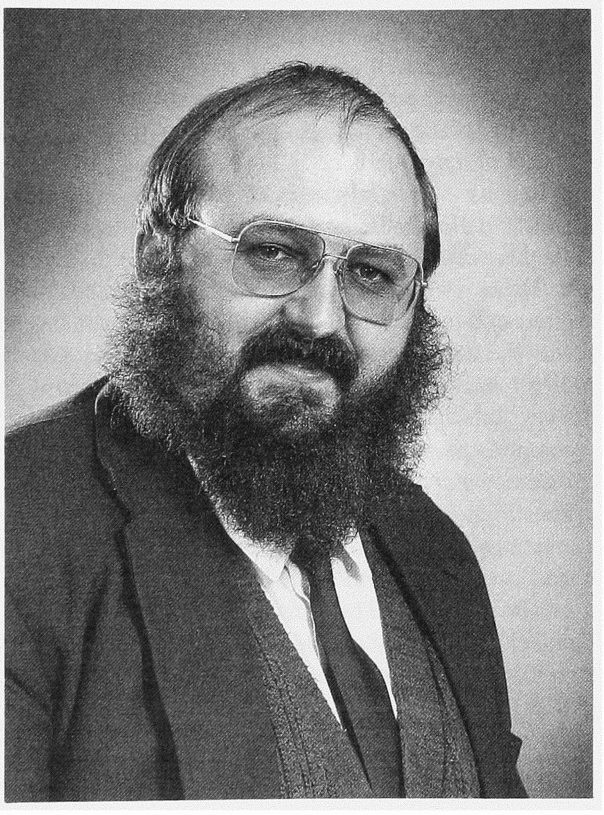
Proceedings of the 1993 Winter Simulation Conference
G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles (eds.)

KEYNOTE ADDRESS: SIMULATION SHOULD BE EASY AND FUN!

John D. Salt

CACI Products Division
Coliseum Business Centre
Watchmoor Park Riverside Way
Camberley Surrey GU15 3YL



It may seem absurd to assert that stochastic discrete-event simulation should be easy. The technique is used to study systems exhibiting complex, time-dependent behavior - systems that cannot reasonably be studied by any other means. Where direct observation of the live system is impractical; where linear programming is insufficient; where queueing theory is inadequate; on these problems, we use simulation. Simulation enables us to study systems that are otherwise intractable. How then can it be easy?

It is no part of my argument that simulation modeling is, or should be, trivial. I believe that simulation is one of the most demanding fields of the programmer's art. I further believe that the construction of useful models requires special skills, quite independent of programming or project management ability, which will be increasingly in demand as mainstream computing comes to adopt more of the techniques developed by simulationists. It is my contention that one of the most important of these skills is the ability to keep things simple. Yes, real problems are messes, but it is not desirable to model them by creating an automated mess. Simplification is the essence of simulation. The whole modeling process consists in capturing the important elements of a system, and leaving out the rest. How many projects do you know that failed because they were too simple?

A paper by Ward (1989) distinguishes between "transparency" and "constructive simplicity". Transparency is a perception of the model's user, and depends on factors such as the user's familiarity with modeling methods, interest in the problem, and the degree to which the builder explains the model. Constructive simplicity is a characteristic of the model itself, and could in principle be objectively measured. Thus a model of a given degree of constructive simplicity that appears transparent to one user might seem opaque to a user who is less skilled, less experienced, or who has had less explanation of the model. My concern here is mainly with constructive simplicity.

Constructively simple simulation models generally take less time to write, less time to run, and require less input data than complex ones. This makes it easier to use them in an exploratory fashion. Users can form an impression of system behavior by trying out a number of scenarios and comparing the results.

There is a real danger, with large models, that this will not be possible. The user may take decisions based on too few cases, and too little understanding of the way the system parameters interact. Worse still, a constructively complex model will have more parameters to explore than a simple one. Even if the same number of scenarios were tested in each model, the proportion of the possible number of execution paths investigated will still be smaller for the complex model. The simple model has a double advantage.

