code was created. Each case separated conceptual modeling from the model translation step.

Teaching the "modeling process" with modern simulation software requires we first "turn off" the simulation software. This means developing some aggregate-level process flow, refining this aggregate view to add modeling detail, and developing some concept of a data dictionary of resources, sets, variables, schedules, etc. It means we re-enforce the top-down systematic modeling process. Academic or textbook problems, complete with process details and distributional data, project a "bottom-up" approach. The best way to confuse a novice modeler is to mire them in details in front of a simulation software tool.

The use of the simulation language should be treated like using a foreign language. To converse in a foreign language, we formulate our concept in our native language, use the foreign language structures to translate our concept, and execute the structures to communicate the concept to the target recipient. In simulation, our native language might be as simple as a flowchart. The structures are the simulation language icons (and their underlying syntax and data structures), and our target recipient is the computer (via the underlying simulation language). Simulation modeling and programming is simply a translation process from our conceptual model to some computer simulation code. For example, while covering some of the ARENA modules during class I try to convey what the module accomplishes algorithmically. Further, how does parameterization of a particular module tie back to the data structures defined in the various spreadsheet views? Again, trying to convey the need to be conversant in the specifics of the language.

To encourage a modeling perspective I emphasize

- aggregate system descriptions is some "native" form;
- pre-defining model data requirements;
- understanding the syntax and semantics of the simulation language; and
- executable model development as a translation process,

while avoiding

- delving immediately into the simulation software;
- leading students to believe data acquisition is easy; and
- ignoring the underlying fundamental simulation concepts.

Icon-based simulation languages are wonderful; their benefits far outweigh any concerns. However, as we educate the next generation of simulation professionals we must keep in mind an overriding need for developing modeling professionals conversant in their simulation language of choice.

4 CATHERINE HARMONOSKY

"Why model?" That is the question that provides the motivation for my approach to teaching simulation to undergraduate industrial engineering majors. The answer, "To gain a detailed understanding of the performance of a system through experimentation when it is difficult (or impossible) to experiment with the real thing," spurs another question: "How do you gain understanding?" Primarily, we gain understanding through analyzing system performance output data generated by the modeling process. These two questions and answers are the foundation and wood framing on which the drywall and pretty pictures of my course are hung.

Regarding modeling, our students learn many different modeling approaches, and it is important that they understand where simulation fits into their toolkit. The biggest differentiator is that simulation is a heuristic technique that specifically incorporates system variability into the analysis through sampling from distributions of input parameters. Since there is variability in the inputs, there is variability in the outputs. Making sure this connection between input and output is clear early in the simulation education process is extremely important, because the output analysis techniques they must use are predicated upon this fact. Because engineers will look at the output data analysis and draw conclusions and make recommendations, their understanding of the variability in the process and what those numbers really mean is crucial for their professional survival.

So, my emphasis in an undergraduate simulation course is to concentrate on the fundamentals of understanding the logic of the real system, translating that real system logic into a simulation model, applying proper output analysis techniques for that system and understanding how inputs affect outputs. At Penn State, we made a fundamental change in our course in 1998 from straight lectures to a lecture and laboratory structure. This structure allows for a complimentary division of concepts that nicely supports this emphasis. Lecture material focuses on general simulation concepts and issues that are applicable across languages, e.g. random numbers, random variate generation and output analysis techniques. Labs give hands-on practice with the general concepts, sometimes through traditional practice modeling of detailed "pseudo-real-world" scenarios and sometimes through a structured exercise focusing on one very specific concept, such as good validation and verification techniques. However, even in the labs, we emphasize "transferable skills", such as developing good logic flow diagrams of the system being modeled, which could then be used to translate the real world logic into any specific simulation language.

One hurdle with this approach is making sure the lectures and labs are well connected. I really work hard to synchronize the lecture topics with the lab experiences. Further, I have found that clearly making references to lab experiences that relate to lecture topics and making clear references to specific topics/statements from lectures in the lab sections is crucial to tie it all together so it does not appear disjoint.

