Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

The methodology roles in the realization of a model development environment

Richard E. Nance
Systems Research Center
Virginia Tech
Blacksburg, Virginia 24061

and

James D. Arthur
Department of Computer Science
Virginia Tech
Blacksburg, Virginia 24061

ABSTRACT

The definition of "methodology" is followed by a very brief review of past work in modeling methodologies. The dual role of a methodology is explained: (1) conceptual guidance in the modeling task, and (2) definition of needs for environment designers. A model development environment based on the conical methodology serves for specific illustration of both roles.

1. INTRODUCTION

The search for a better way to accomplish assigned tasks is one of the distinctive characteristics of the human species, as Balmer (1987) notes in his quotation from Bronowski (1973):

> The most powerful drive in the ascent of man is his pleasure in his own skill. He loves to do what he does well and, having done it well, he loves to do it better.

A disconcerting conclusion, based on almost 20 years of research, is that the search for improved modeling techniques has not kept pace with the ever increasing expectations regarding the problems to be solved. Computing technology has served as a constant driver: the desires for faster processors and larger storage capacity have been repeatedly realized.

The capabilities requisite in the evolution of more powerful systems or the progressing need to solve more challenging problems converge in a single focused question: Can a better means of resolving modeling complexities be found? To date, the positive answers to that question have been few, and those, generally disappointing. One answer takes the form of a "methodological" approach to modeling tasks. Exploring this form leads to the identification of the two roles of a methodology. Understanding both roles and the consequential implications for a modeling environment are the subjects of this paper.

1.1 "Methodology" Overused or Underutilized?

Methodologies have been a major subject of development in the software engineering domain. Examples such as Software Cost Reduction (Heninger, *et. al.* 1978), SREM (Alford 1985), and SADT (SofTech 1976) are touted as means for effectively dealing with the very costly "software problem." Colter (1982) describes the evolution of structured methodologies, beginning in the 1960's when the apparent difficulty of creating large software systems became universally recognized. This description of software development notes the sometimes subtle changes in methodology emphasis as the understanding of root problems increases, e.g., recognition of the high cost of maintenance, definition of structured programming principles, and preoccupation with tools and techniques.

In the software development domain, little attention has been given to the evaluation of the numerous software development methodologies. Among those that do treat evaluation, e.g., Bergland (1981), Basili and Weiss (1985), and Sawyer and Dawson (1984), the comparison of benefits has tended to focus on those derived from utilizing a methodological approach as opposed to having none at all. Only recently, have studies appeared comparing software development methodological techniques or specific methodologies (Card, McGarry and Page (1987) is an example of the former; Henry, Arthur and Nance (1985), the latter).

In contrast to software engineering, modeling oriented disciplines have been slower to recognize the need for methodological approaches. Nevertheless, a few pioneers have recognized that the analysis and diagnosis of model representations mandates some form of disciplined development and algorithmic representation. Greenberg (1983), Kurator and O'Neill (1980), Greenberg and Maybee (1981), and Greenberg (1983) are early examples of this recognition within the mathematical programming community. Meeraus (1983) with GAMS and Geoffrion (1987) with structured modeling represent two methodological approaches that are still in development. In particular, structural modeling is rapidly evolving as an instructive environment for model development (Geoffrion 1988).

Concerns for methodology-related issues in discrete event simulation can be traced to the early work of Lackner (1962) and the seminal RAND Corporation reports of Kiviat (1967 and 1969). Zeigler (1976) and Nance (1977 and 1981) have drawn attention to methodological needs. Recent research in modeling methodologies for discrete event simulation has touched on the relationships with software engineering (Sheppard, Friel, and Reese (1984), and artificial intelligence (Oren and Zeigler (1979), Oren (1982), and Rozenblit and Zeigler (1985).

1.1.1 The Importance of a Methodology

The development of a small program or a small model requires little discipline and almost no supportive techniques. Both can be developed fairly rapidly and with little control on the conceptual representation and the eventual implementation. In part, this explains the criticism often leveled at new graduates in computer science or engineering: they do not know how to solve real world problems. Underlying this observation is the realization that only problems of sufficient size and complexity introduce difficulties in human communication, differences in concepts, organizational impedance, and the countless other "opportunities" for failure. The necessity for small problems that can be solved within the constraints of an academic semester gives rise to a false premise that those techniques successfully applied in this domain can be easily extrapolated into the domain of large, complex problems. The error of this perception is now quite apparent.