Simple models are easier to change than complex models. This is an important consideration in all fields of computing, but especially so in simulation. It is wholly unrealistic to expect the specification for a simulation program to remain static over time. The perception of the problem is bound to change as more is understood about it - why else do a simulation study, if not to improve understanding? So, not only are simple models easier to use in an exploratory fashion; they are also easier to develop in an exploratory fashion. It is also quite possible for the system under study to change. There is little point in having a highly detailed model of the way a system used to work last year. This has less chance of happening during model development if the development time is kept short. Again, the simple model has a double advantage.

It is easier to throw a simple model away and start again. This makes it easier to keep up with the latest languages and techniques in new studies. Also, one might hope that users will request more simulation projects if they are perceived as short-term and relatively low-risk activities.

Simple models are easier to make transparent to the user, and so it is more likely that decisions will be based on simulation results. It is no good having a first-class recommendation for action that the user does not act on because the reasoning behind it is not understood.

Given all the advantages of simple models, why do people write complex ones? I believe that there are a number of reasons for simulation models becoming complex. Many of these reasons are entirely non-technical in nature, but arise from human factors, and so are quite independent of the area of study.

The first consideration is that many programmers enjoy creating intricate programs. This is understandable, and, as I shall argue later, it is desirable that programmers should be absorbed in their work. However, it is tempting to lavish detail on those parts of the system that are well understood, and to skimp on those parts that are not. This is the reverse of what is wanted. If we are to increase understanding of the problem, it is the mysterious parts that provide the greatest opportunities for learning. There is little benefit in having any part of the system much more detailed than the least-well understood part.

Another temptation is to do something just because it is possible. Computing power is still halving in price every year, and so each successive generation of computers can run increasingly large and complex programs at acceptable speeds. However, simulation models are not something to do, like climbing Mount Everest, just "because it's there", and the temptation must be resisted.

It is all the more likely that the programmer or model-builder will wander off into irrelevant detail if the goal of the model has been poorly defined. As Russell (1983) points out, the mere desire to build a model of a system is not a legitimate goal; the model must have some purpose. Where the overall aim is poorly defined, the anxious simulationist may draw the bounds of the model too wide, in the hope of including whatever it is that the user is really interested in. Better to refine the aims of the project with the user - perhaps by means of a few "quick and dirty" throw-away studies - than to embark straight away on the construction of a baggy and over-complicated monster.

Even if a model starts life as a simple and elegant creation, it can become complex. It is easy enough to add complexity to a model; it is very difficult to subtract it out. The effects of the "ever-growing model" syndrome become more pernicious still as the staff involved in developing it changes. Perceptions of the model may change as the staff changes, and this will tend to blur the original aim. Fortunately, there are some technical fixes for this problem. The adoption of a modular programming style can give a program "pluggability" and, equally important "unpluggability". Neatly interfaced modules can be unplugged and replaced with simpler versions in order to keep the overall complexity of the model down.

It is easy to confuse detail with realism. They are not the same. It is quite possible to create a model that is at the same time finely detailed and wildly inaccurate. Indeed, questionable assumptions can

hide much more easily in a detailed model than in a simple one. Complexity can be used to baffle people into acceptance, "blinding them with science". However, the users will want some reassurance that they are getting their money's worth for a project. The easiest way to measure programmer productivity is by the number of lines of code written. The easiest way to measure the output of a simulation study is by the bulk of the final report. As long as size is the measure used, simplicity and elegance will go unrewarded.

The same is true of the model's input data. A model that requires an enormous amount of input data to function may appear very impressive, but as the volume of data required increases, so does the problem of checking it become more acute. Reliance on a large amount of input data provides a ready-made excuse for the development team; if the results of the study turn out to be unrealistic or misleading, they can always be blamed on inaccuracies in the data. In this way, fear of failure may, paradoxically, be a motivation for making a model complex.

Once a large and complex model has been created, it will be difficult ever to get rid of it. It is much harder to admit the failure of a million-dollar system that a thousand-dollar one. Even if the model has become wholly inappropriate, managers may wish to retain it because of the perceived investment it represents. Consequently, there are millions of lines of FORTRAN legacy code still in use today.