Of course, you can't accomplish hands-on simulation practice without a simulation language. (Well, actually, you *could* have them write their own code from scratch, but why reinvent the wheel?) I hope I follow the philosophy, "Language as facilitator, I as educator". It is a delicate balance to teach them enough about a particular language's capability to give them adequate simulation experience, yet not allow the students to get so hung up on the particular language that they lose sight of the general concept being emphasized. This is where the laboratory section has really helped.

The type of modeling assignments now covered in the lab section used to be given as homework assignments done completely outside of the class in the lecture-only course structure. This meant 2 things: 1) I took time in lecture to cover specific commands/modules of the language to at least point them in the right direction, and (2) because they were not working with the commands/modules immediately in class, by the time they got to their homework, they had forgotten my main points making modeling a lengthy process and leaving them rushing through the analysis, which *should* be the most important part of the assignment. Both, the students and I were frustrated. I have found the dedicated lab time provides an appropriate structure to ensure that the most important general concepts are being emphasized and they are not being left in the dust of developing the model. However, we walk the line between too much handholding and just the right amount of direction, which is a potential drawback of the laboratory environment. So, we have taken a bit of a "design studio" approach, where there is some instruction, but the majority of the time students can work on the modeling problem at their own pace, asking questions as needed of the lab instructor and getting some valuable one-on-one tutoring.

Even though we have the lab section, I still try to postpone using a particular language as long as possible. The first lab assignment is primarily a class exercise with a standard simulation done by hand of a single-server system, complete with die rolling and coin flipping for generating samples from distributions. But, this simple example done well in the structured lab environment effectively demonstrates many fundamentals of simulation—random variates, simulation clock, event calendar, time-persistent and observation-based statistics, multiple replications and confidence interval estimation. As the semester develops, this lab experience can be referenced frequently to relate to the "behind the scenes" work a simulation language does for us. I am sometimes amazed at how much mileage I get out of this one lab.

When a simulation language is introduced, we use Arena. The major hurdle with any language is making the connection between these lovely icons in the GUI and actual program code. Thankfully, Arena 5.0's flow-chart style icons for logic modules has made this easier compared to Arena 3.0. Also, I do require them to include a copy of the complete model code (both *.mod and *.exp files) in their lab reports. So, at least I know that they have *seen* the code and they realize it is there.

Also, I have found the hierarchical nature of Arena to be helpful. This allows us to get up and running quickly early in the semester with simple systems using higher-level modules. But, when system logic becomes just a bit more complex, they quickly learn the drawbacks of the additional structure of the higher levels (akin to simulators) and understand the trade-offs between increased modeling flexibility and ease-of-use. We have also made good use of Arena's Input Analyzer for analyzing input data and the Output Analyzer for various output analysis techniques, including steady state analysis with batch means.

Of course, I see some opportunities for improvement in Arena to better support my educational goals. The output report is certainly loaded with information and graphs. But, with the information spread out over many, many pages, it is a bit overwhelming and frustrating for the beginner to just find the data they need, such as determining under what subtitle the work-in-process measure is reported. Consequently, we spend more lab time than I would like just reviewing the output report structure instead of focusing on output analysis. Also, clearer error messages written with the "beginner" in mind along with a much more user-friendly interactive debugging capability would allow the students to more independently correct really tough logic problems.

In closing, I find teaching simulation continues to be challenging, stimulating, rewarding and fun—despite the heavy workload of such a course!

5 JOAN DONOHUE: TEACHING SIMULATION TO BUSINESS STUDENTS USING AWESIM

This part of the panel discussion focuses on the following four issues:

1. How to attract business students into taking a course in computer simulation.
2. Why AweSim is a suitable software package for teaching simulation to business students.
3. Types of simulation projects undertaken by undergraduate business students.
4. Ability of students to conduct meaningful simulation projects at the end of a one-semester course.

A course entitled "Simulation of Business Systems" is offered each semester in the Business School at the University of South Carolina. It is an elective course for students majoring in Production and Operations Management, Management Information Systems, or Quantitative Business Analysis. Space permitting (up to a maximum of 35 students), non-business majors can enroll in the class and they are typically Computer Engineering students. Since most undergraduate business students know nothing about computer simulation, they are not likely to enroll unless previous students recommend it to them, or if it happens to fit into an empty time slot in their schedule. Unfortunately, very few business students choose to take the course because they want to learn how to solve business problems using simulation models. Most students do not know the purpose of computer simulation and they often fear it will involve writing computer programs and becoming an expert with computers. So, how should professors generate student interest in taking a computer simulation course?