Despite recognizing the need for teaching methodology, the academic community is challenged by how to do so within traditional confines. A plausible recourse is likely to take form in project courses that extend beyond the campus

boundaries. Such a recourse should be recognized as an opportunity for more cooperation between university and industry; unfortunately, such recognition seems slow to develop.

1.1.2 The Variant Forms of Modeling Methodologies

Methodology designers must by necessity be slightly arrogant persons. Because they claim to understand *how* the task should be done, their approach often takes an evangelistic character. Different forms of the evangelism emerge, but the emphasis on *application domain* versus *solution technique* is a prominent differentiator. The former attempts to describe THE way to solve the problems in THIS application domain. The latter bases a methodology around the technique that is believed to be broadly, if not universally, applicable. Both forms offer advantages and impose deficiencies.

1.2 The Dual Roles of a Methodology

Ignoring for the moment a definition of the term, a methodology serves two roles. The first role is that of its *conceptual* contribution to understanding the development task, be it software- or model-oriented. In a subsequent section describing the conceptual role, we provide a characterization that extends well beyond the simple definition and, in so doing, crystallizes the expectations that differentiate a methodology from a simple method.

A characterization, based on the stipulation of relationships among *objectives*, *principles*, and *attributes*, constitutes the basis for evaluating development methodologies (see Arthur, Nance and Henry (1986), Arthur and Nance, (1987)). Such an evaluation procedure can be made largely objective, rather than subjective, and is applicable to the comparative assessment of methodology capabilities.

Methodology principles are the foundation for the second role: a practical design guide. A following section describes how methodology principles serve as a blueprint for development environment tools. Further, such principles define the interfaces among tools that are crucial in achieving an integrated development environment.

1.3 Illustration: The Conical Methodology

To transition the role description from an abstract to a more concrete level, the conical methodology serves as an example for both the conceptual and practical roles. The reader interested in obtaining an in-depth understanding might profit from consulting two references (Nance 1981), (Nance 1988).

The conical methodology (CM), developed specifically for simulation modeling tasks, prescribes a top-down model definition followed by a bottom-up model specification. The model definition phase resembles the classical object-oriented programming paradigm in that the modeler performs an object decomposition, assigning attributes to objects based on the system being described and the objectives of the simulation study. Attributes are strongly typed, and the CM advocates extraction of the maximum information from the modeler during the definition phase, e.g., attribute dimensions and value range definition along with attribute typing. The model specification phase utilizes the basically static representation produced in the definition phase to construct a dynamic representation through the changes in attribute values (and object states). Objects must be defined prior to specification.

For almost five years, the conical methodology has served as the blueprint for a model development environment project based on rapid prototyping. This work is described in several references, both for modeling tasks in general (Overstreet and Nance 1985), (Balci 1986) and for discrete event simulation in particular (Balci and Nance 1987).

2. THE CONCEPTUAL ROLE

We view the term "paradigm" as more philosophical and less prescriptive than "methodology." In that sense a paradigm may influence a methodology or may condition the stipulation of how a methodology is applied. With respect to the conical methodology, two paradigms exert significant influence: the previously mentioned object-oriented paradigm (OOP) and the entity-relationship (ER) paradigm (Chen 1976). Clear distinctions between these two do not exist, for the ER paradigm, which finds its roots in database management research, emphasizes many of the same concepts as the earlier OOP, which traces its roots to SIMULA.

The automation-based paradigm (Balzer, Cheatham and Green 1983, Balzer 1985) has had significant influence on the model development environment (MDE) research as well. In that sense, it might be said to have an indirect influence on the conical methodology interpretation in the realization of the MDE prototypes.

2.1 What is a Methodology?

Following the definition advanced in (Arthur, Nance and Henry 1986, page 4), we consider that a methodology should:

(1) organize and structure the tasks comprising the effort to achieve global objectives,

(2) include methods and techniques for accomplishing individual tasks (within the framework of global objectives),

(3) prescribe an order in which certain classes of decisions are made and the ways of making those decisions that lead to the desired objectives.

A methodology, in contrast with a method, is a collection of complementary methods and a set of rules for applying them. Viewed in the context of simulation, the conical methodology is intended to respond to the needs of all users: manager, modeler, analyst and programmer. Through identification of the objectives to be achieved in the simulation modeling task, the methodology guides the modeler in the execution of the process to achieve those objectives.