Dixon (1976) mentions that authoritarian personalities show a preference for impracticably difficult tasks rather than moderate risks, so that failure will appear more excusable. Authoritarian traits are also linked to obsessive behavior. Either of these might be a motive for preferring detailed to simple models - I regret that authoritarian personality types are not unknown in the world of computing. Fear of failure by insecure personalities can also explain reluctance to establish a clear aim. Further, authoritarian attitudes go and-in-hand with conservatism, which might partly explain some of the reluctance to dispose of legacy models. It would be most interesting to conduct a survey of the correlation between authoritarian attitudes, model complexity and model success.

Simulation seeks to solve problems that are by their nature complex, so one could argue that a degree of complexity is unavoidable. This is true. However, that complexity can be managed by "chunking" into digestibly small portions. As General Creighton Abrams said, "When eating an elephant, take one bite at a time". This is another argument for a highly modular programming style, and perhaps for a series of small, tightly-focused models in place of one big one.

Even if constructive simplicity cannot be attained, transparency is still possible if communications between user and modeler are good, and the modelers speak the language of the problem domain. What is not acceptable is the introduction of unnecessary complications motivated by a fear of failure of appearing mistaken. It is possible to view simulation primarily as an exercise in making mistakes. By making mistakes in a simulated environment, rather than a real one, we hope to avoid expensive or dangerous errors in the real thing. In this view, it is a duty of both the developers and the users of simulation to make their mistakes clearly and openly, so that they can be corrected. DeMarco and Lister (1987) go so far as to suggest that software developers should have a quota for errors; anyone who hasn't made any mistakes obviously isn't trying hard enough.

The atmosphere in which simulation developers ply their trade should be relaxed enough to permit mistakes. More than that, it should encourage people to play. It seems to me that using a simulation for "what-if" investigations is no different in kind to the make-believe play of young children, who use fantasy to help make sense of the world about them (Millar, 1968).

This is not to say that mistakes should not be corrected. The first step in correcting a mistake is to detect that a mistake has been made. Fear of blame is only likely to ensure that mistakes stay hidden. From the point of view of project success, it does not matter where the blame for a mistake lies, as long as it is corrected.

Mitrani (1982) distinguishes three kinds of mistakes that occur in simulation projects: programming errors, logic errors, and "errors of judgement". The first two categories of error can be handled by mainstream software development techniques. One of the best such techniques is the technical inspection, peer review, or structured walkthrough. Whichever name it takes, this technique relies on the idea of "egoless programming" - if software is the property of the whole development team, blame cannot be attributed to individuals. My experience is that a

properly-conducted inspection can be lively and enjoyable. According to Yourdon, many managers report that the introduction of walkthroughs boosts staff morale. He even says that managers should "be prepared to walk past the conference room where the walkthrough is being held and hear raucous laughter, shrieks and giggles, and all the other evidence of a drunken orgy" (Yourdon, 1977).

Mitrani's last category, "errors of judgement", really deals with whether the simulation is an adequate representation of the real system. It is harder to decide what constitutes a mistake here. The judgement is better made by the domain experts than the simulationists in this case. Validation in a strict sense may not be possible, particularly in the case of hypothetical systems. Law and Kelton (1991) offer the more realistic goal of making simulation models credible. This requires the intimate involvement of the user and domain experts.

An excellent technique to involve users and improve understanding is manual simulation, as recommended by Paul and Balmer (1993) and others. Participation in such an exercise is an excellent medium of communication between developers and domain experts, and is able to establish much deeper levels of understanding than any number of memoranda. Nothing improves understanding of a system as well as pretending to be a functioning component of that system. To investigate difficult execution paths, Coad (1992) recommends exactly this technique, with the slogan "If in doubt, act it out". By impersonating an object, the analyst obtains a new perspective on the system, and can answer questions such as "what do I need to know?" and "what do I need to do?" from the object's point of view. The popularity of wargaming and role-playing as hobbies is testimony to the fact that manual simulation can be great fun. As with walkthroughs, the exercise is probably best carried out among peers, without the deadening effect of a senior management presence. If a walkthrough can sound like a drunken orgy, a manual simulation will sound like a riot. Nevertheless, total immersion in the problem provides rich insights into it. That people are enjoying themselves doing something together can only have beneficial effects on creating a "jelled" team (Demarco and Lister, 1987).