The obvious answer is to promote the course in a variety of ways. For example, prerequisite courses such as Business Statistics should inform students of the class, explain the purpose of computer simulation, and encourage interested students to take the class. On a larger scale, other business academic departments could allow their majors to choose simulation as one of their elective courses. This approach would be more difficult since it requires convincing faculty in areas such Accounting, Finance, Marketing, Management, etc. that a computer simulation class would be beneficial to their students.

Concerning the software used in an introductory business simulation class, any of the block-based simulation languages, such as AweSim, GPSS, and Arena would be appropriate. While engineering and computer science students might consider the use of such software to be "programming with pictures," it is often the closest that business students ever get to computer programming. The block-based software packages enable students to learn how to logically structure a computer program without getting involved with syntax, subroutines, etc. The author uses AweSim and the associated textbook (Pritsker and O'Reilly 1999) in the business simulation course because it is a good general purpose software package and it has a reasonably priced student version. Students like using the software and few have trouble understanding it. Three drawbacks of using the AweSim software are high cost for the commercial version (required for large models and for user-written inserts with C or Visual Basic), the textbook is too advanced for an introductory class, and the software is not widely used in the business world. Students would prefer that the software were available as an Excel add-in so that they could easily use it after they graduate. However, as a teaching tool, AweSim works well because it allows students to gain a basic understanding of simulation with minimal startup costs. Those students who truly understand the course material (about half of those in the author's classes) will, in the future, be able to perform simulations on any software platform that is accessible to them.

In the course taught by the author, the first half of the semester is devoted to learning how to develop models of real-world systems and translate them into AweSim networks. Another important aspect of this first part of the course is learning what output statistics are provided, how they are computed, and how to interpret them. The second half of the course is devoted to two projects, a midterm project and a final project. The midterm project is completed individually by each student. The instructor chooses the real-world system and develops the simulation model. The students collect their own input data for random processes such as arrival times, service times, probabilities of taking particular paths through the system, etc. Students are required to organize their data on an Excel worksheet and to find the best-fitting probability distributions using the Stat::Fit software. Students enter their input data into the AweSim model and perform ten runs of four different scenarios (the real world system and three alternatives). Output statistics from AweSim are imported into Minitab and appropriate statistical analyses are performed. Results vary considerably from one student to the next due to the varying times over which they collected data. Lastly, students write a report that explains, in detail, the entire simulation project and the implications for the real-world system. Examples of midterm projects that have been used in the past include traffic stoplights, fast food restaurants, laundromats, and gas stations. After completing the midterm project, students begin working on a final project. The final project is carried out in groups of three students and the system to be simulated is selected by the students. The instructor works closely with each group to help them carry out a meaningful simulation project. The requirements for the final project are similar to the midterm project but, in addition, a PowerPoint presentation is required. Examples of final projects that students have completed include a UPS package sorting facility, a John Deere chainsaw chain manufacturing plant, and a model of particle coagulation and granulation.

One might wonder if business students are capable of conducting meaningful projects at the end of a one-semester course. Over the years, through trial and error, the author has found that students can conduct meaningful simulation projects provided the following things are done:

- Prior to the selection of the final project, each student individually prepares a midterm project of similar difficulty. The instructor points out errors and explains how to correct them so that similar errors are not made in the final project.
- Final projects are prepared in groups of three students. Having the diverse skills three students greatly improves the chances of the project being correct and meaningful.
- Groups must develop an idea for the project and the instructor works closely with each group to develop a model, specify inputs and outputs, etc. This is the most important way of ensuring that the projects will be correct and at least somewhat meaningful. Some groups need very little help while others need the instructor to guide them through almost every step of the project.
- An oral presentation of the group's project using PowerPoint is required. Most students want to make a good impression in front of their peers. Therefore, the quality of these presentations is very good and this improves the chances of the accompanying written report being of high quality.