2.2 The Objective/Principle/Attributes Characterization

The above definition proves unsatisfactory in fully explaining the conceptual role of a methodology. More definitive is a characterization that takes the following form:

> An *objective* is "something aimed at or striven for." Within the modeling or software development context, an objective is an expression of a project desirable, i.e., a characteristic judged in a holistic fashion at the completion of a project. For example, the claim that a software system is reliable or that a model is valid.
>
> A *principle* describes that which is employed in the process to achieve one or more objectives. In a sense, a principle is a governing rule of "right conduct." For example, *abstraction* is a guiding principle of both model and software development.

Attributes are the intangible characteristics of product components. An attribute may be present or absent in any given part of the product, but an inherently intangible nature renders establishment of an attribute quite difficult. For example, one submodel might be quite understandable; yet another, perhaps developed by a different individual or team, could be very abstruse.

Tables 1, 2 and 3 identify the objectives, principles and attributes that are applicable to modeling methodologies. The O/P/A characterization expresses the foundational statement that every methodology should clearly set forth its objectives, define the principles necessary to achieve those objectives, and identify those attributes to be expected in the model that is produced following the stated principles. The characterization summarized in Tables 1, 2 and 3 explicates the conceptual role of a methodology.

Table 1: Modeling Methodology Objectives

- Adaptability: Accommodating changing requirements
- Correctness: Meets validation criteria
- Maintainability: Ability to correct inadequacies
- Portability: Transferability to different run-time hosts
- Reliability: Error-free use over time
- Reusability: Reuse of model components in other studies
- Testability: Ability to perform model verification and validation

Table 2: Modeling Methodology Principles

- Abstraction: Use of varying levels of descriptive detailed
- Concurrent Documentation: Maintaining document support throughout the lifecycle
- Functional Decomposition: Components partitioned along functional boundaries
- Hierarchical Decomposition: Components defined top-down
- Information Hiding: Insulating internal details of component behavior
- Iterative Refinement: Expanding detail of model behavior
- Lifecycle Verification: Progressive assurance of correct model transformations
- Progressive Elaboration: Addition of model descriptiveness

Table 3: Model Attributes Realized from Modeling Principles

- Cohesion: Locality of component description
- Controlled Complexity: Minimize efforts to utilize model components
- Early Error Detection: Identification of faults in design and specification prior to implementation
- Ease of Change: Capability for refinement or extensions
- Reduced Coupling: Minimize dependencies among model components
- Understandability: Ease in comprehending component representations
- Visibility of Behavior: Provision of review process for error checking
- Well Defined Interfaces: Clarity and completeness of a shared boundary between two model components

2.3 O/P/A Characterication of the Conical Methodology

While a methodology should not be expected to stress all the objectives listed in Table 1, it should set forth those objectives considered as primary. Five primary objectives are stressed in the conical methodology:

(1) Model correctness: the methodology proposes that model objectives define a level of tolerance by which comparison between model and system can enable the validation process to be performed.

(2) Testability: different model specifications should be comparable so that the evolution of the model following the lifecycle shown in Figure 1 is possible.

(3) Adaptability: changes in successive model specifications should be accomplished with relative ease so that extensibility is achieved without excessive cost.

(4) Reusability: model components should be extracted and made accessible for subsequent modeling tasks.

(5) Maintainability: model specifications should enable their modification to meet needs originally unstated.

To achieve these objectives, five principles are enunciated in the conical methodology. Their specific statement is shown in Table 4 along with the related modeling principles.

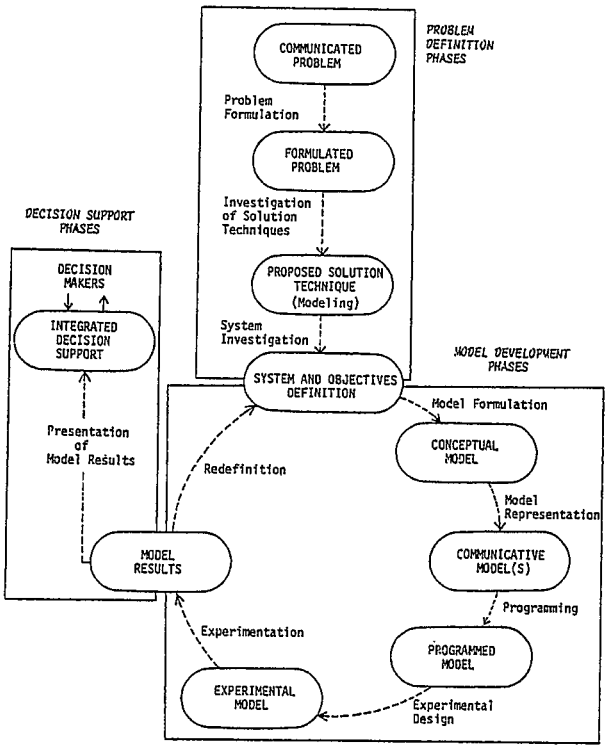

Figure 1. Phases in the Chronological Periods of the Model Life Cycle (Nance 1984).