Pirsig (1974) holds that this identification with the object - the suppression of the normal subject-object relationship - is the key to quality. This seems to apply to individuals as well as groups. DeMarco and Lister (1987) point out that creative brain-work is done in a state that psychologists call "flow". This a meditative, mildly euphoric state, with decreased awareness of the passage of time, when ideas flow without effort. Unfortunately, it is very easy to interrupt a state of flow. The key offender in this respect - what Pirsig would refer to as a "gumption trap" - is the telephone. The office environment for brain workers should permit them to use their brains without endless interruptions.

To conclude, I believe that the easiest way for management to prevent the development of high-quality simulation software is to take the attitude that fun is unprofessional. Building simulations is a creative activity, and people cannot be forced to be creative. They must be poor creatures indeed who do not understand Brooks (1975) when he writes of the joy of making things:

"As the child delights in his mud pie, so the adult enjoys building things, especially things of his own design. I think this delight must be an image of God's delight in making things, a delight shown in the distinctness and newness of each leaf and each snowflake."

No matter what else you do at this conference, have fun!

REFERENCES

Brooks, F.P. 1975. The mythical man-month: essays on software engineering. Addison-Wesley, Reading Massachusetts.

Coad, P. 1992. Object-oriented analysis (course notes). Object International, Austin, Texas.

Demarco, T. and T. Lister. 1987. Peopleware: productive projects and teams. Dorset House, New York.

Dixon, N.F. 1976. On the psychology of military incompetence. Jonathan Cape, London.

Law, A.M. and W.D. Kelton. 1991. Simulation Modeling & Analysis. McGraw-Hill, New York.

Millar, S. 1968. The psychology of play. Penguin, Hamondsworth, Middlesex.

Mitrani, I. 1982. Simulation techniques for discrete event systems. Cambridge University Press, Cambridge.

Paul, R. and D. Balmer. 1993. Simulation modelling. Chartwell-Bratt, Bromley, Kent.

Pirsig, R.M. 1974. Zen and the art of motorcycle maintenance. The Bodley Head, London.

Russell, E.C. 1983. Building simulation models with SIMSCRIPT II.5 CACI, Los Angeles.

Ward, S.C. 1989. Arguments for constructively simple models. *J. Opl Res. Soc.*, 40:2, 141-153.

Yourdon, E. 1977. Structured walkthroughs. Prentice-Hall, Englewood Cliffs, New Jersey.

AUTHOR BIOGRAPHY

JOHN SALT is a Senior Simulation Engineer working for CACI Products Division in Camberley, UK, where he is responsible for technical support and customer training. He is UK trainer for the SIMSCRIPT II.5 and MODSIM II simulation languages.

John has been in the simulation business for five years. In January 1988 he joined Hunting Engineering Limited (HEL), producing simulation models in GPSS/H for the British Ministry of Defence. Most of this work dealt with operational availability models for helicopter squadrons or flights. He successfully persuaded HEL to adopt their first object-oriented lanugage, Simula, in place of GPSS/H.

He then moved to Eurotunnel, the company that will operate the channel tunnel. Here he was responsible for setting up an in-house simulation capability from scratch, and so became the first civilian user of MODSIM II in the United Kingdom. The series of simulation models he created dealt with topics such as the management of road traffic in the Folkestone and Coquelles terminals, the movement of vehicle streams onto loading platforms, the effects of high winds on shuttle operating policies, the manning levels required at security checkpoints, and the effects of different safety policies on shuttle loading patterns. An article on the early work done at Eurotunnel made the cover of OR/MS Today magazine.

John has previously spoken at Simulation Conference 16 in San Diego, Software Development '92 in Wembley, Simulation Conference 18 at Maastricht, and at simulation seminars in England, France, Germany, Italy and Poland.

He holds a BA(Hons) in Russian and French from the University of Exeter and an M.Sc. in Computing Science from the University of Newcastle-upon-Tyne. Among his more unusual qualifications, he holds a Certificate of Military Training from Exeter UOTC; a first descent certificate from the Royal Marines and Royal Navy Sport Parachuting Association; and a Certificate of Excellence in Russian from a Polytechnical Institute in what was then Leningrad. He has recently enrolled at Brunel University for a part-time PhD in simulation modelling.