In summary, many business students enjoy taking a course in computer simulation and feel it may be a useful tool in their careers. AweSim provides a user-friendly platform for teaching simulation to business students who have no computer programming background. Incorporating projects into the class is an important aspect of teaching students how to solve business problems using computer simulation.

6 W. DAVID KELTON

I've been teaching simulation (or trying to) for 20something years now, with university audiences coming from engineering schools, business schools, computer science, and a host of other departments (predictable ones like math and forestry, less-predictable ones like medicine and economics, and unpredictable ones like Spanish literature). I've also taught in non-academic settings such as corporate seminars, industry conferences, and military training. The experience/age profile has included the usual 20something college students, Marine colonels, as well as both retiring high-school math teachers and their 15-year-old pupils.

With that seemingly wide range of settings and audiences, it has surprised me how little variation, both over time and settings, there has been in the interests and underlying topics (maybe the latter is my fault), other than the obvious progression of software. So this panel on "teaching the classics" was intriguing to me as a possible vehicle for discovering why things seem to be so much the same, and whether that's good or not.

One of the (many) great things about being late in supplying this paper is that I had the unfair advantage of reading the papers of my colleagues on this panel, and I am largely in agreement with the points they make, so I won't repeat their convincing arguments. What I'll try to do here is conduct a debate against myself on the degree to which the "classics" should be taught, and to whom. So, first things first.

6.1 What Are the Classics and Should They Be Taught?

Of course, what's a "classic" is in the eye (and age) of the beholder. I'm still trying to get used to text-entered simulation software and always try to map it back onto FORTRAN 66 no matter what. (One of the co-editors of

these *Proceedings* once gently told me, when I asked him to review some of my programs as I was trying to teach myself C, that I'd done a nice job of writing FORTRAN in C, sort of like the way I speak English in German.)

In this context, a "classic" to me is a general-purpose programming language that is not a simulation language at all — like C, C++, Java, Pascal (if anyone still speaks it), and, yes, any variety of FORTRAN. Spreadsheets don't count; it has to be a *real* programming language. At some point (but not at the beginning) of such instruction I'd admit utility routines (best if they're home-brewed) for common simulation chores such as list processing, generation of random numbers and random variates and random processes, and statistical accumulation and reporting.

Now, for my intradebate.

6.2 Dump the Classics

In order to build good simulation models, exercise them well, and conduct effective simulation projects, nobody needs to know this stuff any more, so we should not teach it. In just a few hours (or maybe minutes) with what Ray Hill calls an icon-based simulation language, it's pretty true that "anybody" can learn to simulate at least simple things (a state of affairs that is, as Ray points out, both good and bad). And some additional exposure, perhaps self-taught, brings not-so-simple things to within reach. Further, high-level simulation software is almost immediately applicable to real problems, as legions of project-hunting students (many of mine are on co-op so they have inside access to solid rustbelt companies) discover, sometimes with measurable impact.

True, all high-level software is going to have shortcomings and irritations (as well as "anomalous features"), but this situation is getting better all the time. And the prices are falling as well, making just-post-graduation implementation in a hiring organization ever more feasible. Moreover, students seem to like icon-based simulation languages and the classes fill up (a practical issue for faculty, if nothing else). And, speaking of practicalities in universities (if that's not oxymoronic), there's an increasing trend to compress OR-type topics, sometimes to oblivion, especially in business schools, and going straight to high-level software makes it possible to do something of value in, say, three weeks with, say, MBA students; I do it all the time and it works.

Teaching old-fashioned programming is simply no longer needed, any more than is teaching how to carve cuneiform into clay tablets now that we have LaTeX (though Word wouldn't be enough to surrender the clay tablets). It's a waste of time and we should get over it, grow up, and move on.

6.3 The Classics are Essential

If all we ever teach, and thus if all anybody will know in a generation or so, is high-level point-and-click software, there will be a general dumbing down of the simulating population that could have serious long-term impact. For one thing, who will write future simulation software if nobody understands anything of what's going on under the hood? I own and drive a car yet know nothing of how it works, but I do know that Tater down the street at Tater's Blue Ash Auto Repair knows my car inside out and I'm glad he does.