Table 4. The Conical Methodology Principles

CM Principle	Modeling Methodology Principles
(1) Top-down model definition is followed by bottom-up model specification	Hierarchical Decomposition Functional Decomposition
(2) Documentation and specification are inseparable	Concurrent Documentation
(3) Iterative refinement and progressive elaboration are essential	Abstraction Information Hiding Hierarchical Decomposition Functional Decomposition Position Stepwise Refinement
(4) Verification must begin with communicative models and continue throughout the development process	Lifecycle Verification
(5) Model specification should be independent of model implementation	Abstraction Information Hiding

3. THE PRACTICAL ROLE OF A METHODOLOGY

Restricted to the conceptual role, many of us might express the opinion attributed to John Crookes by Balmer (1987):

> I find reading about modeling methodolgies very boring. Nevertheless, I continue to think that methodologies are quite important.

Those with a strong practitioner bent are clearly justified in asking, "but isn't there more?" Gelovani (1984, p.78), in describing an interactive modeling system for complex socio-economic models, states that:

> Clearly, existing modeling methodology for the range of problems under consideration offers no adequate modeling technique corresponding to practical requirements.

3.1 Methodology-Based Environments

The practical role of a methodology is embodied in the principles. The principles, which are described above as the *nucleus* of a methodology, serve also as the blueprint for a set of tools constituting an environment supporting that methodology. This claim certainly seems reasonable, for the governing principles employed in the process must be supported by computer-assisted tools if the process is to be accomplished properly. Further, the principles establish the basis for defining tool functionality and consequently the boundary interfaces that exist among tools. Without well defined boundaries, an *integrated* set of software support tools cannot be achieved.

3.2 Deriving a Model Development Environment Based on the Conical Methodology

Figure 2 shows the structure of a model development environment based on the conical methodology. The architecture of this environment has been described in several papers (Balci 1986), and a detailed description is not warranted here. Suffice it to say that the identified set of tools have clearly defined functionality, and communication is permitted only through the kernel interface. Consequently, prototypes of tools can be inserted and removed without the effects rippling through other tools, which might occur if direct communication were allowed.

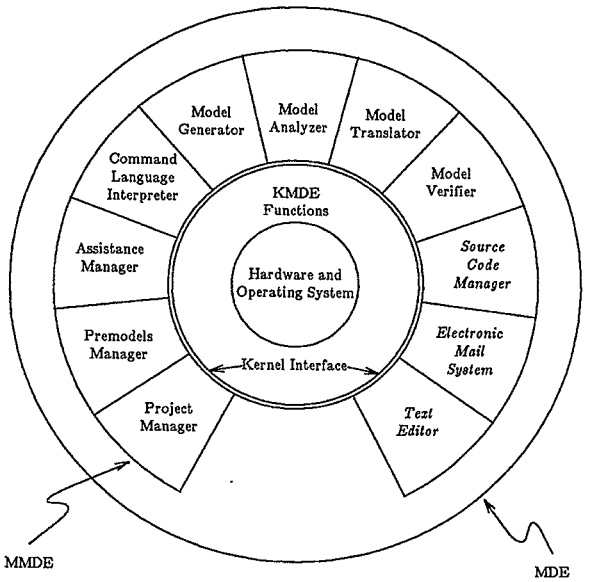

Figure 2. The structure of model development environments (Balci 1986)