So we still need at least a few Taters in simulation who know how it all works inside, and can fix it when it's broke. And so we still need to teach at least a little low-level simulation to at least a few people, if nothing else just to make sure students know that it's there, and know that there will always be some Taters around who can help if needed. I liked Catherine Harmonosky's idea of forcing students to hand in the Arena .mod and .exp files, and not just the flowchart view, simply to make sure they know they're there and at least vaguely what they can do.

Even to nonplussed MBA students, I still force-feed a simulation by hand, as does Catherine to her students in their lab exercises. This gives them a feel for how to structure the data in a model (which I think is the most important aspect, even more so than the network topology), which promotes sound modeling once we start pointing and clicking.

In a recent class, I returned to teaching quite a lot of low-level simulation programming (using C by popular vote of the class), and it did not go well. One problem is that, even among engineering students, the facility with programming is getting pretty rough around the edges. Another problem is that they knew that a high-level software package was coming up and they were anxious to get to it. So maybe the upstream education and short attention spans now argue against such a classics-heavy course, but at least I know that that particular group of students went out knowing how simulations really work (and they could point and click too).

There's a reason that Columbia hangs on steadfastly to The Core of Homer, Herodotus, and friends for all undergraduates. If we succumb to the siren call of teaching icon-based simulation software only, we're short-changing not only our current students' fundamental understanding of simulation, but also the ability of future generations of simulators to progress.

6.4 The Answer

The problem is, I actually believe both sides of the argument, though they're pretty much incompatible. And to some extent, I am still trying to work both sides of the street in my teaching. Perhaps my inability to let go of the classics completely (for good reason, I believe) is one of the underlying currents that have anchored the topics over quite a few years; another underlying current is the need to introduce statistical design and analysis early and often.

As in most issues, The Answer probably depends, in this case on the audience, their needs, and the intended take-aways.

6.5 Different Audiences, Different Approaches

This is purely opinion, though based on some experiences, some successful and others less so. And this assumes that we're talking about a first course in simulation modeling, not a second course in simulation analysis:

- For university students, either advanced undergraduate or beginning graduate, in a first "modeling" course, try to do a mixture, though not a whole lot of time (20% max) on the classic programming approach. This will show them that there *is* something underneath the hood, and indicate how they might get themselves out of a jam in an emergency. A problem here is the aforementioned programming near-illiteracy.
- For computer-science and some engineering students, devote more time to the classics but still cover simulation software in some depth. If they don't know a programming language upon arrival to the course, they need to teach one to themselves in the first week.
- For a brief (and thus shallow) introduction, e.g. as a three-week module in some kind of an O.R.-survey course, devote about one sentence to the classics and just rush to the point and click and try to get as far as you can. It's better than nothing (I think), which is probably the only alternative in most academic-political environments.
- For an industry/military audience, put the heavy emphasis on the simulation software, but probably step through at least one programming exercise.

Let me close by describing how this can map onto an unorthodox audience, high-school juniors. I taught a one-semester course on simulation modeling to such a group at Cincinnati Country Day School (to the great embarrassment of my daughters, both of whom were in school there), on a strictly *pro bono* basis, and the math teacher sat in on it as well. Now I must admit that this group might not have been typical, as one went to Harvard on a full Intel scholarship, one went to Cal Tech, one went to Case on a full scholarship, and one went to Pomona (to major in English, though I still think he's smart). But I found that, even with so young a group, the classic principles were easy to get across, and they programmed simple examples in Java (I think ... I don't speak that language). We then moved on to a high-level simulation package, which they absorbed easily. They all did projects that were of reasonable complexity (even if on sophomoric topics like the effect the cafeteria operations of putting pop in the drinking fountains). Now I don't pretend to have the reach into high schools that Henry Herper has had, in his program of training the trainers, but this was still, in my view, a very successful experiment.

So I believe that it's possible (and desirable) to get most people up to speed on a high-level simulation package, yet not ignore "the classics" that are essential, in varying degrees, to understanding simulation mechanics, not to mention making sure that we have a steady supply of people who really understand The Core.

REFERENCES

Herper, H. and I. Ståhl. 1999. Micro-GPSS on the Web and for Windows - A Tool for Introduction to Simulation in High Schools. In P.A. Farrington, H.B. Nembhard, D.T. Sturrock and G.W. Evans (eds.) *Proceedings of the 1999 Winter Simulation Conference*, 298-306. SCS, Phoenix.

Pritsker, A. A. and J. J. O'Reilly. 1999. *Simulation with Visual SLAM and AweSim*. 2d ed. New York: John Wiley and Sons.

AUTHOR BIOGRAPHIES

INGOLF STÅHL is a Professor at the Stockholm School of Economics, Stockholm, and has a chair in Computer Based Applications of Economic Theory. He was visiting Professor, Hofstra University, N.Y., 1983-1985 and leader of a research project on inter-active simulation at the International Institute for Applied Systems Analysis, Vienna, 1979-1982. He has taught GPSS for twenty-five years at universities and colleges in Sweden and the USA. Based on this experience, he has led the development of the micro-GPSS and WebGPSS systems. He is also consultant in simulation to Swedish banks and industry. His email address is <ingolf.stahl@hhs.se> and the web address for his WebGPSS system is <www.webgpss.com/>.

HENRY HERPER is in the Institute for Simulation and Graphics at the Otto-von-Guericke University, Magdeburg. His research interests include the modeling of logistical and manufacturing systems and the development of simulation tools for introduction to simulation. He is a member of ASIM and the GPSS-Users' Group Europe. His e-mail and web addresses are <henry@isg.cs.uni-magdeburg.de> and <wwwisg.cs.uni-magdeburg.de/isg/henry.html>.

RAYMOND HILL is an Associate Professor of Industrial and Human Factors Engineering with the Department of Biomedical, Industrial, & Human Factors Engineering of Wright State University where he runs the Advanced Modeling, Optimization, & Systems Laboratory. His Ph.D. is from The Ohio State University. His research interests include agent-based modeling, applied simulation modeling,

and applied optimization modeling. His email address is <ray.hill@wright.edu>.

CATHERINE M. HARMONOSKY is an Associate Professor in the Harold and Inge Marcus Department of Industrial and Manufacturing Engineering at Penn State University. She received her B.S.I.E. from Penn State University and her M.S. and Ph.D. in Industrial Engineering from Purdue University. Her research and teaching interests are in simulation and production planning and control, and she has been teaching simulation since 1987. She is a member of IIE, SME and SWE. Her email address is <cmhie@engr.psu.edu>.

JOAN M. DONOHUE is an Associate Professor of Management Science in the Moore School of Business at the University of South Carolina. She has a B.S. degree in Chemical Engineering from the University of Delaware and an M.B.A. and Ph.D. degree in Management Science from Virginia Tech. Her research interests include the design and statistical analysis of simulation experiments and the application of these methods to manufacturing systems. Her email address is <donohue@moore.sc.edu>.

W. DAVID KELTON is a Professor in the Department of Quantitative Analysis and Operations Management at the University of Cincinnati. He received a B.A. in mathematics from the University of Wisconsin-Madison, an M.S. in mathematics from Ohio University, and M.S. and Ph.D. degrees in industrial engineering from Wisconsin. His research interests and publications are in the probabilistic and statistical aspects of simulation, applications of simulation, and stochastic models. He is co-author of *Simulation Modeling and Analysis* (3d ed., 2000, with Averill M. Law), and *Simulation With Arena* (2nd ed., 2002, with Randall P. Sadowski and Deborah A. Sadowski), both published by McGraw-Hill. Currently, he serves as Editor-in-Chief of the *INFORMS Journal on Computing*, and has been Simulation Area Editor for *Operations Research*, the *INFORMS Journal on Computing*, and *IIE Transactions*, as well as Associate Editor for *Operations Research*, the *Journal of Manufacturing Systems*, and *Simulation*. From 1991 to 1999 he was the INFORMS co-representative to the Winter Simulation Conference Board of Directors and was Board Chair for 1998. In 1987 he was Program Chair for the WSC, and in 1991 was General Chair. His email and web addresses are <david.kelton@uc.edu> and <www.cba.uc.edu/faculty/keltonwd>.