Table 5. Procedural Guidance for Environment Design Tool Functionality

	Conical Methodology Principle	Procedural Guidance Derived From the CM Principle	Environment Tools Affected
1.	Top-down model definition/bottom-up model specification	1.1 Definition must precede specification	Model Generator Pre-models Manager
2.	Documentation and specification are inseparable	2.1 Model documentation is produced during model specification 2.2 The model specification and consequent documentation should support different views (aspects) of the modeling task	Assistance Manager Project Manager Model Generator
3.	Iterative refinement and progressive elaboration	3.1 The degree of detail of submodel description should be controlled by the modeler; submodel stubbing should be supported so that later addition of detail is facilitated 3.2 The functional expansion (progressive elaboration) of the model should be supported	Pre-Models Manager Model Generator
4.	Verification must begin with communicative models and continue throughout the development process	4.1 Diagnosis of model representation should begin as early as possible, certainly prior to the program form 4.2 Automated or semi-automated diagnosis is a requirement	Project Manager Model Analyzer Model Verifier
5.	Model specification is independent of model implementation	5.1 The execution (implementation) details should be ignored in the model development (specification) process	Model Generator Model Analyzer Model Translator Model Verifier

Table 5 shows the procedural guidance derived from each CM principle and identifies the particular tool(s) that are affected by this guidance.

4. CONCLUDING SUMMARY

The key points in this paper have emerged in the explication of the dual role of a methodology in the development process, be it software, model, or systems development. The objective/principle/attributes characterization of a methodology provides an explanation in terms of a foundational understanding that can impart an appreciation for both the conceptual and practical roles of a methodology. In either case the principles form the nucleus: the statements of "right rules of conduct" so that the task can be accomplished properly and effectively. Environments lacking a methodological foundation can be subject to difficulties in the partitioning of tool functionality and in the integration necessary to support large project needs.

ACKNOWLEGEMENTS:

The support of the U.S. Navy under BOA N 60921-83-G-A165 is gratefully acknowledged. The contributions of all members of the model development environment research group are acknowledged. Particular appreciation is expressed to Osman Balci for his sharing of many thoughts and comments on the above topics.

REFERENCES:

Alford, M. (1985). SREM at the age of eight: the distributed computing design system. *IEEE Computer* 18, 36-54.

Arthur, J.D., Nance, R.E. and Henry, S.M. (1986). A procedural approach to evaluating software development methodologies: the foundation. SRC-86-008, Systems Research Center, Virginia Tech, Blacksburg, Virginia.

Arthur, J.D. and Nance, R.E. (1987). Developing an automated procedure for evaluating software development methodologies and associated products. SRC-87-007, Systems Research Center, Virginia Tech, Blacksburg, Virginia.

Balci, O. (1986). Requirements for model development environments, *Computers and Operations Research 13,* 53-67.

Balci, O. and Nance, R.E. (1987). Simulation model development environments: a research prototype, *Journal of the Operational Research Society 38,* 753-763.

Balmer, D.W. (1987). Modeling styles and their support in the CASM environment. In: *Proceedings of the 1987 Winter Simulation Conference* (A. Thesen, H. Grant, and W.D. Kelton, eds.), 478-485.

Balzer, R., Cheatham, T.E. and Green, C. (1983). Software technology in the 1990's using a new paradigm, *IEEE Computer 16,* 39-45.

Balzer, R. (1985). A 15-year perspective on automatic programming, *IEEE Transactions on Software Engineering SE-11,* 1257-1268.

Basili, V.R. and Weiss, D.M. (1985). A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering 11,* 157-168.

Bergland, G.D. (1981). A guided tour of program design methodologies. *IEEE Computer 14,* 13-36.

Bronowski, J. (1973). *The Ascent of Man,* British Broadcasting Corporation.

Card, D.N., McGarry, F.E. and Page, G.T. (1987). Evaluating software engineering technologies. *IEEE Transactions on Software Engineering SE-13,* 845-854.

Chen, P.P. (1976). The entity-relationship model -- toward a unified view of data, *ACM Transactions on Database Systems 1*, 9-36.

Colter, M.A. (1982). Evolution of the structured methodologies. In: *Advanced System Development/Feasibility Techniques* (J.D. Couger, M.A. Colter, and R.W. Knapp, eds.) John Wiley, 74-95.

Gelovani, V.A. (1984). An interactive modeling system as a tool for analyzing complex socio-economic problems. In: *Models of Reality: Shaping Thought and Action*, Lomond Books.

Geoffrion, A.M. (1987). An introduction to structured modeling. *Management Science 33*, 547-588.

Geoffrion, A.M. (1988). SML: a model definition language for structured modeling. Working Paper No. 360, Western Management Science Institute, University of California, Los Angeles.

Greenberg, H.J. and Maybee, J.S. (1981). *Computer-Assisted Analysis and Model Simplification.* Academic Press, New York.

Greenberg, H.J. (1983). A functional description of ANALYZE: a computer-assisted analysis system for linear programming models. *ACM Transactions Mathematical Software 9*, 18-56.

Heninger, K.L., Kallender, J.W., Shore, J.E., and Parnus, D.L. (1978). Software requirements for the A-7E aircraft. NRL Memorandum Report 3876, Naval Research Laboratory, Washington, D.C.

Henry, S.M., Arthur, J.D. and Nance, R.E. (1985). A procedural approach to evaluating software development methodologies. SRC-85-008, Systems Research Center, Virginia Tech, Blacksburg, Virginia.

Kiviat, P.J. (1967). Digital computer simulation: modeling concepts, RAND Memorandum RM-5378-PR, Santa Monica, California.

Kiviat, P.J. (1969). Digital computer simulation: computer programming languages, RAND Report RM-5883-PR, Santa Monica, California.

Kurator, W.G. and O'Neill, R.P. (1980). An interactive system for mathematical programs. *ACM Transactions Mathematical Software 6*, 489-509.

Lackner, M.R. (1962). Toward a general simulation capability. In: *Proceedings of the SJCC*, AFIPS Press, 1-14.

Meeraus, A. (1983). An algebraic approach to modeling. *Journal Economic Dynamic Control 5*, 81-108.

Nance, R.E. (1977). The feasibility of and methodology for developing Federal documentation standards for simulation models. Final Report to the National Bureau of Standards, Department of Computer Science, Virginia Tech, Blacksburg, Virginia.

Nance, R.E. (1981). Model representation in discrete event simulation: the conical methodology. Technical Report CS81003-R, Department of Computer Science, Virginia Tech, Blacksburg, Virginia.

Nance, R.E. (1984). Model development revisited. In: *Proceedings of the 1984 Winter Simulation Conference* (S. Sheppard, U.W. Pooch and C.D. Pegden, eds.), 75-80.

Nance, R.E. (1988). The conical methodology: a framework for simulation model development. In: *Proceedings of the Conference on Methodology and Validation* (O. Balci, ed.), Simulation Series 19, Society for Computer Simulation, San Diego, California.

Oren, T.I. (1982). Computer aided modeling systems. In: *Progress in Modeling and Simulation* (F.E. Collier, ed.), Academic Press, London.

Oren, T.I. and Zeigler, B.P. (1979). Concepts for advanced simulation systems. *Simulation 32*, 69-82.

Overstreet, C.M. and Nance, R.E. (1985). A specification language to assist in analysis of discrete event simulation models. *Communications of the ACM 28*, 190-201.

Sawyer, S.K. and Dawson, C.A. (1984). Practical applications of software engineering theorems and methodologies to software system design problems. UCCEL Corporation, Dallas, Texas.

SofTech, Inc. (1976). *An Introduction to SADT: Structured Analysis and Design Technique,* Document No. BCS-10167.

Zeigler, B.P. (1976). *Theory of Modeling and Simulation,* John Wiley.

AUTHOR'S BIOGRAPHIES

Dr. Richard E. Nance is the Dahlgren Professor of Naval Computing Systems at Virginia Tech. Formerly the Director of the Systems Research Center, he serves as Project Manager for the study of Model Development Environments. He has chaired the TIMS College on Simulation and Gaming and the ACM Special Interest Group on Simulation (SIGSIM). He has held editorial positions involving simulation publications in *Operations Research* and *IIE Transactions*, and currently serves on the editorial panel of the *Communications of the ACM* for research contributions in simulation modeling and statistical computing. He is the area editor for simulation of the *ORSA Journal on Computing*.

Dr. Richard E. Nance
Department of Computer Sciences
562 McBryde Hall
Virginia Tech
Blacksburg, Virginia 24061, U.S.A.
(703) 961-6144

Dr. James D. Arthur received his B.S. and M.A. from the Department of Mathematics at the University of North Carolina in 1972 and 1973, respectively. After spending several years in industry, Mr. Arthur continued his formal education at Purdue University and received the M.S. and Ph.D. degrees in Computer Science in 1981 and 1983, respectively. Since 1983 Dr. Arthur has pursued a research and instructional career as as Assistant Professor of Computer Science at Virginia Tech. His research interests include software engineering, user support environments, human/machine interaction, and programming languages and systems. Dr. Arthur is a member of ACM and IEEE CS.

Dr. James D. Arthur
Department of Computer Sciences
562 McBryde Hall
Virginia Tech
Blacksburg, Virginia 24061, U.S.A.
(703) 961-7538